Metalogic for Students

Metalogic for Students

By

Roderic A. Girle

Cambridge
Scholars
Publishing

Metalogic for Students

By Roderic A. Girle

This book first published 2024

Cambridge Scholars Publishing

Lady Stephenson Library, Newcastle upon Tyne, NE6 2PA, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-1-0364-1102-2
ISBN (Ebook): 978-1-0364-1103-9

CONTENTS

PREFACE

This is a combined text and workbook for students of Metalogic. It is a book for postgraduate and advanced undergraduates, and the main aim is to present metalogic in a maximally accessible, step by step, didactic manner.

In both chapter 1 and in chapter 9 there are some serious, but slightly skeptical, ruminations of a philosophical kind about classical first-order logic.

The Method of Presentation

The method of this text is to interact with students and readers as much as can possibly be done in a written text. For a good understanding it is important to take time to get basics established and not to try to rush through metalogic to Gödel's second theorem in three or four weeks of nine or ten lectures or seminars or a few pages of text.

This text assumes that the student or reader has completed an introductory course in first-order logic with identity, preferably but not necessarily a course which introduced the student or reader to semantic or analytic tableaux. This text is directed at enabling the average advanced or postgraduate student to understand the basics of metalogic and then build upon them.

There is a considerable amount of question-answer material in the text. It is presented in two columns. The left column contains questions with space to write in answers. The right column contains model answers. The question-answer material is directed at teaching, ***not*** at assessment. If the reader fills in the correct answer it confirms understanding. If the reader gets the answer wrong, then there is the opportunity to revise their understanding.

Any text which has problems and but no answers will make learning more difficult than it needs to be for a student, especially if they are working at home.

One telco in New Zealand has TV advertisements explaining how students can record lectures, email their lecturer or tutor with pictures of their work asking for explanations of why they are mistaken, or asking for confirmation that they have “got it right”. This text is intended to be used in that kind of interactive context.

The text contains two topics that are not new but are not often seen in metalogic texts. In particular, there is material about *pictorial semantic*s in the form of Venn and Euler diagrams and Carroll and Karnaugh maps. There is also a discussion of the *reliability* of first order logic as a means for the evaluation of premise-conclusion arguments couched in ordinary language.

ACKNOWLEDGEMENTS

I wish to acknowledge the work of my *students* in making suggestions, and pointing out omissions and typographical errors. I also acknowledge the brilliant teaching of the late Malcolm Rennie. His works, published only within philosophy departments, have guided my efforts. Any mistakes or obscurities are mine. *Mea culpa*!

CHAPTER 1

THE LANGUAGES OF PROPOSITIONAL LOGIC

1.1 Introduction

Modern formal logic comes to us in many systems. Most people first meet logic in introductory courses in either Philosophy or Computer Science. Logic courses in Philosophy used to be courses in syllogistic, the logic that came to us from Aristotle' *Prior Analytics*. But in the 1960s there was a huge change in the Anglosphere. Modern formal logic began to appear in Philosophy and gradually became dominant. But Syllogistic was not forgotten. In some courses it was the stepping off point for logic. Logic courses usually started at one of two points. Some courses began with Syllogistic (Barker 1965) and then introduced propositional logic. Others began with the translation of ordinary language into the formal notation of Propositional Logic (Lemmon 1965). Syllogistic was relegated to the side lines.

Syllogistic is formulated as regimented ordinary language: *Some frogs are green.* becomes *Some F are G.* There is a mix of ordinary language and symbolic letters.

Propositional logic requires translation from ordinary language to a formal language: *Kermit is a frog and Kermit is green.* becomes (*F* & *G*).

The letters *F* and *G* appear in both *Some F are G.* and (*F* & *G*), yet their meaning is quite different. Metalogic has the task of explaining such differences. Metalogic compares formal languages and logics. In general, metalogic is the logic of logics. Metalogic compares logics, looks at relationships between logics and proves theorems about the relationships between logics.

One difference between syllogistic and propositional logic is that syllogistic has a limited number of argument forms but propositional logic is not limited. (see *A Profile of Mathematical Logic*, Delong 1971, 14-24) The discussion of this difference is a metalogical discussion. The discussion can be more or less formal, and at times the discussion can become philosophical. In this text the main emphasis is on the formal presentation of metalogical discussion.

