REASONING IN ARTIFICIAL INTELLIGENCE
Course Notes
for
First International Summer Institute in Cognitive Science
State University of New York at Buffalo
Buffalo, NY, U.S.A
July, 1994

Stuart C. Shapiro
Department of Computer Science
and Center for Cognitive Science
State University of New York at Buffalo
Buffalo, NY 14260-2000
U.S.A.
`shapiro@cs.buffalo.edu`

July 14, 1994

Contents

1 Introduction

We are interested in techniques for programming computers to reason in ways that are similar to the ways that people reason, and in the formal underpinnings of those techniques. **Logic** is the study of correct reasoning. It is not a particular language for knowledge representation and reasoning. There are many systems of logic (or logics). Some of these are quite unlike the way people think; some are closer. All are alike in being systems of reasoning that do not take one from truths to falsities. So if one starts with truths, and reasons according to some logic (i.e., reasons logically), then one's conclusions are also true. Of course, one might start with some truths and some falsities. Then, even reasoning logically, one can end up with falsities. Logic is the underpinning of computer reasoning programs, so we will begin with a study of various logics that have been used in AI.

A complete study of a logic would include the following:

1. **Syntax:** The rules for constructing well-formed sentences of the logic.

2. **Semantics:** The rules for determining the meaning of a sentence of the logic.

3. **Syntactic Inference:** The rules for determining the theorems of the logic.

4. **Semantic Inference:** The rules for identifying the logically valid sentences of the logic.

We will not go through all of the details in each of these areas for each logic we study, but will do enough to give a flavor and to establish the basics of computer reasoning.

2 Propositional Calculus

Propositional (or Sentential) Calculus is a class of logics that do not analyze information below the level of the sentence (or proposition). We will look at several propositional calculi. First, let us look at one "micro-world" we will use as an example.

2.1 Seesaw World

Seesaw (or "teetertotter") world is illustrated in Figure 1. This world can be described (for our purposes) using 4 sentences:

1. *John is down.*

2. *John is up.*

3. *Betty is down.*

4. *Betty is up.*

2.2 The "Standard" Propositional Calculus

We will begin with, and examine in most detail the standard, classical, propositional calculus.

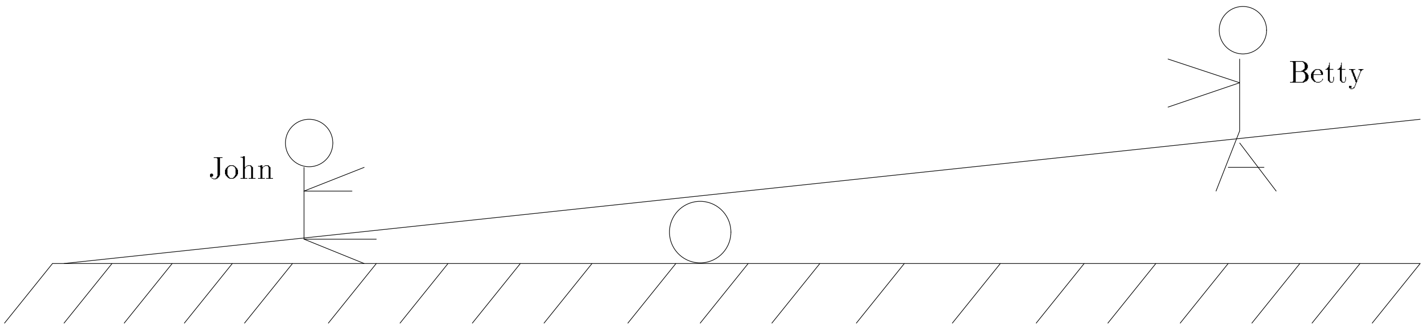

Figure 1: Seesaw World

2.2.1 Syntax

Atomic Propositions: An atomic sentence is any letter of the alphabet, or any letter with a numeric subscript, or any alphanumeric string. For example, P, Q_3 and the four propositions of Seesaw World are atomic propositions.

Well-Formed Propositions (WFPs):

1. Every atomic proposition is a WFP.
2. If P is a WFP, then so is $\neg P$.
3. If P and Q are WFPs, then so is
   (a) $P \wedge Q$
   (b) $P \vee Q$
   (c) $P \Rightarrow Q$
   (d) $P \Leftrightarrow Q$
4. Nothing else is a WFP

For example, *John is down.* $\Rightarrow$ *Betty is up.* and *John is up.* $\Leftrightarrow \neg$ *John is down.* are WFPs.

Note that we have still described a class of propositional calculi because we didn't specify exactly which atomic propositions are in the language. For example, the formalization of Seesaw World is better specified, because we specified the four atomic propositions that are in that language. No other atomic proposition is in our formalization of Seesaw World.

2.2.2 Semantics

The semantics of a logical language can be defined *intensionally* or *extensionally.* An extensionally defined semantics depends on a particular *model, interpretation,* or *situation,* (which we will take to be equivalent terms), while an intensional semantics covers multiple situations. For purposes of reasoning in AI, the intensional semantics is very important, since we want a formalization to cover various particular situations, any of which we might want to reason about. The intensional semantics of an atomic proposition is given by an informal English statement that allows us to determine the truth or falsity of the proposition in each particular situation. For example, the intended intensional semantics of the four atomic propositions in Seesaw World are:

John is down.: The individual named John is in the lower position on his end of the seesaw.

John is up.: The individual named John is in the higher position on his end of the seesaw.

Betty is down.: The individual named Betty is in the lower position on her end of the seesaw.

Betty is up.: The individual named Betty is in the higher position on her end of the seesaw.

Notice, however, that it is not proper to guess the intensional semantics from the English sentence the atomic proposition looks like. We *could* have specified the intensional semantics of the four sentences of Seesaw World as:

John is down.: The individual named Betty is in the higher position on her end of the seesaw.

John is up.: The individual named John is in the lower position on his end of the seesaw.

Betty is down.: The individual named John is in the higher position on his end of the seesaw.

Betty is up.: The individual named Betty is in the lower position on her end of the seesaw.

or we could have made the four sentences of Seesaw World, A, B, C, and D, with the semantics:

A: The individual named John is in the lower position on his end of the seesaw.

B: The individual named John is in the higher position on his end of the seesaw.

C: The individual named Betty is in the lower position on her end of the seesaw.

D: The individual named Betty is in the higher position on her end of the seesaw.

The *extensional* semantics, as we said above, of an atomic proposition is relative to an interpretation. Each interpretation assigns each atomic proposition either the value *True* or the value *False*. (We will use T as synonymous to *True*, and F as synonymous to *False*.) So in one particular Seesaw World situation, we might have the four atomic propositions have the values:

John is down.: *True*

John is up.: *False*

Betty is down.: *False*

Betty is up.: *True*

Clearly, with four atomic propositions, there are $2^4 = 16$ theoretically possible situations. More about this later.

Given the extensional semantics of the atomic propositions of some propositional calculus, we can define the extensional semantics of the WFPs. (The intensional semantics should be obvious from the extensional definitions.) The standard semantics for our standard version of propositional calculus is given by the following rules:

1. $\neg P$ is *True* if P is *False*, and *False* if P is *True*.
2. $P \wedge Q$ is *True* if P is *True* and Q is *True*; otherwise, it is *False*.
3. $P \vee Q$ is *True* if P is *True* or Q is *True* (or both); otherwise, it is *False*.
4. $P \Rightarrow Q$ is *False* if P is *True* and Q is *False*; otherwise, it is *True*.
5. $P \Leftrightarrow Q$ is *True* if both P and Q are *True* or if they are both *False*; otherwise, it is *False*.

We could also specify these semantics by *Truth Tables:*

P	$\neg P$
T	F
F	T

and

P	Q	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

For example, in the Seesaw World situation in which *John is up.* is *False,* and *John is down.* is *True,* *John is up.* $\Leftrightarrow \neg$ *John is down.* is *True.*

2.2.3 Syntactic Inference

Specifying the syntactic inference methods of a logic determines what well-formed sentences of that logic are *theorems.* It also specifies when a given well-formed sentence can correctly be derived from a set of (other) well-formed sentences. The standard notation for saying that P is a theorem is $\vdash P$, and for saying that P follows from $A_1, \ldots, A_n$ is $A_1, \ldots, A_n \vdash P$. The $\vdash$ (pronounced "turnstyle") is sometimes subscripted by a symbol indicating the system of logic in which P is a theorem or in which P follows from $A_1, \ldots A_n$. First, however, we must specify our notion of *proof.* We will show two different ways to do this for the standard propositional calculi: Hilbert-style proofs, and Fitch-style, Natural Deduction proofs.

Hilbert-style Inference Hilbert-style inference systems rely on a set of *axioms* and a very small set of *Rules of Inference.* A proof of a theorem P is a list of WFPs such that each one is either an axiom or follows from previous WFPs in the list by one of the rules of inference, and such that the last WFP in the list is P. A derivation of P from $A_1, \ldots, A_n$ is just like a proof of P, except a WFP is also allowed in the list if it is one of $A_1, \ldots, A_n$.

One set of axioms for the standard propositional calculus is[1]:

(A1). $(A \Rightarrow (B \Rightarrow A))$

(A2). $((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$

(A3). $((\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B))$

Note that these axioms use only the connectives $\Rightarrow$ and $\neg$. The others may be defined as abbreviations of these, or additional axioms may be given for them. For our purposes, we will not bother about them.

There are two attitudes taken about axioms such as these:

1. They are *axiom schemata.* The actual axioms (There are an infinite number of them.) may be derived from them by taking any one axiom schema, and substituting any WFP whatsoever for all occurrences of each of the letters, A, B, and C. We will take this attitude in the example below.

2. They are actual axioms, and the rule of Substitution, sketched above, is a Rule of Inference.

Whether (A1) – (A3) are axioms or axiom schemata, the (other) Rule of Inference is *modus ponens,* which says that if any WFP A is a line of a proof, and a WFP of the form $A \Rightarrow B$ is also a line in the proof, for any WFP B, then B can be added as a line in the proof. This rule of inference can be summarized as

$$\frac{A \Rightarrow B}{B}$$

An example proof of the theorem $A \Rightarrow A$ is:

(1)	$(A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))$	Instance of A2
(2)	$A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$	Instance of A1
(3)	$(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)$	From 1, 2 by MP
(4)	$A \Rightarrow (A \Rightarrow A)$	Instance of A1
(5)	$A \Rightarrow A$	From 3, 4 by MP

[1]These axioms, and the following example, come from E. Mendelson, *Introduction to Mathematical Logic,* (Princeton: D. Van Nostrand) 1964, pp. 31–32.

As a second example, let's derive ¬ *Betty is down.* from *John is down.* ⇒ *Betty is up.*, *Betty is up.* ⇒ ¬ *Betty is down.*, and *John is down.*:

(1)	*John is down.*	Assumption
(2)	*John is down.* ⇒ *Betty is up.*	Assumption
(3)	*Betty is up.*	From 1, 2 by MP
(4)	*Betty is up.* ⇒ ¬ *Betty is down.*	Assumption
(5)	¬ *Betty is down.*	From 3, 4 by MP

Notice that "rules" in the sense of "rule-based systems" correspond not to rules of inference, but to non-atomic assumptions. Rules of inference are used by the underlying "inference engine" of rule-based systems. We sometimes call assumptions *non-logical axioms* because they are used in proofs the same way axioms are, but they are not actually axioms of any logical system. The rules of rule-based systems are actually non-logical axioms. Sometimes, they are also called *domain rules* to distinguish them from rules of inference.

Fitch-style Natural Deduction There are several different versions of "natural deduction" systems, all characterized by having no axioms but many rules of inference. Typically, there are a few *structural* rules of inference, and two rules of inference for each logical connective: an *introduction* rule, and an *elimination* rule. The natural deduction system given here is based on that of F. B. Fitch, *Symbolic Logic: An Introduction,* (New York: Ronald Press), 1952.

A proof of a theorem P in this system is an ordered list of WFPs and subproofs, each of which is also an ordered list of WFPs and subproofs, recursively. P must be the last WFP in the outermost proof. As we will see below, each subproof begins with an assumption or *hypothesis.* A proof of P from assumptions $A_1, \ldots, A_n$ is an outermost subproof whose hypothesis is $A_1 \wedge \ldots \wedge A_n$, and whose last line is P.

The structural rules of inference we will use are:

$$\begin{array}{|l} A \quad Hyp \\ \hline \\ \end{array} \qquad \begin{array}{|l} A \\ \ldots \\ A \quad Rep \end{array} \qquad \begin{array}{|l} A \\ \ldots \\ \begin{array}{|l} A \quad Reit \end{array} \end{array}$$

The introduction and elimination rules are:

$$\begin{array}{|l} \begin{array}{|l} A \\ \hline \ldots \\ B \\ \neg B \end{array} \\ \neg A \quad \neg I \end{array} \qquad \begin{array}{|l} \neg\neg A \\ A \quad \neg E \end{array}$$

$$\begin{array}{|l} A \\ \ldots \\ B \\ A \wedge B \quad \wedge I \end{array} \qquad \begin{array}{|l} A \wedge B \\ A \quad \wedge E \end{array} \qquad \begin{array}{|l} A \wedge B \\ B \quad \wedge E \end{array}$$

$$\begin{array}{|l} A \\ A \vee B \quad \vee I \end{array} \qquad \begin{array}{|l} B \\ A \vee B \quad \vee I \end{array} \qquad \begin{array}{|l} A \vee B \\ \begin{array}{|l} A \quad Hyp \\ \hline \ldots \\ C \end{array} \\ \begin{array}{|l} B \quad Hyp \\ \hline \ldots \\ C \end{array} \\ C \quad \vee E \end{array}$$

$$\begin{array}{|l} \begin{array}{|l} A \quad Hyp \\ \hline \ldots \\ B \end{array} \\ A \Rightarrow B \quad \Rightarrow I \end{array} \qquad \begin{array}{|l} A \\ \ldots \\ A \Rightarrow B \\ B \quad \Rightarrow E \end{array}$$

$$
\begin{array}{|l}
\begin{array}{|ll}
A & Hyp \\ \hline
\ldots & \\
B &
\end{array} \\
\\
\begin{array}{|ll}
B & Hyp \\ \hline
\ldots & \\
A &
\end{array} \\
A \Leftrightarrow B \qquad \Leftrightarrow I
\end{array}
\qquad
\begin{array}{|ll}
A & \\
\ldots & \\
A \Leftrightarrow B & \\
B & \Leftrightarrow E
\end{array}
\qquad
\begin{array}{|ll}
B & \\
\ldots & \\
A \Leftrightarrow B & \\
A & \Leftrightarrow E
\end{array}
$$

For some examples, we will first show the Natural Deduction versions of the proof and the derivation from assumptions of the previous section.

$$
\begin{array}{l}
\begin{array}{|ll}
A & Hyp \\ \hline
A & Rep
\end{array} \\
A \Rightarrow A \qquad \Rightarrow I
\end{array}
$$

$$
\begin{array}{|lr}
John\ is\ down. \wedge (John\ is\ down. \Rightarrow Betty\ is\ up.) \wedge (Betty\ is\ up. \Rightarrow \neg Betty\ is\ down.) & Hyp \\ \hline
John\ is\ down. & \wedge E \\
John\ is\ down. \Rightarrow Betty\ is\ up. & \wedge E \\
Betty\ is\ up. & \Rightarrow E \\
Betty\ is\ up. \Rightarrow \neg Betty\ is\ down. & \wedge E \\
\neg Betty\ is\ down. & \Rightarrow E
\end{array}
$$

Next, we will show the proof of Axiom (A1).

$$
\begin{array}{l}
\begin{array}{|ll}
A & Hyp \\ \hline
\\
\begin{array}{|ll}
B & Hyp \\ \hline
A & Reit
\end{array} & \\
B \Rightarrow A & \Rightarrow I
\end{array} \\
A \Rightarrow (B \Rightarrow A) \qquad \Rightarrow I
\end{array}
$$

For a final example, we will prove $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$.

$$
\begin{array}{ll}
\mid \underline{\neg(A \wedge B)} & Hyp \\
\mid \quad \mid \underline{\neg(\neg A \vee \neg B)} & Hyp \\
\mid \quad \mid \quad \mid \underline{A} & Hyp \\
\mid \quad \mid \quad \mid \quad \mid \underline{B} & Hyp \\
\mid \quad \mid \quad \mid \quad \mid A & Reit \\
\mid \quad \mid \quad \mid \quad \mid B & Rep \\
\mid \quad \mid \quad \mid \quad \mid A \wedge B & \wedge I \\
\mid \quad \mid \quad \mid \quad \mid \neg(A \wedge B) & Reit \\
\mid \quad \mid \quad \mid \neg B & \neg I \\
\mid \quad \mid \quad \mid \neg A \vee \neg B & \vee I \\
\mid \quad \mid \quad \mid \neg(\neg A \vee \neg B) & Reit \\
\mid \quad \mid \neg A & \neg I \\
\mid \quad \mid \neg A \vee \neg B & \vee I \\
\mid \quad \mid \neg(\neg A \vee \neg B) & Rep \\
\mid \neg\neg(\neg A \vee \neg B) & \neg I \\
\mid (\neg A \vee \neg B) & \neg E \\
 & \\
\mid \underline{(\neg A \vee \neg B)} & Hyp \\
\mid \quad \mid \underline{A \wedge B} & Hyp \\
\mid \quad \mid A & \wedge E \\
\mid \quad \mid B & \wedge E \\
\mid \quad \mid \neg A \vee \neg B & Reit \\
\mid \quad \mid \quad \mid \underline{\neg A \quad Hyp} & \\
\mid \quad \mid \quad \mid \neg A \quad Rep & \\
\mid \quad \mid \quad \mid \underline{\neg B} & Hyp \\
\mid \quad \mid \quad \mid \quad \mid \underline{A \quad Hyp} & \\
\mid \quad \mid \quad \mid \quad \mid B \quad Reit & \\
\mid \quad \mid \quad \mid \quad \mid \neg B \quad Reit & \\
\mid \quad \mid \quad \mid \neg A & \neg I \\
\mid \quad \mid \neg A & \vee E \\
\mid \neg(A \wedge B) & \neg I \\
 & \\
\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B) & \Leftrightarrow I
\end{array}
$$

Exercise Prove $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$.
Exercise A complete set of domain axioms for Seesaw World is:

1. *John is up.* $\Leftrightarrow \neg$ *John is down.*
2. *John is down.* $\Leftrightarrow$ *Betty is up.*
3. *Betty is up.* $\Leftrightarrow \neg$ *Betty is down.*

Show that from these and the additional domain axiom *John is up.* it follows that *Betty is down.*

2.2.4 Semantic Inference

The semantic inference methods allow one to identify the logically valid sentences of the logic. We say "identify" rather than "determine," because the logically valid sentences are determined by the semantics of the logic. However, it is not always easy to decide which sentences of the logic are logically valid. Semantic inference methods can be used to make such decisions.

A WFP P is logically valid (We write this as $\models P$ and pronounce $\models$ "double turnstyle.") if and only if it is *True* under all interpretations. A set of WFPs $A_1, \ldots, A_n$ *logically implies* a WFP P (written $A_1, \ldots, A_n \models P$) if and only if every interpretation that makes $A_1, \ldots, A_n \models P$ *True* also makes P *True.* If there is no model for which P is *True,* P is said to be *contradictory.* If P is *True* in some models and *False* in some others, than P is said to be *satisfyable.* If P is logically valid, we also say that P is a *tautology.*

Recall that every interpretation assigns to each atomic proposition either the value *True* or the value *False.* Thus, we can examine all the interpretations of a non-atomic WFP that is formed from n atomic propositions, by setting up a truth table with one line for each of the 2^n possible truth value assignments. Let's do this for Axiom A1:

A	B	$B \Rightarrow A$	$A \Rightarrow (B \Rightarrow A)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

It is apparent that Axiom A1 is logically valid. In fact *every axiom of a Hilbert-style logic is logicaly valid.*

Next, let's examine the truth table for the theorem $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$:

A	B	$\neg A$	$\neg B$	$A \wedge B$	$\neg(A \wedge B)$	$\neg A \vee \neg B$	$\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$:
T	T	F	F	T	F	F	T
T	F	F	T	F	T	T	T
F	T	T	F	F	T	T	T
F	F	T	T	F	T	T	T

So this theorem is logically valid. As, indeed, is every theorem of the propositional calculi we have been describing.

Finally, let us construct a truth table to see if

(John is down.), (John is down. $\Rightarrow$ *Betty is up.), (Betty is up.* $\Rightarrow \neg$ *Betty is down.)* $\models \neg$ *(Betty is down.)*

First, we will be complete, and construct all 16 rows of Seesaw World, but we'll abbreviate to save space:

JUp	$JDown$	BUp	$BDown$	$(JDown \Rightarrow BUp)$	$(BUp \Rightarrow \neg BDown)$	$\neg BDown$
T	T	T	T	T	F	F
T	T	T	F	T	T	T
T	T	F	T	F	T	F
T	T	F	F	F	T	T
T	F	T	T	T	F	F
T	F	T	F	T	T	T
T	F	F	T	T	T	F
T	F	F	F	T	T	T
F	T	T	T	T	F	F
F	T	T	F	T	T	T
F	T	F	T	F	T	F
F	T	F	F	F	T	T
F	F	T	T	T	F	F
F	F	T	F	T	T	T
F	F	F	T	T	T	F
F	F	F	F	T	T	T

Now, since

(John is down.), (John is down. $\Rightarrow$ *Betty is up.), (Betty is up.* $\Rightarrow \neg$ *Betty is down.)* $\models \neg$ *(Betty is down.)*

if and only if every interpretation that makes the antecedents *True* also makes the consequent *True,* we can look at only those lines for which the three antecedent WFPs are *True:*

JUp	$JDown$	BUp	$BDown$	$(JDown \Rightarrow BUp)$	$(BUp \Rightarrow \neg BDown)$	$\neg BDown$
T	T	T	F	T	T	T
F	T	T	F	T	T	T

So we see that the argument is valid.

Three important properties of logical systems are *soundness, consistency* and *completeness.* A logical system is *sound* if, for every WFP P, $\vdash P$ implies $\models P$. That is, every WFP that is provable is valid. A logical system is *consistent* if, for every WFP P, it is not the case that both $\vdash P$ and $\vdash \neg P$. That is, there is no WFP such that both it and its negation are provable. If at most one of P and $\neg P$ can be valid, soundness implies consistency. A logical system is *complete* if $\models P$ implies $\vdash P$. That is, every valid WFP is provable. It is the essence of what we mean by correct reasoning that any logical system we use be sound. Completeness, however, is less important because it says nothing about how long a proof will take. We might give up on a proof that is taking too long, and then we will not know if the WFP we were trying to prove is a theorem or not. Nevertheless, the Propositional Calculi we have been describing are complete as well as sound. (The Gödel Incompletness Theorem says that any formal system that is strong enough to represent arithmetic is either inconsistent or incomplete. However, propositional calculus is not that strong.)

Wang's Algorithm Since the standard propositional calculus is sound, we can use a semantic method for proving theorems:

1. Try to construct a line of the truth table in which the WFP being tested is *False.*
2. If you succeed, the WFP is not a theorem.
3. If you cannot construct such a line, the WFP must be valid, and therefore, must be a theorem.

An organized (and programmable) way to do this is Wang's Algorithm[2]. Wang's Algorithm works on two lists of WFPs, a list of *True* WFPS, and a list of *False* WFPs. To see if some WFP P is a theorem, start with an empty list of *True* WFPs and P as the only WFP in the list of *False* WFPs. To see if $A_1, \ldots, A_n$ logically imply P, start with $A_1, \ldots, A_n$ as the list of *True* WFPs, and P as the only *False* WFP. Then keep doing the following steps until one of the termination conditions becomes true:

1. If a WFP is in both the *True* list and the *False* list, terminate and return T: a falsifying line cannot be found in this way.
2. If both lists contain only Atomic WFPs, terminate and return F: the two lists specify a falsifying line of the truth table.
3. Remove a non-atomic WFP from either list, and call the algorithm recursively with the top-level operands of the WFP in the appropriate list(s). This may need to be done two ways, in which case, return T if both ways return T, and return F if either way returns F.

The details of step (3) should be as easy to understand in the following COMMON LISP code as they would be in English.

[2] Wang, H. 1960. Toward Mechanical Mathematics. IBM Journal of Research and Development 4, 2-22. Reprinted in K. M. Sayre and F. J. Crosson (Eds.) The Modeling of Mind: Computers and Intelligence. Simon and Schuster, New York, 1963.

```
(defun wang (TrueWffs FalseWffs)
  "Returns True if no way to make consistent truth value assignments."
  (if (intersection TrueWffs FalseWffs :test #'equal) t
    (let (wff)
      (cond ((setf wff (find-if-not #'atomic TrueWffs))
             (cond ((negation wff)
                    (wang (remove wff TrueWffs) (addwff (second wff) FalseWffs)))
                   ((conjunction wff)
                    (wang (addwff (second wff) (addwff (third wff) (remove wff TrueWffs)))
                          FalseWffs))
                   ((disjunction wff)
                    (and (wang (addwff (second wff) (remove wff TrueWffs)) FalseWffs)
                         (wang (addwff (third wff) (remove wff TrueWffs)) FalseWffs)))
                   ((implication wff)
                    (and (wang (addwff (third wff) (remove wff TrueWffs)) FalseWffs)
                         (wang (remove wff TrueWffs) (addwff (second wff) FalseWffs))))
                   ((biconditional wff)
                    (and (wang (addwff (second wff) (addwff (third wff) (remove wff TrueWffs)))
                               FalseWffs)
                         (wang (remove wff TrueWffs)
                               (addwff (second wff)
                                       (addwff (third wff) (remove wff FalseWffs))))))))
            ((setf wff (find-if-not #'atomic FalseWffs))
             (cond ((negation wff)
                    (wang (addwff (second wff) TrueWffs) (remove wff FalseWffs)))
                   ((conjunction wff)
                    (and (wang Truewffs (addwff (second wff) (remove wff Falsewffs)))
                         (wang Truewffs (addwff (third wff) (remove wff Falsewffs)))))
                   ((disjunction wff)
                    (wang TrueWffs
                          (addwff (second wff) (addwff (third wff) (remove wff FalseWffs)))))
                   ((implication wff)
                    (wang (addwff (second wff) TrueWffs)
                          (addwff (third wff) (remove wff FalseWffs))))
                   ((biconditional wff)
                    (and (wang (addwff (second wff) TrueWffs)
                               (addwff (third wff) (remove wff FalseWffs)))
                         (wang (addwff (third wff) TrueWffs)
                               (addwff (second wff) (remove wff FalseWffs)))))))
            (t nil)))))
```

Let's see some uses of this program. First, let's try the non-theorem $\neg(A \vee B) \Leftrightarrow (\neg A \vee \neg B)$.

```
> (wang '() '((<=> (- (or A B)) (or (- A) (- B)))))
 0: (WANG NIL ((<=> (- (OR A B)) (OR (- A) (- B)))))
   1: (WANG ((- (OR A B))) ((OR (- A) (- B))))
     2: (WANG NIL ((OR A B) (OR (- A) (- B))))
       3: (WANG NIL (A B (OR (- A) (- B))))
         4: (WANG NIL ((- A) (- B) A B))
           5: (WANG (A) ((- B) A B))
           5: returned T
         4: returned T
       3: returned T
     2: returned T
   1: returned T
   1: (WANG ((OR (- A) (- B))) ((- (OR A B))))
     2: (WANG ((- A)) ((- (OR A B))))
       3: (WANG NIL (A (- (OR A B))))
         4: (WANG ((OR A B)) (A))
           5: (WANG (A) (A))
           5: returned T
           5: (WANG (B) (A))
           5: returned NIL
         4: returned NIL
       3: returned NIL
     2: returned NIL
   1: returned NIL
 0: returned NIL
NIL
```

Here, we are using `-` for negation, `or` for disjunction, and `<=>` for biconditional. `NIL` is Common Lisp's way of saying *False*. We see that assigning B the value *True* and A the value *False* will give a value of *False* to the original WFP.

Now, let's try the axiom $A => (B => A)$

```
> (wang '() '((=> A (=> B A))))
 0: (WANG NIL ((=> A (=> B A))))
   1: (WANG (A) ((=> B A)))
     2: (WANG (B A) (A))
     2: returned T
   1: returned T
 0: returned T
T
```

Here, we are using `=>` for implication, and we see that the WFP is valid.

Finally, let's try the Seesaw World argument for which we constructed the truth table above.

```
(wang '((John is down)
        (=> (John is down) (Betty is up))
        (=> (Betty is up)(- (Betty is down))))
      '((- (Betty is down))))
 0: (WANG ((JOHN IS DOWN) (=> (JOHN IS DOWN) (BETTY IS UP))
           (=> (BETTY IS UP) (- (BETTY IS DOWN))))
          ((- (BETTY IS DOWN))))
   1: (WANG ((BETTY IS UP) (JOHN IS DOWN)
             (=> (BETTY IS UP) (- (BETTY IS DOWN))))
            ((- (BETTY IS DOWN))))
     2: (WANG ((- (BETTY IS DOWN)) (BETTY IS UP) (JOHN IS DOWN))
              ((- (BETTY IS DOWN))))
     2: returned T
     2: (WANG ((BETTY IS UP) (JOHN IS DOWN))
              ((BETTY IS UP) (- (BETTY IS DOWN))))
     2: returned T
   1: returned T
   1: (WANG ((JOHN IS DOWN) (=> (BETTY IS UP) (- (BETTY IS DOWN))))
            ((JOHN IS DOWN) (- (BETTY IS DOWN))))
   1: returned T
 0: returned T
T
```

So Wang's Algorithm confirms that this is a valid argument.

Notice that the program of Wang's Algorithm constitutes an AI reasoning system in which one can specify a "Knowledge Base" of domain rules, and ask the system if some proposition is true in the situation described by that Knowledge Base.

2.3 Clause Form Propositional Calculus

2.3.1 Syntax

Atomic Propositions: An atomic sentence is any letter of the alphabet, or any letter with a numeric subscript, or any alphanumeric string. For example, P, Q_3 and the four propositions of Seesaw World are atomic propositions.

Literals: If P is an atomic proposition, P and $\neg P$ are literals. P is called a *positive* literal, and $\neg P$ is called a *negative* literal.

Clauses: If $L_1, \ldots, L_n$ are literals, then the set $\{L_1, \ldots, L_n\}$ is a clause.

Sets of Clauses: If $C_1, \ldots, C_n$ are clauses, then the set $\{C_1, \ldots, C_n\}$ is a set of clauses.

2.3.2 Semantics

As in the standard propositional calculus, the semantics of clause form propositional logic is given intensionally and extensionally, within the following constraints.

Atomic Propositions: Intensionally, the meaning of an atomic proposition is some proposition in the domain. Extensionally, it is one of the truth values *True* or *False*.

Literals:

1. If L is a literal and also an atomic proposition, then its meaning as a literal is the same as it is as an atomic proposition.

2. If $\neg A$ is a literal, then it is *True* if A is *False,* and *False* if A is *True.*

Clauses: If $\{L_1, \ldots, L_n\}$ is a clause, it is *True* if any one of $L_1, \ldots, L_n$ is *True;* otherwise, it is *False.*

Sets of Clauses: If $\{C_1, \ldots, C_n\}$ is a set of clauses, it is *True* if every one of $C_1, \ldots, C_n$ is *True;* otherwise, it is *False.*

2.3.3 Syntactic Inference: Resolution

A set of clauses in clause form logic corresponds to a proof in the standard logics. There is one Rule of Inference, called *resolution,* that specifies when a new clause can be added to a set of clauses. In particular, if a set of clauses contains a clause of the form $\{A, P_1, \ldots, P_n\}$, where A is a positive literal and $P_1, \ldots, P_n$ are any literals, and a clause of the form $\{\neg A, Q_1, \ldots, Q_m\}$, where $Q_1, \ldots, Q_m$ are any literals, then one may add to the set of clauses the clause $\{P_1, \ldots, P_n, Q_1, \ldots, Q_m\}$. Symbolically:

$$\frac{\{A, P_1, \ldots, P_n\}, \{\neg A, Q_1, \ldots, Q_m\}}{\{P_1, \ldots, P_n, Q_1, \ldots, Q_m\}}$$

Resolution is a sound rule of inference, but it is not complete. For example, even though $\{\{P\}\}$ logically implies $\{\{P\}, \{P, Q\}\}$, resolution will not produce the latter set of clauses from the former.

2.3.4 Semantic Inference: Resolution Refutation

Although resolution is not a complete rule of inference for syntactic inference, it is complete for semantic inference in clause form propositional calculus, if used in the correct way. If a set of clauses is logically contradictory, that is, there is no consistent assignment of truth values to the atomic propositions that makes the set of clauses *True,* then repeated application of resolution is guaranteed to eventually produce $\{\}$, the *empty clause.* So, to see if a clause C is logically implied by a set of clauses $S = \{C_1, \ldots, C_n\}$, take the *negation* of C, which will be a set of clauses, union that set with S, and apply resolution until either you can't get anything new or you generate the empty clause. In the latter case, you have shown that no interpretation that satisfies S also satisfies $\neg C$, so every interpretation that satisfies S also satisfies C, and so S logically implies C. This is called a *refutation* proof.

The negation of a clause $\{L_1, \ldots, L_n\}$ is the set of clauses $\{\{\neg L_1\}, \ldots \{\neg L_n\}\}$, where $\neg L$ is just $\neg L$ if L is a positive literal, but is P if L is the negative literal $\neg P$.

For example, to show that $\{\{P\}\}$ logically implies $\{P, Q\}$, apply resolution repeatedly to the set of clauses $\{\{P\}, \{\neg P\}, \{\neg Q\}\}$. We will do this by showing the set of clauses as a numbered list of clauses, and using resolution to add clauses to the list:

1.	$\{P\}$	Assumption
2.	$\{\neg P\}$	Assumption
3.	$\{\neg Q\}$	Assumption
4.	$\{\}$	by Resolution from (1) and (2)

A complete set of clause form domain axioms for Seesaw World are:

$\{\{JUp, JDown\}, \{\neg JUp, \neg JDown\}, \{BUp, BDown\}, \{\neg BUp, \neg BDown\}, \{JUp, BUp\}, \{JDown, BDown\}\}$

Let's show that this set of clauses, plus $\{JUp\}$ logically implies $\{BDown\}$:

1.	$\{\{JUp, JDown\}$	Assumption
2.	$\{\neg JUp, \neg JDown\}$	Assumption
3.	$\{BUp, BDown\}$	Assumption
4.	$\{\neg BUp, \neg BDown\}$	Assumption
5.	$\{JUp, BUp\}$	Assumption
6.	$\{JDown, BDown\}$	Assumption
7.	$\{JUp\}$	Assumption
8.	$\{\neg BDown\}$	Negation of clause in question
9.	$\{JDown\}$	by Resolution from (6) and (8)
10.	$\{\neg JUp\}$	by Resolution from (2) and (9)
11.	$\{\}$	by Resolution from (7) and (10)

It is relatively easy to implement resolution refutation theorem proving. In the program to be demonstrated below, atomic propositions are represented by LISP symbols, negative literals such as $\neg P$ are represented by lists of the form `(- P)`, and clauses such as $\{P, Q, R\}$ are represented by lists of the form `(P or (Q or R))`. The main function, `prove` takes two arguments: a list of assumption clauses, and a single "theorem" clause. The function `prove` tries to show that the theorem is logically implied by the assumptions. We will show this function operating on the two proofs shown above.

```
> (prove '(P) '(P or Q))

 1  (P)             Assumption
 2  ((- P))         From Theorem
 3  ((- Q))         From Theorem
 4  NIL             R  2  1
QED

> (prove '(
           (JUp or  JDown)
           ((- JUp) or (- JDown))
           (BUp or BDown)
           ((- BUp) or (- BDown))
           (JUp or BUp)
           (JDown or BDown)
           JUp
           )
         'BDown)

 1  (JUP)           Assumption
 2  (JUP JDOWN)     Assumption
 3  ((- JUP) (- JDOWN)) Assumption
 4  (BUP BDOWN)     Assumption
 5  ((- BUP) (- BDOWN)) Assumption
 6  (JUP BUP)       Assumption
 7  (JDOWN BDOWN)   Assumption
 8  ((- BDOWN))     From Theorem
 9  (BUP)           R  8  4
10  (JDOWN)         R  8  7
11  ((- JUP))       R 10  3
12  ((- BDOWN))     R  9  5
13  NIL             R 11  1
QED
```

2.4 SNePS Propositional Calculus

3 Predicate Calculus

Although the Propositional Calculi are the basic logical systems, they are not expressive enough because they ignore the information internal to sentences. We will motivate the move to the Predicate Calculi by looking again at Seesaw World.

3.1 Seesaw World

Recall, the four propositions relevant to Seesaw World are:

1. *John is down.*

2. *John is up.*
3. *Betty is down.*
4. *Betty is up.*

In propositional calculus, these four propositions have nothing to do with each other. (Except as specified by the domain axioms.) Propositional calculus ignores the fact that these four propositions share two people (John and Betty) and two properties (up and down). Predicate calculus can recognize this. As we will see, in predicate calculus, we can express these four propositions as:

1. *Down(John)*
2. *Up(John)*
3. *Down(Betty)*
4. *Up(Betty)*

Making the two properties and the two individuals clearer. Also, instead of specifying the domain axioms as:

1. *John is up.* $\Leftrightarrow \neg$ *John is down.*
2. *John is down.* $\Leftrightarrow$ *Betty is up.*
3. *Betty is up.* $\Leftrightarrow \neg$ *Betty is down.*

we can just say:

1. $\forall x[Up(x) \Leftrightarrow \neg Down(x)]$
2. $\forall x, y[Up(x) \Leftrightarrow Down(y)]$

Unfortunately, in the standard predicate calculus, domain axiom (2) implies $[Up(John) \Leftrightarrow Down(John)]$, since both variables can be replaced by the same individual. So in the standard predicate calculus, we'd have to make the domain axioms:

1. $\forall x[Up(x) \Leftrightarrow \neg Down(x)]$
2. $\forall x, y[(x \neq y) \Rightarrow (Up(x) \Leftrightarrow Down(y))]$
3. $John \neq Betty$
4. $Betty \neq John$

Next, we will go through a more careful description of the standard Predicate Calculus.

3.2 The "Standard" Predicate Calculus

3.2.1 Syntax

Individual Constants: An individual constant is any letter of the alphabet (preferably early in the alphabet), or any (such) letter with a numeric subscript, or any character string not containing blanks nor other punctuation marks. For example, a, B_{12}, *John*, *John's_mother-in-law.*

Variables: A variable is any letter of the alphabet (preferably late in the alphabet), or any (such) letter with a numeric subscript. For example, u, v_6.

Skolem Constants: A Skolem constant is any letter of the alphabet (preferably early in the alphabet), or any (such) letter with a numeric subscript.

Function Symbols: A function symbol is any letter of the alphabet (preferably in the early middle of the alphabet), or any (such) letter with a numeric subscript, or any character string not containing blanks nor other punctuation marks. Each function symbol must have a particular *arity.* When we want to make the arity explicit, we will write it as a superscript on the function symbol. For example, $f, g_2, \mathit{mother\text{-}of}^1$.

Skolem Function Symbols: A Skolem function symbol is any letter of the alphabet (preferably in the early middle of the alphabet), or any (such) letter with a numeric subscript. Each Skolem function symbol must have a particular arity. When we want to make the arity explicit, we will write it as a superscript on the Skolem function symbol.

Predicate Symbols: A predicate symbol is any letter of the alphabet (preferably in the late middle of the alphabet), or any (such) letter with a numeric subscript, or any character string not containing blanks nor other punctuation marks. Each predicate symbol must have a particular arity. When we want to make the arity explicit, we will write it as a superscript on the function symbol. For example, $P, Q_4, \mathit{odd}, \mathit{Up}^1$.

- In any specific predicate calculus language the sets of individual constants, variables, Skolem constants, function symbols, Skolem function symbols, and predicate symbols must each be disjoint from all the others.

Terms:

1. Every individual constant, every Skolem constant, and every variable is a term.
2. If f^n is a function symbol or Skolem function symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $f^n(t_1, \ldots, t_n)$ is a term. (The superscript may be omitted if no confusion results.)
3. Nothing else is a term.

Atomic Formulae: If P^n is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $P^n(t_1, \ldots, t_n)$ is an atomic formula. (The superscript may be omitted if no confusion results.)

Well-Formed Formulae (wffs):

1. Every atomic formula is a wff.
2. If P is a wff, then so is $\neg P$.
3. If P and Q are wffs, then so is
   (a) $P \wedge Q$
   (b) $P \vee Q$
   (c) $P \Rightarrow Q$
   (d) $P \Leftrightarrow Q$
4. If P is a wff and x is a variable, then $\forall x P$ and $\exists x P$ are wffs. $\forall$ and $\exists$ are called the *universal quantifier* and the *existential quantifier,* respectively. P is called the *scope* of quantification. Every occurrence of x in P, not in the scope of some other occurrence of $\forall x$ or $\exists x$, is said to be *free* in P and *bound* in $\forall x P$ and $\exists x P$. Every occurrence of every variable other than x that is free in P is also free in $\forall x P$ and $\exists x P$. A wff with no free variables is called *closed,* otherwise it is called *open.*
5. Nothing else is a wff.

Parentheses (either round, square, or curly) may be used to reduce syntactic confusion or omitted if that doesn't cause confusion. Any term or wff that doesn't contain any variable is called *ground.*

3.2.2 Semantics

Any particular predicate calculus language, formulated to formalize or represent a particular domain, or set of situations, must be formulated with a particular, finite set of individual constants, function symbols, and predicate symbols, and arbitrarily large sets of variables, Skolem constants, and Skolem function symbols. The individual constants, function symbols, and predicate symbols should be given informal, intensional semantics to make it clear what they are supposed to represent in any situation within the domain, and more formal extensional semantics, that should not vary across situations for individual constants, but may vary across situations for function symbols and predicate symbols. In the version of predicate calculus presented here, variables will not have semantics, and Skolem constants and Skolem function symbols will get semantics as they are used, as will be explained below.

First, we will sketch the semantics for the class of standard predicate calculi. Then, we will specify the semantics for a predicate calculus formalization of Seesaw World.

Predicate calculus semantics assumes a *domain* of individuals, functions on individuals, sets of individuals, and relations on individuals. In defining the semantics, we will say that some syntactic expression *denotes* some individual, set, or relation, and if e is a syntactic expression, $[\![e]\!]$ will be what e denotes.

Individual Constants: If a is an individual constant, then $[\![a]\!]$ is some particular individual in the domain.

Skolem Constants: If sc is a Skolem constant, created for the Skolemization of the wfp A, then $[\![sc]\!]$ is the individual described by A. (See below.)

Function Symbols: If f^n is a function symbol of arity n, then $[\![f^n]\!]$ is some particular function from the cartesian product of n sets of individuals into the set of individuals.

Skolem Function Symbols: If sf^n is a Skolem function symbol of arity n, created for the Skolemization of the wfp A, then $[\![sf^n]\!]$ is the n-ary function described by A. (See below.)

Predicate Symbols:

1. If P is some unary predicate symbol, then $[\![P]\!]$ is some particular set of individuals.
2. If P^n is some n-ary predicate symbol, then $[\![P^n]\!]$ is an n-ary relation on n sets of individuals.

Functional Terms: If f^n is some function symbol or Skolem function symbol, and $t_1, \ldots, t_n$ are ground terms, then $[\![f^n(t_1, \ldots, t_n)]\!]$ is $[\![f^n]\!]([\![t_1,]\!] \ldots, [\![t_n]\!])$.

Atomic Formulae:

1. If P is some unary predicate symbol, and t is some ground term, then $[\![P(t)]\!]$ is *True* if $[\![t]\!] \in [\![P]\!]$, and *False* otherwise.
2. If P^n is some n-ary predicate symbol, and $t_1, \ldots, t_n$ are ground terms, then $[\![P^n(t_1, \ldots, t_n)]\!]$ is *True* if $\langle [\![t_1]\!], \ldots, [\![t_n]\!] \rangle \in [\![P^n]\!]$, and *False* otherwise.

Well-Formed Formulae:

1. $[\![\neg P]\!]$ is *True* if $[\![P]\!]$ is *False.* Otherwise, it is *False.*
2. $[\![P \wedge Q]\!]$ is *True* if $[\![P]\!]$ is *True* and $[\![Q]\!]$ is *True.* Otherwise, it is *False.*
3. $[\![P \vee Q]\!]$ is *True* if $[\![P]\!]$ is *True* or $[\![Q]\!]$ is *True.* Otherwise, it is *False.*
4. $[\![P \Rightarrow Q]\!]$ is *True* if $[\![P]\!]$ is *False* or $[\![Q]\!]$ is *True.* Otherwise, it is *False.*
5. $[\![P \Leftrightarrow Q]\!]$ is *True* if $[\![P]\!]$ and $[\![Q]\!]$ are both *True,* or both *False.* Otherwise, it is *False.*
6. $[\![\forall x P]\!]$ is *True* if $[\![P\{t/x\}]\!]$ is *True* for every ground term, t, where $P\{t/x\}$ is P with every free occurrence of x replaced by t. Otherwise, it is *False.*
7. $[\![\exists x P]\!]$ is *True* if there is some ground term, t such that $[\![P\{t/x\}]\!]$ is *True.* Otherwise, it is *False.*

Example—Seesaw World:

Intensional Semantics:

Individual Constants:

John: The individual person named John.

Betty: The individual person named Betty.

Predicate Symbols:

$Up^1(x)$: The individual $[\![x]\!]$ is in the higher position on his/her end of the seesaw.

$Down^1(x)$: The individual $[\![x]\!]$ is in the lower position on his/her end of the seesaw.

$x \neq y$: $[\![x]\!]$ and $[\![y]\!]$ are different individuals.

Extensional Semantics: We will give the extensional semantics for the particular situation pictured in Figure 1.

Individual Constants:

$[\![John]\!]$ is the individual person named John.

$[\![Betty]\!]$ is the individual person named Betty.

Predicate Symbols:

$[\![Up]\!]$ is {Betty}.

$[\![Down]\!]$ is {John}.

$[\![\neq]\!]$ is $\{\langle$John, Betty $\rangle, \langle$ Betty, John$\rangle\}$.

3.2.3 Syntactic Inference

Hilbert-style Inference The Hilbert-style inference system is just like that for propositional calculus, except there are two more axioms[3]:

(A4). $\forall x A \Rightarrow A\{t/x\}$, where t is any term not containing a variable any of whose occurrences gets bound as a result of the substitution.

(A5). $(\forall x(A \Rightarrow B)) \Rightarrow (A \Rightarrow \forall x B)$ if A is a wff containing no free occurrences of x.

and one more Rule of Inference[4]:

$$\frac{A}{\forall x A}$$

Here, $\exists x A$ is just an abbreviation of $\neg\forall x \neg A$.

We will not bother showing any proofs in Hilbert-style predicate calculus.

Fitch-style Natural Deduction The Fitch-style natural deduction system for predicate calculus is just like that for propositional calculus, except it has four more Rules of Inference[5]:

$$\begin{array}{|l} \begin{array}{|l} \underline{sc \qquad SkolemI} \\ \ldots A \end{array} \\ \forall x A\{x/sc\} \qquad\qquad \forall I \end{array} \qquad\qquad \begin{array}{|l} \forall x A \\ A\{t/x\} \quad \forall E \end{array}$$

[3]Mendelson, *op. cit.,* p. 57.

[4]*Ibid.*

[5]Adapted from D. Kalish and R. Montague, *Logic: Techniques of Formal Reasoning,* (New York: Harcourt, Brace & World), 1964, pp. 99–101.

$$\begin{array}{|ll}
A(t) & \\
\exists x A(x) & \exists I
\end{array}
\qquad\qquad
\begin{array}{|ll}
\exists x A & \\
A\{sc/x\} & \exists E
\end{array}$$

where, sc is a Skolem constant never used before in the proof; in $A\{t/x\}, t$ contains no variable any of whose occurrences get bound as a result of the substitution; $A(t)$ is a wff containing t as a term, $A(x)$ is $A(t)\{x/t\}$ and no occurrence of x in this substitution is bound in $A(t)\{x/t\}$. In $\forall I$, $[\![sc]\!]$ is an arbitrary individual about which nothing is known except what can be proved in the subproof introduced by $SkolemI$. Therefore, anything True about $[\![sc]\!]$ is true about all individuals. In $\exists E$, $[\![sc]\!]$ is the individual claimed to exist by the wff $\exists x A$.

Example We will prove $\neg\forall x A(x) \Leftrightarrow \exists x \neg A(x)$.

$$\begin{array}{ll}
\mid \neg\forall x A(x) & Hyp \\
\hline
\mid \quad \mid \neg\exists x\neg A(x) & Hyp \\
\mid \quad \mid \quad \mid s_1 & SkolemI \\
\mid \quad \mid \quad \mid \quad \mid \neg A(s_1) & Hyp \\
\mid \quad \mid \quad \mid \quad \mid \exists x\neg A(x) & \exists I \\
\mid \quad \mid \quad \mid \quad \mid \neg\exists x\neg A(x) & Reit \\
\mid \quad \mid \quad \mid \neg\neg A(s_1) & \neg I \\
\mid \quad \mid \quad \mid A(s_1) & \neg E \\
\mid \quad \mid \forall x A(x) & \forall I \\
\mid \quad \mid \neg\forall x A(x) & Reit \\
\mid \neg\neg\exists x\neg A(x) & \neg I \\
\mid \exists x\neg A(x) & \neg E
\end{array}$$

$$\begin{array}{ll}
\mid \exists x\neg A(x) & Hyp \\
\hline
\mid \quad \mid \forall x A(x) & Hyp \\
\mid \quad \mid \exists x\neg A(x) & Reit \\
\mid \quad \mid \neg A(s_2) & \exists E \\
\mid \quad \mid A(s_2) & \forall E \\
\mid \neg\forall x A(x) & \neg I \\
\neg\forall x A(x) \Leftrightarrow \exists x\neg A(x) & \Leftrightarrow I
\end{array}$$

3.2.4 Semantic Inference

3.3 Clause Form Predicate Calculus

3.3.1 Syntax

3.3.2 Semantics

3.3.3 Inference

3.4 Substitutions

A *substitution* is a set of pairs $\{t_1/v_1, \ldots, t_n/v_n\}$, where each t_i is a term, each v_i is a variable, and $\forall i, j(v_i \neq v_j)$. The v_i are called the variables of the substitution, and the t_i are called the terms of the substitution. If some term in the substitution happens to be a variable, it is still not called a variable of the substitution unless it is also in variable position in some pair.

In Lisp it is common to represent a substitution as an association list (list of dotted pairs) with the order of the pairs reversed: `((`v_1 `.` t_1`)` ... `(`v_n `.` t_n`))`.

If A is a wff and σ is a substitution, $A\sigma$ is the result of *applying* the substitution σ to the wff A. This is done by replacing each variable in A that is a variable of σ with the term that variable is paired with in σ.

The Common Lisp function `sublis` applies a substitution to an expression. Some examples are

```
> (sublis '((x . a) (y . b)) '(P (f x) y))
(P (F A) B)

> (sublis '((x . y) (y . x)) '(P x y))
(P Y X)
```

We need an operation on substitutions, $\sigma \circ \tau$, called the *composition* of σ with *tau* such that, for any formula, $A, A\sigma\tau = (A\sigma)\tau = A(\sigma \circ \tau)$. Composition is performed in the following way:

1. Replace every term t_i in σ by $t_i\tau$.
2. Add into σ every pair of τ whose variable is not already a variable of σ.

In Common Lisp this can be expressed as

```
(defun compose (sigma tau)
  "Returns the composition of the substitution sigma with the
   substitution tau."
  (append (mapcar #'(lambda (pair)
                      (cons (first pair) (sublis tau (rest pair))))
                  sigma)
          (remove-if #'(lambda (pair)
                         (member (first pair) sigma :key #'first))
                     tau)))
```

3.5 Unification

```
(defun unify (s1 s2 &optional b)
  "If s1b and s2b are unifiable (where sib means the result of
      applying the substitution b to the S-expression si)
      with most general unifier c, returns bc.
      I.e., in that case, returns the most general unifier of s1 and s2.
   Otherwise, returns FAIL.
   Assumes that no variable of b occurs in a term of b."
  (cond ((eql b 'FAIL) 'FAIL)
        ((eql s1 s2) b)
        ((variablep s1) (unifyVar s1 s2 b))
        ((variablep s2) (unifyVar s2 s1 b))
        ((or (atom s1) (atom s2)) 'FAIL)
        ((/= (length s1) (length s2)) 'FAIL)
        (t (unify (rest s1)
                  (rest s2)
                  (unify (first s1) (first s2) b)))))

(defun unifyVar (var term b)
  "If the variable VAR does not occur in TERM
      returns b composed with {term/var};
   Else returns FAIL."
  (if (bboundp var b) (unify (bindingOf var b) term b)
    (let ((newterm (sublis b term)))
      (cond ((eql var newterm) b)
            ((occursIn var newterm) 'FAIL)
            (t (compose b `((,var . ,newterm)))))))
```

3.6 Translating Standard Wffs into Clause Form

Example: $\forall x[parent(x) \Leftrightarrow (person(x) \wedge \exists y(person(y) \wedge child(y,x)))]$

1. Eliminate occurrences of $\Leftrightarrow$ using the theorem $(A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$.

   Example:

   $$\forall x[(parent(x) \Rightarrow (person(x) \wedge \exists y(person(y) \wedge child(y,x)))) \\ \wedge((person(x) \wedge \exists y(person(y) \wedge child(y,x))) \Rightarrow parent(x))]$$

2. Eliminate occurrences of $\Rightarrow$ using the theorem $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$.

   Example:

   $$\forall x[(\neg parent(x) \vee (person(x) \wedge \exists y(person(y) \wedge child(y,x)))) \\ \wedge(\neg(person(x) \wedge \exists y(person(y) \wedge child(y,x))) \vee parent(x))]$$

3. Translate to *miniscope* form—all occurrences of $\neg$ governing only atomic formulae, using the theorems

   $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$

   $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$

   $\neg \forall x A \Leftrightarrow \exists x \neg A$

   $\neg \exists x A \Leftrightarrow \forall x \neg A$

   Example:

   $$\forall x[(\neg parent(x) \vee (person(x) \wedge \exists y(person(y) \wedge child(y,x)))) \\ \wedge((\neg person(x) \vee \forall y(\neg person(y) \vee \neg child(y,x))) \vee parent(x))]$$

4. Rename apart: If any two quantifiers bind the same variable, rename all occurrences of one of them.

   Example:

   $$\forall x[(\neg parent(x) \vee (person(x) \wedge \exists y(person(y) \wedge child(y,x)))) \\ \wedge((\neg person(x) \vee \forall z(\neg person(z) \vee \neg child(z,x))) \vee parent(x))]$$

5. Skolemize: Whenever a formula of the form $\exists y A(y)$ is in the scope of universal quantifiers $\forall x_1 \ldots \forall x_n$, delete the "$\exists y$" and replace all other occurrences of y by the functional term $f(x_1,\ldots,x_n)$, where f is a new Skolem function not used anywhere else in the set of clauses being constructed.

   Example:

   $$\forall x[(\neg parent(x) \vee (person(x) \wedge (person(f(x)) \wedge child(f(x),x)))) \\ \wedge((\neg person(x) \vee \forall z(\neg person(z) \vee \neg child(z,x))) \vee parent(x))]$$

6. Discard all occurrences of "$\forall x$" for any variable x.

   Example:

   $$[(\neg parent(x) \vee (person(x) \wedge (person(f(x)) \wedge child(f(x),x)))) \\ \wedge((\neg person(x) \vee (\neg person(z) \vee \neg child(z,x))) \vee parent(x))]$$

7. CNF: Translate the wff into Conjunctive Normal Form (CNF), using the theorems

   $(A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$.

   $(A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$.

   Example:

   $$[((\neg parent(x) \vee person(x)) \wedge ((\neg parent(x) \vee person(f(x))) \wedge (\neg parent(x) \vee child(f(x),x)))) \\ \wedge((\neg person(x) \vee (\neg person(z) \vee \neg child(z,x))) \vee parent(x))]$$

8. Use the associativity and commutativity of $\wedge$ and $\vee$ to discard extra parentheses. The wff should now look like $((L_{11} \vee \ldots \vee L_{1k_1}) \wedge \ldots \wedge (L_{n1} \vee \ldots \vee L_{nk_n}))$.

   Example:

   $$\begin{array}{l}[(\neg parent(x) \vee person(x)) \\ \ \wedge(\neg parent(x) \vee person(f(x))) \\ \ \wedge(\neg parent(x) \vee child(f(x), x)) \\ \ \wedge(\neg person(x) \vee \neg person(z) \vee \neg child(z, x) \vee parent(x))]\end{array}$$

9. Turn each disjunction into a clause, and the conjunction into a set of clauses, getting

   $$\{\{L_{11}, \ldots, L_{1k_1}\}, \ldots, \{L_{n1}, \ldots, L_{nk_n}\}\}.$$

   Example:

   $$\begin{array}{l}[\{\neg parent(x), person(x)\}, \\ \ \{\neg parent(x), person(f(x))\}, \\ \ \{\neg parent(x), child(f(x), x)\}, \\ \ \{\neg person(x), \neg person(z), \neg child(z, x), parent(x)\}]\end{array}$$

10. Rename the clauses apart so that no variable occurs in more than one clause. If a variable x occurs in two clauses, just substitute some other variable for all occurrences of x in one of the clauses.

    Example:

    $$\begin{array}{l}[\{\neg parent(x_1), person(x_1)\}, \\ \ \{\neg parent(x_2), person(f(x_2))\}, \\ \ \{\neg parent(x_3), child(f(x_3), x_3)\}, \\ \ \{\neg person(x_4), \neg person(z_4), \neg child(z_4, x_4), parent(x_4)\}]\end{array}$$

3.7 SNePS Predicate Calculus