Relationships between the Connectives

Demorgan's Laws

$\sim(\sim a ? \sim B)$

&	V
$A \& B$	$\sim(\sim A \vee \sim B)$
$\sim(A \& B)$	$\sim A \vee \sim B$
$\sim(\sim A \& \sim B)$	$A \vee B$
$\sim A \& \sim B$	$\sim(A \vee B)$

Arrow

$\sim a ? B$

→	V
$A \to B$	$\sim A \vee B$
$\sim(A \to B)$	$\sim(\sim A \vee B)$
$\sim A \to B$	$A \vee B$
$\sim(\sim A \to B)$	$\sim(A \vee B)$

Arrow

$\sim(a ? \sim B)$

→	&
$A \to B$	$\sim(A \& \sim B)$
$\sim(A \to B)$	$A \& \sim B$
$\sim(A \to \sim B)$	$A \& B$
$A \to \sim B$	$\sim(A \& B)$

Exercises Convert each of the following formulas into 3 equivalent formulas, the first using only & and ~, the second using only ∨ and ~ and the third only → and ~.

$p \to q$

$p \& \sim p$

$(p \& q) \to (r \vee s)$

$p \to (q \to r)$

$p \leftrightarrow (p \& q)$

Relations Between the Connectives
Practice Sheet

	$\rightarrow$	$\vee$
$A \;\&\; B$	$\sim(A \rightarrow \sim B)$	$\sim(\sim A \vee \sim B)$
$A \;\&\; \sim B$	$\sim(A \rightarrow B)$	$\sim(\sim A \vee B)$
$\sim A \;\&\; B$	$\sim(\sim A \rightarrow \sim B)$	$\sim(A \vee \sim B)$
$\sim A \;\&\; \sim B$	$\sim(\sim A \rightarrow B)$	$\sim(A \vee B)$
$\sim(A \;\&\; B)$	$A \rightarrow \sim B$	$\sim A \vee \sim B$
$\sim(A \;\&\; \sim B)$	$A \rightarrow B$	$\sim A \vee B$
$\sim(\sim A \;\&\; B)$	$\sim A \rightarrow \sim B$	$A \vee \sim B$
$\sim(\sim A \;\&\; \sim B)$	$\sim A \rightarrow B$	$A \vee B$

	$\&$	$\rightarrow$
$A \vee B$	$\sim(\sim A \;\&\; \sim B)$	$\sim A \rightarrow B$
$A \vee \sim B$	$\sim(\sim A \;\&\; B)$	$\sim A \rightarrow \sim B$
$\sim A \vee B$	$\sim(A \;\&\; \sim B)$	$A \rightarrow B$
$\sim A \vee \sim B$	$\sim(A \;\&\; B)$	$A \rightarrow \sim B$
$\sim(A \vee B)$	$\sim A \;\&\; \sim B$	$\sim(\sim A \rightarrow B)$
$\sim(A \vee \sim B)$	$\sim A \;\&\; B$	$\sim(\sim A \rightarrow \sim B)$
$\sim(\sim A \vee B)$	$A \;\&\; \sim B$	$\sim(A \rightarrow B)$
$\sim(\sim A \vee \sim B)$	$A \;\&\; B$	$\sim(A \rightarrow \sim B)$

	$\&$	$\vee$
$A \rightarrow B$	$\sim(A \;\&\; \sim B)$	$\sim A \vee B$
$A \rightarrow \sim B$	$\sim(A \;\&\; B)$	$\sim A \vee \sim B$
$\sim A \rightarrow B$	$\sim(\sim A \;\&\; \sim B)$	$A \vee B$
$\sim A \rightarrow \sim B$	$\sim(\sim A \;\&\; B)$	$A \vee \sim B$
$\sim(A \rightarrow B)$	$A \;\&\; \sim B$	$\sim(\sim A \vee B)$
$\sim(A \rightarrow \sim B)$	$A \;\&\; B$	$\sim(\sim A \vee \sim B)$
$\sim(\sim A \rightarrow B)$	$\sim A \;\&\; \sim B$	$\sim(A \vee B)$
$\sim(\sim A \rightarrow \sim B)$	$\sim A \;\&\; B$	$\sim(A \vee \sim B)$

Goal: Complete in 5 minutes
with no errors.

How to Make a Tree

There is a major defect with the truth table method for determining whether an argument is valid. When there are many different letters in an argument, the number of rows in the truth table can be very large. Since the number of rows doubles with each added letter (for 2 letters - 4 rows, 3 letters - 8 rows, 4 letters - 16 rows, etc..) the table contains more than a thousand rows for an argument with only 10 letters. The tree method is a variation on the truth table method which can vastly shorten validity calculation. True, a tree may take as long as the corresponding table in the worst case, but the chances are very good that it will yield massive savings in effort.

The main idea behind the tree method is to avoid calculating any parts of the table which do not contribute to the main objective in the validity test, namely to find a row where all the premises are T and the conclusion is F. If there is such a row, the argument is invalid, because validity requires that it is impossible for (all) the premises to be T and the conclusion F. Let us call a row with (all) T premises and a F conclusion a counterexample. The idea behind tables is to examine all possible combinations of truth values to see whether there is a counterexample. If there is, the argument is invalid, and if not, it is valid. The tree method is more economical than tables because it calculates only those portions of the table where a counterexample can be. To help illustrate the point, consider this table for the following invalid argument: $Q\rightarrow\sim P \vdash P\rightarrow Q$.

P	Q	$Q\rightarrow\sim P$	$\vdash P\rightarrow Q$	
T	T	F	T	
F	T	T	T	
T	F	T	F	← Counterexample
F	F	T	T	

Note that the third row is a counterexample to the argument - it shows that the argument is invalid because it is possible for the premise to be T and the conclusion F. If we had some way to know to calculate only this third row, and to ignore the other three, we would have the answer (invalid) with only 1/4 the effort.

So let us see whether we can find a method to locate rows where counterexamples will be. Suppose that the argument $Q\rightarrow\sim P \vdash P\rightarrow Q$ does have a counterexample. Then there must be a row where $Q\rightarrow\sim P$ is T and $P\rightarrow Q$ is F. But there is only one way for a conditional like $P\rightarrow Q$ to be F, namely when the antecedent (P) is T and the conclusion (Q) is F. This means that if there is a counterexample at all, it must be on the row with P = T and Q = F, i.e. the third row. No other rows could possibly count as a counterexample, because on these rows the conclusion will be T. At this point we can easily check that the premise $Q\rightarrow\sim P$ is indeed T on this row, verifying that row 3 is indeed a counterexample.

Notice how this reasoning focuses our attention on the crucial third row, and helps us see why it is pointless to calculate out the other three rows. By reasoning backwards from the fact that a counterexample requires an F conclusion to the values the letters must have, it may be possible to narrow down a search for a counterexample to one or two rows. (The same idea may also be used on the premises: knowing that a premise has to be T, also yields information on what values the letters have to have in a counterexample.)

You may get the impression that this idea only works for invalid arguments where a counterexample exists. But it works for valid arguments as well. Let us use $P\rightarrow\sim Q \vdash Q\rightarrow\sim P$ to illustrate. If this argument were to have a counterexample, then there would have to be a row where $P\rightarrow\sim Q$ = T and $Q\rightarrow\sim P$ = F. From the latter fact, it follows that the row would have to have Q = T and $\sim P$ = F. There is only one row like this, namely where Q = T and P = T. But if this row is a counterexample it must make $P\rightarrow\sim Q$ = T. However, it does not, since P = T and Q = T makes $\sim Q$ = F, with the result that $P\rightarrow\sim Q$ = F. The only possible counterexample to our argument is the row where P = T and Q = T. But we have just seen that it does not make the premise T, so it does not qualify as a counterexample. It follows that the argument has no counterexample, and so it must be valid. Notice that even in the case of this valid argument, we only needed to consider calculating out values for

one of the four rows.

So far we have only explained ideas behind the tree method. It is time to learn the notation and details. The basic idea is to develop a coordinated search for a counterexample to our argument by using the information that the premises must be T and the conclusion F. One notation for doing this involves labeling formulas and their parts with Ts and Fs to indicate their values. However, this can be both conceptually and visually confusing. Instead, the tree method uses the convention that every sentence we write down is considered to be T. If we want to indicate that sentence P is F, we simply write down ~P instead, (for if ~P is T, P must be F). Let us illustrate this idea by actually building the tree for the argument P→~Q ⊢ Q→~P. We begin with the information that if the argument is to have a counterexample in some row, then P→~Q must be T and Q→~P must be F:

P→~Q
~(Q→~P)

Note that to indicate Q→~P = F, we have written ~(Q→~P) instead. Now if ~(Q→~P) is to be T, i.e. Q→~P = F, then there is only one possibility: Q = T and ~P = F. Entering this information using our convention, we have the following.

P→~Q
① ~(Q→~P)
Q
~~P

The circled 1 is there to indicate that we have worked on the second line - the numbers will indicate the order in which we worked. We have carried out the tree rule for negative arrows. In general, the rule has the form:

~(𝒜→ℬ)
𝒜
~ℬ

This indicates that whenever we have ~(𝒜→ℬ) in a tree, we should then write down 𝒜 and ~ℬ below it. This principle is strongly related to the AR (Arrow) rule. You probably remember the following useful move: ~(𝒜→ℬ) ⊢ 𝒜&~ℬ. Notice that the tree rule for a negative arrow amounts to almost the same thing, for from ~(𝒜→ℬ), we obtain 𝒜&~ℬ by AR, from which 𝒜 and ~ℬ follow by &Out.

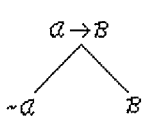
Notice that ~~P now appears on the tree. Using the double negation rule:

~~𝒜
𝒜

we may simplify this to P:

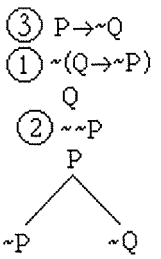
P→~Q
① ~(Q→~P)
Q
② ~~P
P

The next step in building our tree corresponds to checking to see whether P→~Q can be T. To do so, we must introduce the tree rule for positive conditionals. In general the rule has the following form:

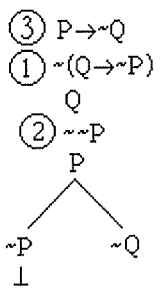


The branch in the rule indicates that for 𝒜→ℬ to be true, there are two (and only two) possibilities:

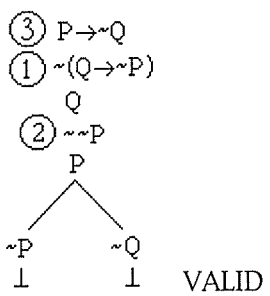
either $\mathcal{A}$ must be F or $\mathcal{B}$ must be T. Review the truth table for → to convince yourself that this is correct. Another way to understand (and remember) this rule is to think of it as a variation on the AR rule for converting from v to →: $\mathcal{A} \rightarrow \mathcal{B} \vdash \sim \mathcal{A} \vee \mathcal{B}$. When we apply this principle to our tree, we have the following:



The circled numbers show that we have worked on every complex line of our tree. It is now time to evaluate the argument. The tree shows that if there is to be a counterexample to our argument, there are two possibilities. The first, is recorded by the left branch, which contains the simple sentences Q, P and ~P. This indicates (according to our convention) that Q must be T (Q), P must be T (P) and P must be F (~P). But it is impossible for P to be both T and F on the same row, so this possibility can be ruled out. To indicate that this situation is impossible, we indicate that this branch is "dead" by adding the contradiction sign:



Notice that the right hand branch also contains a contradiction, for the counterexample it describes requires that Q be both F and T. So we mark that branch with ⊥ as well:

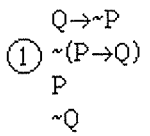


Let us review what this tree tells us about validity. It provides a complete record of all the possible ways in which a counterexample can be constructed for P→~Q ⊢ Q→~P. There were two options (described by the left and right branches), but both of these turned out to be impossible. The verdict: there cannot be a counterexample to this argument. To put it another way, the argument is valid.

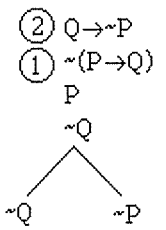
Now we will work out the tree for Q→~P ⊢ P→Q to see what happens with an invalid argument. We begin by writing down the premise and the negative of the conclusion:

Q→~P
~(P→Q)

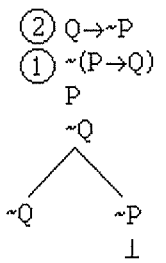
Applying the rule for negative conditionals we obtain:



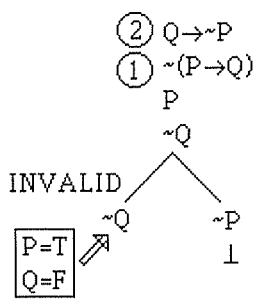
Now we apply the rule for positive conditionals to Q→~P:



Note that the right-hand branch contains a contradiction, so we mark it closed.



However, the left hand branch is still open. Since we have worked on every complex sentence, we have a full account of the possibilities for counterexamples for this argument. The left hand branch reports that there is a counterexample, namely one with P, ~Q, and ~Q all true. That amounts to saying that there is a counterexample with P = T and Q = F. We list this counterexample by putting the values in a box:



Here is a review of the tree construction process.

1. Write down the premises and the negative of the conclusion.
2. Apply tree rules to every complex sentence.
3. Mark every branch that contains a contradiction with ⊥.
4. If all branches are closed (contain ⊥), the argument is valid; if any branch is open (no ⊥ on it), then this branch describes a counterexample and the argument is invalid.

Let's review the process with a more complex argument: (P&Q)→R, P ⊢ Q→R. First enter the premises and the negative of the conclusion:

(P&Q)→R
P
~(Q→R)

Now apply the negative conditional rule to the last line:

(P&Q)→R
P
① ~(Q→R)
Q
~R

Now apply the positive conditional rule to the first premise:

② (P&Q)→R
P
① ~(Q→R)
Q
~R
~(P&Q) R
⊥

Notice that the right hand branch is closed because it contains ~R and R, so we have marked it with ⊥. Now we must apply a rule to ~(P&Q). The rule for negative conjunctions is similar to De Morgan's Law: ~(𝒜&ℬ) ⊢ ~ 𝒜 v~ℬ. If 𝒜&ℬ is F then either 𝒜 is false or ℬ is false:

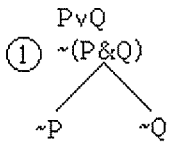
~(𝒜&ℬ)
~𝒜 ~ℬ

Applying this rule to ~(P&Q), we obtain:

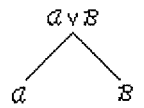
② (P&Q)→R
P
① ~(Q→R)
Q
~R
③ ~(P&Q) R
⊥
~P ~Q VALID
⊥ ⊥

Since all branches close, the argument is valid.

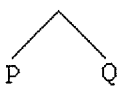
There are still a few details about the tree method that need explaining. To illustrate, we will work out the tree for PvQ ⊢ P&Q. We begin entering the premise and negative conclusion as usual, and we have already applied the negative & rule to the last step:



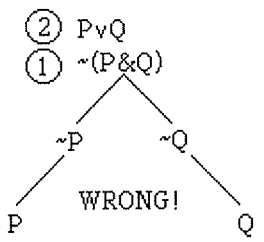
Now we must work on PvQ. It is easy enough to see what rule we want here. If $\mathcal{A}$vB is T then either $\mathcal{A}$ is T or B is T:



The problem is where to put the result of applying this rule. We have two open branches in our tree, one ending with ~P and the other ending with ~Q, and neither one is closed. So where do we place the new branch generated by PvQ:



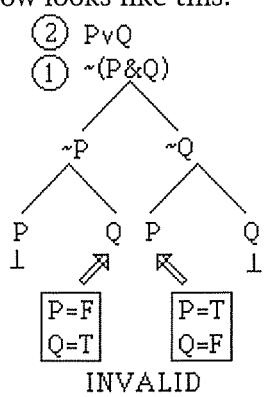
on the left (below ~P) or on the right (below ~Q)? It would be wrong to do either one, and it would also be wrong to divide the results of applying the rule between the two sides like this:



To see what we should do, it helps to remember what the tree represents. The two branches ending with ~P and ~Q represent two alternatives: for there to be a counterexample, either P = F or Q = F. However, the fact that PvQ is T means that <u>on top of</u> these two alternatives there are two more, either P = T or Q = T. So the correct thing to do is represent a total of <u>four</u> alternatives. Beneath the alternative ~P, we need to record the alternative P or Q, and beneath ~Q we must do so as well. This illustrates a basic principle about building a tree when there is more than one open branch.

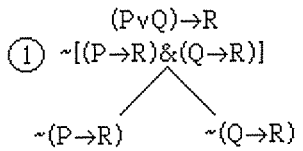
The results of applying a rule to a line must be entered on <u>every open branch</u> below that line.

Following this principle, our tree now looks like this:

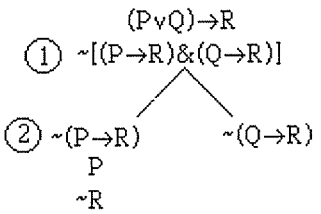


Note that contradictions appeared on two of the branches, so we marked them closed: Since we have worked on every step, the tree is complete and we can evaluate the argument. Note that there are two open branches indicating two different counterexamples to our argument. So the argument is invalid. It is a good idea to calculate the truth table for this argument to verify that it has these two counterexamples. One nice thing about trees is that they always give a full account of all the counterexamples to an argument.

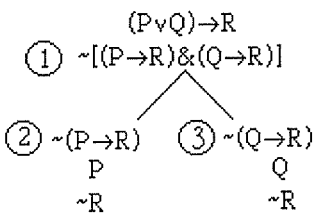
Our next project will be to show that (PvQ)→R ⊢ (P→R)&(Q→R) is valid using trees. We begin by negating the conclusion, and applying the negative & rule:



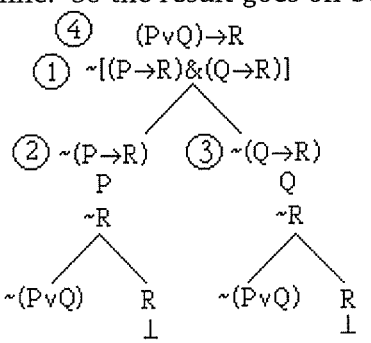
Next we will work on the ~(P→R) on the left hand branch. According to the negative → rule we enter P and ~R below ~(P→R). **It is not necessary to place P and ~R below the right hand branch.** Remember the rule is to put the results of applying a rule on every open branch below the formula you are working on.



After working on ~(Q→R) on the right branch, we obtain.



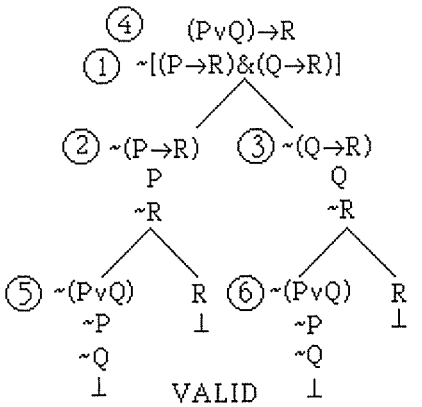
Now we will work on the first line: (PvQ)→R. In this case, we must place the result of applying the rule on all open branches below this line. So the result goes on both branches:



All that remains is to work on the ~(PvQ) on each branch. The rule for a negative v is as follows:

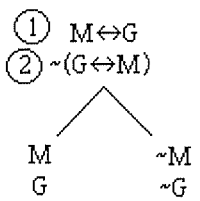
$$\sim(\mathcal{A} \vee \mathcal{B})$$
$$\sim\mathcal{A}$$
$$\sim\mathcal{B}$$

This rule is related to DeMorgan's Law. The idea is that when $\mathcal{A} \vee \mathcal{B}$ is F, then both $\mathcal{A}$ and $\mathcal{B}$ are F. Applying this rule on each branch we complete the tree:

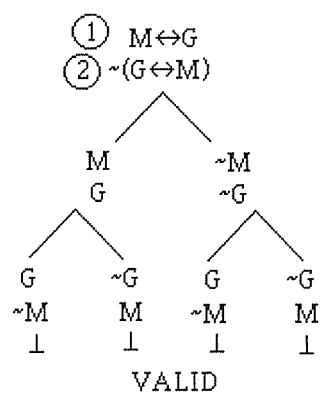


There is a final detail about tree construction worth mentioning. You may have been curious about the order in which I have carried out the steps in the trees. You might want to experiment with the problems we have already done to see what happens if you carry out the steps in a different order. In each case I have ordered the steps to keep the tree as simple as possible. There is a simple recipe to help economize in tree construction. The branching rules tend to create open branches, and subsequent steps must be copied onto all these open branches. For this reason, it is a good idea to **do the non-branching steps first**. Then when branching steps are finally carried out, there is a better chance that the branches created will close.

As a last note, we must discuss the ↔ rules. The rules may appear complex. But there is a simple way to remember them. In case $\mathcal{A}$↔$\mathcal{B}$ is T, the values of $\mathcal{A}$ and $\mathcal{B}$ match. So either $\mathcal{A}$ and $\mathcal{B}$ are both T or $\mathcal{A}$ and $\mathcal{B}$ are both F. So the positive ↔ rule indicates these two possibilities on two branches. In case $\mathcal{A}$↔$\mathcal{B}$ is F, then the values of $\mathcal{A}$ and $\mathcal{B}$ disagree. So either $\mathcal{A}$ is T and $\mathcal{B}$ is F or $\mathcal{A}$ is F and $\mathcal{B}$ is T. This is why the negative ↔ rule shows two alternatives, one containing $\mathcal{A}$ and ~$\mathcal{B}$ and the other containing ~$\mathcal{A}$ and $\mathcal{B}$. Here is a tree for the argument M↔G ⊢ G↔M to help you practice these rules.

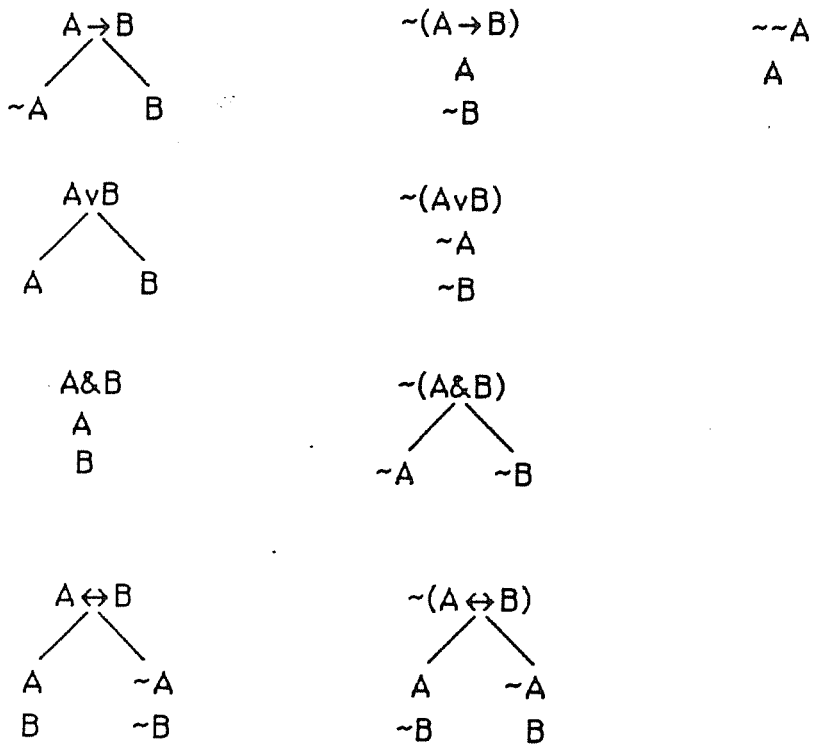


Since there are two open branches, the result of applying the negative ↔ rule to ~(M↔G) goes on both sides:

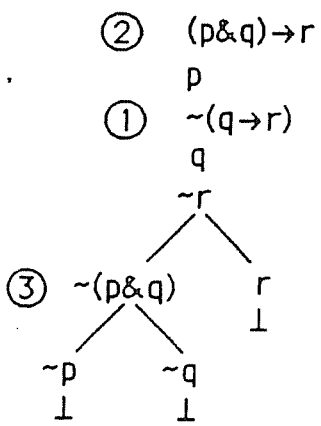


Note that all branches are closed. It is not necessary for there to be a contradiction for M and a contradiction for G to close a branch. **One contradiction on a branch is enough to close it.**

Tree Rules for Propositional Logic



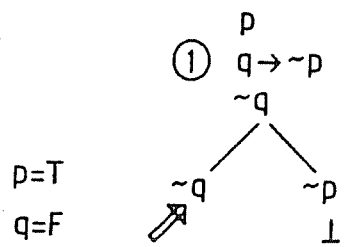
Example: To show the argument $(p\&q)\rightarrow r,\ p \vdash q\rightarrow r$ is valid, negate the conclusion, and then show that there is no assignment that makes the premises and the negated conclusion all T.



EXERCISES.
1) $p\rightarrow(q\rightarrow r) \vdash q\rightarrow(p\rightarrow r)$
2) $p\rightarrow(p\rightarrow q) \vdash p\rightarrow q$
3) $p \vdash q \vee (r\rightarrow p)$
4) $p \vdash (p\rightarrow r)\leftrightarrow r$
5) $p \vee q,\ p\rightarrow r,\ q\rightarrow s \vdash r \vee s$

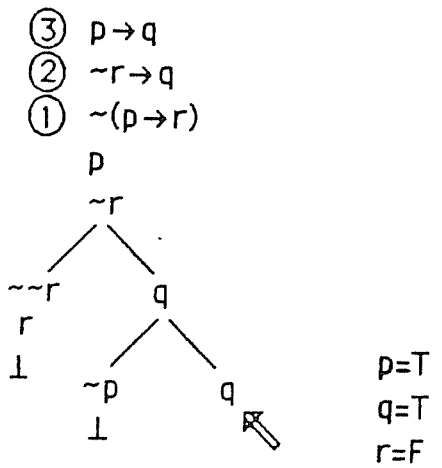
Finding Counterexamples with Trees

To find a counterexample to this invalid argument: p, q→~p ⊢ q first build its tree:



The left branch is open (marked with ⇗). Now construct the open branch assignment, by assigning T to all variables that appear unnegated in the open branch, and F to those that appear negated. In this case we have p=T and q=F. Calculate the values of the sentences in this argument, and you will find that the premises are T and the conclusion is F in this case.

Here is tree for the argument p→q, ~r→q ⊢ p→r.



The open branch assignment is p=T, q=T, r=F. Calculate the values of the sentences in this argument to show that the premises are T and the conclusion is F.

EXERCISES. 1) p→q ⊢ q→p
2) p→q, p→r ⊢ ~r→q
3) p→q ⊢ (pvr)→q
4) (p&r)→q ⊢ p→q

S A simple Propositional Logic

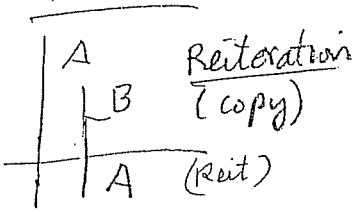
2010

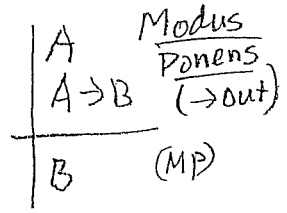
A. $\rightarrow$ $\perp$ are the only symbols. All others are defined

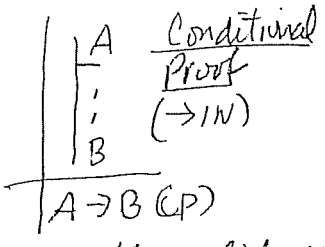
$-A =_{df} A \rightarrow \perp$ $\quad A \,\&\, B =_{df} -(A \rightarrow -B)$ $\quad A \vee B =_{df} -A \vee B$

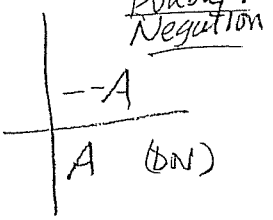
$A \leftrightarrow B =_{df} (A \rightarrow B) \,\&\, (B \rightarrow A)$

B. Rules



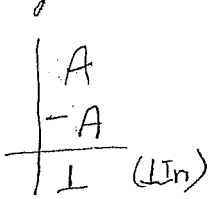


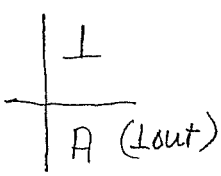


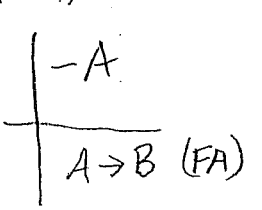


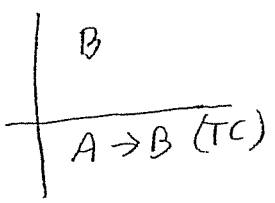
C. These rules are sufficient for proving all valid arguments in propositional logic, i.e. system S is complete.

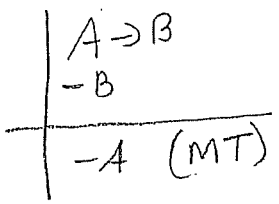
D. Project: Build a useful logic from S by showing rules you like are derived. Here are some candidates:

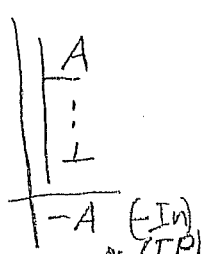


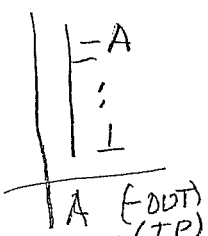


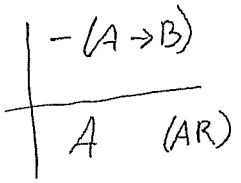


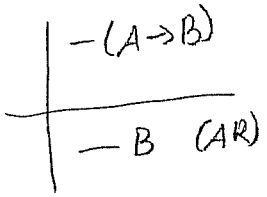


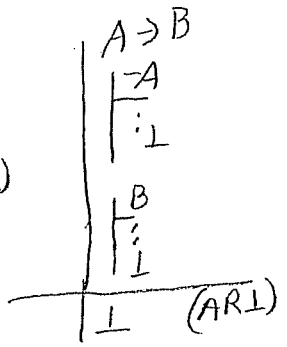




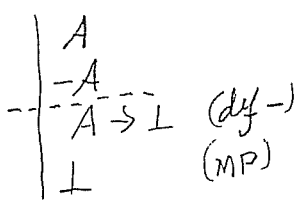


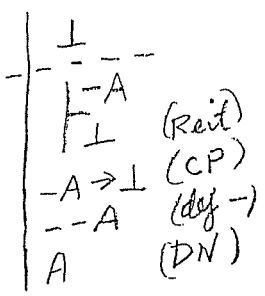


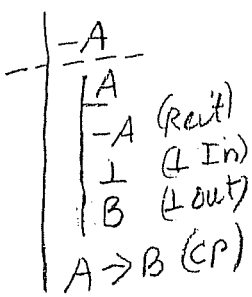


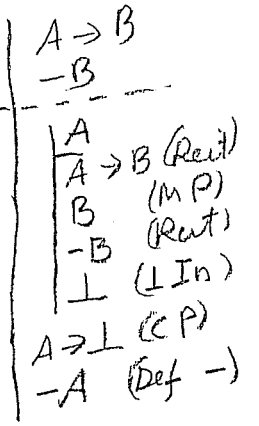


E. Exercize: Show all these rules are derivable in S. For example here are solutions for (⊥In), (⊥out), (FA) and (MT)









P: A useful Propositional Logic

A. Probably everyone will want to know the rules of P

$$\frac{A \& B}{A} \ (\&\text{Out}) \ (\text{Simp}) \qquad \frac{A \& B}{B} \ (\&\text{Out}) \ (\text{Simp})$$

$$\frac{A \quad B}{A \& B} \ (\&\ \text{In}) \ (\text{Conj})$$

$$\frac{A \leftrightarrow B}{A \rightarrow B} \ (\leftrightarrow \text{Out}) \qquad \frac{A \leftrightarrow B}{B \rightarrow A} \ (\leftrightarrow \text{Out})$$

$$\frac{A}{A \vee B} \ (\vee \text{In}) \ (\text{Add}) \qquad \frac{B}{A \vee B} \ (\vee \text{In}) \ (\text{Add})$$

$$\frac{A \vee B \quad \begin{array}{|l} A \\ \vdots \\ C \end{array} \quad \begin{array}{|l} B \\ \vdots \\ C \end{array}}{C} \ (\vee \text{Out}) \ (\text{Dilemma})$$

$$\frac{A \rightarrow B \quad B \rightarrow A}{A \leftrightarrow B} \ (\leftrightarrow \text{In})$$

B. Exercise: Show all these rules are derivable in S.
For example, here are some solutions

$$\frac{A \& B}{\neg(A \rightarrow \neg B)} \ (\text{def } \&)$$

$$\begin{array}{|l} \neg A \\ A \rightarrow B \ (\text{FA}) \\ \neg(A \rightarrow \neg B) \ (\text{Reit}) \\ \bot \ (\bot \text{ In}) \\ A \ (\neg \text{out}) \end{array}$$

$$\begin{array}{|l} A \\ \hline \begin{array}{|l} \neg A \\ A \ (\text{Reit}) \\ \bot \ (\bot \text{ In}) \\ B \ (\bot \text{ out}) \end{array} \\ \neg A \rightarrow B \ (\text{CP}) \\ A \vee B \ (\text{Def } \vee) \end{array}$$

$$\begin{array}{|l} A \leftrightarrow B \\ \hline (A \rightarrow B) \,\&\, (B \rightarrow A) \\ \quad (\text{def} \leftrightarrow) \\ A \rightarrow B \ (\& \text{ out}) \end{array}$$

$$\begin{array}{|l} A \vee B \\ \begin{array}{|l} A \\ \vdots \\ C \end{array} \\ \begin{array}{|l} B \\ \vdots \\ C \end{array} \\ \hline \neg A \rightarrow B \ (\text{def } \vee) \\ A \rightarrow C \ (\text{CP}) \\ B \rightarrow C \ (\text{CP}) \\ \begin{array}{|l} \neg C \\ \neg A \ (\text{MT}) \\ B \ (\text{MP}) \\ C \ (\text{MP}) \\ \bot \ (\bot \text{In}) \\ C \ (\neg \text{OUT}) \end{array} \end{array}$$

C. Here are some more useful rules. Prove them in S and adopt them if you like — or invent your own.

$$\frac{A \rightarrow B}{\neg B \rightarrow \neg A} \ (\text{CN}) \qquad \frac{\neg(A \vee B)}{\neg A \,\&\, \neg B} \ (\text{DM}) \qquad \frac{A \vee B \quad \neg A}{B} \ (\text{DA})$$

$$\frac{A \vee B \quad \neg B}{A} \ (\text{DA}) \qquad \frac{\neg(A \,\&\, B)}{A \rightarrow \neg B} \ (\text{AR}) \qquad \text{etc ...}$$

PROOF STRATEGY for System P

0. **Can I apply &Out, MP, DN or ↔Out to lines already available?**
Yes? Then do it.

1. **What is my goal?**

2. **Is $\alpha \vee \beta$ avaliable?**
Yes? Then set up vOut. That means set α and β, as new hypotheses and put down G as a goal in each of the subproofs just created, where G is your old goal.

3. **Is my goal $\alpha \rightarrow \beta$, $\alpha \& \beta$, or $\alpha \leftrightarrow \beta$?**
Yes? Then assume it will come by the corresponding In rule. For example, if it is $\alpha \rightarrow \beta$, make a subproof with α a hypothesis, and set β as a new goal. When the subproof is completed obtain $\alpha \rightarrow \beta$ with CP.
If it is $\alpha \& \beta$, set α and β as new goals. When the goals are achieved use &In.
If it is $\alpha \leftrightarrow \beta$, set $\alpha \rightarrow \beta$ and $\beta \rightarrow \alpha$ as new goals. When the goals are achieved use ↔In.

3. **Is my goal $\alpha \vee \beta$?**
Yes? Then if either α or β is available, use vIn. <u>Otherwise work elsewhere.</u>

4. **Does my goal appear to the right of → in an available line?**
Yes? Then set what is to the left of → as a new goal. When you get that goal, use MP to get what was on the right of →. (Your old goal.)

6. **Is $\sim(\alpha \vee \beta)$ available?**
Yes? Then convert to $\sim\alpha \& \sim\beta$ with DM.

7. **Is $\sim(\alpha \rightarrow \beta)$ or $\sim(\alpha \& \beta)$ available?**
Yes? Then use AR to convert this to $\alpha \& \sim\beta$ or $\alpha \rightarrow \sim\beta$ respectively.

8. **Are your stuck?**
Yes? Then use IP. Make a subproof with the opposite of your goal a hypothesis and chose a contradiction as a new goal. In chosing a contradiction look for negative sentences in lines available. Long negations are a good choice. Do not use the goal for IP as part of your contradiction, and do not use contradictions previously used for IP.

Strategies for Proof Finding

If you see This:		then Do this:		
		A | B GOAL		
A → B	GOAL	A→B ~~GOAL~~ C.P.		
A & B	GOAL	A B A&B ~~GOAL~~ &IN		
A ↔ B	GOAL	A→B GOAL B→A GOAL A↔B ~~GOAL~~ ↔IN		
A ∨ B	GOAL	WAIT! WORK ELSE-WHERE... THEN TRY	A GOAL A∨B ~~GOAL~~ ∨In	OR B GOAL A∨B ~~GOAL~~ ∨In
A → G	(PROVEN)	A→G (PROVEN)		
⋮				
G	GOAL	A GOAL A→G Reit G ~~GOAL~~ M.P.		
A ∨ B	(PROVEN)	A∨B (PROVEN)		A∨B (PROVEN)
⋮		A | G GOAL	OR	~A→B ARROW
G	GOAL	B | G GOAL G ~~GOAL~~ ∨OUT		G GOAL
~(A & B)		convert to A → ~B	(Arrow)	
~(A → B)		convert to A & ~B	(Arrow)	
~(A ∨ B)		convert to ~A & ~B	(DeMorgan)	
G	GOAL (nothing works)	~G | B GOAL | ~B GOAL G ~~GOAL~~ I.P.	Try Indirect Proof	

Choose a contradiction as a new goal.

1. Prefer long negated formulas already available,
2. Avoid formulas already used as contradictions for I.P.
3. Avoid using G and ~G as your contradiction.

Predicate Logic Translation Chart

Singular:

(Use a lower case letter s for the subject and a upper case letter P for the predicate and write: Ps.)

Examples: John, David's cat, he*, she*, it*, that*, the king*

* on occasion, these may be general

General:

A. $\forall x(Ax \rightarrow Bx)$

All As are Bs
Every A is a B
As are Bs
Any A is a B
Each A is a B
An A is a B*
Only Bs are As
None but Bs are As

* Usually 'a' means some, but it may mean all. It is NEVER singular.

E. $-\exists x(Ax \& Bx)$ or $\forall x(Ax \rightarrow -Bx)$

No A is a B
None of the As are Bs
Not any A is a B

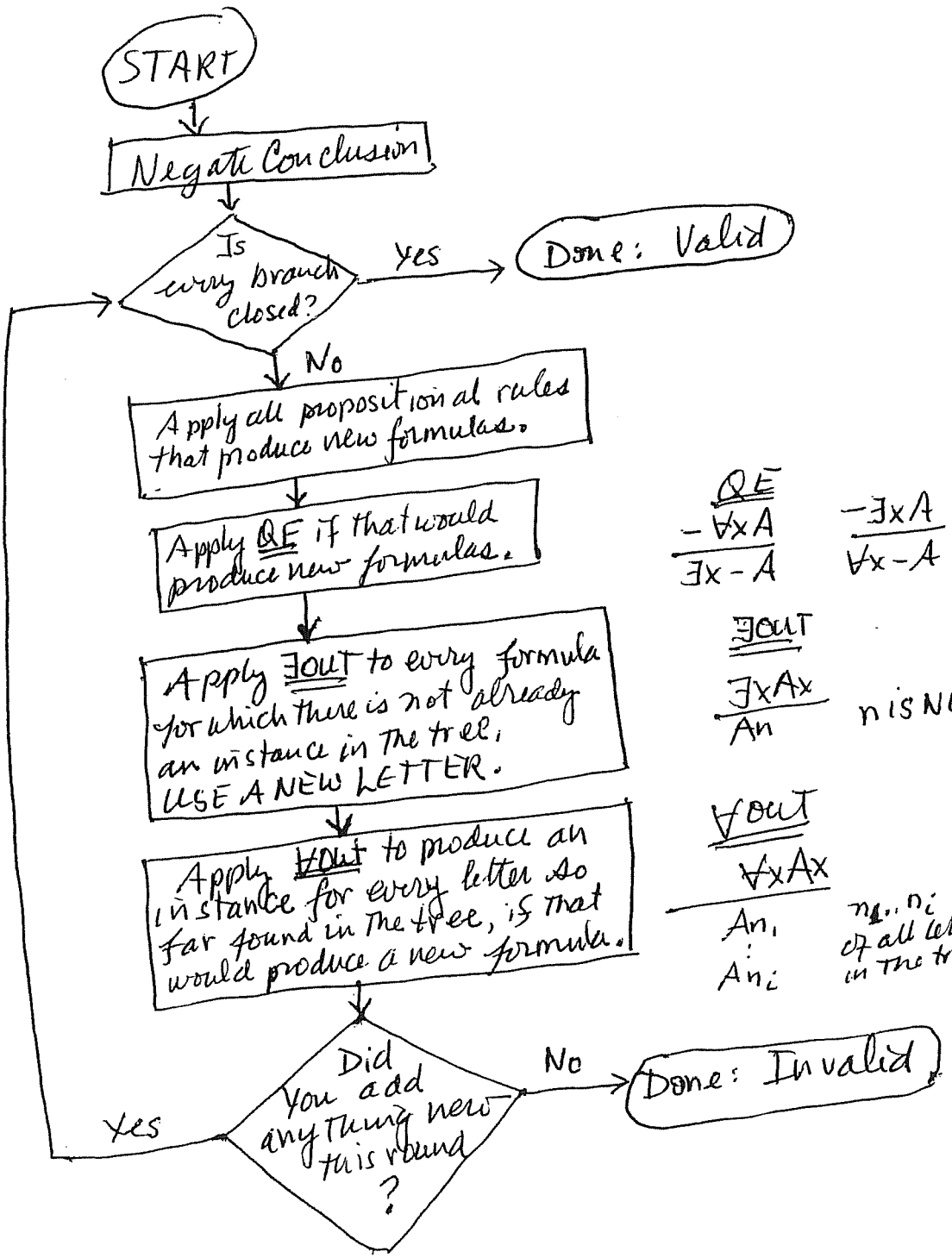
I. $\exists x(Ax \& Bx)$

Some As are Bs
Some A is a B
There are As that are Bs
A least one A is a B
As are sometimes Bs
An A is a B*
Many A are B
There exist As that are Bs

O. $\exists x(Ax \& -Bx)$ or $-\forall x(Ax \rightarrow Bx)$

Some As are not Bs
Some As are non-Bs
Not all As are Bs

Trees in Predicate Logic



How to Convert a Tree to a Proof

1. Let's assume that &, v, and ↔ are translated into -> and ~. If this is done, any tree can be constructed using only the rules for A->B, ~(A->B), ~~A, and the quantifier rules ∀Out, ∃Out, and QE.

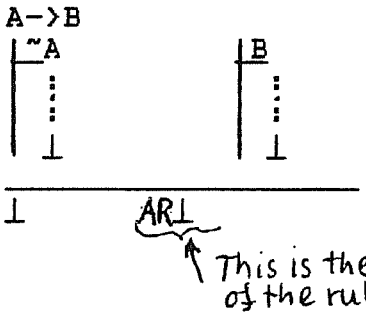
2. The rules for the quantifiers are already in our proof system, and the rules for ~~A and ~(A->B) are clearly derivable:

EXERCISE. Show that the following rules are derivable:

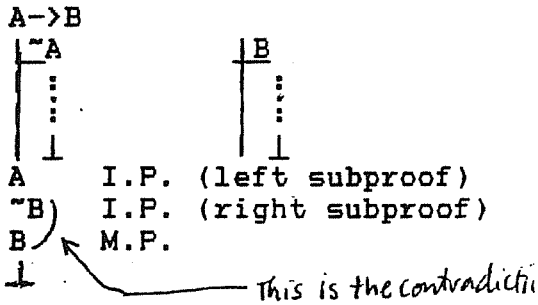
$$\frac{\sim(A\to B)}{A} \qquad \frac{\sim(A\to B)}{\sim B}$$ Call them: AR (for Arrow)

3. So now we know that all steps of a tree correspond to steps we can carry out in a proof, except for the tree rule for (A->B). In that case, the tree branches. We need to come up with some rule of proof that corresponds to the branching (A->B) tree rule. The rule we need, called ⊥, resembles VOut. The symbol '⊥' stands for any contradiction.

AR⊥ (Arrow to a ⊥)

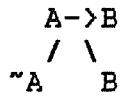


PROOF OF DERIVABILITY:

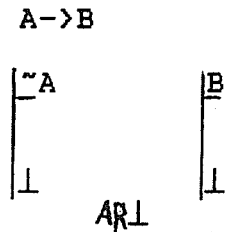


5. Each branching step of the tree corresponds to steps of a proof in the following way:

TREE:



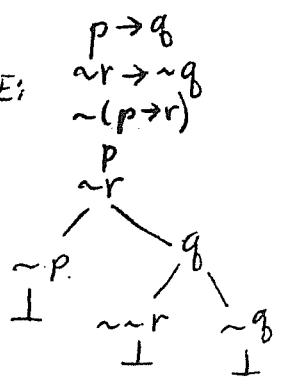
PROOF:



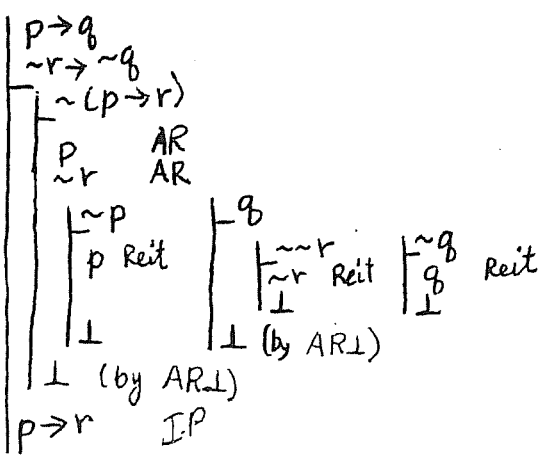
6. To convert a whole tree to a proof, begin by setting up the negation of the conclusion for I.P., then copy down all steps of the tree as lines of the proof. In case the tree branches, set up two subproofs needed for the ->⊥ rule. Since contradictions will turn up on all branches of the tree, the corresponding contradictions will appear in each of the subproofs so constructed, and the proof will be done.

EXAMPLE:

TREE:



PROOF:



Determining Truth Values of Predicate Logic Sentences

$\forall x Ax$ is T iff An is T for <u>every</u> letter n in the Universe of Discourse

$\exists x Ax$ is T iff An is T for <u>some</u> letter n in the Universe of Discourse

Examples: Let us calculate the truth values of a few sentences for the following assignment:

UD: a b

Aa = F	Ba = F
Ab = T	Bb = F

a) $\forall x Ax \rightarrow \forall x Bx$

The main connective is → so we must <u>separately</u> calculate the values of $\forall x Ax$ and $\forall x Bx$.
To calculate $\forall x Ax$, take all the instances: Aa and Ab.
Since Aa = F, one of the instances is false, and so it is not the case that all instances are T, so $\forall x Ax$ is F.
But then the left side of the → in a) is F, making a) T.

b) $\forall x(Ax \rightarrow Bx)$

The main connective here is $\forall x$. To determine the value of b), then, we must take all its instances:

$Aa \rightarrow Ba$
$Ab \rightarrow Bb$

The first instance has a false antecedent (Aa), so it is T. However, the second instance has a true antecedent and a false conclusion, so it is F. Since one of the instances was false, b) is false.

EXERCISE: Calculate the truth values of the following two sentences: $\exists x(Ax \& Bx)$, $\exists x Ax \ \& \ \exists x Bx$ on the following interpretation.

UD: a b

Aa	~Ba
Bb	~Ab

Proof that the Tree Method Works

Theorem: **If a tree for an argument has an open branch then the assignment constructed for that branch (called the acb) is a counterexample to the argument.** (To put this another way: the acb makes the premises T and the conclusion F.)

Proof of the Theorem. We will prove something stronger:

1) **Every sentence on an open branch is T on the acb.**

When 1) is proven, then the theorem holds because the premises and negated conclusion of the argument appear on the open branch and so 1) guarantees that the acb makes the premises T and the conclusion F.

Let us call the sentences on the open branch that result from applying a tree rule to a sentence A the children of A. For example, the children of -(B→C) are B and -C. Sentences (such as Fab or -Gc) to which no rule applies are called atoms.

Proof of 1. First we prove 2). I'll explain how this gives us 1) later

2) **If all the children of A on the branch are T on the acb, then A is T on the acb.**

Proof of 2. Assume that all the children of A on the open branch are T. We will show A is T. A must have one of the following shapes and we can prove A is T in each case:

1. A has shape --B. Then by the tree rules, the child B must be on the branch. By the assumption that all A's children are T, B=T. So -B=F and --B (or A) = T.
2. A has shape -(B→C). The tree rules guarantee that B and -C appear on the branch. By the assumption B=T, -C=T. This makes C=F and B→C=F, so -(B→C) (or A) = T.
3. A has shape B→C. The tree rules guarantee that one of the children (either -B or C) is on the branch. Given the children on the branch are T, then either -B=T or C=T. If -B=T then B=F and B→C=T, and if C=T then B→C=T. So B→C (or A) =T either way.
4. A has shape ∀xBx. The tree rules guarantee that children Bn are on the open branch for each letter n in the UD. By the assumption all these children are T. So ∀xBx (or A) = T.
5. A has shape ∃xBx. EXERCISE.
6. A has shape -∀xBx. Then the child ∃x-Bx is on the branch and by the assumption ∃x-Bx=T. So -Bn is T for some n in the UD, so Bn=F, which means ∀xBx=F. So A =T.
7. A has shape -∃xBx. EXERCISE.

2) is now proven. 1) follows from 2) by this reasoning: The acb makes all the atoms T. Now consider parents of the atoms. By 2) these sentences are all T. Now consider parents of these parents. By 2) again these sentences are T. By repeating this argument over and over we can show that any ancestor of atoms on the branch is T. SInce any sentence on a branch is either an atom or an ancestor of atoms, every sentence on the open branch = T.

Basic Translation Patterns

All a B

$\forall x (x\,a \rightarrow x\,B)$

Sample: a B
All dogs love John
$\forall x (x\,dog \rightarrow x\,loves\,John)$
$\forall x (Dx \rightarrow Lxj)$

Some a B

$\exists x (x\,a \,\&\, x\,B)$

Sample: a B
a dog loves John
$\exists x (x\,dog \,\&\, x\,loves\,John)$
$\exists x (Dx \,\&\, Lxj)$

No a B

$-\exists x (x\,a \,\&\, x\,B)$

Sample: a B
No dog loves John
$-\exists x (x\,dog \,\&\, x\,loves\,John)$
$-\exists x (Dx \,\&\, Lxj)$

B all a

$\forall x (x\,a \rightarrow B\,x)$

Sample: B a
John loves all dogs
$\forall x (x\,dog \rightarrow John\,loves\,x)$
$\forall x (Dx \rightarrow Ljx)$

B some a

$\exists x (x\,a \,\&\, B\,x)$

Sample: B a
John loves a dog
$\exists x (x\,dog \,\&\, John\,loves\,x)$
$\exists x (Dx \,\&\, Ljx)$

B no a

$-\exists x (x\,a \,\&\, B\,x)$

Sample: B a
John loves no dog
$-\exists x (x\,dog \,\&\, John\,loves\,x)$
$-\exists x (Dx \,\&\, Ljx)$

Simple Translation Problems

1. John loves some dog.
2. Some dog loves John.
3. John loves a dog.
4. A dog loves John.
5. No dog loves John.
6. John loves no dog.
7. John loves a green vegetable.
8. A green vegetable loves John.
9. John loves any green vegetable.
10. Any green vegetable loves John.
11. No green vegetable loves John.
12. John does not love any green vegetable.
13. John loves a vegetable which is green.
14. John loves any vegetable which is green.
15. John loves no vegetable which is green.
16. No vegetable which is green loves John.
17. John loves every dog who loves him.
18. Every dog who loves John loves him.
19. John loves every dog which loves itself.
20. No dog who loves itself loves John.

Vocabulary

j John
Dx x is a dog
Lxy x loves y
Gx x is green
Vx x is a vegetable

Answers to Simple Translation Problems

1. $\exists x(Dx \ \& \ Ljx)$
2. $\exists x(Dx \ \& \ Lxj)$
3. $\exists x(Dx \ \& \ Ljx)$
4. $\exists x(Dx \ \& \ Lxj)$
5. $\neg\exists x(Dx \ \& \ Lxj)$ or $\forall x(Dx \rightarrow \neg Lxj)$
6. $\neg\exists x(Dx \ \& \ Ljx)$ or $\forall x(Dx \rightarrow \neg Ljx)$
7. $\exists x((Gx \ \& \ Vx) \ \& \ Ljx)$
8. $\exists x((Gx \ \& \ Vx) \ \& \ Lxj)$
9. $\forall x((Gx \ \& \ Vx) \rightarrow Ljx)$
10. $\forall x((Gx \ \& \ Vx) \rightarrow Lxj)$
11. $\neg\exists x((Gx \ \& \ Vx) \ \& \ Lxj)$ or $\forall x((Gx \ \& \ Vx) \rightarrow \neg Lxj)$
12. $\neg\exists x((Gx \ \& \ Vx) \ \& \ Ljx)$ or $\forall x((Gx \ \& \ Vx) \rightarrow \neg Ljx)$
13. $\exists x((Vx \ \& \ Gx) \ \& \ Ljx)$
14. $\forall x((Vx \ \& \ Gx) \rightarrow Ljx)$
15. $\neg\exists x((Vx \ \& \ Gx) \ \& \ Ljx)$ or $\forall x((Vx \ \& \ Gx) \rightarrow \neg Ljx)$
16. $\neg\exists x((Vx \ \& \ Gx) \ \& \ Lxj)$ or $\forall x((Vx \ \& \ Gx) \rightarrow \neg Lxj)$
17. $\forall x((Dx \ \& \ Lxj) \rightarrow Ljx)$
18. $\forall x((Dx \ \& \ Lxj) \rightarrow Lxj)$
19. $\forall x((Dx \ \& \ Lxx) \rightarrow Ljx)$
20. $\neg(\exists x)((Dx \ \& \ Lxx) \ \& \ Lxj)$ or $\forall x((Dx \ \& \ Lxx) \rightarrow \neg Lxj)$

DECISION PROCEDURES

1. A system has a <u>decision procedure</u> iff

   there is a method (computer program, set of instructions) which will correctly distinguish the valid from the invalid arguments of the system. The method must provide an answer for each argument in a finite amount of time.

2. <u>Propostional Logic has a decision procedure</u>. (Trees and truth tables are both examples of decision procedures for for propostional logic.)

3. <u>Predicate Logic has no decision procedure.</u> (This was proved in the 1930s by Alonso Church.)

4. The reason that trees do not work as a decision procedure for predicate logic is that some open branches of a tree may be infinite. If you are working on an infinite branch you will never know (in a finite amount of time) whether it will close or not.

Here is an example of a tree which has an infinite open branch:

Argument: $\forall x \exists y Cyx \;/\; \exists y \forall x\, Cyx$

(8) (6) (4) (2)	$\forall x \exists y\, Cyx$	
(1)	$-\exists y \forall x\, Cyx$	
	$\forall y - \forall x\, Cyx$	QE
(3)	$\exists y\, Cya$	
	Cba	b is new because of ∃Out
(5)	$\exists y\, Cyb$	∀Out from first line, because I must have an instance for all letters.
	Ccb	∃Out again
(7)	$\exists y\, Cyc$	UOut again, now for c
	Cdc	
	.	
	.	
	.	

FOREVER!

Semantical Consistency and Completeness

1. a. An argument is **valid** iff it has no counterexample.
   b. A **counterexample** is an assignment that makes the premises of the argument T and the conclusion F.

2. We are interested in showing the correctness of the logical system we have used in this course, namely the system PL of **Predicate Logic** consisting of MP, CP, IP, QE, ∀Out and ∃Out. We want to show that PL is adequate, i.e. validity and provability in PL correspond.

Adequacy: An argument is valid iff it can be proven in PL.

3. To show adequacy of PL, we must show both directions of the iff:

   Consistency: If an argument can be proven in PL then it is valid.
   Completeness: If an argument is valid, then it can be proven in PL.

4. Demonstration of the **Completeness of PL.**
Given the proof that the tree method works, and given the method we explained for converting each tree into a proof, we can easily show the completeness of PL.

A. If an argument is valid, then the tree for the argument is closed.

Proof. Suppose we have a valid argument, and assume for the purposes of Indirect Proof that its tree has an open branch. By the correctness of the tree method, we know that an open branch corresponds to a counterexample to the argument. So the argument would have to be invalid, contrary to our first assumption. So the assumption that the tree has an open branch is wrong, and we conclude that when an argument is valid, its tree is closed.

B. If the tree for an argument is closed, then the argument can be proven in PL.

Proof. If the tree for an argument is closed, we can convert it into a proof using the method for converting trees to proofs.

Combining A and B using *Chain* , we have the completeness result:

C. If an argument is valid then it can be proven in PL.

Semantical Consistency of Predicate Logic

To show PL is <u>consistent</u>, we must show:
If an argument is provable, it is valid.

Each line of a proof has a corresponding argument. The <u>corresponding argument</u> has the line as a conclusion and all hypotheses under which it lies as premises.

EXAMPLE:

1.	p		corresponds to $p \vdash p$
2.	| q		corresponds to $p, q \vdash q$
3.	| p 1, reit	corresponds to $p, q \vdash p$	
4.	q→p 2-3 CP	corresponds to $p \vdash q \rightarrow p$	

Notice that the last line of this proof corresponds to the argument $p \vdash q \rightarrow p$, which is the argument proven in lines 1-4.

EXERCISE. Construct the corresponding arguments for each line of your solution of the following problems: $A \rightarrow (B \rightarrow C) \vdash (\sim C \& B) \rightarrow \sim A$ and $(x)(Ax \rightarrow Bx), (\exists x)Ax \vdash (\exists x)Bx$. Convince yourself that each of the corresponding arguments is valid.

We will show that every provable argument is valid by showing instead that every line of a proof (including the last one) corresponds to a valid argument. This will be shown by demonstrating that the hypothesis lines correspond to valid arguments, and that the rules <u>preserve validity</u>, i.e. if the line(s) to which a rule applies correspond to valid arguments then so does the result of applying that rule. When this has been shown, it will follow that all lines of a proof correspond to valid arguments, for every line is either a hypothesis or follows from hypotheses by rules that preserve validity.

It is clear that all assumption lines correspond to valid arguments because those arguments always include the conclusion in their list of premises. (See lines 1 and 2 of the example.) Now we must show that each rule preserves validity:

<u>Reiteration</u>

H ⋮ B	corresponds to $H \vdash B$	To show that Reit preserves validity, we show that if $H \vdash B$ is valid then so is $H, A \vdash B$. But if $H \vdash B$ is valid then $H, A \vdash B$ has to be valid, because if not, $H, A \vdash B$ would have a counterexample with H=T, A=T and B=F, and this would be a counterexample to $H \vdash B$.
| A | ⋮ B	corresponds to $H, A \vdash B$	

Conditional Proof

H
:
| A
| :
| B corresponds to H, A ⊢ B

A→B corresponds to H ⊢ A→B

To show CP preserves valdity, we show that if H, A ⊢ B is valid then so is H ⊢ A→B. If H, A ⊢ B is valid, then H ⊢ A→B must also be valid, for otherwise there is a counterexample with H=T, A→B = F. If A→B = F, A=T and B=F and this is a counterexample to H, A ⊢B.

EXERCISE: In similar fashion, show that MP, IP, QE and UO preserve validity.

Unfortunately we can not show that ∃Out preserves validity because it does not. However, there is a related rule C∃Out that does preserve validity, and all proofs using ∃Out can be rewritten using C∃Out.

C∃Out (Correct Existential Out)
H
:
∃xAx corresponds to H ⊢ ∃xAx
| An
| :
| C corresponds to H, An ⊢ C

C corresponds to H ⊢ C

RESTRICTION: The letter n must not be in H, ∃xAx or C

To show C∃Out preserves validity, assume H ⊢ ∃xAx and H, An ⊢ C are both valid, and that the letter n is not in H, ∃xAx or C. Assume for Indirect Proof that H ⊢ C does have a counterexample with H=T and C=F. Since H ⊢ ∃xAx is valid, this assignment makes ∃xAx = T. But if ∃xAx is T, then for some o in the UD of this assignment, Ao is T. Now modify this assignment in the following way. Take each sentence Bn containing n and adjust its value so that it agrees with the value of Bo. The new assignment will agree with the old one for values of H, and C since n does not appear in these sentences, so the new assignment rules that H=T and C=F, since the old one did. Furthermore, the new assignment has An=T, and so it is a counterexample to H, An ⊢ C. But H, An ⊢ C was assumed to be valid, hence it cannot have a counterexample. We conclude that our assumption for Indirect Proof is wrong, and that H ⊢ C could not have a counterexample. If follows that if both H ⊢ ∃xAx and H, An ⊢ C are valid then so is H ⊢ C.

Logic II Final Study Guide

CONCEPTS:

Universe of Discourse, Decision Procedure, Counterexample, Completeness, Consistency, Derivable Rule, Vacuous Quantification, Reflexive, Irreflexive, Nonreflexive, Symmetric, Asymmetric, Nonsymmetric, Transitive, Intransitive, Nontransitive, Valid Argument

SKILLS:

TREES in predicate logic (Notes pp. 3-12 and p. 19)

-- Use predicate logic rules only on sentences with predicate logic main connectives, and propositional rules only on sentences with propositional logic connectives

-- New letter with ∃Out

-- Need an instance for every letter in the tree with ∀Out (so you may need to return to universal sentences)

-- Know how to construct formal counterexample from an open tree

TRANSLATION

-- Distinguish singular (individual) from general terms.

-- Know all expressions on the Pred. Logic Trans. Chart (Notes p. 18).

-- Know how to adjust translation to the Universe of Discourse given.

-- Know how to translate one expression at a time.

-- Be able to distinguish sentences with propositional logic forms from ones with predicate logic forms. (Signs: 1) 'if' 'both' or 'either' at the beginning, 2) two quantifier expressions, 3) pronoun cross reference)

-- Quickly translate simple problems as in Notes p. 23.

-- Correctly handle examples like: Dogs and cats are pets.

-- Know difference between focused (∃x first) and unfocussed (∀x first) interpretations and be able to translate each.

-- Identify premises and conclusions of an argument using premise indicator (because, since) and conclusion indicator (thus, therefore) words.

-- Know to replace 'x is a father' with 'x is a father of someone' (∃yFxy) in case Fxy is the only vocabulary allowed.

-- Know how to manage pronoun reference to a quantifier in a relative clause: 'Anyone who owns a donkey feeds it.' (Move ∃ in antecedent to a universal on the outside.)

PROOFS in predicate logic (Ch 11 style) (Notes p. 12, 13, plus QE, ∃Out and ∀Out)

-- Know to apply ∃Out and QE before ∀Out.

-- Know to use sentences such as ∀xFx→∀xGx, by setting the antecedent ∀xFx as a goal, not an assumption.

-- Know that it is sometimes necessary to use ∀Out on the same line with many different instances.

-- Know how to select instances for ∀Out by matching sentences already in the proof.

-- Convert from one connective to another quickly (Notes, p. 1, 2)

METALOGIC

-- Convert a tree into a proof, using →⊥ (Notes, p. 20)

-- Calculate the truth value of a predicate logic sentence (Notes, p. 21)

-- Create informal counterexamples.

-- Know the information about decision procedures (Notes pp. 25)

-- Prove that the tree method works (Notes, p. 22)

-- Prove completeness of predicate logic (Notes, p. 26)