We begin with propositional logic. Although it might look as if the introduction of propositional logic is quite simple and straightforward, that is not so. There are usually two forms of propositional logic. One is a form in which capital letters are used for translation. The letters are *translation* letters and are given meaning in a *dictionary* such as:

F = *Kermit is a frog*. and *G* = *Kermit is green.*

The other form of propositional logic is where lower case letters such as *p* and *q* are used as *propositional variables*. *Formalising* replaces translation. Formalising is proposed as a way of exposing the *logical form* or logical structure of ordinary language sentences and arguments. This difference is of immediate interest to metalogic. What exactly is the relationship between (*F* & *G*) and (*p* & *q*)? The answer is a metalogical answer.

The topic following translation or formalising to propositional logic, however extensive or simple, is usually the introduction of truth-tables for evaluating formulas and for evaluating translated premise-conclusion arguments. The use of truth-values means that the formal logic is developed in terms of *truth-value semantics*.

Semantics are not confined to truth-value semantics. There is *pictorial semantics*. Pictorial semantics includes systems of logic maps and logic diagrams. Logic diagrams include Euler diagrams, Venn diagrams, Carroll diagrams, Karnaugh maps and electronic or switching circuit diagrams.

The pictorial semantics most closely related to propositional logic is switching circuit diagrams as in *Logic: Theory and Practice* (Rennie and Girle 1972, 99-103) and *Deductive Logic* (Halpin and Girle 1981, 248-265). It has to be remembered that the diagrams are not actual circuits. A switching circuit diagram gives a picture of an ideal circuit.

There is some "by the way" discussion of diagrams for predicate logic in chapter 6. The main focus in this text will be on truth-value semantics and deductive proof systems of logic and their inter-relation.

Many introductory logic texts from the 1960s such as *Introduction to Logic* (Copi 1961), *Natural Deduction* (Anderson and Johnstone 1962), The Elements of Logic (Barker 1965) and *Deductive Logic and Descriptive Language* (Harrison 1969), use truth-tables as stepping off point for propositional logic. Nothing has really changed much since then except for the slow but gradual increase in the use of truth-trees since *Formal Logic: Its Scope and Limits* (Jeffrey 1967).

Some very few texts do not take a semantic approach. They take a quite different approach and introduce systems of deduction and proof from the very beginning. Deduction systems are set out in terms of basic argument forms, and deduction rules of one kind or another and sets of logical equivalences. It is also typical for such texts to begin with truth-tables, sometimes almost in passing, and then move to a deduction system. The four texts cited together above all shift to the use of deduction and proof. They begin with semantics but move to deduction and proof. The use of deduction means that the logic is shifted from the semantic system of truth-tables to a *proof* system.

There is a lesser used method of deductive development. It is the method of axiom systems. There are standard propositional logic axiom systems to be found in texts such as *Logic: Theory and Practice* (Rennie and Girle 1973, 105-122), and *Elementary Logic* (Simco and James 1976, 82-100).

Some texts such as *Logic A First Course* (Blumberg 1976) and *The Logic Book* (Bergman Moore and Nelson, 1998) present both extended semantic systems with truth-trees and proof systems.

Truth-tables are the mainstay of the semantic approach to propositional logic. Proofs and deductions are the mainstay of proof systems for propositional logic. Those two kinds of systems are the main focus of the metalogic in the first four chapters of this text. What is the relationship between these systems? The answer is a metalogical answer.

In what follows we will set out and compare some of these semantic and proof systems, and show how they relate to each other. We begin with two standard truth-value systems that are very similar but crucially different. They are standard truth-table systems which can be found in many introductory logic texts such as *Introduction to Logic* 2nd Edition (Girle 2008, 69-96). The study will then move to axiomatic proof systems rather than natural deduction systems.

After showing that all the valid arguments of truth-value logic are also valid in axiomatic logic, we move on to make similar points about predicate logic, its formulas, its truth-value semantics and its axiomatic deductive logic.

To summarise: the logic systems considered here are formulated on a base of some artificial or formal symbolic language. In this chapter we look at the languages of propositional logic, and discuss the way in which their symbols are structured into the formal languages.

