9.1 Analytic Tableaux

Truth tables were based on the definition that *a formula is a tautology if it is true under every interpretation*. However, using truth tables for evaluating the truth of a formula quickly becomes infeasible. For larger formulas, we should therefore look for a better method – something that is fairly schematic, but doesn't rely checking individual valuations anymore.

Instead, it is better to reason about truth and falsehood as such and to *analyze* the conditions for the truth of a formula under an interpretation based on what we know about its subformulas. For this purpose let us rephrase the axioms for boolean valuations in terms of truth and falsehood.[1]

B$_1$:: If $\neg X$ is true then X is false
If $\neg X$ is false then X is true

B$_2$:: If $(X \wedge Y)$ is true then X and Y are true
If $(X \wedge Y)$ is false then X is false or Y is false

B$_3$:: If $(X \vee Y)$ is true then X is true or Y is true
If $(X \vee Y)$ is false then X and Y are false

B$_4$:: If $(X \Rightarrow Y)$ is true then X is false or Y is true
If $(X \Rightarrow Y)$ is false then X is true and Y is false

So, to prove a formula X true, we look at its outer structure, apply the appropriate axiom and then check the remaining conditions for the subformulas. Let us look at a simple example.

Example 9.1 *How would you prove $P \Rightarrow P$?*

We know that $P \Rightarrow P$ is true if P is false or P is true. And this is the case for any interpretation v_0, since v_0 can only assign `f` or `t` to the variable P. □

Yat was easy. But actually, instead of proving a formula true, it is even easier to prove that it cannot be false, that is to assume that X is false and to derive a contradiction from that assumption. Sometimes this is called an *indirect proof* or a *refutation proof*.

How would that work for $P \Rightarrow P$?

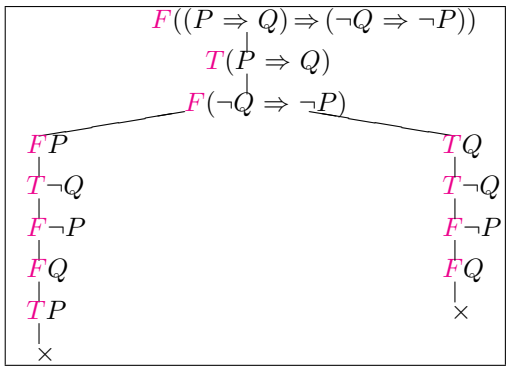
If we assume that $P \Rightarrow P$ is false then P must be both true and false, and this certainly cannot be the case.

The *Tableaux Method* is schematic version of the technique we just applied. It replaces the statement "X *is true*" by the *signed formula* TX and the statement "X *is false*" by the signed formula FX. It replaces logical reasoning by the schematic application of syntactic maniulations to a signed formula and takes the rules for doing that from the above observations. Before we formulate these rules, let us look at an example.

[1] Note that these are actually equivalences.

Example 9.2 To prove the formula $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$, we assume it to be false and see if we can derive a contradiction from that. So we start with the signed formula

$F((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P))$



Keeping in mind that FX stands for "X *is false*", we use observation B_4 and derive $T(P \Rightarrow Q)$ and $F(\neg Q \Rightarrow \neg P)$, which we write below the initial signed formula. Smullyan calls these two formulas *direct consequences* of the signed formula $F((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P))$.
Look at the first of these two formulas. According to B_4 it is true, if either P is false or Q is true. This means we have to investigate two possibilities independently. We denote that
by opening two *branches*, one for each argument that we have to follow.

Next we use the second of the new formulas. Again we use B_4 and derive $T\neg Q$ and $F\neg P$, which we write as established facts into each of the two branches.

Let's look at the left branch and pick the first signed formula that we can still decompose: $T\neg Q$. Using B_1 we derive FQ and in the same way we derive TP from $F\neg P$. Let us look at the branch a little closer.

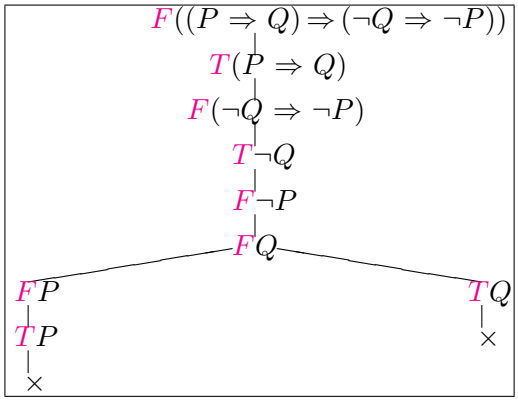
We see both FP and TP in that branch, that is a signed formula and its *conjugate* (i.e. the same basic formula with a different sign – Smullyan denotes conjugates by $\bar{F}$). That means, we have shown than P is both false and true. Since both are consequences of our initial assumption, the branch is *contradictory* and we *close* it by marking it with a $\times$.

But we're not done yet. We still have to look at the other branch, because this branch described just one of two possibilities. So we decompose $T\neg Q$ using B_1 and derive FQ. Now we also have a contradiction in this branch and check it off as well.

We now have shown that the assumption that $((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P))$ is false definitely leads to a contradiction, so $((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P))$ must be true.

□

Note that we have used a top down order to decompose formulas. This is not necessary. We could also decompose the formula $F(\neg Q \Rightarrow \neg P)$ first because we use its consequences in both branches. For the same reason we would then decompose $T\neg Q$ before we finally branch by decomposing $T(P \Rightarrow Q)$.
The resulting proof would look like this

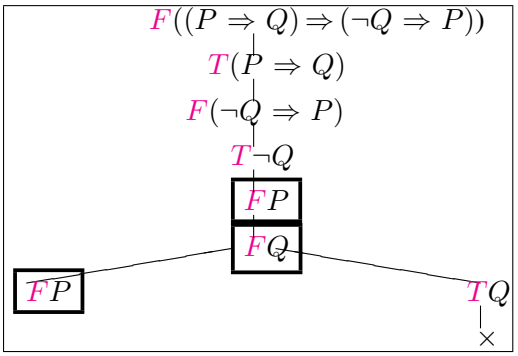


Example 9.3 Let us look at another example, where we change the formula just a little bit to $((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow P))$. Is this formula still valid?

