

Formal Logic Handouts
What Philosophers Always Wanted to Know
About Reading McCawley

October 9, 2001

Contents

Handout I	3
1 Introduction	3
2 Formation Rules	3
2.1 A Simple Grammar	3
2.2 McCawley's Rules	4
3 Another Set of Formation Rules	6
3.1 McCawley's Rules for Propositional Logic	6
4 Special Topic: Parentheses and Ambiguity	7
5 Special Topic: Linguistic Trees	8
Handout II	9
1 Inference Rules	9
2 McCawley's Inference Rules	10
2.1 McCawley's Inference Rules for Quantifiers	10
2.2 McCawley's Rules for Propositional Connectives	10
	1

3	Special Topic: Subproofs	12
4	Special Topic: Proof Strategies	12
	Handout III	13
1	Truth Tables	13
2	Which Connectives are Necessary	13
3	Special Topic: Completeness	13
	Handout IV	14
1	Special Topic: Restricted Quantification	14
2	Special Topic: Branching Quantifiers	14
	Appendix	15

Phil 31045 Handout I (Revision 2): Formation Rules and Syntax

1 Introduction

There are three kinds of rules that make up a logical system:

1. *Formation Rules*: determine what does or does not constitute a well-formed instance (either trees or linear formulas) of the logical system. These are referred to as *wfs* or *wfws*.
2. *Inference Rules*: define necessarily truth preserving relations between well-formed instances. That is when can *wfws* (well-formed whatevers), A, B and C, be said to entail some conclusion D.
3. *Truth Values*: Some method of assigning truth values to the various instances of the system, so that it the truth of statements on which inferences are being performed can be evaluated. More daring philosophers, and almost all linguists, would look at these rules as the way that the purely syntactical and uninterpreted system developed by the first two steps can be given content.

So, far we have looked at McCawley's own version of the formation rules of quantifier logic in the tree form. We've also already seen that these rules can't be formulated without some attention to the way that truth values are to be assigned. The point of the coherence conditions on variables is to avoid having to assign truth-values to sentences that are irremediably deviant.

2 Formation Rules

2.1 A Simple Grammar

Formation rules are a set of rules intended to clearly identify the strings of symbols that are an acceptable part of a language. They are often called grammars and are, in many ways analogous to a grammar of a natural language. That is, if a interpreter of some sort were working with the language consisting a string of *a*'s followed by a string of *b*'s of equal length (ie, ab, aabb,...), then the string *aaabab* would cause a problem analogous to an

English speaker if they encountered a sentence like “a lecture for him to understand is difficult”. Strictly speaking, the interpreter has no well-defined idea of the strings mean (a failure of semantics) or what to do with them (a failure of inference rules). In most realistic cases, an interpreter will have tools available to them that will allow them to recover something from strings that are not too mangled.

Let’s take a closer look at this specialized language of a ’s and b ’s. (The $aabb$ language is useful for demonstrating formation rules, and not much else.)

1. If A is a wff of L_1 , then aAb is a wff of L .
2. ab is a wff of L_1 .

This language is called L_1 . That name is for the purposes of this handout only. Should you become fluent in L_1 , you’ll find that other speakers use other names for it.

There are some things to note about the formation rules provided above. “ A ” is not a wff, it is a variable which can be replaced by any wff of the language. This means that the language is defined *recursively*. Most interesting grammars are recursive, but some aren’t (such as the language consisting of a finite number of strings that could be listed.) The rules of L_1 are sufficient to describe all strings in the language of the form $a_1 \dots a_n b_1 \dots b_n$, no matter how large n may be.

2.2 McCawley’s Rules

For your convenience, McCawley’s Rules are as follows: (Remember, the way to interpret these rules is whatever is on the left side of the colon may be replaced by whatever is on the right.)