1.2 Symbols and Formulas

There are no surprises in the artificial language of propositional logic. We have some symbols and some grammatical or syntactical rules for building formulas from the symbols. These grammatical rules are the *formation rules* for the logic. In what follows "WSPL" stands for "Well formed formula of Standard Propositional Logic". These rules are like the rules set out in (Girle 2008, Chapter 2). The rules here will be somewhat more detailed.

Formally, there are the symbols:

As many *Propositional Translation Letters* (PTL) as we need:

$$A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1, \ldots, A_n, B_n, C_n, D_n, \ldots$$
$$\text{(where } n \geq 0\text{)}$$

As many *Propositional Variables* (PV) as we need:

$$p_0, q_0, r_0, s_0, p_1, q_1, r_1, s_1, \ldots, p_n, q_n, r_n, s_n, \ldots$$
$$\text{(where } n \geq 0\text{)}$$

There are infinitely many PTL and PV, if needs be.
In practice it is usual to leave off the sub-scripts. So "p_0" is written as "p", and "A_0" as "A".
There are, besides the PTL and PV, eight:

Incomplete symbols: $\sim$ & $\vee$ $\not\equiv$ $\supset$ $\equiv$ ( )

This collection of propositional letters and incomplete symbols is referred to as the *primitive symbols* of Standard Propositional Logic (*SPL*), not because they are uncouth, but because they are the basic building blocks for the artificial language or languages.

The incomplete symbols divide three ways.

Monadic Operator:	$\sim$
Dyadic Operators:	& $\vee$ $\supset$ $\equiv$ $\not\equiv$
Parentheses:	( )

We use the two phrases, *propositional letter* and *connective*, in the following way:

A *propositional letter* is either a Propositional Variable or a Propositional Translation Letter.

A *connective* is either a Monadic Operator or a Dyadic Operator.

Two logic languages are set out. The first is apt for translation purposes. There are no PVs in translation, just PTLs. The language is Applied Propositional Logic, APL. It's the language for translation.

The Well-formed formulas of APL (WAPL) are defined by:

Formation Rules:

BPTL:	Any PTL standing alone is a WAPL.
R~:	If α is a WAPL then $\sim\alpha$ is a WAPL.
R&:	If α and β are WAPL then $(\alpha \,\&\, \beta)$ is a WAPL.
R∨:	If α and β are WAPL then $(\alpha \vee \beta)$ is a WAPL.
R⊃:	If α and β are WAPL then $(\alpha \supset \beta)$ is a WAPL.
R≢:	If α and β are WAPL then $(\alpha \not\equiv \beta)$ is a WAPL.
R≡:	If α and β are WAPL then $(\alpha \equiv \beta)$ is a WAPL.
T	The set of WAPL is the smallest set compatible with the rules above.

The several rules for dyadic operators can be condensed into one rule:

R*:	If α and β are WAPL and * is a dyadic operator, then $(\alpha * \beta)$ is a WAPL.

There are ***five*** important points to note about these formation rules, and the formulas they define.

1. The formation rules, **BPTL**, **R~**, **R&**, **R∨**, **R⊃**, **R≢**, **R≡** and **T**, constitute a *recursive definition*.
   Recursive definitions in metalogic have three parts.

   (i) There are one or more *basis clauses*. In defining WAPL there is just one basis clause: **BPTL**.
   The basis clause states flatly or categorically that a symbol has a property, in this case, the property is *being a formula of* APL.

(ii) There are one or more *recursive clauses*. **R~**, **R&**, **R∨**, **R≢**, **R⊃** and **R≡** are the recursive clauses above.
The Recursive clauses are conditionals that say that IF something has the property, then something, usually more complex, also has the property of *being a formula of* APL.

(iii) There is a *terminal clause*. **T** is the terminal clause.
The terminal clause does not allow things such as:
p* or *p~* or □∘p~q*.
to count as formulas.

In this text the important Recursive Definitions are placed in a box. This is done because in some cases there will be exercises for developing such definitions, and the final correct definition will need to be made quite clear in one place.

So we have the condensed Recursive Definition of the WAPL:

RDef: WAPL (Well formed formulas of APL).

BPTL:	Any PTL standing alone is a WAPL.
R~:	If α is a WAPL then ~α is a WAPL.
R*:	If α and β are WAPL and * is a dyadic operator, then (α * β) is a WAPL.
T	The set of WAPL is the smallest set compatible with the rules above.

