# PROJECT PROPOSAL

**October 12, 2021**

Michael Humphrey
Colorado School of Mines
CSCI598 Program Verification and Synthesis

## Introduction

Converting First Order Logic propositions to Conjunctive Normal Form (CNF) allows for easier solving of logical propositions and allows for faster evaluation of complex expressions by simplifying them to a more standardized form. For this reason, I propose to build a CNF solver that converts complex First Order Logical expressions to CNF. This CNF solver will be able to take many of the common connectives used in first order logic (Conjunctions, Disjunctions, Negations, Implications, and Biconditionals). These will be parsed in using standard representations of these connectives. For example:

- Conjunction: $\wedge, \&, AND$
- Disjunction: $\vee, |, OR$
- Negation: $\neg, !, NOT$
- Implication: $\Rightarrow, =>, IMPLIES$
- Biconditional: $\Leftrightarrow, <=>, IFF$

Each of these symbols will be able to be detected to make the solver more general and able to operate on more propositions.

These connectives will be expected to be in the following forms:

- Binary Connectives (Conjunction, Disjunction, Implication, Biconditional): $(\phi)f(\psi)$ where $\phi$ and $\psi$ are the expressions of the connective, and $f$ is the connective itself.
- Unary Connectives (Negation): $f(\phi)$ where $f$ is the connective and $\phi$ is the operator.

Additionally, for simplicity's sake, my CNF solver will expect the inputs to be fully parenthesized, and will output a fully parenthesized CNF version of this proposition. This CNF solver will be written as a multi-threaded C program and will be written using best practices of C. Although performance is not the primary goal of this project, it will be considered during development.

## Proposed Approach

My approach will be very similar to the algorithm described in lecture. It will similarly consist of three steps: implication elimination, conversion to Negation Normal Form (NNF) and finally, conversion from NNF to CNF. The algorithms for each are illustrated below:

```
Algorithm 1: ImpElim
   Input: $E : \phi, \sigma, \psi$ ;                        // left, op, right
   Output: $E' : \phi, \sigma, \psi$ ;                      // left, op, right
1  Function f($E$):
2      if $term(E)$ then
3          return $E$
4      else if $\sigma = Implication$ then
5          return $E(f(\phi), \vee, f(E(\neg, \psi)))$
6      return $E(f(\phi), \sigma, f(\psi))$
7  return $f(E)$
```

```
Algorithm 2: BiconElim
   Input: $E : \phi, \sigma, \psi$ ;                        // left, op, right
   Output: $E' : \phi, \sigma, \psi$ ;                      // left, op, right
1  Function f($E$):
2      if $term(E)$ then
3          return $E$
4      else if $\sigma = Biconditional$ then
5          return $E(f(ImpElim(\phi, \Rightarrow, \psi)), \wedge, f(ImpElim(\psi, \Rightarrow, \phi)))$
6      return $E(f(\phi), \sigma, f(\psi))$
7  return $f(E)$
```

**Algorithm 3:** NNF

```
   Input: $E : \phi, \sigma, \psi$ ;                    // left, op, right
   Output: $E' : \phi, \sigma, \psi$ ;                  // left, op, right
1  Function f($E$):
2      if $term(E)$ then
3          return $E$
4      else if $E = (\neg, E(\neg, E(\phi, \sigma, \psi)))$ then
5          return $f(\psi)$
6      else if $E = (\neg, E(\phi, \vee, \psi))$ then
7          return $f(E(\neg, \phi)), \wedge, f(E(\neg, \phi))$
8      else if $E = (\neg, E(\phi, \wedge, \psi))$ then
9          return $f(E(\neg, \phi)), \vee, f(E(\neg, \phi))$
10     return $f(\phi, \sigma, \psi)$
11 return $f(E)$
```

**Algorithm 4:** CNF

```
   Input: $E : \phi, \sigma, \psi$ ;                    // left, op, right
   Output: $E' : \phi, \sigma, \psi$ ;                  // left, op, right
1  Function Dist($E_1, E_2$):
2      if $E_1 = (\phi, \wedge, \psi), E_2 = \_\_$ then
3          return $E(dist(\phi, E_2) \wedge dist(\psi, \wedge, E_2))$
4      else if $E_1 = \_\_, E_2 = (\phi, \wedge, \psi$ then
5          return $E(dist(E_1, \phi), \wedge, dist(E_1, \psi))$
6      return $E(E_1, \vee, E_2)$
7  Function f($E$):
8      if $term(E)$ then
9          return $E$
10     else if $E = (\phi, \wedge, \psi)$ then
11         return $E(f(\phi), \wedge, f(\psi))$
12     return $dist(f(\phi), f(\psi))$
13 return $f(E)$
14
```

## Examples

INPUT:
$\neg(p(\neg(q \wedge (\neg pq))))$

Implication Elimination:
$\neg(p(\neg(q \wedge (\neg pq))))$ // input
$\neg(\neg p \vee (\neg(q \wedge (\neg\neg p \vee q))))$ // replace implications with disjunctions

NNF:
$\neg(\neg p \vee (\neg(q \wedge (\neg\neg p \vee q))))$ // input
$\neg(\neg p \vee (\neg(q \wedge (p \vee q))))$ // remove double negation
$(\neg\neg p \wedge \neg(\neg(q \wedge (p \vee q))))$ // propagate negation
$(\neg\neg p \wedge (\neg\neg(q \wedge (p \vee q))))$ // simplify expression
$(p \wedge (q \wedge (p \vee q)))$ // remove double negations

CNF:
$(p \wedge (q \wedge (p \vee q)))$ // already in CNF
Note, to be more explicit, this can be de-parenthesized to:
$(p \wedge q \wedge (p \vee q))$ // remove unneeded parenthesis
Similarly, to be completely explicitly in CNF:
$((p \vee \bot) \wedge (q \vee \bot) \wedge (p \vee q))$ // add implicit False disjunctions

## Proposed Evaluation

I plan to evaluate my program using a testing script that will test each component of the program (Removing Negations, Performing DeMorgan's, and Applying the Distributive property) as well as testing the program as a whole. This will be done using a testing shell script that executes the program on many different inputs, and expects the output to match a precomputed value. This evaluation will be written most likely as a shell script. Below is an example of a given input and output for each step of the algorithm.

INPUT:
$\neg(p \Rightarrow (\neg(q \wedge (\neg p \Rightarrow q))))$

Implication Elimination OUTPUT:
$\neg(\neg p \vee (\neg(q \wedge (\neg\neg p \vee q))))$

Negation Normal Form OUTPUT:
$(p \wedge ((q \wedge (p \vee q))))$

Conjunctive Normal Form OUTPUT:
$(p \wedge q \wedge (p \vee q))$

The testing script will ensure that each of these values match what is outputted by the program. This will be done using simple string comparison.