1 Introduction

[The ideas behind this parser can be found in the book "A Mathematical Introduction to Logic," by Herbet B. Enderton, and also below. Here, we present the theoretical foundation behind propositional formulas, and discover 6 properties of wffs that the parsing algorithm uses, along with the idea of construction sequences, to conclude whether user-entered strings are wffs. The last section consists of the pseudo-code for the algorithm, and an argument concerning its correctness based on the theoretical foundation.]

To begin, we define our alphabet A to be the set $\{a-z, A-Z, ', (,), \tilde{}, \&, |, <, =, >\}$. Note that $a-$z means the 26 lowercase letters from $a-z$ inclusive; an analogous comment applies to $A-Z$. Note that our alphabet also contains the apostrophe symbol.

Now we define our universal set U to be the set of all strings of finite, non-zero length formed using the characters in A.

Let V be the set of propositional variables; that is, V is the subset of U such that every element of V begins with a letter followed by zero or more apostrophes.

We now define:

- $f_{\sim} : U \to U$ to be such that $f(x) = \sim x$
- $f_{=>} : (U \times U) -> U$ is such that $f_{=>}(x, y) = (x => y)$
- $f_{\&} : (U \times U) \to U$ is such that $f_{\&}(x, y) = (x \& y)$
- $f_{||} : (U \times U) \to U$ is such that $f_{||}(x, y) = (x || y)$
- $f_{<=>} : (U \times U) \to U$ is such that$f_{<=>}(x, y) = (x <=> y)$

We will call these the connective functions; $f_{\sim}$ is the unary connective function, and the rest are the binary connective functions.

2 An Intuitive Construction of the Set of Wffs

First, we define a *construction sequence* to be a finite sequence of elements of U, say $(a_1, \ldots, a_n)$, where every term a_i is either a propositional variable or is the result of applying one of the connective functions to a term or to terms in the sequence with index/indices less than i. The *length* of a construction sequence is the number of terms in the sequence.

If there is a construction sequence whose last term is a string x, then we say that x *has a construction sequence.*

Now we define:

$F_* =$ set consisting of all elements of U that have construction sequences.

This is our set of wffs.

Observe:

Every term in any construction sequence is a wff.

To see this, observe that any construction sequence of length one has, as its only term, a propositional variable. So any term of any construction sequence of length one is a wff.

Now, simply note that for $n > 1$, each of the $n-1$ strict subsequences (a_1), (a_1, a_2),..., $(a_1, \ldots, a_{n-1})$ of a construction sequence $(a_1, \ldots, a_n)$ are themselves construction sequences, and that, by definition of F_*, the last terms of construction sequences are wffs, and so $a_1, a_2, ..., a_n$ are all wffs.

Observe that here, it is fairly easy to show that something is a wff: one merely exhibits a construction sequence. It is, in general, not so easy to show that something is not a wff using the definition of F_*; to address this, we provide an alternate, equivalent construction of the set of wffs below:

3 An Equivalent Construction

First, a few more definitions:

If S is a subset of U, and if $x, y \in S$ implies that $\sim x, (x => y), (x\&y), (x||y), (x <=> y)$ are all in S, then we say that S is *closed* under the five functions defined above; for brevity, we will simply say S is *closed under* $\{f\}$.

If $V \subset S \subset U$, and if S is closed under $\{f\}$, then we say that S is a *inductive* set.

We now define:

$F^* =$ The intersection of all inductive sets.

Note that F^* itself is inductive.

To see this: V is obviously a subset of F^*, and if $x, y \in F^*$, then x, y are in every inductive set, and so $\sim x, \ldots, (x <=> y)$ are in every inductive set, and so $\sim x, \ldots, (x <=> y)$ are in F^*. That is, $F^* \supset V$, and F^* is closed under $\{f\}$; therefore, F^* is inductive.

We now show that $F_* = F^*$:

First, we'll show $F_* \subset F^*$:

Suppose $x \in F_*$, and so x has a construction sequence $(a_1, \ldots, x)$. a_1 is a propositional variable, and since F^* is inductive, a_1 is in F^*. Now suppose that terms $1, \ldots, n$ $(n \geq 1)$ of the sequence are in F^*. Since term $n+1$ is either a propositional variable or something formed via application of one of the connective functions to some of the preceding term(s), and since F^* is closed under $\{f\}$, term $n+1$ must be in F^*. So by induction, $x \in F^*$.

Now we'll show $F^* \subset F_*$.