1. $S : Q' S$
2. $Q' : Q N'$
3. $S : \text{Arg Pred}'$
4. $\text{Pred}' : \text{Pred} (\text{arg}) \dots (\text{arg})$
5. $Q : \text{All, Each, Every, Any, Some, Most, } \dots$

6. Arg : $x, x_1, x_2, \dots, y, y_1, y_2, \dots, a, a_1, \dots, b, b_1, \dots$

7. Pred : Sleep, Breathe, Tall, ...

8. Pred : Admire, Love, ... (Arg)

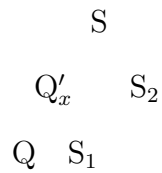
9. Pred: Between (Arg) (Arg)

Given a variable x and any expression of the formula expressed by tree (1) below.

10. S_1 must contain x .

11. S_2 must contain x .

12. S_1 and S_2 must not contain Q'_x .



3 Another Set of Formation Rules

A Quantifier is a part of a sentence that controls how a predicate relates to a class. McCawley presents a discussion based on virtually unlimited number of quantifiers, symbolized as Q.

In ordinary, linear notation, there are two quantifiers and they are expressed as \forall (for all) and \exists (exists).

Give a preference for these two quantifiers and for a linear notion, one might use the following formation rules for quantificational logic: (These are a simplified set of the rules given on p. 52-53 of A. G. Hamilton's *Logic for Mathematicians* (1987, CUP).

1. variables and individual constants are terms
2. if f_i^n is a function letter, than $f_i^n(t_1, \dots, t_n)$ is a term (if the t s are terms) (NB, terms are not *wf*).
3. if A_j^k is a predicate letter, than $A_j^k(t_1, \dots, t_k)$ is an atomic formula and a *wf*.
4. If A and B are *wfs* so are $(\neg A)$, $(A \rightarrow B)$. and $(\forall x_i)A$
5. All formulas generated according to the above template are *wffs*.

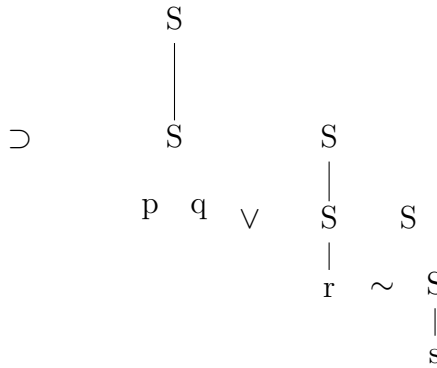
3.1 McCawley's Rules for Propositional Logic

The Formation Rules for First Order Propositional Connectives

1. S: $\forall S^n$ ($n \leq 2$)
2. S: $\wedge S^n$ ($n \leq 2$)
3. S: $\sim S$
4. S: $\supset S S$
5. S: p
6. S: q
7. S: r

8. ...

Bear in mind that each of these formation rules means that a sentence (or S) node in a tree may be replaced by each of the elements to the right of a full colon, each of them appearing as their own node and in the order that they appear. The atomic symbols and the logical connectives each form “leaves” that are not replaced by further replacements. The following is a valid tree according to the rules listed above.



4 Special Topic: Parentheses and Ambiguity

McCawley uses a loose version of so-called Polish Notation. This notation was originally derived by Łukasiewicz (1878-1956), the most important Polish logician or philosopher before Alfred Tarski. We’ll see more of Tarski once we have some semantics under our belt. The most evident feature of Polish notation is the absence of parentheses. This is accomplished by using a prefix notations, an operator always precedes whatever it operates upon. Hence, $A \wedge B$ becomes $\wedge A, B$. Some formulations in ordinary notation are ambiguous without parentheses. For instance, $A \wedge B \supset C \vee D$ could be $((A \wedge B) \supset C) \vee D$ or $((A \wedge (B \supset C)) \vee D)$ or any number of others. In strict Polish notation, the primary operator comes first, anything that comes later is within the “scope” of that operator, so $\wedge A, \supset B, \vee C, D$ is unambiguous. If we had a clear convention for when a certain symbol begins and ends, then the commas in this example could be done away with.

Of course, this does not mean that there is no role for any parentheses. Human readability matters, and parentheses are useful for that purpose, so unless there is a reason to write something in a very strict notation, feel free

to write $(\wedge A(\supset B(\vee D, C)))$. Translating back into the bracket free form is as simple as erasing the parens.

In addition to prefix notations, there are also postfix notations (the operator comes after whatever it operates on). For reasons having to do with computer architecture, postfix notation (also called Reverse Polish) has been utilized in the design of some important computer languages. In general, these do not preserve the bracket-free nature of the notation. So, reverse polish notation isn't really polish notation.

McCawley's choice is odd, if one considers the lengths that he goes to in order to preserve the special status of the grammatical subject of a proposition in his formation rules.

5 Special Topic: Linguistic Trees

Phil 31045 Handout II (Revision 1): The Syntax of First Order Logic w/ Inference Rules

1 Inference Rules

Inference rules give exact conditions under which some conclusion can be drawn from some premises. That is, what syntactical manipulations preserve truth. Another way to say this is that they define the conditions under which some set of symbols (premises) may be true (whatever that means, remember we haven't gotten that far yet) *only* when some other (the conclusion) is true, without any concern for what the strings themselves actually *mean* (again, whatever that means, semantics is coming.)

Inference Rules take many forms. For the sake of comparison, Hamilton's rules of inference for the Universal Quantifier take the following form:

1. $((\forall x_i A(x_i)) \rightarrow A(t))$ if $A(x_i)$ is a *wf* of L and t is a term in L which is free for x_i in $A(x_i)$.
2. $(\forall x_i (A \rightarrow B) \rightarrow (A \rightarrow (\forall x_i) B))$, if each A contains no free occurrences of the variable x_i .

Each of the above of inference schemes that can be used to derive an infinite number of possible equivalences. (That is, sentences that may replace each other while conserving truth.)

Hamilton's statement calculus contains a single inference rule equivalent to the traditional rule of Modus Ponens.

From A and (A → B), B is a direct consequence. (1)

Despite their superficial differences, Hamilton's rules are logically equivalent to McCawley's. This means that, given a way of translating between *wfws* of the two systems, any conclusion derivable in one is derivable in the other. From McCawley's rules, given below, it is easy to see his rule \supset -exploitation is Modus Ponens, so all of Hamilton's logic should flow from just that one rule. In fact, we could write rules to show that all of the inferences in McCawley's system could be derived from just the \supset -exploitation rule. However, the sacrifices we make for the simplicity in our original set of rules will come back to haunt us in the form of terribly difficult proofs.

Hamilton's system requires both \supset and \sim as propositional connectives, but the \sim does not have any special inference rules associated with it. It is possible to build a logical system of equal extension (that is, what can be proved in one system can be proved in the other) to our logical system with only a single connection, generally called "Nand" and symbolized $|$. We will not be too concerned with nand in this course.

2 McCawley's Inference Rules

2.1 McCawley's Inference Rules for Quantifiers

McCawley's inference rules for quantifiers have the following form:

$\begin{array}{l} \forall\text{-introduction} \\ \begin{array}{ l} Fu \\ \dots \\ Gu \end{array} \\ \hline (\forall:Fx)_x(Gx) \end{array}$	$\begin{array}{l} \forall\text{-exploitation} \\ (\forall:Fx)_x(Gx) \\ Fa \\ Ga \end{array}$
$\begin{array}{l} \exists\text{-introduction} \\ Fa \\ Ga \\ (\exists:Fx)Gx \end{array}$	$\begin{array}{l} \exists\text{-exploitation} \\ (\exists:Fx)Gx \\ \begin{array}{ l} Fu \\ Gu \\ \dots \\ A \end{array} \\ A \end{array}$
<p style="text-align: center;">Where u does not occur in A.</p>	

2.2 McCawley's Rules for Propositional Connectives

The inference rules for propositional connectives, in McCawley's system, follow:

\wedge -introduction A_1 A_2 \dots A_n $\wedge(A_1, A_2, \dots, A_n)$	\wedge -exploitation $\wedge(A_1, A_2, \dots, A_n)$ $A_i [1 \leq i \leq n]$
\vee -introduction A_1 $\vee(A_1, \dots, A_n) [1 \leq i \leq n]$	\vee -exploitation $\vee(A_1, \dots, A_n)$ $\begin{array}{ l} \dots \\ \hline B \end{array}$ \dots $\begin{array}{ l} \dots \\ \hline B \end{array}$ B
\sim -introduction $\begin{array}{ l} A \\ \hline \dots \\ B \\ \sim B \end{array}$ $\sim A$	\sim -exploitation $\sim \sim A$ A
\supset -introduction $\begin{array}{ l} A \\ \hline \dots \\ B \end{array}$ $\supset AB$	\supset -exploitation $\supset AB$ A B
Reiteration A $\begin{array}{ l} \dots \\ \hline \dots \\ A \end{array}$	

3 Special Topic: Subproofs

Subproofs are our chance to ask “what if?”. The sentences that propositions that we have accepted have some consequences that might not be evident until we consider different possibilities. This is most clearly true for the case of conjunctions. Each of the conjuncts has its own consequences. The consequences of the entire conjunction are those things implied by both of its parts. So, we can test to see if a certain proposition is implied by each of its conjuncts in turn. It’s not difficult to imagine a conjunction in which the two parts have incompatible consequences ($A \vee (\sim A \wedge B$ comes to mind.) and it may take different steps to get from one of the conjuncts to the desired conclusion. So, we segregate the sub-proofs leading from each conjunct from the rest of the proof. *Conclusions within each sub-proof are not valid anywhere else except in the sub-proof in which they are derived or subsequent sub-proofs of that sub-proof.* This is because each line of a subproof supposes information that is unavailable elsewhere in the proof.

Suppose that we were trying to exploit a conjunction ($A \vee B$). Each half would require its own subproof. Suppose further that A has the form ($A_1 \vee A_2$), then A_1 and A_2 would each require their own (second level) sub-proofs. Something proven in subproof A could be utilized further on subproof A or in either subproof A_1 , subproof A_2 . Something proven in subproof A_1 could only be used in subproof A_1 , not A_2 or A and certainly not in B .

Subproofs are book-keeping measures to keep steps that has been derived from certain suppositions limited to just those cases when the supposition can be expected to be true. They are not for any other purpose. So if one were trying to prove a conjunction, the two conjunctions would not require separate sub-proofs. They might each be proven in distinct lines of the proof, but they would not require subproofs at all. If they were put into sub-proofs, then we would need to find some way to get the information out (that is to show that the conclusion is true, even if the supposition of the sub-proof isn’t made). Subproofs are a powerful, but should be used carefully.

4 Special Topic: Proof Strategies

Phil 31045 Handout III: Semantics for Logical Connectives

1 Truth Tables

p	q	$p \supset q$	p	q	$p \vee q$	p	q	$p \wedge q$	p	$\sim p$
T	T	T	T	T	T	T	T	T	T	F
T	F	F	T	F	T	T	F	F	T	F
F	T	T	F	T	T	F	T	F	F	T
F	F	T	F	F	F	F	F	F	F	T

2 Which Connectives are Necessary

I mentioned earlier that the \supset relation was the only one necessary for the entire system of first order logic, this is because propositions using only the \supset rules can be constructed with truth conditions identical to propositions using any other logical connectives. Truth tables provide the easiest way to the truth of this statement. Consider the following

3 Special Topic: Completeness

1 Special Topic: Restricted Quantification

Restricted Quantification is the preferred method of linguists and some database designers (at least those who use logical quantification). Unrestricted quantification is generally preferred by philosophers and logicians.

There are at least three reasons to prefer restricted quantification (in order of decreasing plausibility).

1. Intuitive Plausibility
2. Avoids Contradiction
3. Simpler Proof Structures, in some cases

2 Special Topic: Branching Quantifiers

The following are various mistakes that I've found so far in McCawley's book.

p. 101, exercise i should read $\supset (\vee AB, C) \dashv\vdash (\wedge(\supset AC, \supset BC)$