# High Level History and Overview of Mathematical and Logic Techniques Used in Mechanical Theorem Proving

By: Elita Cheung

### A Quick Logic Lesson:

Reading about mechanical theorem proving led our group to encounter many terms in logic which seemed at first alien to us. Realizing that understanding basic logic principles and definitions is crucial to the topic in discussion, we will introduce the readers unfamiliar with symbolic logic to the essential basics to gain a better insight on the intellectual excitement of the theories behind mechanical theorem proving. The weird symbols one encounters in the explanations of *resolution*, and other methods of theorem proving, can be easily defined with a quick lesson in symbolic logic.

### What is Symbolic Logic?

Symbolic Logic considers languages whose essential purpose is to symbolize reasoning encountered in mathematics as well as daily life. One of the simplest symbolic logic is propositional logic (or propositional calculus). We shall use propositional logic and its language to explain the general concepts of *resolution* as well as the overall concept and complexities involved in theorem proving. The more general symbolic logic is the first-order logic (or first-order predicate calculus) which we will also quickly cover. However, the emphasis of this section is to help the reader learn the basics and later use them to understand the high level overview of the theorem proving methods and development.

---

### Propositional Logic

The following is an example and some basic definitions to familiarize the reader with symbolic logic quickly and painlessly. This section is designed for readers with no background in logic, and can be read quickly to obtain a grasp of the vocabular used in the discussions to follow.

### An Example and Definitions

A **proposition** is a declarative sentence that is either true or false (it cannot be both). Examples of propositions: "Humidity is high", "Temperature is high", "One feels comfortable".

We can denote the above propositions as:

B ≙ Humidity is high
C ≙ Temperature is high
D ≙ One feels comfortable

Symbols, such as B, C, D, that are used to denote propositions are called **atoms** (atomic formulas).

Like all math, we use symbols in logic. Just as the symbol "+" is defined as "add", the following symbols have their own definitions.

| Symbol | Definition |
|---|---|
| ~ | Not |
| ^ | And |
| v | Or |
| → | if... then |
| ↔ | if and only if |

**Helpful example:** The sentence "If the humidity is high and the temperature is high, then one does not feel comfortable" can be represented by (( B ^ C) → (~D)). As we see, this compound proposition can represent a complicated idea that we deal with in everyday life.

The **truth value** of the proposition is the "true" or "false" that is assigned to a proposition.

**Formulas** are defined recursively as:

1. An atom is a formula
2. If G is a formula, then (~G) is a formula
3. if G and H are formulas, then (G ^ H), (G v H), (G →H), and (G↔H) are formulas
4. All formulas are generated by applying the above rules

Let G and H be two formulas. Then the truth values for the above formulas have the relationships shown below:

1. ~G is false when G is true. ~G is true when G is false. ~G is called the **negation** of G.
2. (G ^ H) is true if G and H are both true; otherwise, (G ^ H) is false. (G ^ H) is the **conjunction** of G and H.
3. (G v H) is true if either G or H is true; otherwise, (G v H) is false. (G v H) is the **disjunction** of G and H.

4. (G $\rightarrow$H) is false if G is true and H is false; otherwise, (G $\rightarrow$H) is true. It is read as "If G, then H."
5. (G $\leftrightarrow$H) is true whenever G and H have the same truth values; otherwise (G$\leftrightarrow$H) is false.

Below is the **truth table** that contains all the information from the above. The table allows us to find out the truth value of the different formulas by choosing the corresponding truth values of G and H.

| G | H | ~G | (G ^ H) | (G v H) | (G $\rightarrow$H) | (G $\leftrightarrow$H) |
|---|---|---|---|---|---|---|
| T | T | F | T | T | T | T |
| T | F | F | F | T | F | F |
| F | T | T | F | T | T | F |
| F | F | T | F | F | T | T |

**Interpretations of Formulas in Propositional Logic:**

Lets use an example to learn what an *interpretation* of a formula really is.

Consider the formula

$$G \triangleq (A \wedge B) \rightarrow (C \leftrightarrow (\sim D))$$

If the atoms A, B, C, and D are have the truth values *T, F, T*, and *T* respectively, then formula G is *T*. Lets work it out step by step to see how we got that answer.

(A^ B) is false because one of them is false. (~D) is false because D is true. Then, (C $\leftrightarrow$ (~D)) is false because C and (~D) does not have the same truth value. Easy way to think about it: C if and only if (~D); ( C $\leftrightarrow$ (~D)) is"true (C) if and only if (~D) is true". Because (~D) is false, that atom is false.