2. Formulas constructed by use of a basis clause alone are *atomic formulas*. They are the simplest formulas. Formulas constructed by use of either a basis clause only or a basis clause and just one use of **R~** are called *literals*. In other words, atomic formulas and the negations of atomic formulas are literals.

3. We use the definition to show how formulas are *constructed* or *assembled*. There are other ways of constructing formulas. We look at one of them below.

4. The formulas are *finite strings* of symbols. The definition can only ever give us a formula of finite length. Some logics allow for formulas that are infinitely long, but not APL. This finiteness is important for the discussion of topics later in this text.

5. The complex formulas of APL ***by themselves*** mean nothing. In this text this language is not treated as an abbreviation, symbolisation or regimentation of English or any other natural language. Just as French is not a symbolisation of English, APL is not either. The formulas only mean something if we ***give*** the symbols, or strings of symbols, some meaning.

   But, until we give them such meanings in a semantics, they are actually just symbols in defined finite strings.

When APL is used for translation the atomic formulas are given meaning in a translation dictionary such as:

A = Attitudes change over time.

B = The sky is blue.

The incomplete symbols are given a meaning for translation, but it is a "make do" intuitive meaning for translation. The formal meaning is usually truth-value meaning such as: If *A is true* then *~A is false*. If *Attitudes change over time.* is true then *Attitudes do not change over time.* is false. That's the meaning of the ~ symbol. A detailed account of truth-value meaning is set out in detail in Chapter 2.

But, no use has been made of the PVs in the definition of WAPLs. If **BPTL** is replaced by **BPV** we have a definition of the *Well-formed formulas of Pure Propositional Logic*: WPPL.

RDef: WPPL (Well formed formulas of PPL).

BPV:	Any PV standing alone is a WPPL.
R~:	If α is a WSPL then ~α is a WPPL.
R*:	If α and β are WPPL, not necessarily different, and * is a dyadic operator, then (α * β) is a WPPL.
T	The set of WPPL is the smallest set compatible with the rules above.

At once the question is, "What is the meaning of the formulas of pure propositional logic?" The answer often given is that the formulas of pure propositional logic display the *logical form* or *pattern* of propositions and premise-conclusion arguments.

1.3 Constructing Formulas

The formation rules can be used to show how a formula can be constructed or assembled from the primitive symbols in an *assembly line*. For example, consider the simple formula of PPL:

$(p \vee q)$

It is built out of two atomic formulas: *p* and *q* as follows:

1.	p	**BPV**
2.	q	**BPV**

The propositional letters are collected and then assembled into the formula:

3.	$(p \vee q)$	from formulas 1 and 2 by Clause **R∨**

In three steps we have *assembled* the formula, showing how the formula conforms to the definition of a WPPL. There is a well-formed formula at each step in the assembly line. The annotation to the right can be shortened from "from formulas 1 and 2 by Clause **R∨**" to "1, 2, **R∨**" Now consider the more complex formula:

$(\sim p \mathbin{\&} (q \vee r))$

It is assembled out of three atomic formulas: *p*, *q*, and *r* as follows:

1.	p	**BPV**
2.	q	**BPV**
3.	r	**BPV**
4.	$\sim p$	1, **R~**
5.	$(q \vee r)$	2, 3, **R∨**
6.	$(\sim p \mathbin{\&} (q \vee r))$	4, 5, **R&**

It's now time for the reader to assemble formulas. Fill in the spaces on the left. The correct answer is on the right or, in some cases, below the question. Here you can join in doing some metalogic:

1.	p	**BPV**	
2.	q	**BPV**	
3.	_______________	1, 2, **R⊃**	$(p \supset q)$

4.	________________	3, **R~**	$\sim(p \supset q)$

There are more examples below.

1.	p	**BPV**	
2.	q	**BPV**	
3.	r	**BPV**	
4.	________________	1, **R~**	$\sim p$
5.	________________	2, 3 **R∨**	$(q \vee r)$
6.	________________	4, 5 **R&**	$(\sim p \,\&\, (q \vee r))$

Assemble another formula:

1.	p	**BPV**	
2.	q	**BPV**	
3.	r	**BPV**	
4.	________________	2, 3 **R∨**	$(q \vee r)$
5.	________________	1, 4 **R&**	$(p \,\&\, (q \vee r))$
6.	________________	5, **R~**	$\sim(p \,\&\, (q \vee r))$

You can see that the last two assembly lines above place the ~ in slightly different places in the formulas at each line 6. The different placing is important for logic. In the first assembly line the ~ was built onto just *p*. In the second assembly line the ~ was built onto $(p \,\&\, (q \vee r))$. The small difference in place is an important difference in logic.