First, observe that F_* is inductive: $F_* \supset V$, and if $x, y \in F_*$, then they have construction sequences $(a_1, ..., x)$ and $(b_1, ..., y)$, and so $(a_1, \ldots, x, \sim x)$ and $(a_1, \ldots, x, b_1, \ldots, y, g(x, y))$ are themselves construction sequences, where g is any binary connective function. So F_* is closed under $\{f\}$. So if $x \in F^*$, then x is in every inductive set, and so $x \in F_*$.

From now on, we will refer to the set of wffs using F, with the understanding that

$$F = F_* = F^*$$

4 The Induction Principle

As promised, we will now address the issue of showing that something is not a wff:

To do so, we first observe that F "absorbs" any inductive set that is a subset of F:

That is, if S is inductive, and S is a subset of F, then $S = F$.

To see this, we repeat the argument above: since $F =$ the intersection of all inductive sets, then F is a subset of every inductive set, including S, and so $S = F$.

This gives us something very useful: we can specify a property P, form a subset S of F that

consists of all the elements of F that have property P, and if we can show that S is a inductive set, then we can conclude that all members of F have property P.

This is very similar to mathematical induction, and so we call this the induction principle.

The induction principle allows us to assert that wffs have the 6 properties below, and that any string that lacks any one of these properties is not a wff. We will give proofs for our claims that wffs have property P_5 and P_6; the other properties can be easily verified.

5 Properties of Wffs

1. Any wff x has property P_1: its number of left parentheses is equal to its number of right parentheses (we will say that x is "balanced").

2. Any wff x has property P_2: it begins with either a letter, the $\sim$ character, or a (.

3. Any wff x has property P_3: If x begins with a letter, then x is a letter followed by zero or more apostrophes.

4. Any wff x has property P_4: If x begins with (, then it ends with).

5. Any wff x has property P_5: If x begins with (, then x has at least one proper initial segment, (meaning a segment of x that starts at the very beginning and is not equal to x) and any proper initial segment of x is such that its number of left parentheses is greater than its number of right parentheses (we say that the segments are left-heavy).

   Proof:

   Suppose S is the set of all elements of F with property P_5. Then all propositional variables are vacuously in S, and so $S \supset V$. If $x, y \in S$, then $\sim x$ is vacuously in S. Consider $g_C(x, y) = (xCy)$, where C is any binary connective. By inspection, (xCy) must have a proper initial segment. By the fact that x, y are also in F, and by property P_1, x and y are both balanced, and so any proper initial segment is necessariy left-heavy, by virtue of the leftmost left bracket. So $(xCy) \in S$.

   Hence, S is closed under $\{f\}$, and by the induction principle, $S = F$.

6. Any wff x has property P_6: If x begins with $\sim$, then $x = \sim \cdots \sim p' \ldots'$ or $x = \sim \cdots \sim (\ldots)$. Here, p stands for any letter, and $\sim \cdots \sim$ means one or more $\sim$'s and $' \ldots'$ means zero or more apostrophes. (...) means something beginning with '(', ending with), and also something that is balanced and having at least one proper initial segement, and whose proper initial segments are left-heavy.

   Proof:

   Suppose S is the set of all elements of F with property P_6. Then all propositional variables are vacuously in S, and so $S \supset V$. If $x \in S$, then x either begins with $\sim$ or doesn't. If it does, then $x = \sim \cdots \sim p' \ldots'$ or $x = \sim \cdots \sim (\ldots)$, and so $\sim x \in S$. If x doesn't begin with $\sim$, then, by the fact that $x \in F$ as well, and by property P_1 above, x begins with either (or a letter.

   If x begins with a letter p, then, by property P_3, $x = p' \ldots'$. So $\sim x$ is in S. If x begins with (, then by property P_1, P_4, and P_5, $x = (\ldots)$. So $\sim x \in S$.

   For any $x, y \in S$, and any binary connective function g, $g(x, y)$ is vacuously in S.

   So S is closed under $\{f\}$ and is thus inductive. Therefore, $S = F$, and every element of F has property P_6

6 The Algorithm

Let us define a *tree* to be a collection of ordered pairs (s, n), where s is a string and n is a natural number. We will call these pairs *nodes* and a node's *content* is its string s and a node's *level* is its number n. A tree's *root* is the node of the tree with level 0; every tree has exactly one root. Each node (s, n) may have, at most, two children, each with level $n + 1$. Every node, except the root, has exactly one parent. A *minimal* node is one with no children, and we say that a node is *parsed* if it has been fed, as an input, into our algorithm; otherwise, it is *unparsed*. Lastly, let $h(N)$ be a node N's content and let $l(N)$ be its level. Our algorithm makes heavy use of this tree structure.