Let us see what our method reveals. We begin as before and get almost the same tableau. However, we cannot close off the left branch, since we cannot decompose FP while we could decompose $F\neg P$ before.

What does that mean?

Since we decomposed all the formulas, there is nothing else we can do, which means that our initial assumption is not contradictory.



But there is more information in the tableau. All the signed formulas in the branch are consequences of the assumption that $((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow P))$ is false. That means that $(P \Rightarrow Q)$ must be true, $(\neg Q \Rightarrow P)$ must be false, $\neg Q$ must be true, and P and Q must be false.

So the tableau does not only show that the formula is not a tautology, but also gives us a *counterexample*, i.e. an interpretation of the variables that makes the formula false. □

9.2 Rules for the construction of Tableaux

Let us review what we have just done. We have applied a schematic method that decomposes signed formulas according to the 4 axioms of boolean valuations. No semantic arguments were involved – just a "stupid" application of *syntactic rules*. Smullyan states all these rules on page 17 of the book: for each connective there are two rules, one for the sign T and one for F.

1) $\dfrac{T\neg \mathrm{X}}{F\mathrm{X}}$ $\quad$ $\dfrac{F\neg \mathrm{X}}{T\mathrm{X}}$

2) $\dfrac{T(\mathrm{X} \wedge \mathrm{Y})}{T\mathrm{X}\\ T\mathrm{Y}}$ $\quad$ $\dfrac{F(\mathrm{X} \wedge \mathrm{Y})}{F\mathrm{X} \mid F\mathrm{Y}}$

3) $\dfrac{T(\mathrm{X} \vee \mathrm{Y})}{T\mathrm{X} \mid T\mathrm{Y}}$ $\quad$ $\dfrac{F(\mathrm{X} \vee \mathrm{Y})}{F\mathrm{X}\\ F\mathrm{Y}}$

4) $\dfrac{T(\mathrm{X} \Rightarrow \mathrm{Y})}{F\mathrm{X} \mid T\mathrm{Y}}$ $\quad$ $\dfrac{F(\mathrm{X} \Rightarrow \mathrm{Y})}{T\mathrm{X}\\ F\mathrm{Y}}$

These rules express exactly the same as the axioms B_1 to B_4, but now in schematic form and as instructions for performing *inferences*.

The first rule, for instance, states that from $T\neg$X we can directly infer FX and *add this signed formula to any branch passing through* $T\neg X$.

The rule for $T(\mathrm{X} \wedge \mathrm{Y})$ states that we can infer both TX and TY and thus add both formulas to any branch passing through $T(\mathrm{X} \wedge \mathrm{Y})$.

In contrast to that, the rule for $F(\mathrm{X} \wedge \mathrm{Y})$ states that either FX or FY can be inferred. Thus the tableau has to branch if we apply the rule: to any branch passing through $F(\mathrm{X} \wedge \mathrm{Y})$ we may add two new branches at the leaf – one containing FX and the other FY.

So for the propositional calculus we have two types of rules – one that adds *direct consequences* and one that *branches*. Smullyan assigns two *types* to the corresponding signed formulas. Type A (or α) to those that result in direct consequences (also called *conjunctive type*) and type B (or β) to those that cause branching (also called *disjunctive type*).

Why does he do that?

Assigning types to signed formulas helps reducing inference rules to their essential effect, namely adding direct consequences or branching. When we later prove that the tableaux method actually works correctly, it makes quite a difference whether we have to consider 2 cases or 8. When you try to implement the method on a computer, it makes a difference whether you have to program 2 basic methods for manipulating tableaux or 8 methods, among which many are almost identical.

So Smullyan introduces a *uniform notation* for signed formulas, denoting formulas of type A by the symbol α and formulas of type B by the symbol β. For each of these formulas he defines component formulas α_1, α_2 and β_1, β_2 respectively as follows (see tables on page 21).

α	$T(X \wedge Y)$	$F(X \vee Y)$	$F(X \Rightarrow Y)$	$T\neg X$	$F\neg X$
α_1	TX	FX	TX	FX	TX
α_2	TY	FY	FY	–	–

β	$F(X \wedge Y)$	$T(X \vee Y)$	$T(X \Rightarrow Y)$
β_1	FX	TX	FX
β_2	FY	TY	TY

These are exactly the formulas that we find in the corresponding rules, so we we know that – under any interpretation – a signed formula α is true iff both α_1 and α_2 are true and β is true iff at least one of β_1, β_2 is true. Thus we can summarize the 8 rules by the following two:

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

What is the point here?

Logic is much about abstraction, reducing logical reasoning to the essentials. Using an abstract language of formulas instead of informal text is one step. Using schematic rules to create proofs is the next. Finding an elegant representation of the proof system of tableaux leads us even further.

There are more steps that one could go if one were to build an efficient proof system in a computer, but we'll get to that later. The current form is the most appropirate one for another important task: showing that the tableaux method is correct and sufficient – i.e. that only true formulas can be proven and that we can in fact find a tableaux proof for every true formula. We will look into these issues in one of the next lectures.

9.3 A definition of tableaux

Now that we have explained how the tableaux method works, let us define tableaux precisely.

Definition 9.4 *An analytic tableau $\mathcal{T}$ for a signed formula X is a dyadic ordered tree, whose points are formulas and that satisfies the following conditions.*

(1) *The root of $\mathcal{T}$ is X.*

(2) *If a node y has one successor then there is some α on the path P_y and the successor of y is α_1 or α_2.*

(3) *If a node y has two successors then there is some β on the path P_y and β_1, β_2 are the successors of y.*

A branch ϑ of a tableau $\mathcal{T}$ is closed *if it contains a formula and its conjugate (or its negation, for unsigned formulas). A* tableau $\mathcal{T}$ is closed *if all its branches are closed.*

A proof *of an unsigned formula X is a closed tableau for FX (or $\neg X$).*

A tableau $\mathcal{T}_2$ is a direct extension *of $\mathcal{T}_1$ if $\mathcal{T}_2$ is the result of applying the α-rule or the β-rule to one of the branches of $\mathcal{T}_1$.*