A second thing to note is that every time a dyadic operator was added in an assembly line, one left parenthesis and one right parenthesis are also added to the formula. At no other time are parentheses added. This means that every WPPL should have an equal number of left parentheses, right parentheses and dyadic operators.

Assemble another formula on the next page:

1.	p	**BPV**	
2.	q	**BPV**	
3.	________________	1 **R~**	$\sim p$
4.	________________	2 **R~**	$\sim q$
5.	________________	3, 2 **R&**	$(\sim p \,\&\, q)$
6.	________________	1, 4 **R&**	$(p \,\&\, \sim q)$
7.	________________	1, 2 **R≢**	$(p \not\equiv q)$
8.	______________________	5, 6 **R∨**	$((\sim p \,\&\, q) \vee (p \,\&\, \sim q))$

There are standard abbreviations in which the outside parentheses are left off. There are also abbreviations in which full stops and semi-colons replace the parentheses. Here follow some examples. See if you can work out, by just looking at them, what they are the abbreviations of; or, in other words, what they would un-abbreviate to:

$p \,\&\, q$	________________	$(p \,\&\, q)$
$\sim p \supset q$	________________	$(\sim p \supset q)$
$(p \,\&\, q) \supset r$	________________	$((p \,\&\, q) \supset r)$
$p \,\&\, q \,.\!\supset r$	________________	$((p \,\&\, q) \supset r)$
$p \,\&\, .\, q \supset r$	________________	$(p \,\&\, (q \supset r))$
$\sim (p \,\&\, q) \;:\!\supset\!:\; \sim p \vee \sim q$ ______________________		$(\sim (p \,\&\, q) \supset (\sim p \vee \sim q))$

Formula abbreviations will be avoided in this text. They add a level of complication best avoided. If they were to be frequently used in SPL then most would have to be unlearned when we come to Predicate Logic. Let’s not learn to do something that has to be unlearned later.

Follow the annotations to construct formulas.

1.	q	**BPV**	
2.	p	**BPV**	
3.	r	**BPV**	
4.	__________________	1 **R~**	$\sim q$
5.	__________________	4 **R~**	$\sim\sim q$
6.	__________________	2 **R~**	$\sim p$
7.	__________________	5, 6 **R&**	$(\sim\sim q \ \& \sim p)$
8.	__________________	7 **R~**	$\sim(\sim\sim q \ \& \sim p)$
9.	__________________	8, 3 **R∨**	$(\sim(\sim\sim q \ \& \sim p) \lor r)$
10.	__________________	9 **R~**	$\sim(\sim(\sim\sim q \ \& \sim p) \lor r)$

Fill in the rule clauses for the following

1.	p	____________	**BPV**
2.	q	____________	**BPV**
3.	$(p \ \& \ q)$	____________	1, 2 **R&**
4.	$\sim(p \ \& \ q)$	____________	3 **R~**
5.	$\sim p$	____________	1 **R~**
6.	$\sim q$	____________	2 **R~**
7.	$(\sim p \lor \sim q)$	____________	5, 6 **R∨**
8.	$(\sim(p \ \& \ q) \equiv (\sim p \lor \sim q))$		
		____________	4, 7 **R≡**

Fill in the spaces:

1.	A	**BPTL**	
2.	B	**BPTL**	
3.	C	**BPTL**	
4.	________	1, 2 **R&**	$(A \& B)$
5.	________	3, 4 **R∨**	$(C \vee (A \& B))$
6.	________	5 **R~**	$\sim(C \vee (A \& B))$
7.	________	1 **R~**	$\sim A$
8.	________	7 **R~**	$\sim\sim A$
9.	________	6, 8 **R⊃**	$(\sim(C \vee (A \& B)) \supset \sim\sim A)$