Let us initialize our tree T with the root $(S, 0)$, where S is the user-entered string. At this point, the only node in our tree is the root and the root is unparsed.

Pseudo-code for our algorithm: (Below, "throwing an exception" entails exiting the algorithm)

Step 0: Choose the leftmost, minimal, unparsed node from T, say N, and take this as our input; N is now considered parsed. Go to step 1. If there are no more unparsed nodes, we exit the algorithm without throwing an exception.

Step 1: If $h(N)$ is a propositional variable, go back to step 0. Otherwise, go to step 2.

Step 2: If $h(N)$ does not begin with $\sim$, go to step 3. Otherwise, check to see if the length of $h(N)$ is at least 2. If it's not, throw an exception. If it is, take s to be $h(N)$ with the leftmost $\sim$ chopped off, and let the node $(s, l(N) + 1)$ be a (unparsed) child of N, and thus a node in T. Go back to step 0.

Step 3: If $h(N)$ does not begin with (, throw an exception. Otherwise, check to see if the length of $h(N)$ is at least 2. If it's not, throw an exception. If it is, starting from the second character of $h(N)$, read until you are confronted with a character that is not $\sim$. If you reach the end of $h(N)$ and are never confronted with such a character, throw an exception.

Otherwise, starting from that non-$\sim$ character, which must be either a left bracket or a letter (otherwise, throw an exception), begin reading character-by-character and adding to a balance, which is initially zero, in the following fashion: if you reach a left-bracket, add 1 to the balance, and if you reach a right bracket, subtract 1 from the balance. Otherwise, don't add anything. If you reach a letter, keep reading until you are confronted with a non-apostrophe character. After reading a character (or a string of characters, in the case of a letter followed by apostrophes), check the balance. If it's not zero, keep reading. If it is zero, stop reading.

If you read until the end of $h(N)$ and the balance is non-zero, throw an exception. If the balance does reach zero at some point, let α be the segment of $h(N)$ starting from the second character until and including the character that is read right before we stopped. If there is no binary connective following α, then throw an exception. Otherwise, check to see if $h(N)$ ends with) and that there is something strictly between the aforementioned binary connective and the terminal right bracket. If not, throw an exception. If there is, let this be β, and let $(\alpha, l(N) + 1)$, $(\beta, l(N) + 1)$ be the left and right children of N, respectively; we thus add two unparsed nodes to T. Go back to step 0.

Notice that the algorithm must terminate. That is, if no exception is thrown, then the content of the nodes being added to the tree are strict substrings of their parent node's content. Hence, the time it takes for the algorithm to terminate is bounded by the length of S.

Observe: S is a wff if and only if: we initialize T with $(S, 0)$, we run the algorithm, and the algorithm does not throw an exception.

To see this:

Suppose S is a wff. Then, by P_2, it begins with either a letter, $\sim$, or (. If it begins with a letter, then it is a propositional variable, and the algorithm will quickly terminate without throwing an exception. If it begins with $\sim$, then we go to step 2, and since S has a construction sequence, there must be a substring of S starting from S's second character, and this itself must be a wff (recall

that all members of a construction sequence are wffs).

If it begins with (, then there are two members of its construction sequence, say x and y, such that $S = (xCy)$, where C is a binary connective. In fact, x and y must be unique; that is, they must be the α and β specified by the algorithm. Otherwise, if x were less than α, and $\alpha \in V$, then a binary connective cannot possibly follow x, and so x cannot be used to form S. If x were more than α, then by P_3, x is not a wff, and so also cannot be used to form S.

On the other hand, if α begins with $\sim$, then x being less than it would make x equal to either $\sim \cdots \sim$ or $\sim \cdots \sim$ followed by either a proper inital segment of $p' \ldots'$ or $(\ldots)$. A string of $\sim$'s is not a wff, and x cannot be followed by a binary connective if it is $\sim \cdots \sim$ followed by a proper initial segment of $p' \ldots'$; lastly, by P_5, any proper initial segment of $(\ldots)$ is not a wff, and so x is not a wff. If x were more than α, then, by P_3 and P_5, it is also not a wff.

Now, using just property P_5, if we consider the case where α begins with (, we see that $x = \alpha$; for if x were less than α, x would be left-heavy and hence not a wff, and if x were more than α, α would be left-heavy, contradicting the specification of step 3 of the algorithm. And since x must be α, y must be β, and no exception is thrown.

Repeating the same arguments above for the subformulas of S, and further subformulas, we see that the algorithm must terminate without throwing an exception.

Now, suppose the algorithm does not throw an exception. Then it terminates and the resulting tree is simply a construction sequence for S arranged vertically. So S has a construction sequence and is thus a wff.