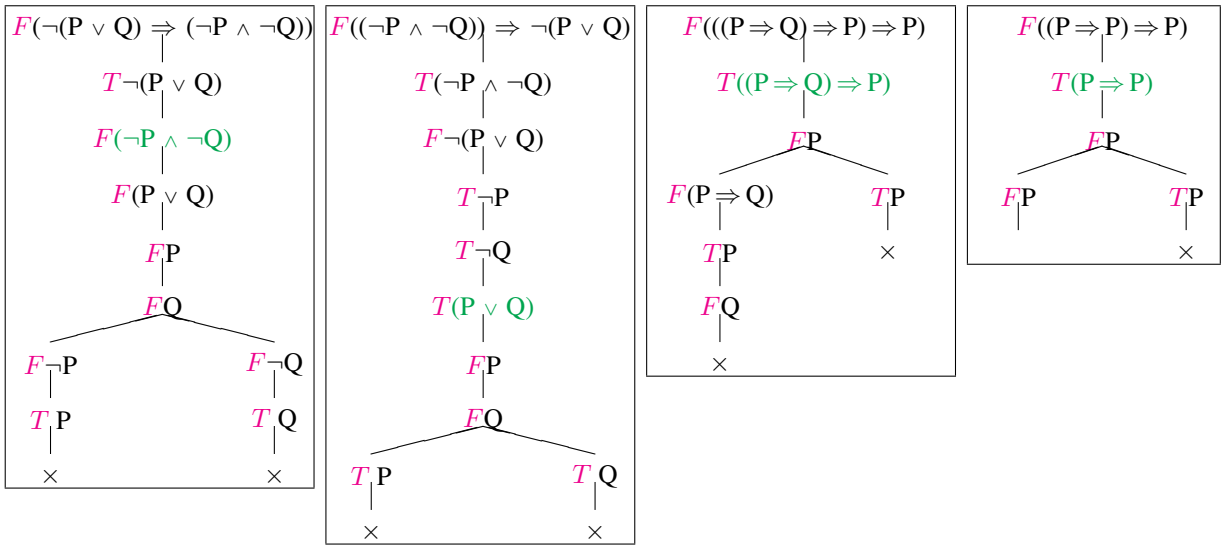
A branch ϑ of a tableau $\mathcal{T}$ is complete *if for every α on ϑ both α_1 and α_2 occur on ϑ and if for every β on ϑ at least one of β_1, β_2 occur on ϑ. A* tableau $\mathcal{T}$ is complete *if all its branches are either closed or complete.*

9.4 Proof Strategies

A tableaux proof for a formula X is constructed by iteratively extending the root tableau FX until it is complete. If the resulting tableau is closed, the formula is true, otherwise it is not. In one of the next lectures we will prove this fact.

A tableau can be extended by applying either an α-rule or a β-rule. We have a certain degree of freedom here, since we may pick an arbitrary branch and an arbitrary α- or a β-formula on this branch. Once a formula has been used, it doesn't make sense to use it again on the same branch. Since every rule decomposes a formula into smaller components (i.e. reduces the degree of the sub-components), the extension process must eventually terminate.

There are several proof strategies one may follow. The simplest one is to just *decompose the formulas in top down order* and mark them off as used. However, as our initial examples showed, it is usually more effective to *decompose α-formulas first*, since this will keep the proof from branching too early. The following examples show how this strategy works (β nodes are marked green). On the left there are proofs for $(\neg(P \vee Q) \Rightarrow (\neg P \wedge \neg Q))$ and it's reverse $((\neg P \wedge \neg Q)) \Rightarrow \neg(P \vee Q)$.

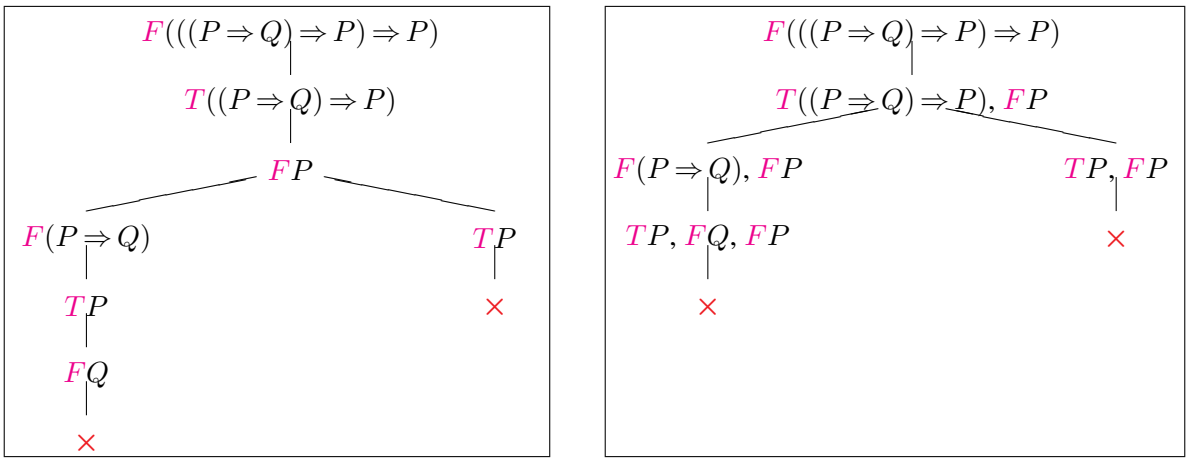


The third example is $(((P \Rightarrow Q) \Rightarrow P) \Rightarrow P)$. Before we try to prove it or find a counterexample, let us try to make sense of it. Who would believe that "*if the fact that P implies something implies P, then P is true*" is actually true? But the above proof shows that it actually is valied and that the Q didn't really matter in the proof. Does that mean we can simply remove the Q? No, since in this case we get FP as a counterexample.

9.5 Block Tableaux

One of the disadvantages of the original tableau method from the perspective of a human user is that it is difficult to keep track of those formulas in the proof tree that still can be decomposed, thus generating new signed formulas that may be essential for closing the current branch. One essentially has to look at the whole path in the proof tree between the root and the current proof node and determine which of these nodes haven't been used yet.

For interactive use it would be much better to redesign the calculus in a way that it supports *local* reasoning. Essentially this means that we add all the formulas to a proof node that are above it in the tree but haven't been decomposed yet and can still be analyzed. So what we essentially do is to use a tableau calculus that operates on *sets of signed formulas* instead of a single one and doesn't require a user to look anywhere else but at the current proof node. The following example illustrates the difference

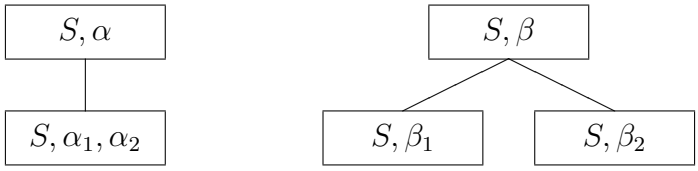


As we can see, the block tableau proof is even shorter because it keeps the two formulas generated by most α-rules within the formula set of a single successor node. Apart from the the proofs are essentially the same.

The formal modifications for the block tableau calculus are simple.

(1) The root of a tableau tree is now a finite set of formulas, usually the set consisting of the one original formula to be refuted.

(2) α and β rules operate on sets with a distinguished formula



In these rules, the comma stands for set union, so the α and β might be elements of S

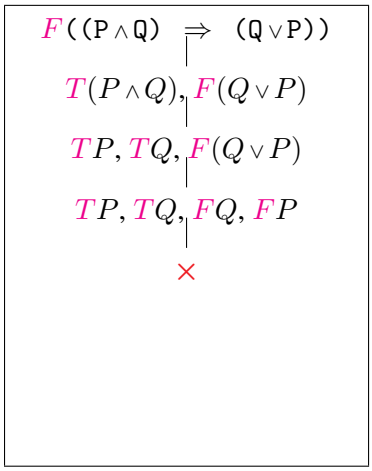
(3) A block tableau is *closed* if each end point contains a formula and its complement and it is *atomically closed* if it contains a (signed) variable and its complement.

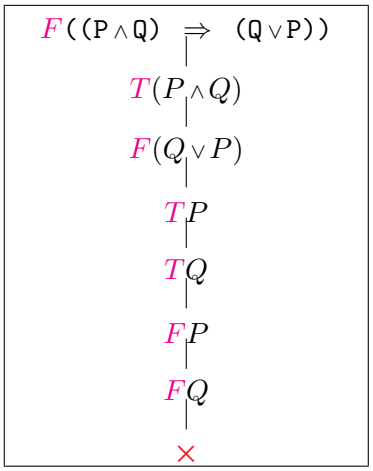
Smullyan uses two α-rules, one that adds only α_1 and the other adds only α_2. But this is not really necessary because adding the other α-successor does not affect whether a branch can be closed or not.
If we spell out these rules for each logical connective and write the rules in a less graphical fashion, but simply use one line for each generated branch, we get the table of rules that I gave you in the handout. The condition for closing a tableau can also be formulated as rule that closes a branch, because it creates no successor.

	T	F	
α	$S, T\mathrm{A} \wedge \mathrm{B}$ $S, T\mathrm{A}, T\mathrm{B}$	$S, F\mathrm{A} \wedge \mathrm{B}$ $S, F\mathrm{A}$ $S, F\mathrm{B}$	β
β	$S, T\mathrm{A} \vee \mathrm{B}$ $S, T\mathrm{A}$ $S, T\mathrm{B}$	$S, F\mathrm{A} \vee \mathrm{B}$ $S, F\mathrm{A}\ , F\mathrm{B}$	α
β	$S, T\mathrm{A} \Rightarrow \mathrm{B}$ $S, F\mathrm{A}$ $S, T\mathrm{B}$	$S, F\mathrm{A} \Rightarrow \mathrm{B}$ $S, T\mathrm{A}, F\mathrm{B}$	α
α	$S, T\neg\mathrm{A}$ $S, T\mathrm{A}$	$S, F\neg\mathrm{A}$ $S, F\mathrm{A}$	α
$*$	$S, T\mathrm{A}, F\mathrm{A}$		

As the above example demonstrated, a conventional tableau for a single formula an immediately be converted into a block tableau for that formula. One just collects all the formulas that have been generated before and not yet been decomposed into a set of formulas for the block tableau. Apart from that and the fact that α-rules in the block tableau generate only one node where the conventional tableau creates two, the block tableau is a *one-to-one simulation* of the analytic tableau, because we apply the same rules to the same nodes. If the convential tableau is closed, so is the block tableau. If it is open, so is the block tableau. Thus *every analytic standard tableau can be simulated by a block tableau* and as a result of that *block tableaux are complete*
Why?
because due to completeness of analytic tableau every valid formula has a closed analytic tableau and hence a closed block tableau.

Let us look at the other direction and show that *every block tableau can be simulated by an analytic standard tableau*. Let us look at another example





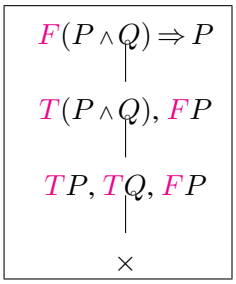
Again we use the same rules at the same nodes in both tableau and notice that -rules in analytic standard create two successors instead of the one generated in the block tableau. Thus *block tableaux are consistent* since every formula that has a closed block tableau also has a closed analytic tableau and thus, by consistency of analytic tableau, is valid.

Thus we have shown that *the block tableau calculus is consistent and complete* and thus provided a foundation for proving Gentzen systems consistent and complete.

9.6 Transforming tableaux proofs into Refinement logic proofs

The tableaux proof method is an elegant approach to finding proofs for the validity of formulas. However, it is rooted in truth table semantics and does not provide the necessary evidence. But we can observe a certain similarity between tableaux proofs and proofs in refinement logic.

- Both calculi construct a proof tree by decomposing a goal into subgoals until each branch can be closed.




```
⊢ (P∧Q)⇒P     by impliesR
1. P∧Q ⊢ P     by andL
1.1. P,Q ⊢ P   by axiom
```

  Often, the proofs have a very similar structure.

- Both calculi have two rules per connective and one rule to close off a branch.

T		F	
α	$S, T\mathrm{A} \wedge \mathrm{B}$ $S, T\mathrm{A}, T\mathrm{B}$	$S, F\mathrm{A} \wedge \mathrm{B}$	β $S, F\mathrm{A}$ $S, F\mathrm{B}$

left		right	
andL	$H, A \wedge B, H' \vdash C$ $H, A, B, H' \vdash C$	$H \vdash A \wedge B$	andR $H \vdash A$ $H \vdash B$

  In the tableaux calculus there is one rule for a connective with sign T one for the sign F. In refinement logic one rule operates on the assumptions (left) and one on the conclusion (right). If we compare the rules, we notice a strong similarity if we associate the T-formulas with assumptions and F-formulas with the conclusion.

- Using the above association both calculi have the same initial goal. In the tableaux calculus it has the form FX, where X is the formula to be proven, while in refinement logic it has the form $\vdash X$.

But there are also differences between the tableaux calculus and refinement logic.

- Block tableaux proofs are based on sets of signed formulas while refinement logic operates on lists. This is a technicality, as we usually write sets in the form of lists, but it causes a slight difference in the notation for the rules. When using lists, we have to indicate the position of the target formula in the assumptions and we also have to explicitly mention this formula in the subgoal sequents if we want it to be preserved. In set notation the order doesn't matter and we also may assume that the target formula occurs in the remainng set of formulas as well so there is no need to repeat it.

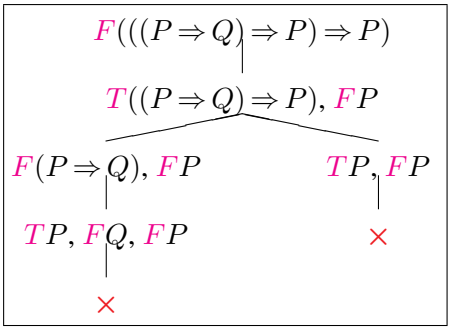
  In a sense, sets are a simpler notation for doing proofs on paper. But for implementation purposes tableaux rules need to be written in the form of lists ayway.

- The main difference between the calculi is the underlying semantics. The tableaux calculus is based on a boolean (truth-table) semantics, which contains the implicit assumption that every proposition is decidable, i.e. that $P \vee \neg P$ holds for every propositional variable. Refinement logic is based on evidence semantics and does not make that assumption. However, if we explicitly add the assumption $P \vee \neg P$ for every variable that occurs in a formula, then the semantical foundations become the same.
- The difference in semantics is also reflected in the fact that tableaux rules may create multiple F-formulas in the nodes of a proof tree while refinement proofs are only permitted to have a single conclusion. This is more than a technicality, since the refinement rules `orR1`, `orR2`, `impliesL`, and `notL` have to drop one of the conclusions when the corresponding tableaux rule introduces an additional F-formula.

	T	F	
α	$S, T\mathrm{A} \wedge \mathrm{B}$ $S, T\mathrm{A}, T\mathrm{B}$	$S, F\mathrm{A} \wedge \mathrm{B}$ $S, F\mathrm{A}$ $S, F\mathrm{B}$	β
β	$S, T\mathrm{A} \vee \mathrm{B}$ $S, T\mathrm{A}$ $S, T\mathrm{B}$	$S, F\mathrm{A} \vee \mathrm{B}$ $S, F\mathrm{A}$, $F\mathrm{B}$	α
β	$S, T\mathrm{A} \Rightarrow \mathrm{B}$ $S, F\mathrm{A}$ $S, T\mathrm{B}$	$S, F\mathrm{A} \Rightarrow \mathrm{B}$ $S, T\mathrm{A}, F\mathrm{B}$	α
α	$S, T\neg\mathrm{A}$ $S, F\mathrm{A}$	$S, F\neg\mathrm{A}$ $S, T\mathrm{A}$	α
*	$S, T\mathrm{A}, F\mathrm{A}$		

	left	right	
`andL`	$H, A \wedge B, H' \vdash C$ $H, A, B, H' \vdash C$	$H \vdash A \wedge B$ $H \vdash A$ $H \vdash B$	`andR`
`orL`	$H, A \vee B, H' \vdash C$ $H, A, H' \vdash C$ $H, B, H' \vdash C$	$H \vdash A \vee B$ $H \vdash A$, $\boxed{B}$ $H \vdash A \vee B$ $H \vdash B$, $\boxed{A}$	`orR1` `orR2`
`impliesL`	$H, A \Rightarrow B, H' \vdash C$ $H, A \Rightarrow B, H' \vdash A$, $\boxed{C}$ $H, B, H' \vdash C$	$H \vdash A \Rightarrow B$ $H, A \vdash B$	`impliesR`
`notL`	$H, \neg A, H' \vdash C$ $H, \neg A, H' \vdash A$, $\boxed{C}$	$H \vdash \neg A$ $H, A \vdash \mathrm{f}$	`notR`
		$H, A, H' \vdash A$	`axiom`

Since the conclusions dropped by the refinement rules may be necessary to complete a proof, there are formulas that can be proven by tableaux but not in refinement logic. A typical example is Peirce's Law $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$. It has a simple tableaux proof but any proof attempt in refinement logic will get stuck.




```
      ⊢ ((P ⇒ Q) ⇒ P) ⇒ P          by impliesR
1     (P ⇒ Q) ⇒ P ⊢ P              by impliesL

1.1   (P ⇒ Q) ⇒ P ⊢ P ⇒ Q          by impliesR
1.1.1 (P ⇒ Q) ⇒ P, P ⊢ Q           by ???

1.2   P ⊢ P                        by axiom
```

The refinement proof cannot succeed since the application of `impliesL` had to drop the formula P in the first subgoal, while P is used to close the corresponding branch in the tableaux proof. However, if we add the assumption $P \vee \neg P$ that is implicitly used in the tableau proof, we will be able to complete the refinement proof.

$P \vee \neg P \vdash ((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$ by impliesR
1 $P \vee \neg P, (P \Rightarrow Q) \Rightarrow P \vdash P$ by orL
1.1 $P, (P \Rightarrow Q) \Rightarrow P \vdash P$ by axiom
1.2 $\neg P, (P \Rightarrow Q) \Rightarrow P \vdash P$ by impliesL
1.2.1 $\neg P, (P \Rightarrow Q) \Rightarrow P \vdash P \Rightarrow Q$ by impliesR
1.2.1.1 $\neg P, (P \Rightarrow Q) \Rightarrow P, P \vdash Q$ by notL
1.2.1.1.1 $\neg P, (P \Rightarrow Q) \Rightarrow P, P \vdash P$ by axiom
1.2.2 $\neg P, P \vdash P$ by axiom

Note that this proof uses the same proof steps as the tableau proof but requires a few extra steps to mimic the use of multiple F-formulas. Via `orL` it makes explicit use of the decidability of the variables that occur in the conclusion that will be dropped by `impliesL` and proves the conclusion under the assumption P.

The essential difference to the failed proof attempt is that the conclusion P that is dropped by `impliesL` is now preserved in negated form as assumption and will be restored via `notL` when it is needed.

Using the negation of a conclusion C as assumption (or the negation of an F-formula as T-formula) is a simple trick to avoid multiple conclusions without sacrificing completeness. It is justified under the assumption that every proposition is decidable, which in turn is the case if every variable in that proposition is decidable.[2] Since this is the implicit assumption of the tableaux calculus every tableaux proof can be converted into a refinement logic proof of similar structure if these assumptions are made explicit in the initial goal of the proof.

Instead of stating the decidability of all propositional variables at the beginning of a refinement proof we could also provide an additional proof rule that introduces these assumptions *on demand*, i.e. immediately before an application of `impliesL`, `notL`, `orR1`, or `orR2`. This will give us an opportunity ignore decidability assumptions that are not relevant for the proof. We will call this rule `Use_decidability_of` P[3] , where P is the proposition whose decidability we want to use.

$H \vdash C$ by `Use_decidability_of` P
$H, P \vee \neg P \vdash C$

With this additional rule, we can describe a straighforward simulation of tableaux proofs in (the now extended) refinement logic.

(1) In a first step we convert the block tableaux rules into rules that generate proofs with only one F-formula. Since the initial goal has the form FX we only have to make sure that the rules single out the F-formula (we denote the set of T-formulas by S_T) and that they do not introduce additional ones. The latter is accomplished by converting additional F-formulas into T-formulas by negating them: a formula FC is converted into $T\neg C$ and $F\neg C$ is converted into TC. This step can be justified by the negation rules of the tableaux calculus.

In the case of the $T \Rightarrow$ and $T\neg$-rules the existing F-formula will be moved to make room for the new one. In the case of $F \vee$, one of the two alternatives will have to be moved, which leads to two rules that are redundant, but with a different focus.

[2] We know from the homework that decidablity is preserved under conjunction, disjunction, implication, and negation. By induction over the structure of formulas we can show that every formula is decidable if all its variables are.

[3] This rule used to be called `magic` but the name `Use_decidability_of` P seems to be more fitting.

	T	F	
α	$S, T\,A \wedge B$ $S, T\,A, T\,B$	$S, F\,A \wedge B$ $S, F\,A$ $S, F\,B$	β
β	$S, T\,A \vee B$ $S, T\,A$ $S, T\,B$	$S, F\,A \vee B$ $S, F\,A\ ,\ F\,B$	α
β	$S, T\,A \Rightarrow B$ $S, F\,A$ $S, T\,B$	$S, F\,A \Rightarrow B$ $S, T\,A, F\,B$	α
α	$S, T\neg A$ $S, F\,A$	$S, F\neg A$ $S, T\,A$	α
$*$	$S, T\,A, F\,A$		

$\mapsto$

	T	F	
α	$S_T, T\,A \wedge B, F\,C$ $S_T, T\,A, T\,B, F\,C$	$S_T, F\,A \wedge B$ $S_T, F\,A$ $S_T, F\,B$	β
β	$S_T, T\,A \vee B, F\,C$ $S_T, T\,A, F\,C$ $S_T, T\,B, F\,C$	$S_T, F\,A \vee B$ $S_T, T\neg A\ ,\ F\,B$ $S_T, F\,A \vee B$ $S_T, T\neg B, F\,A$	α
β	$S_T, T\,A \Rightarrow B, F\,C$ $S_T, T\neg C\ ,\ F\,A$ $S_T, T\,B, F\,C$	$S_T, F\,A \Rightarrow B$ $S_T, T\,A, F\,B$	α
α	$S_T, T\neg A, F\,C$ $S_T, T\neg C\ ,\ F\,A$	$S_T, F\neg A$ $S_T, T\,A$	α
$*$	$S_T, T\,A, F\,A$		

(2) The second step is a simple syntax translation. We separate the T-formulas from the F-formulas by putting a $\vdash$ between them and then drop the signs. The empty list of F-formulas generated by the $F\neg$-rule is represented by f. This leads to the following set of rules.

	T	F	
α	$S_T, T\,A \wedge B, F\,C$ $S_T, T\,A, T\,B, F\,C$	$S_T, F\,A \wedge B$ $S_T, F\,A$ $S_T, F\,B$	β
β	$S_T, T\,A \vee B, F\,C$ $S_T, T\,A, F\,C$ $S_T, T\,B, F\,C$	$S_T, F\,A \vee B$ $S_T, T\neg A\ ,\ F\,B$ $S_T, F\,A \vee B$ $S_T, T\neg B, F\,A$	α
β	$S_T, T\,A \Rightarrow B, F\,C$ $S_T, T\neg C\ ,\ F\,A$ $S_T, T\,B, F\,C$	$S_T, F\,A \Rightarrow B$ $S_T, T\,A, F\,B$	α
α	$S_T, T\neg A, F\,C$ $S_T, T\neg C\ ,\ F\,A$	$S_T, F\neg A$ $S_T, T\,A$	α
$*$	$S_T, T\,A, F\,A$		

$\mapsto$

	left	right	
α	$S, A \wedge B \vdash C$ $S, A, B \vdash C$	$S \vdash A \wedge B$ $S \vdash A$ $S \vdash B$	β
β	$S, A \vee B \vdash C$ $S, A \vdash C$ $S, B \vdash C$	$S \vdash A \vee B$ $S, \neg A \vdash B$ $S \vdash A \vee B$ $S, \neg B \vdash A$	α
β	$S, A \Rightarrow B \vdash C$ $S, \neg C \vdash A$ $S, B \vdash C$	$S \vdash A \Rightarrow B$ $S, A \vdash B$	α
α	$S, \neg A \vdash C$ $S, \neg C \vdash A$	$S \vdash \neg A$ $S, A \vdash \mathsf{f}$	α
$*$	$S, A \vdash A$		

(3) In the third step we translate the set-representation into list form (a trivial step) and provide "implementations" of the resulting rules in the extended refinement logic. The rules $T\wedge$, $F\wedge$, $T\vee$, $F\Rightarrow$, $F\neg$, and * correspond directly to the rules `andL`, `andR`, `orL`, `impliesR`, `notR`, and `axiom`. For the other rules we provide small proof fragments that generate the same subgoals and thus make it possible to simulate each application of a tableaux *locally* rule by a corresponding fragment in refinement logic.

$F\vee$**1:** $H \vdash A \vee B$ by `Use_decidability_of` A

1 $H,\ A \vee \neg A \vdash A \vee B$ by `orL`

1.1 $H,\ A \vdash A \vee B$ by `orR1 THEN axiom`

1.2 $H,\ \neg A \vdash A \vee B$ by `orR2`

1.2.1 $H,\ \neg A \vdash B$

$F\vee$**2:**	$H \vdash A \vee B$	by Use_decidability_of B
1	$H,\ B \vee \neg B \vdash A \vee B$	by orL
1.1	$H,\ B \vdash A \vee B$	by orR2 THEN axiom
1.2	$H,\ \neg B \vdash A \vee B$	by orR1
1.2.1	$H,\ \neg B \vdash A$	
$T\Rightarrow$**:**	$H,\ A \Rightarrow B,\ H' \vdash C$	by Use_decidability_of C
1	$H,\ A \Rightarrow B,\ H',\ C \vee \neg C \vdash C$	by orL
1.1	$H,\ A \Rightarrow B,\ H',\ C \vdash C$	by axiom
1.2	$H,\ A \Rightarrow B,\ H',\ \neg C \vdash C$	by impliesL
1.2.1	$H,\ A \Rightarrow B,\ H',\ \neg C \vdash A$	
1.2.2	$H,\ B,\ H',\ \neg C \vdash C$	
$F\neg$**:**	$H,\ \neg A,\ H' \vdash C$	by Use_decidability_of C
1	$H,\ \neg A,\ H',\ C \vee \neg C \vdash C$	by orL
1.1	$H,\ \neg A,\ H',\ C \vdash C$	by axiom
1.2	$H,\ \neg A,\ H',\ \neg C \vdash C$	by impliesL
1.2.1	$H,\ \neg A,\ H',\ \neg C \vdash A$	

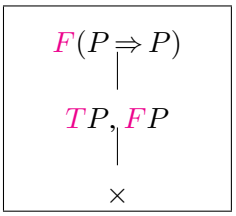
If we convert a $F\neg C$ into a TC instead of FC into a $T\neg C$ the proof fragments would prove subgoal 1.2 by `axiom` and continue with subgoal 1.1. and the assumption C.

Note that the above three steps were only introduced to justify the transition between tableaux rules and proof fragments in refinement logic. In an actual translation they can be performed as a single step, which converts the block tableaux rules $T\wedge$, $F\wedge$, $T\vee$, $F\Rightarrow$, $F\neg$, and * into the rules `andL`, `andR`, `orL`, `impliesR`, `notR`, and `axiom`, $F\vee$ into the fragment for $F\vee 1$ (or $F\vee 2$ – it doesn't matter), and $T\Rightarrow$ and $F\neg$ into their proof fragments.

Note also, that the use of `Use_decidability_of` C should be replaced by using the decidability of all the propositional variables occurring in C if C is a compound formula and that C and $\neg C$ should be assembled from that. This increases the size of the proof fragment needed for the simulation but doesn't affect the step-by-step simulation itself.

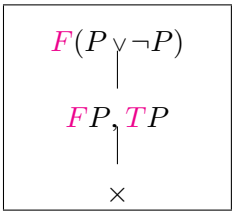
Let us conclude with a few examples.

- The tableaux proof for $P \Rightarrow P$ translates immediately without requiring decidabilities



	$\vdash P \Rightarrow P$	by impliesR
1	$P \vdash P$	by axiom

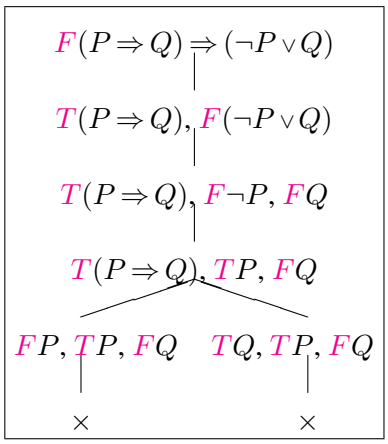
- The tableaux proof for $P \vee \neg P$ looks similar but generates a very different refinement proof



	$\vdash P \vee \neg P$	by Use_decidability_of P
1	$P \vee \neg P \vdash P \vee \neg P$	by orL
1.1	$P \vdash P \vee \neg P$	by orR1 THEN axiom
1.2	$\neg P \vdash P \vee \neg P$	by orR2
1.2.1	$\neg P \vdash \neg P$	by axiom

Note that this is a schematic translation. Obviously we could have used `axiom` already in the second step

- The tableaux proof for $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$ and its translation are shown below




```
 ⊢ (P ⇒ Q) ⇒ (¬P ∨ Q)        by impliesR
1 P ⇒ Q ⊢ ¬P ∨ Q             by Use_decidability_of P
1.1 P ⇒ Q, P ⊢ ¬P ∨ Q        by orR2
1.1.1 P ⇒ Q, P ⊢ Q           by impliesL
1.1.1.1 P ⇒ Q, P ⊢ P         by axiom
1.1.1.2 P ⇒ Q, Q, P ⊢ Q      by axiom
1.2 P ⇒ Q, ¬P ⊢ ¬P ∨ Q        by orR1 THEN axiom
```

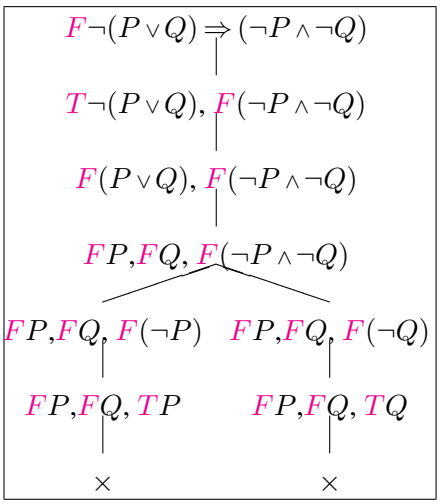
  Not that we have used an optimization in subgoal 1.1.1. Following the scheme we should have used `Use_decidability_of` Q before applying `impliesL` to preserve Q in the first subgoal. However, we see in the left branch of the tableaux proof that Q is not needed to complete the proof.

There are some optimizations that lead to better translations and possibly to refinement proofs of lower dimension.

- As the last example shows, we do not have to introduce decisions for all variables in a conclusion that will be eliminated by `impliesL`, `notL`, or `orR`i. We only need to preserve the ones that are actually used to close the proof branch. In the example, the conclusion Q did not have to be preserved at all.
- Even if a decision for a variable has been introduce, it may not be necessary to apply `orL` to it right away to assemble a proof of the conclusion C that shall be preserved. This is the case, if C is decomposed in tableau proof before being used.
- Sometimes it makes sense to modify the tableau proof and postpone the application of the critical rules $F\vee$, $T\Rightarrow$ and $T\neg$ until the proof node under consideration has no F-formula (or one of the F-formulas generated by $F\vee$ is not used in the remaining proof anymore). In this case we can translate the rules directly into `orR1`,, `orR2`, `impliesL`, or `notL` without going throug the simulation with `Use_decidability_of`.

The following example shows the effect of these optimizations.

Example 9.5 Consider the formula $\neg(P \vee Q) \Rightarrow (\neg P \wedge \neg Q)$. A standard tableaux proof would create the following translation.

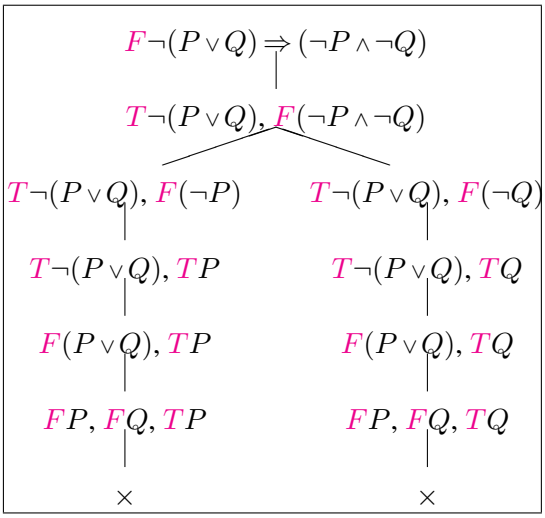



```
 ⊢ ¬(P∨Q)⇒(¬P∧¬Q)                          by impliesR
1 ¬(P∨Q) ⊢ ¬P∧¬Q                           by Use_..(¬P∧¬Q)/orL
1.1 ¬(P∨Q),¬P∧¬Q ⊢ ¬P∧¬Q                   by axiom
1.1 ¬(P∨Q),¬(¬P∧¬Q) ⊢ ¬P∧¬Q                by notL
1.1.1 ...,¬(¬P∧¬Q) ⊢ P∨Q                   by Use_..P/orL
1.1.1.1 ...,¬(¬P∧¬Q),P ⊢ P∨Q               by orR1 THEN axiom
1.1.1.2 ...,¬(¬P∧¬Q),¬P ⊢ P∨Q              by orR2
1.1.1.2.1 ...,¬(¬P∧¬Q),¬P ⊢ Q              by Use_..Q/orL
1.1.1.2.1.1 ...,...,¬P,Q ⊢ Q               by axiom
1.1.1.2.1.2 ...,¬(¬P∧¬Q),¬P,¬Q
                                    ⊢ Q    by notL
1.1.1.2.1.2.1 ....¬P,¬Q ⊢ ¬P∧¬Q             by andR
1.1.1.2.1.2.1.1 ...,...,¬P,¬Q ⊢ ¬P          by axiom
1.1.1.2.1.2.1.2 ...,...,¬P,¬Q ⊢ ¬Q          by axiom
```

While the above proof is simple as a tableaux proof, it uses $F\vee$ and $T\neg$ very early, which results in a complex translation. If we use the decidability of P and Q separately instead of the one of $(\neg P\wedge\neg Q)$ and postpone the `orL` on $Q\vee\neg Q$ we get the following result

```
   ⊢ ¬(P∨Q)⇒(¬P∧¬Q)                        by impliesR
1  ¬(P∨Q) ⊢ ¬P∧¬Q                          by Use_decidability of P THEN orL
1.1  ¬(P∨Q),P ⊢ ¬P∧¬Q                       by notL
1.1.1  ¬(P∨Q),P ⊢ P∨Q                       by orR1 THEN axiom
1.2  ¬(P∨Q),¬P ⊢ ¬P∧¬Q                      by Use_decidability of Q THEN orL
1.2.1  ¬(P∨Q),¬P,Q ⊢ ¬P∧¬Q                  by notL
1.2.1.1  ¬(P∨Q),¬P,Q ⊢ P∨Q                  by orR2 THEN axiom
1.2.2  ¬(P∨Q),¬P,¬Q ⊢ ¬P∧¬Q                 by andR
1.2.2.1  ¬(P∨Q),¬P,¬Q ⊢ ¬P                  by axiom
1.2.2.2  ¬(P∨Q),¬P,¬Q ⊢ ¬Q                  by axiom
```

This proof is simpler and still follows the pattern $T\neg$, $F\vee$, and $F\wedge$ in the tableau proof. As a result it still has dimension 2. If we modify the the tableau proof by postponing the $T\neg$ and the $F\vee$, the translation will give us a simple refinement proof of dimension 0.




```
 ⊢ ¬(P∨Q)⇒(¬P∧¬Q)          by impliesR
1 ¬(P∨Q) ⊢ ¬P∧¬Q           by andR

1.1 ¬(P∨Q) ⊢ ¬P            by notR
1.1.1 ¬(P∨Q),P ⊢ f         by notL
1.1.1.1 ¬(P∨Q),P ⊢ P∨Q     by orR1
1.1.1.1.1 ¬(P∨Q),P ⊢ P     by axiom

1.2 ¬(P∨Q) ⊢ ¬Q            by notR
1.2.1 ¬(P∨Q),Q ⊢ f         by notR
1.2.1.1 ¬(P∨Q),Q ⊢ P∨Q     by orR2
1.2.1.1.1 ¬(P∨Q),Q ⊢ Q     by axiom
```

This proof mimics the tableaux proof without extra proof fragments, since the $F\vee$ rule was used in the final step. □