1.	A	________	**BPTL**
2.	B	________	**BPTL**
3.	$(A \supset B)$	________	1, 2 **R⊃**
4.	$\sim A$	________	1 **R~**
5.	$\sim B$	________	2 **R~**
6.	$\sim\sim B$	________	5 **R~**
7.	$(\sim\sim B \vee \sim A)$	________	6, 4 **R∨**
8.	$((\sim\sim B \vee \sim A) \equiv (A \supset B))$	________	7, 3 **R≡**

1.4 The Main Operator

The last operator added to a formula in its assembly line is the *main operator* of the formula. Atomic formulas do not have a main operator. The *main operator* of a formula is marked out by an arrow under it:

$$\underset{\uparrow}{\sim}(p \,\&\, {\sim}p)$$

Mark the main operators in the following formulas with an arrow. (For the first five, check the assembly lines above.)

$({\sim}p \,\&\, (q \vee r))$	$({\sim}p \underset{\uparrow}{\&} (q \vee r))$
${\sim}(p \,\&\, (q \vee r))$	$\underset{\uparrow}{\sim}(p \,\&\, (q \vee r))$
$((p \not\equiv q) \equiv (({\sim}p \& q) \vee (p \& {\sim}q)))$	$((p \not\equiv q) \underset{\uparrow}{\equiv} (({\sim}p \& q) \vee (p \& {\sim}q)))$
${\sim}({\sim}({\sim}{\sim}q \,\&\, {\sim}p) \vee r)$	$\underset{\uparrow}{\sim}({\sim}({\sim}{\sim}q \,\&\, {\sim}p) \vee r)$
$({\sim}(p \,\&\, q) \equiv ({\sim}p \vee {\sim}q))$	$({\sim}(p \,\&\, q) \underset{\uparrow}{\equiv} ({\sim}p \vee {\sim}q))$
$((((p \supset q) \supset q) \supset q) \supset q)$	$((((p \supset q) \supset q) \supset q) \underset{\uparrow}{\supset} q)$
$((p \vee q) \not\equiv ((p \supset q) \supset q))$	$((p \vee q) \not\equiv ((p \supset q) \underset{\uparrow}{\supset} q))$

We now have two distinct kinds of formulas, some formulas with only PTL's, and some formulas with only PV's. Let us use the original terminology: Standard Propositional Logic, SPL for a formal language that has both kinds of formulas.

In set theory terms the Well-formed formulas of SPL, WSPL, are simply the set of both WAPL and WPPL.

$$\text{WSPL} = \text{WAPL} \cup \text{WPPL}$$

If you do not understand this notation above then look at the Appendix for explanations of sets, set union and other simple operations on sets.

All the connectives other than $\sim$ and $\supset$ can be replaced by formulas containing only those two. For example, where α and β are SPL of just one kind, not necessarily different, the following is a tautological equivalence:

$$(\alpha \lor \beta) \equiv (\sim \alpha \supset \beta)$$

In the context of an artificial language in which only $\sim$ and $\supset$ occur we define the other operators:

$$(\alpha \lor \beta) =_{df} (\sim \alpha \supset \beta)$$

("$=_{df}$" means "is defined to abbreviate")

So, we can always replace any formula of the form $(\alpha \lor \beta)$ with a formula of the form $(\sim\alpha \supset \beta)$. This would mean that we could do without *vel* as an incomplete symbol so long as we have *tilde* and *hook*.
Do just that in the following formulas. Un-abbreviate them:

$(p \lor q)$	$(\sim p \supset q)$
$(\sim p \lor q)$	$(\sim\sim p \supset q)$
$(\sim p \lor \sim q)$	$(\sim\sim p \supset \sim q)$
$((p \,\&\, q) \supset (p \lor q))$	$((p \,\&\, q) \supset (\sim p \supset q))$
$((\sim p \lor \sim q) \supset \sim (p \,\&\, q))$	$((\sim\sim p \supset \sim q) \supset \sim (p \,\&\, q))$

We could do without & if we added:

$$(\alpha \,\&\, \beta) =_{df} \sim(\alpha \supset \sim\beta)$$

Apply this to the last formula answer above, and a couple more:

________	$((\sim\sim p \supset \sim q) \supset \sim\sim(p \supset \sim q))$
$((\sim p \,\&\, \sim q) \supset \sim(p \vee q))$	
________	$(\sim(\sim p \supset \sim\sim q) \supset \sim(\sim p \supset q))$
$(((p \,\&\, q) \supset r) \supset (p \supset (q \supset r)))$	
________	$((\sim(p \supset \sim q) \supset r) \supset$
________	$(p \supset (q \supset r)))$

