Post Canonical System: Examples First

The definition follows some examples.

A Post system for sums:

\[
\begin{align*}
\bar{N} & \quad \bar{N}x \\
N & \quad Nx \\
\quad + y = y \\
\quad x + y = z \\
\end{align*}
\]

A Post system for Hofstadter's MU puzzle:

\[
\begin{align*}
\bar{MI} & \quad \bar{xI} \\
MI & \quad xI \bar{U} \\
\quad \bar{M}x \quad Mx \\
\quad x \bar{IIIy} \quad xIIIy \\
\quad x \bar{UUy} \quad xUUy \\
\quad xUy \\
\quad xy \\
\end{align*}
\]
A Post system consists of a list of signs, a list of variables, and a finite set of productions. The signs form the alphabet of the canonical system. A term is a string of signs and variables, a word is a string of signs, and a production is a figure of the form

\[ t_1 \quad t_2 \quad \cdots \quad t_n \]
\[ \overline{t} \]

where \( t, t_1, \ldots, t_n \) (\( n \geq 0 \)) are all terms. The \( t_i \) are called the premises and \( t \) the conclusion of the production. A production without premises (\( n = 0 \)) is called an axiom. An instance of a production is obtained from a production by substituting strings of signs for all the variables, the same string being substituted for all occurrences of one and the same variable.
Like Turing machines, Markov algorithms, Herbrand-Gödel recursive functions, and lambda calculus, Post systems are a computational formalism. But they are more akin to unrestricted grammars (which can also be viewed as a computational formalism). Both are good at describing strings. Grammars (especially CFG) are more intuitive, but Post systems are good at pattern matching.

<table>
<thead>
<tr>
<th>Post system</th>
<th>grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>signs</td>
<td>terminals</td>
</tr>
<tr>
<td>variables</td>
<td>nonterminals</td>
</tr>
<tr>
<td>productions</td>
<td>productions</td>
</tr>
<tr>
<td>string</td>
<td>sentential form</td>
</tr>
<tr>
<td>word</td>
<td>sentence</td>
</tr>
</tbody>
</table>
Church (or Church-Turing) Thesis

*All computational formalisms have the same power.* Every general, computational formalism so far proposed is equivalent to all the others.
Two productions:

\[ \frac{N}{N} \quad \frac{Nx}{Nx} \]

This Post system derives the word \( N \). Other words can be derived from \( N \) using the second production. Instances of the second production are:

\[ \frac{N}{N} \quad \frac{N}{N} \quad \frac{N}{N} \quad \frac{N}{N} \]

In the first case the variable \( x \) has been replaced by the empty string. This instance can be used in the proof of \( N \):
Tally Notation Equivalent to $\mathbb{N}$

\begin{align*}
0 & \quad N \\
1 & \quad N | \\
2 & \quad N || \\
3 & \quad N ||| \\
4 & \quad N |||| \\
\vdots & \\
\end{align*}

Hence, this Post system derives strings of the form $N \mid \cdots \mid$. The set of derivable strings has a one-to-one correspondence with the natural numbers $\mathbb{N}$. 
Post System: Addition Tables

The following Post system makes use of the tally notation for natural numbers previously introduced. This Post system derives addition equations of the form \( x + y = z \). It uses the variables \( x, y, \) and \( z \), and the signs \( \{ N, |, +, = \} \).

\[
\begin{array}{cccc}
N & Nx & Ny & x + y = z \\
\hline
N & Nx | & + y = y & x | + y = z |
\end{array}
\]

The last two rules reflect the recursive definition of addition in terms of the successor function (concatenating \( | \) to a string is like adding one). From this Post system correct equations are derivable like \( 2 + 2 = 4 \):

\[
\begin{array}{cccc}
N & N | & N || & x + y = z \\
\hline
N | & N || & + || = || \\
N || & + || = || |
\end{array}
\]
Other interesting Post systems can be built using just a few rules. The MIU system of Hofstadter can be framed in terms of a Post system.

\[
\begin{align*}
& \text{MI} \quad \frac{xI}{xIU} \quad \frac{Mx}{Mxx} \quad \frac{xIIIy}{xUy} \quad \frac{xUUy}{xy}
\end{align*}
\]

This Post system produces strings beginning with M and containing I and U.
A Proof In The MIU Post system

\[
\begin{align*}
\text{MI} & \quad \text{axiom} \\
\text{MII} & \quad \text{rule2} \\
\text{MI} & \quad \text{rule2} \\
\text{MII} & \quad \text{rule1} \\
\text{MIII} & \quad \text{rule3} \\
\text{MIII} & \quad \text{rule2} \\
\text{MIIIU} & \quad \text{rule4} \\
\text{MIU} & \\
\text{MIU} & \\
\text{MIU} & \\
\text{MIU} & \\
\text{MIU} & \\
\text{MIIU} &
\end{align*}
\]
Is MU Derivable?

**FIGURE 11.** A systematically constructed “tree” of all the theorems of the MIU-system. The Nth level down contains those theorems whose derivations contain exactly N steps. The encircled numbers tell which rule was employed. Is MU anywhere in this tree?
Theorem

All strings derivable in the Hofstadter’s MIU system are red.