The formula G is then "if (A ^ B), then false". Since (A ^ B) is false, G would be true. Therefore, the formula G is *T* if A, B, C, and D are assigned *T, F, T,* and *T* respectively. The assignment of truth values {*T, F, T, T*} to {A, B, C, D}, respectively, will be called an **interpretation** of the formula G. There are a total of 2^4=16 interpretations of the formular G because each A, B, C, D can be assigned either *T* or *F.*

Like mathematics, symbolic logic has its own rules for the manipulation of formulas. For example, (P$\rightarrow$Q) is equivalent to (~P v Q). You can verify this equivalency by writing out the truth table of both the formulas and compare the truth values. There are associative laws, commutative laws, distributive laws

as shown in the following table. In the propositional logic, let ■ denote the formula that is always true and Å denote the always false formula.

| Equivalent formulas in Propositional Logic | | |
|---|---|---|
| 1 | F ⟷ G = (F → G) ^ (G → F) | |
| 2 | F → G = ~F v G | |
| 3 | F v G = G v F | F ^ G = G ^ F |
| 4 | (F v G) v H = F v (G v H) | (F ^ G) ^ H = F ^ (G ^ H) |
| 5 | F v (G ^ H) = (F v G) ^ (F v H) | (F ^ G) ^ H = (F ^ G) v (F ^ H) |
| 6 | F v Å = F | F ^ ■ = F |
| 7 | F v ■ = ■ | F ^ Å = F |
| 8 | F v ~F= ■ | F ^ ~F = Å |
| 9 | ~(~F) = F | |
| 10 | ~(F v G) = ~F ^ ~G | ~(F ^ G) = ~F v ~G |

With the above equivalent formulas, we can now reduce a complicated formula or concept into a more simple one (just like what we do in algebra).

**Example of Propositional Logic Usage:**

Suppose that the test scores goes down if the difficulty of the test increases. Also, suppose that most students are unhappy when their test scores goes down. Assume that the difficulty of the test does increase. Show that you can conclude that most students are unhappy.

To show the above conclusion, use propositional logic:

T ≙ Difficulty of test increases
S ≙ Test score goes down
U ≙ Most students are unhappy

The four following statements correspond to this example:

1. If the difficulty of the test increases, the test scores go down
2. If the test scores go down, most students are unhappy
3. The difficulty of the test increases
4. Most students are unhappy

Above can be symbolized as

1'. T → S
2'. S → U

3'. T
4'. U

Now, we can show that (4') is true whenever (1')^(2')^(3') is true. By using the equivalent formulas and manipulating the equation, we can transform (( T→S) ^ (S→U) ^ T) (representing (1')^(2')^(3')) into the form of T^S^U.

Therefore, if (( T→S) ^ (S→U) ^ T) is true, then (T^S^U) is true. Since T^S^U is true only if T, S, and U are all true, we conclude that U is true. U is called a **logical consequence** of ( T→S), (S→U), and T.

---

## First Order Logic

First order logic is a more expressive logic than prepositional logic. For example, if we denote

P: Every man is mortal
Q: Confucius is a man
cR: Confucius is mortal

then R is not a logical consequence of P and Q within the framework of the propositional logic. This is because these structures are not used in propositional logic. However, first order logic is more expressive. It has three more logical notions: **terms**, **predicates**, and **quantifiers** than propositional logic. Most of mathematical and everyday language can be symbolized by the first-order logic.

A **predicate** *GREATER(*x,y) can be defined to mean "x is greater than y." A predicate is a relation. "y is greater than 5" can be expressed as *GREATER*(y,5). It is also possible to use function symbols in first order logic. For example, we can use *subtract*(x,y) to denote "x - y". X and y here are individual symbols, and *subtract* and *GREATER* are function symbols.

There are also new symbols associated with first order logic. ∀*x* is read as "for all x", "for each x". And (∃*x*) is read as "there exists an x." ∀ and ∃ are called the **universal** and **existential** quantifiers, respectively. For example, it is easy to symbolize "for every x, there exists a y such that x is greater than y":

(∀*x*)(∃*y*)*GREATER(x,y)*

Just like in propositional logic, first order logic also has **interpretations** for a formula. However, because there are variables involved, it is more complicated

than propositional logic. We will give you a formal definition taken from one of the texts we used [1].

"An **interpretation** of a formula *F* in the first-order logic consists of a nonempty domain *D,* and an assignment of 'values' to each constatn, function symbol, and predicate symbol occuring in *F* as follows:

1. To each constatn, we assign an element in *D*.
2. To each n-place function symbol, we assign a mapping from *D^n* to *D*.
3. To each n-place predicate symbol, we assign a mapping from D^n *to {**T, F**}*."

This will become more clear by reading through the following example.

**Example of First Order Logic interpretations**

Consider the formula

$$(\forall x)(\exists y)Q(x,y).$$

By defining the domain *D* as

$$D = \{0,1\}$$

then we have to assign a truth value to each of the following: *P(0,1), P(0,0), P(1,0), P(1,1)* to define an interpretation.

| *P(0,0)* | *P(0,1)* | *P(1,1)* | *P(1,0)* |
|---|---|---|---|
| ***T*** | ***F*** | ***T*** | ***F*** |

If x = 0, there is a y such that *P(0,y)* would be **T**rue. (When y = 0).
If x = 1, there is also a *y* such that *P(0,y*) would be **T**rue. (When y =1).

In the above interpretation (the domain the the T/F assignments) we defined, ( ∀x)(∃y)*P(x,y).* i.e. for every *x* in *D*, there is a *y* such that *P(*x,y) is **T**.

**Definitions for first order logic:**

**Satisfiable** - A formula *P* is satisfiable (**consistent**) if and only if there exists an interpretation *I* such that *P* has a truth value of **T**rue in *I*.
**Unsatisfiable** - A fomula *P* is unsatisfiable (inconsistent) if there is no interpretation that can satisfy it.

---

**Herbrand's Theorem and Resolution, and a Little History...**

Church and Turing in 1936 proved that is was impossible to find a general decision to verify the inconsistency of a formula. However, there are proof procedures that will verify that a fomula is valid if the formula is in fact valid. For invalid formulas, these general procedures will never terminate (as we have discussed in our P=NP Question lecture).

One of the most significant developments in automated theorem proving occured in the 1930's and 1960's. In 1930, Herbrand proved an important theorem that changed the idea of a mechanical theorem prover into a more feasible one. He developed an algorithm to find an interpretation that can falsify a given formula. Because a valid formula is one that is true under all interpretations, this interpretation cannot exist if the formula is indeed valid, and his algorithm will halt after going trying out a finite amount of interpretations. However, it was impossible to implement the refutation procedure which Herbrand's theorem led to because going through the procedure by hand would take forever. (A casual definition of this refutation procedure is: instead of proving the validity of a theorem, the procedure proves that the negation of the formula is invalid.)

It was not until 1960's when Herbrand's procedure can be implemented on digital computer. In 1960, Gilmore implemented the procedure on a digital computer, and Davis and Putnam followed with a more efficient procedure. However, the most important breakthrough in the 60's in the automated theorem proving world was J. A. Robinson's **resolution principle** (1965). His single inference rule was highly effective and it reduced the exponentially increasing amount of time needed to implement Herbrand's procedure. Since then, the resolution procedure has been refined to increase its efficiency for applications.

**Herbrand's Theorem**

First, let us learn some more logic vocabulary in order to understand the concept under discussion.

Definitions:
**Clause** - a disjunction of literals
**Literal** - an atom or the negation of an atom

For example, the disjunctions $\sim P(x) \vee R(x)$ and $Q(x) \vee R(x)$ are clauses. And

$$\{\sim P(x) \vee R(x),\ Q(x) \vee R(x)\}$$

is a set of clauses.

A set of clauses is **unsatisfiable** if and only if it is false under all interpretations over all domains (by definition).

However, it is impossible to consider all the interpretations over all the domains. This is the importance of Herbrand's contribution. He developed a special domain *H* such that a set of clauses *S* is unsatisfiable if and only if *S* is false under all the interpretations over this domain. This **Herbrand universe of S** is defined as follows according to Chang and Lee [1]:

"Let $H_0$ be the set of constants appearing in *S.* If no constant appears in *S*, then *H0* is to consist of a single constant, say $H_0 = \{a\}$. For i = 0, 1, 2,..., let $H_i$+1 be the union of $H_i$ and the set of all terms of the form $f^n(t_1, .... t_n)$ for all *n*-place functions *f^n* occuring in *S,* where $t_j$, *j*= 1, ..., n, are members of the set $H_i$. Then each $H_i$ is called the *i-level constant set* of *S,* and $H_\infty$ is called the *Herbrand universe of S."*

Example:

Let $S = \{P(a), P(x) \vee \sim P(f(x))\}$. Then

$H_0 = \{a\}$
$H_1 = \{a, f(a)\}$
$H_2 = \{a, f(a), f(f(a))\}$
.....
$H_\infty = \{a, f(a), f(f(a)), f(f(f(a))), ...\}$

Definition: A **ground instance** of a clause *C* of a set *S* of clauses obtained by replacing variables in *C* by members of the Herbrand universe of *S.*

**Herbrand's Theorem:** A set *S* of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.

The above information basically contains the essence of Herbrand's theorem. Since the emphasis of our project is automated theorem proving in a general sense, we will not discuss the details. Basically, Herbrand's contribution opened a world of automated theorem proving that was not possible before. Unlike before where there is an infinite of different domains to test, now the domain is confined by Herbrand's universe. This breakthrough is the basis of *Resolution* which made automated theorem proving more effective.

## Resolution

Even though Herbrand's procedure was a major breakthrough, it has a major drawback. It requires generation of sets of ground instances of clauses which grows exponentially. It will literally take forever to prove interesting theorems. **Resolution principle** is the idea that will decrease the number of generation of sets of ground instances required by Herbrand's procedure.

The basic idea of the resolution principle is to check rather any set *S* of clauses contains the empty clause Å. If *S* contains Å, then *S* is unsatisfiable. If *S* does not contain Å, then check to see if Å can be derived from *S.* If it can, then it is also unsatisfiable.

**Resolution Principle for the Propositional Logic**

The resolution principle is easier to understand in propositional logic than in first-order logic. For propositional logic, the principle can be roughly described as the following: combine the literals that are complementary to each other so that they cancel out (e.g. P and ~P are complementary). Below is a quick example that demonstrate the concept.

**Example:**

Consider the set *S*

1. Q v ~P
2. P
3. ~Q

From (1) and (2), we obtain a resolvent

4. Q

From (4) and (3), we obtain a resolvent

5. Å

Here is a more rigourous definition of resolution principle from Chang and Lee [1]:

For any two clauses C and D, if there is a literal L1 in C that is complementary to a literal in L2 in D, then delete L1 and L2 from C and D, respectively, and construct the disjunction of the remaining clauses. The constructed clause is a **resolvent** of C and D.

The resolution principle is a very powerful inference rule. It is complete, that is: as set S of clauses is unsatisfiable if and only if there is a resolution deduction of the empty clause Å from S. The proof of its completeness can be found in Chang and Lee's book[1].

**Substitution and Unification**

The resolution principle for first-order logic requires understanding of two principles: **substitution** and **unification** for most general applications. However, I will use a first-order resolution example here that does not require those two technical concepts. Substitution is required for first-order resolution because clauses now contain variables. Look at the following clauses:

C1: P(x) v Q(x)
C2: ~P(*f*(x)) v R(x)

There is no literal in C1 that is complementary in C2 so that they can cancel out. Even though the predicates P and ~P look like they should cancel. Substitution is a procedure that manipulates the clauses so that they do cancel out in the end. It might remind readers to some of the ideas of algebra and calculus they tmight have studied.

Unification is also used to combine functions in clauses are are not exactly complementary. However, this report will not go in depth into these two ideas. Further information can be found in *Symbolic Logic and Mechanical Theorem Proving*.

**Resolution for First-Order Logic**

Most of the application of the resolution principle in first-order logic will require the use of substitution and unification. However, in order to give a general concept intsead of an in-depth math lesson, we will not show how those two concepts work in the resolution example. The following example is taken from Chang and Lee's *Symbolic Logic and Mechanical Theorem Proving*. I think it is a great example to show off the concept of resolution without concentrating on the math.

**Example**

Show that alternate interior angles formed by a diagonal of a trapezoid are equal.

Let T(x, y, u, v) mean that xyuv is a trapezoid with upper-left vertex x, upper-right vertex y, lower-right vertex u, and lower-left vertex v.

Let P(x, y, u, v) mean that the line segment xy is parallel to the line segment uv.

Let E(x,y,z,u,v,w) mean that the angle xyz is equal to the angle uvw. Then we have the following axioms:

A1: (∀x)(∀y)(∀u)(∀v) [*T*(x,y,u,v) → P(x,y,u,v)] ---definition of a trapezoid

A2: (∀x)(∀y)(∀u)(∀v) [P(x,y,u,v) → E(x,y,v,u,v,y)] --alternate interior angles of parallel lines are equal]

A3: T(a,b,c,d) --given (a trapezoid with vertices a,b,c,d)

From the above, we should be able to conclude that E(a,b,d,c,d,b) is true, that is

A1 ^ A2 ^ A3 → E(a,b,d,c,d,b) is a valid formula. Since we want to prove this by refutation, we negate the conclusion and prove that

A1 ^ A2 ^ A3 → ~E(a,b,d,c,d,b) is unsatisfiable. Details can be found on the last slide in the powerpoint presentation.

---

## Glossary

Atoms
Conjunction
Clause
Disjunctiion
Existential Quantifier
First Order Logic
Formulas
Ground Instance
Herbrand's Theorem
Herbrand's Universe of S
Interpretation (propositional logic)
Interpretation (first order logic)
Literal
Logical Consequence
Negation
Predicate
Proposition
Satisfiable

Truth Table
Truth Value
Unsatisfiable
Universal Quantifier