In section **1.5** we will look at a pure language very like PPL, but with some interesting additional features. In the meantime we introduce one new symbol. It is the *falsum* symbol, sometime referred to as just "the false":

$$\perp$$

This symbol is a *Propositional Constant*. In standard semantic systems it is always ***false***. In both axiomatic and semantic systems it is often defined as an abbreviation:

$$\perp =_{df} (p \,\&\, \sim p)$$

We add it as another symbol. If we add this symbol we get Standard Propositional Logic with Constants, SPLC. The well formed formulas are WSPLC.

The *falsum* symbol is just like a propositional variable as far as formation rules go. So, we have the additional basis clause:

BPC: A Propositional Constant standing alone is a WSPLC.

We simply note these constants for the moment. There will be further discussion in the next chapter. The *falsum* constant is important in chapter 8.

1.5 Formation Trees

There is at least one other very important way of looking at constructing the formulas of *SPL*. This involves the use of *formation trees*. These are diagrams like truth trees, but they are the other way up – more like real above ground trees than the root systems of truth-trees.

Consider the formula above: $(\sim p \mathbin{\&} (q \lor r))$

The formation tree is this:

$$\begin{tikzcd}
p \arrow[d, dash] & q \arrow[dr, dash] & & r \arrow[dl, dash] \\
\sim p \arrow[dr, dash] & & (q \lor r) \arrow[dl, dash] & \\
& (\sim p \mathbin{\&} (q \lor r)) & &
\end{tikzcd}$$

The tree begins with the propositional variables at the top. As each connective is added the tree develops downward. The monadic operator is added by a move down a vertical path. Each dyadic operator is added by a move down in a joining branch. So we finish with the formula. If you think of the tree growing upward, then the tree breaks the formula down into the sub-formulas. The basic formula elements are at the top of the tree. They are the three PV's.

When we break the formula down in the way the tree does, then the process is sometimes called *parsing* the formula.

Consider the formula: $\sim((p \mathbin{\&} p) \supset (q \lor q))$

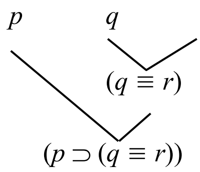
The formation tree is this:

$$\begin{tikzcd}
p \arrow[dr, dash] & & p \arrow[dl, dash] & & q \arrow[dr, dash] & & q \arrow[dl, dash] \\
& (p \mathbin{\&} p) \arrow[drr, dash] & & & & (q \lor q) \arrow[dll, dash] & \\
& & & ((p \mathbin{\&} p) \supset (q \lor q)) \arrow[d, dash] & & & \\
& & & \sim((p \mathbin{\&} p) \supset (q \lor q)) & & &
\end{tikzcd}$$

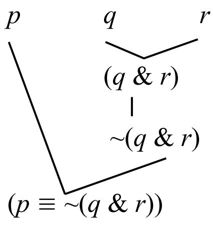
The multiple occurrence of both p and q in the formula is reflected in the multiple occurrence of both p and q at the top of the tree. This means that the tree does not just show what symbols are in the formula, but what *occurrences* of symbols are in the formula. There is a difference between *symbols* and *occurrences of symbols*.

Construct formation trees for the following formulas:

$(p \supset (q \equiv r))$

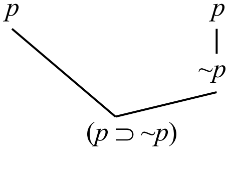


$(p \equiv {\sim}(q \,\&\, r))$

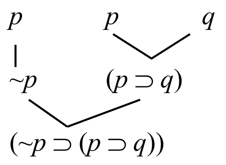


For the next formation tree, you must double list *p* to get what you see in the answer column to the right:

$(p \supset {\sim}p)$

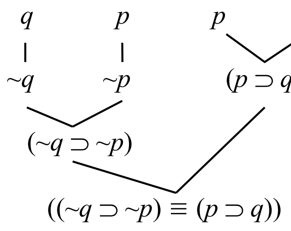


$(\sim p \supset (p \supset q))$



Double listing is required again.