Proof.

The proof is by structural induction. There are five cases based on the five productions of the Post system.

(a) Consider the axiom $\text{MI}$. We show $\text{MI}$ is red.
(b) By induction assume $xI$ is red for any string $x$. We now show that $xIU$ is red.
(c) By induction assume $Mx$ is red for any string $x$. We now show that $Mxx$ is red.
(d) By induction assume $xIIIy$ is red for any strings $x$ and $y$. We now show that $xUy$ is red.
(e) By induction assume $xUy$ is red for any string $x$. We now show that $xUUy$ is red.
A second system and equivalent system for deriving additions equations. It is trickier than the first. It does have a production with more than one premise.

\[
\begin{array}{c|c|c}
N & Nx & Nx \quad Ny \\
\hline
N & Nx & x + y = xy
\end{array}
\]
Propositional logic consists of a collection of propositions $P$, $R$, $Q$, etc., and statements about these propositional symbols using various connectives. For example, $\neg P$, $P \& Q$, $P \Rightarrow Q$.

The Post system uses the set $\{P, |, N, C, F, Th\}$ as signs and $\{x, y, z\}$ as variables. We have two productions for propositions.

\[
\begin{array}{c}
\bar{P} & \quad Px \\
\end{array}
\]

The words of the form $Px$ provable in the Post system are concrete representations of propositions. This set of words is particularly simple. It is just the set

\[
\{P, P |, P ||, \ldots\}
\]
Propositional logic is *not* really about propositions.
Correct logic?

If April is rainy, then flowers will bloom in May and mosquitoes will thrive in June. If mosquitoes thrive in June, then malaria will increase in July. If flowers bloom in May, there will be a lot of honey in September. If April is not rainy, then the lawns will be brown this summer. Hence either there will be a lot of honey in September and malaria will increase in July, or the lawns will be brown this summer.

Consistent?

If the roof needs repair or the house has to be painted, then either the house will be sold or no vacation will be taken this summer. The house will be sold if and only if the roof needs repair and a vacation will be taken this summer. If the house has to be painted, then the house will not be sold or the roof does not need repair. Either a vacation will be taken this summer, or the house has to be painted and the house will be sold.
Correct logic?

If A, then B and M. If M, then J. If B, there will be H. If not A, then L. Hence either there will be H and J, or L.

\[(A \Rightarrow B \& M) \& (M \Rightarrow J) \& (B \Rightarrow H) \& (\neg A \Rightarrow L) \Rightarrow ((H \& J) \lor L)\]

Consistent?

If R or P, then either D or no V. D if and only if R and V. If P, then D or R. Either V, or P and D.

A set \( S \) is inconsistent if for some \( \phi \), \( S \Rightarrow \phi \) and \( S \Rightarrow \neg \phi \).

\( \neg (\phi \Rightarrow \text{false}) \) iff \( \phi \).

\[((R \lor P) \Rightarrow (D \lor \neg V)) \& (D \iff (R \& V)) \& (P \Rightarrow (D \lor R)) \& (V \lor (P \& D))\]
(From Sterling and Shapiro.) There are five houses each of a different color and inhabited by a man of a different nationality, with a different pet, drink and brand of cigarettes.

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. Coffee is drunk in the green house.
4. The Ukrainian drinks tea.
5. The green house is immediately to the right of the ivory house.
7. Kools are smoked in the yellow house.
8. Milk is drunk in the middle house.
9. The Norwegian lives in the first house on the left.
10. The man who smokes Chesterfields lives in the house next to the man with the fox.
11. Kools are smoked in the house next to the house where the horse is kept.
12. The Lucky Strike smoker drinks orange juice.

Who owns the zebra and who drinks water?
After baking a pie for the two nieces and two nephews who are visiting her, Aunt Nellie leaves the pie on her kitchen table to cool. Then she drives to the mall to close her boutique for the day. Upon her return she finds that someone has eaten one quarter of the pie (and even had the nerve to leave her or his dirty plate next to the remainder of the pie). Since no one was in her house that day—except for the four visitors—Aunt Nellie questions each niece and nephew about who ate the piece of pie. The four “suspects” tell her the following:

- Charles: Kelly ate the piece of pie.
- Dawn: I did not eat the piece of pie.
- Kelly: Tyler ate the pie.
- Tyler: Kelly lied when she said I ate the pie.

If only one of these four statements is true and only one of the four committed this heinous crime, who is the vile culprit who Aunt Nellie will have to punish severely?
Babies are illogical. Nobody is despised who can manage a crocodile. Illogical persons are despised. Therefore babies cannot manage crocodiles.

\[ \forall x (B(x) \Rightarrow \neg L(x)) \]

\[ \forall x (C(x) \Rightarrow \neg D(x)) \]

\[ \forall x (\neg L(x) \Rightarrow D(x)) \]

Therefore, \[ \forall x (B(x) \Rightarrow \neg C(x)) \]
We have three productions for formulas.

\[
\begin{align*}
&\frac{Px}{FPx} & & \frac{Fx}{FNx} & & \frac{Fx \ Fy}{FCxy}
\end{align*}
\]

The words of the form \(Fx\) provable in the Post system are concrete representations of formulas of the propositional logic. Strings of the form \(FNx\), correspond to negated formulas and strings of the form \(FCxy\) correspond to implications. The prefix notation is convenient, since no parentheses are required.
For theorems we have four productions. The first three productions correspond to three “axioms” of propositional logic.

\[
\begin{align*}
& Fx \\
\frac{}{ThCCNxxx} \\
& Fx \quad Fy \\
\frac{}{ThCxCNxy} \\
& Fx \quad Fy \quad Fz \\
\frac{}{ThCCxyCCyzCxz}
\end{align*}
\]

Curiously these productions are not axioms in the Post system because of the well-formedness conditions that are the premises of the productions. These conditions are needed to ensure that if a term has the form \( Thx \) then \( Fx \) is a formula. The last production corresponds to \textit{modus ponens}.

\[
\begin{align*}
& ThCxy \quad Thx \\
\frac{}{Thy}
\end{align*}
\]
Propositional Logic

Why these rules?

_modus ponens_ (Latin for “mode that affirms”)

1. If democracy is the best system of government, then everyone should vote.
2. Democracy is the best system of government.
∴ Therefore, everyone should vote.

_modus tollens_ (Latin for “mode that denies”)

Consider an example:

1. If there is fire here, then there is oxygen here.
2. There is no oxygen here.
∴ Therefore, there is no fire here.

Another example:

1. If Lizzy was the murderer, then she owns an axe.
2. Lizzy does not own an axe.
∴ Therefore, Lizzy was not the murderer.
The law of *modus ponens*, or “implication elimination” as it is sometimes called today, expresses the idea that if the formula $\phi$ implies $\psi$ and we know $\phi$ holds, then we can conclude that $\psi$ holds. We can accept or reject this production as we judge most appropriate. This is reminiscent of the Lewis Carroll’s clever story “What the Tortoise said to Achilles.” In the end the exasperated Achilles remarks that logic will take the Tortoise by the throat and force it to accept the law of modus ponens.
Derives some formulas: \( A, B, \) and \((\neg A \Rightarrow A) \Rightarrow A\).

Derives some formulas: \( P_0, P_1, \) and \((\neg P_0 \Rightarrow P_0) \Rightarrow P_0\).
axiom3

\[(\neg A \Rightarrow A) \Rightarrow ((A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow A) \Rightarrow B))\]
Using *modus ponens*

\[
\begin{array}{c}
P \\
\hline
\bar{P} \\
\hline
\bar{P} \\
\hline
\bar{P} \\
\hline
\bar{P} \\
\hline
\bar{P} \\
\hline
\bar{P} \\
\hline
ThCCCPP |
\end{array}
\]

\[
\begin{array}{c}
FPP \\
\hline
FNP \\
\hline
FP \\
\hline
FP \\
\hline
FP \\
\hline
CCNPP |
\end{array}
\]

\[
\begin{array}{c}
FCNPP \\
\hline
FCNPP |
\end{array}
\]

\[
\begin{array}{c}
ThCCCNPPPPCCCPP |
\end{array}
\]

\[
\begin{array}{c}
CCNPPP |
\end{array}
\]

\[
\begin{array}{c}
ThCCNP |
\end{array}
\]

\[
\begin{array}{c}
ThCCNP |
\end{array}
\]

---

axiom 3

\[
((\neg A \Rightarrow A) \Rightarrow A) \Rightarrow ((A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow A) \Rightarrow B))
\]

axiom 1

\[
(\neg A \Rightarrow A) \Rightarrow A
\]

\[
(A \Rightarrow B) \Rightarrow ((\neg A \Rightarrow A) \Rightarrow B)
\]
Proof of $A \Rightarrow A$.

\[
\begin{array}{c}
\text{axiom 1} \\
\hline
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}

\text{ThCCNPPP}

(\neg A \Rightarrow A) \Rightarrow A
\end{array}
\]

\[
\begin{array}{c}
\text{axiom 2} \\
\hline
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}
\begin{array}{c}
\bar{P} \\
FP
\end{array}

\text{ThCPCNPP}

A \Rightarrow (\neg A \Rightarrow A)
\end{array}
\]
axiom3

\[(A \Rightarrow (\neg A \Rightarrow A)) \Rightarrow (((\neg A \Rightarrow A) \Rightarrow A) \Rightarrow (A \Rightarrow A))\]
\[ \frac{P}{FP} \quad \frac{\overline{P}}{\overline{P}} \quad \frac{FP}{FNP} \quad \frac{P}{FP} \quad \frac{\overline{P}}{\overline{P}} \quad \frac{FP}{FP} \quad \frac{P}{FP} \quad \frac{FP}{FP} \quad \frac{\overline{P}}{\overline{P}} \]

\[ \frac{\overline{P}}{FP} \quad \frac{FP}{FCNPN} \quad \frac{P}{FP} \quad \frac{\overline{P}}{FP} \quad \frac{FP}{FP} \quad \frac{\overline{P}}{FP} \quad ThCCPCNPPCCCNPPPPCPPPP \quad ThCPCNPP \quad ThCCNPPPPCPPPP \quad ThCCCNPPPPCPP
\]

\[ ThCPP \]

**axiom3**

\[ \frac{(A \Rightarrow X) \Rightarrow ((X \Rightarrow A) \Rightarrow (A \Rightarrow A))}{(X \Rightarrow A) \Rightarrow (A \Rightarrow A)} \]

**axiom2**

\[ A \Rightarrow X \]

**axiom1**

\[ X \Rightarrow A \]

\[ A \Rightarrow A \]

where \( X \) is \((\neg A \Rightarrow A)\).
Theorem. The term $Thx$ is derivable in the Post system for propositional calculus if, and only if, the formula represented by $Fx$ is valid.

The theorem holds for the usual definition of valid. What is the definition?
The collection of words of the form $Thx$ is not just a random collection of symbols. We are compelled to accept them as useful, because they appear to be true/valid. But what does it mean for a string of the form $Thx$ to be true/valid? We answer this question by giving the usual semantics to propositional logic.

We require, first, a notion of assignment. An assignment is a function $\sigma$ from propositions to the set $\{\top, \bot\} = \text{Bool}$, or equivalently from the natural numbers to $\text{Bool}$. If $\sigma(i) = \top$, then $P_i$ is a true proposition. The set of all possible assignments is denoted by $\Sigma$. 
Example Assignments

fun sigma_0 _ = false;

\[ \sigma_0(i) = \bot \]

fun sigma_1 _ = true;

\[ \sigma_1(i) = \top \]

fun sigma_2 i = i \% 2 = 0;

\[ \sigma_2(i) = \begin{cases} \top, & \text{if } i \text{ is even,} \\ \bot, & \text{otherwise.} \end{cases} \]

fun sigma_3 j = not (sigma_2 j);

fun sigma_4 i = (i=4);
Semantics of Propositional Logic

We associate with each proposition a nonnegative integer in the following manner:

1. \( \mathcal{N}[P] = 0 \)
2. \( \mathcal{N}[P \mid] = \mathcal{N}[P] + 1 \)

Assignments \( \sigma \in \Sigma \) are functions from natural numbers to boolean values.

The semantics for propositional logic is given by a function \( \mathcal{M} \) from \( \Sigma \) to \( \text{Bool} \):

1. \( \mathcal{M}[FPx](\sigma) = \sigma(\mathcal{N}[Px]) \)
Definition. We say that an assignment satisfies or models a formula $Fx$, written $\sigma \models Fx$, if $\mathcal{M}[Fx] \sigma = \top$. If $\mathcal{M}[Fx] \sigma = \bot$, we write $\sigma \not\models Fx$.

Definition. We say that a formula $Fx$ of propositional logic is valid or a tautology if $\sigma \models Fx$ for all assignments $\sigma$, in other words, $\mathcal{M}[Fx] \sigma = \top$ for all assignments $\sigma$. 
Theorem
The formula \( \neg Q \) is not a tautology.

Proof.
Let \( \sigma_0 \) be the assignment that associates \( \top \) to all propositions. From the definition of \( M[\[] \), \( M[\neg Q] \sigma_0 \) is the opposite of \( M[Q] \sigma_0 \). Since \( M[Q] \sigma_0 \) is \( \top \), \( M[\neg Q] \sigma_0 \) is \( \bot \), so the formula \( \neg Q \) is not a tautology.

\[ \square \]

Theorem
For all formulas \( \phi \), the formula \( \phi \Rightarrow \phi \) is a tautology.

Proof.
Given an arbitrary assignment \( \sigma \), either \( M[\phi] \sigma = \top \) or \( M[\phi] \sigma = \bot \). Suppose \( M[\phi] \sigma = \top \). By the definition of \( M[\[] \) for implication, if the consequent if true, \( M[\phi \Rightarrow \phi] \sigma = \top \). Suppose \( M[\phi] \sigma = \bot \). Again by the definition of \( M[\[] \) for implication, if the hypothesis is false, \( M[\phi \Rightarrow \phi] \sigma = \top \). Either way, \( M[\phi \Rightarrow \phi] \sigma = \top \) for all \( \sigma \).  

\[ \square \]
Theorem
If $\sigma(i) = \sigma'(i)$ for all $i$ such that $P_i$ is in $\phi$, then $M[\phi]\sigma = M[\phi]\sigma'$.

Proof.
Proof by induction on $\phi$. If $\phi$ is the proposition $P_i$, then $M[P_i]\sigma = \sigma(i) = \sigma'(i) = M[P_i]\sigma'$. If $\phi$ is the formula $\neg \chi$, then the induction hypothesis is $M[\chi]\sigma = M[\chi]\sigma'$. The conclusion following immediately. If $\phi$ is the formula $\chi \Rightarrow \psi$, then the induction hypothesis is $M[\chi]\sigma = M[\chi]\sigma'$ and $M[\psi]\sigma = M[\psi]\sigma'$. Again the conclusion following immediately.

So, only a finite number of propositions matter, and hence the truth table method works.
Propositions

Strongly-typed, functional languages with recursive data types are good for tree algorithms. The data structure for a proposition is a tree.

datatype prop = (* SML *)
    prop of string |
    neg of prop |
    impl of prop * prop;

data Formula = -- Haskell
    Prop String |
    Neg Formula |
    Impl (Formula,Formula)
Examples in Post system, math, SML, and Haskell:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Impl</th>
<th>Impl</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCPP</td>
<td>$P \Rightarrow P$</td>
<td>impl($P$, $P$)</td>
<td>Impl($p$, $p$)</td>
</tr>
<tr>
<td>FCP</td>
<td>$R \Rightarrow Q$</td>
<td>impl($R$, $Q$)</td>
<td>Impl($r$, $q$)</td>
</tr>
<tr>
<td>FNCPP</td>
<td>$\neg (P \Rightarrow Q)$</td>
<td>neg(impl($P$, $Q$))</td>
<td>Neg(Impl($p$, $q$))</td>
</tr>
<tr>
<td>FCNNPP</td>
<td>$((\neg \neg P) \Rightarrow P)$</td>
<td>impl(neg(neg $P$), $P$)</td>
<td>Impl(Neg(Neg$P$), $P$)</td>
</tr>
<tr>
<td>FCPP</td>
<td>$P \Rightarrow (Q \Rightarrow P)$</td>
<td>impl($P$, impl($Q$, $P$))</td>
<td>Impl($P$, Impl($Q$, $P$))</td>
</tr>
</tbody>
</table>
A propositional formula is a tautology if it is true for all possible assignments to the propositions.

\[
\begin{align*}
(P \Rightarrow P) & \quad \text{impl}(P, P) \\
((\neg\neg P) \Rightarrow P) & \quad \text{impl}(\neg (\neg P), P) \\
(P \Rightarrow (\neg\neg P)) & \quad \text{impl}(P, \neg (\neg P)) \\
(P \Rightarrow (Q \Rightarrow P)) & \quad \text{impl}(P, \text{impl}(Q, P)) \\
(P \Rightarrow (Q \Rightarrow Q)) & \quad \text{impl}(P, \text{impl}(Q, Q)) \\
((\neg P \Rightarrow P) \Rightarrow P) & \quad \text{impl}(\text{impl}(\neg P, P), P) \\
(P \Rightarrow (\neg P \Rightarrow Q)) & \quad \text{impl}(P, \text{impl}(\neg P, Q)) \\
(\neg P \Rightarrow (P \Rightarrow Q)) & \quad \text{impl}(\neg P, \text{impl}(P, Q)) \\
(((\neg (P \Rightarrow P)) \Rightarrow Q) & \quad \text{impl}(\neg (\text{impl}(P, P)), Q) \\
(P \Rightarrow (\neg (P \Rightarrow \neg P))) & \quad \text{impl}(P, \neg (\text{impl}(P, \neg P))) \\
((P \Rightarrow \neg P) \Rightarrow \neg P) & \quad \text{impl}(\text{impl}(P, \neg P), \neg P) \\
\end{align*}
\]
First some preliminary definitions.

(* assignment of infinite number of propositions to their value *)

type assignment = string -> bool;

(* value of a formula given an assignment *)

fun value sg (prop n)       = sg n
  | value sg (impl (h,s)) = not (value sg h) orelse (value sg s)
  | value sg (neg phi)    = not (value sg phi)
;

exception not_found of string;

fun undef n = raise not_found n;

fun update f x y z = if z=x then y else f z;
Semantic tautology checker in SML:

```ml
local
    fun check' phi sg nil = value sg phi
    | check' phi sg (v::vs) =
        check' phi (upd sg v true) vs andalso
        check' phi (upd sg v false) vs
in
    fun check phi =
        check' phi undef (props phi nil);
end;
```
Semantic tautology checker in Haskell:

```haskell
check phi = check' phi undef (props phi [])
where
    check' phi sg [] = value sg phi
    check' phi sg (v:vs) =
        check' phi (upd sg v True) vs &&
        check' phi (upd sg v False) vs
```

The Haskell community considers this vulgar, but it does prune some searching in cases like:

\[ P \Rightarrow (Q \Rightarrow (R \Rightarrow S)) \]
check \( \text{impl}(P, \text{impl}(Q, \text{impl}(R, S))) \) ->
check' undef \( \text{impl}(P, \text{impl}(Q, \text{impl}(R, S))) \) ->
value undef \( \text{impl}(P, \text{impl}(Q, \text{impl}(R, S))) \) handle ... -->

\[ \text{not (value undef P) orelse (value undef (impl(Q,impl(R,S)))))} \] handle ...

\[ \text{not (undef P) orelse (value undef (impl(Q,impl(R,S))))} \] handle ...

\( \text{undef P} \Rightarrow \text{raise not_found P} \) -->
check' (upd undef P true) \( \text{impl}(P, \text{impl}(Q, \text{impl}(R, S))) \) andalso 
check' (upd undef P false) \( \text{impl}(P, \text{impl}(Q, \text{impl}(R, S))) \)
check' (upd undef P true) (impl(P,impl(Q,impl(R,S)))) andalso 
   check' (upd undef P false) (impl(P,impl(Q,impl(R,S)))) -->
   value (upd undef P true) (impl(P,impl(Q,impl(R,S))))
   handle ... andalso ... -->
not (value (upd undef P true) P) orelse (value ... (impl(Q,impl(R,S))))
   handle ... andalso ... -->
not ((upd undef P true) P) orelse (value ... (impl(Q,impl(R,S))))
   handle ... andalso ... -->
  (not true) orelse (value ... (impl(Q,impl(R,S))))
   handle ... andalso ... -->
false orelse (value ... (impl(Q,impl(R,S))))
   handle ... andalso ... -->
value (upd undef P true) (impl(Q,impl(R,S))))
   handle ... andalso ... -->
false andalso
   check' (upd undef P false) (impl(P,impl(Q,impl(R,S)))) -->
false
Semantics of Propositional Logic

**Theorem.** The term $Thx$ is derivable in the Post system for propositional calculus if, and only if, the formula represented by $Fx$ is valid.
Semantics of Propositional Logic

Theorem

*If the word Thw is derivable in the Post system for propositional calculus, then the formula represented by Fw is valid.*

The proof is by induction based on the structure of the derivation of Thw in the Post system for propositional calculus. Essentially, if all theorems start out valid and all rules of inference preserve validity, then all theorems are valid.

**Lemma 1.** If the last step in the derivation in the Post system is an instance of the production \( \frac{Fx}{ThCCNxxx} \), then the formula represented by FCCNxxx is valid.

**Lemma 2.** If the last step in the derivation in the Post system is an instance of the production \( \frac{Fx \ Fy}{ThCxCNxy} \), then the formula represented by FCxCNxy is valid.

**Lemma 3.** If the last step in the derivation in the Post system is an instance of the production \( \frac{Fx \ Fy \ Fz}{ThCCxyCCyzCxz} \), then the formula represented by FCCxyCCyzCxz is valid.

**Lemma 4.** If the last step in the derivation in the Post system is an instance of the production
Completeness of Propositional Logic

Can proofs for all tautologies be constructed starting from just these few axioms? They can, and this result is known as the completeness theorem for propositional logic.

**Theorem**

*If the formula represented by $Fx$ is valid, then the word $Thx$ is derivable in the Post system for propositional logic.*

In his doctoral dissertation of 1920 Post was the first to give a proof. He used the propositional subset of *Principia Mathematica*. (He invented Post systems later and did not use them for logic.) The proof is long. And requires some proof building programs.
Proof Trees

The data structure in SML:

datatype proof =
    assume of prop |
    ax1 of prop | (* Lk1 *)
    ax2 of prop*prop | (* Lk2 *)
    ax3 of prop*prop*prop | (* Lk3 *)
    mp of proof*proof*prop;

The data structure in Haskell:

data Proof =
    Assume Formula |
    Ax1 Formula |
    Ax2 (Formula,Formula) |
    Ax3 (Formula,Formula,Formula) |
    Mp (Proof,Proof,Formula)

Completeness is a function with input prop and output proof.
Proof Trees

fun axiom1 p = impl (impl(neg p,p),p);
fun axiom2 (p,q) = impl (p, impl(neg p,q));
fun axiom3 (p,q,r) =
    impl (impl(p,q),impl(impl(q,r), impl(p,r)));

fun proof_of (assume p) = p
| proof_of (ax1 p) = axiom1 p
| proof_of (ax2 (p,q)) = axiom2 (p,q)
| proof_of (ax3 (p,q,r)) = axiom3 (p,q,r)
| proof_of (mp (_,_,p)) = p
Generalized notion of proof. $A \vdash \phi$ means the formula $\phi$ is derivable using assumptions from the set $A$. $\emptyset \vdash \phi$ or $\vdash \phi$ means the formula $\phi$ is derivable (without any assumptions). Why? Because it enables powerful new proof building techniques.

(* is the formula "p" assumed in the proof? *)

fun occurs p (assume q) = p=q
| occurs p (mp (p1,p2,_)) = occurs p p1 orelse occurs p p2
| occurs p (_,) = false
exception not_implication of prop;
exception not_hypothesis;

(* The constructor of type "proof" should not be used, because it does not (and cannot) check its arguments to see if they are in the right form. *)

local
  fun check (impl(p,q),r) = if p=r then q else raise not_hypothesis
                             | check (p,_) = raise not_implication p
in
  fun modus_ponens (p,q) = mp (p,q,check (proof_of p, proof_of q))
end;

(* example proofs *)
val P = prop"P"; val Q = prop"Q"; val R = prop"R"; val S = prop"S";
val pr1 = ax3 (impl(neg P,P), P, Q);
val pr2 = modus_ponens (pr1, ax1 (P)); (* (P=>Q) => ((˜P=>P)=>Q) *)
val pr3 = modus_ponens (ax3(P,impl(neg P,P),P), ax2(P,P));
signature PROOF = sig
    type proof
    exception not_implication of prop and not_hypothesis
    val assume : prop -> proof
    val ax1 : prop -> proof
    val ax2 : prop * prop -> proof
    val ax3 : prop * prop * prop -> proof
    val modus_ponens : proof * proof -> proof
    val proof_of : proof -> prop
end
structure Proof :> PROOF = struct

datatype proof =
    assume of prop |
    ax1 of prop   | (* Lk1: (~P => P) => P *)
    ax2 of prop*prop | (* Lk2: P => (~P=>Q) *)
    ax3 of prop*prop*prop | (* Lk3: P=>Q => ((Q=>R)=>(P=>R)) *)
    mp of proof*proof*prop;

exception not_implication of prop and not_hypothesis

fun proof_of (assume p) = p
| proof_of (ax1 p) = axiom1 p
| proof_of (ax2 (p,q)) = axiom2 (p,q)
| proof_of (ax3 (p,q,r)) = axiom3 (p,q,r)
| proof_of (mp (_,_,p))= p;

local
    fun check (impl(p,q),r) = if p=r then q else raise not_hypothesis
    | check (p,_) = raise not_implication p

    in
        fun modus_ponens (p,q) = mp (p,q,check (proof_of p, proof_of q))
    end;

end
let
  val pr1 = ax1 A; (* (¬A=>A)=>A *)
  val pr2 = ax2 (A,A);(* A=>(¬A=>A) *)
  val pr3 = ax3 (A,impl(neg A,A),A);
  val pr4 = modus_ponens (pr3,pr2);
in
  modus_ponens (pr1, pr4)
end
Two Initial Lemmas

Lemma (Backward Propagation)

For any propositional formula \( C \), if \( \vdash A \Rightarrow B \), then
\[ \vdash (B \Rightarrow C) \Rightarrow (A \Rightarrow C), \]

Lemma (Transitivity)

If \( \vdash A \Rightarrow B \) and \( \vdash B \Rightarrow C \), then \( \vdash A \Rightarrow C \).
Two Initial Lemmas

(* backward propagation; derived rule of inference; 
  Given any formula C, |-A=>B ==> |- (B=>C) => (A=>C) *
)
fun backward C (pr1) = 
  let
    val impl(A,B) = proof_of (pr1);
  in
    (* ax3: A=>B => ((B=>C)=>(A=>C)) *)
    modus_ponens (ax3(A,B,C), pr1)
  end;
(* transitivity; derived rule of inference;
   |- A=>B, |- B=>C ==> |- A=>C
   *)
fun transitivityity (pr1, pr2) = 
  let
    val impl (A, B) = proof_of (pr1);
    val impl (_, C) = proof_of (pr2);
    (* ax3: A=>B => ((B=>C)=>(A=>C)) *)
    val pr3 = modus_ponens (ax3 (A, B, C), pr1); (* (B=>C)=>(A=>C) *)
in
    modus_ponens (pr3, pr2)
  end;
Other Lemmas

Lemma
If $\vdash \neg B \Rightarrow \neg A$, then $\vdash (A \Rightarrow B)$.

Lemma
For any propositional formula $B$, if $\vdash A$, then $\vdash (B \Rightarrow A)$.

Lemma
If $\vdash B$ and $\vdash A \Rightarrow (B \Rightarrow C)$, then $\vdash (A \Rightarrow C)$.

Lemma
For any propositional formulas $A$ and $B$, $\vdash (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$.

Lemma
For any propositional formulas $A$ and $B$, $\vdash A \Rightarrow (B \Rightarrow A)$.

Lemma
For any propositional formula $A$, $\vdash \neg \neg A \Rightarrow A$. 
fun lemma_2 B pr1 = 
  let 
    val A = proof_of pr1;
    val pr2 = ax2(A,neg B);  (* A => (~A=>~B) *)
    val pr3 = modus_ponens (pr2,pr1);  (* ~A=>~B *)
  in 
    lemma_1 pr3  (* B=>A *) 
  end;
fun lemma_6 A =
   let
      val pr1 = lemma_5(neg(neg A),neg A); (*\neg\neg A => (\neg A => \neg\neg A)*)
      val pr2 = lemma_4(neg A, A); (* (\neg A => \neg\neg A) => (\neg A => A)*)
      val pr3 = transitivity(pr1, pr2); (* \neg\neg A => (\neg A => A)*
   in
      transitivity (pr3, ax1 A) (* \neg\neg A => A *)
   end;
Other Lemmas

Lemma
For any propositional formula $A$, $\vdash A \Rightarrow \neg \neg A$.

Lemma
If $\vdash A \Rightarrow B$ and $\vdash \neg A \Rightarrow B$, then $\vdash B$.

Lemma
For any propositional formulas $A$ and $B$, $\vdash \neg A \Rightarrow (A \Rightarrow B)$.

Lemma
If $\vdash A \Rightarrow (A \Rightarrow B)$, then $\vdash A \Rightarrow B$.

Lemma
If $\vdash A \Rightarrow (B \Rightarrow C)$ and $\vdash A \Rightarrow B$, then $\vdash (A \Rightarrow C)$.
Deduction Theorem

Theorem (Deduction)

Given a proof of $A$ (possibly assuming $B$), there is a proof of $B \Rightarrow A$ without any assumptions of $B$. 
fun deduction B (assume A) =
    if B=A
        (* since A=B, "assume A" is a proof of A using
         an assumption of B. We must get rid of it. *)
    then derived1 (B)   (* B=>B *)
    else lemma_2 B (assume A) (* B=>A *)

| deduction B (mp (p1,p2,_)) =
    (* p1: C=>A  deduction B p1: B=>(C=>A)
     p2: C  deduction B p2: B=>C *)
    lemma_11 (deduction B p1, deduction B p2) (* B=>A *)

| deduction B proofOfA =
    (* proofOfA is an axiom, so it has no assumptions *)
    lemma_2 B proofOfA (* B=>A *)

;
Crucial optimization:

```haskell
fun f p = 
  if occurs a p
  then deduction a p
  else lemma_2 a p;
```
Completeness

Lemma

If $\vdash A$ and $\vdash \neg B$, then $\vdash \neg(A \Rightarrow B)$.

(* Lemma 12. $\vdash A$, $\vdash \neg B$ ==> $\vdash \neg(A\Rightarrow B)$
Requires the deduction theorem and lemma 8. *)

fun lemma_12 (pr1, pr2) =
  let
    val A = proof_of (pr1); val neg B = proof_of (pr2);
    val i = impl (A,B);
    val pr4 = modus_ponens (assume i, pr1); (* i|- B *
    val pr5 = modus_ponens (ax2(B,neg i),pr4); (* i|- \neg B\Rightarrow(A\Rightarrow B) *
    val pr6 = modus_ponens (pr5, pr2); (* i|- \neg(A\Rightarrow B) *
    val pr7 = deduction i pr6; (* (A\Rightarrow B) \Rightarrow \neg(A\Rightarrow B) *)
    val pr8 = derived1 (neg i); (* \neg(A\Rightarrow B) \Rightarrow \neg(A\Rightarrow B) *)
  in
    lemma_8 (pr7, pr8)
  end;
Completeness

Given an assignment $\sigma$, and a formula $\phi$, let $P_\phi$ be the set of propositional formulas occurring in $\phi$, and for each $P \in P_\phi$, let $\hat{P}$ be defined as follows:

$$\hat{P} = \begin{cases} P, & \text{if } \sigma(P) = \top \\ \neg P, & \text{if } \sigma(P) = \bot \end{cases}$$

Let $\hat{P}_{\sigma,\phi}$ be the set of formulas $\hat{P}$ for such that $P$ occurs in $\phi$.

**Lemma**

*Given an assignment $\sigma$, and a formula $\phi$, if $\sigma \models \phi$, then $\hat{P}_{\sigma,\phi} \vdash \phi$. If $\sigma \not\models \phi$, then $\hat{P}_{\sigma,\phi} \vdash \neg \phi$.***

**Proof.**

The proof is by induction on the formula $\phi$. \qed
fun F sg (prop x) = 
    if sg x then assume (prop x) else assume (neg(prop x))

| F sg (neg p) = 
  if value sg p 
      then modus_ponens (lemma_7 p, F sg p) (* |-p=>~p, |-p ==> |-~p *) 
      else F sg p (* |- ~p *)

| F sg (impl(p,q)) = 
  if value sg p 
      then if value sg q 
          then lemma_2 p (F sg q) (* |- q ==> |- p=>q *) 
          else lemma_12 (F sg p, F sg q) (* |-p, |-~q ==> |-~(p=>q) *) 
      else modus_ponens (lemma_9 (p,q), F sg p)

;
Completeness

Theorem (Completeness)

*Given a propositional formula $A$ that is a tautology, then there is a proof of $A$.*

Build a proof of $A$ assuming $P_i$ and $\neg P_i$. (If $A$ is a tautology, this can be done.) Use Lemma 8 to systematically eliminate the use of assumptions $P_1, \neg P_1, P_2, \neg P_2, \ldots$

From

\[\ldots, P \vdash \phi \quad \ldots, \neg P \vdash \phi\]

using the deduction theorem, we get

\[\ldots \vdash P \Rightarrow \phi \quad \ldots \vdash \neg P \Rightarrow \phi\]

and by Lemma 8.

\[\ldots \vdash \phi\]
local
  fun elim v prt prf = lemma_8 (deduction (prop v)prt, deduction (neg (prop v)) prf);

  fun allp phi sg nil = F sg phi |
    | allp phi sg (v::vs) =
      let
        val prt = allp phi (upd sg v true) vs) (* v, ...|- phi *)
        val prf = allp phi (upd sg v false) vs) (* ˜v,...|- phi *)
      in
        elim vprt prf
      end
  in
    fun completeness phi = allp phi undef (prop phi nil)
  end;
local

fun elim v prt prf =
  lemma_8 (deduction (prop v) prt, deduction (neg (prop v)) prf);

fun allp sg phi =
  F sg phi handle not_found v =>
    let
    val prt = allp (update sg v true) phi (* v, ...|- phi *)
    val prf = allp (update sg v false) phi (* ~v,...|- phi *)
    in
    elim v prt prf
    end;

in
  fun completeness phi = allp undef phi
  end;