$((\sim q \supset \sim p) \equiv (p \supset q))$



Note: the set of letters

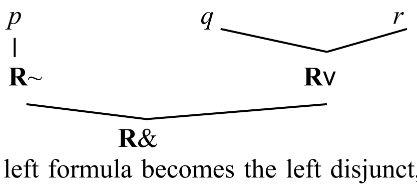
$\{q, p, p, q\}$

is exactly the same set as the set

$\{p, q\}$

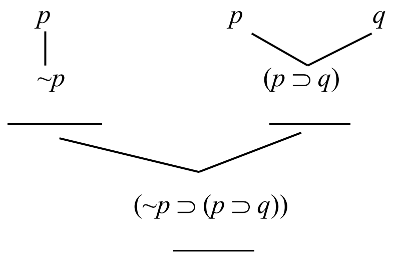
Double listing in a formation tree does not increase the number of letters in the set of letters in a formula. For more information about sets look in the Appendix.

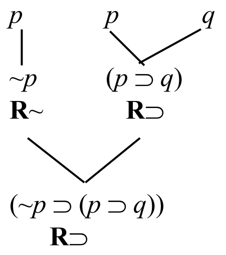
The formulas in the trees above are at the *nodes* of the trees. Each of the trees above could have been labelled in a more abstract way. Instead of formulas we could have used the clause names at the nodes. For example:

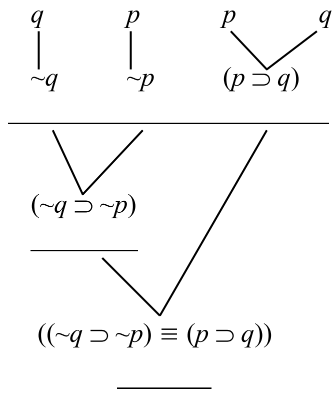


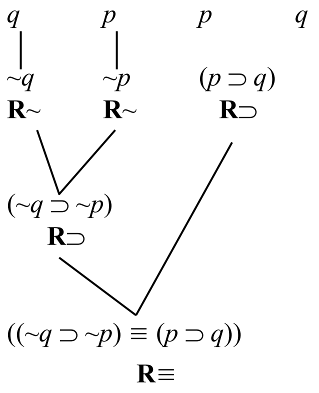
If we assume that the left formula becomes the left disjunct, the right becomes the right disjunct at the **R∨** node, and similarly for every dyadic operator, then for every formula there is a unique tree. The rule labelled tree tells us what the final form of the formula is, and the forms of its sub-formulas, right down to the letters.

In each of the following trees put **rule labels** under each node.









The final rule label tells us what the main operator is. And from that we know the formula's overall logical form.

1.6 Finite Symbols Language

If we have PPL only then we do not have any PTL's. Two more changes can be made. *Metalogic: An Introduction to the Metatheory of Standard First-Order Logic* (Hunter, 1971, 54-55)

The first is a way of limiting the number of symbols to a quite short *finite* list. Look at the list of symbols for SPL. There is an *infinite* list of PTLs and an *infinite* list of PVs.

There is a way of building each these lists out of just two symbols for each list. We will focus attention on the list of PVs. But the same applies to PTLs in the applied language.

Let there be just two symbols, a single letter and the prime symbol:

$p \qquad '$

With just one letter, p, and the prime symbol, $'$, a PV is defined as follows:

> A propositional variable is p followed by any non-zero finite number of $'$.

All the following are PV's:

$p' \qquad p'' \qquad p''' \qquad p''''''' \qquad p''''''''''''''$

We can then *abbreviate* for practical purposes. p' is abbreviated to p, p'' is abbreviated to q, p''' is abbreviated to r, p'''' is abbreviated to s, and so on with lower case letters. Since, in practice we are unlikely to use formulas with more than 20 PV's, less than the number of letters in the alphabet, abbreviation is unlikely to cause any practical problems.

If we follow this finite symbol path, then our language will be built out of just the following ***nine*** symbols:

$p \; ' \sim \& \vee \not\equiv \supset \equiv (\,)$

The second change is to simplify even further. This is done by cutting the nine symbol list to just ***six*** symbols:

$p \; ' \sim \supset (\,)$

With just six symbols we get the language FPPL–a finite pure propositional language. We let "WFPPL" stand for "Well formed formula of Finite Pure Propositional Language". The recursive definition of the formulas of the language would be: