On the development of a logic calculator: a novel tool to perform logical operations

Oscar Chávez-Bosquez[1], Pilar Pozos-Parra[1], and Kevin McAreavey[2]

[1] Universidad Juárez Autónoma de Tabasco, División Académica de Informática y Sistemas. Cunduacán, Tabasco, Mexico,
{oscar.chavez, pilar.pozos}@ujat.mx,
[2] School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Northern Ireland,
kevin.mcareavey@qub.ac.uk

Abstract. In this paper we propose a Logic Calculator with three operation modes: evaluation of logical formulae; logical entailment and conversion of a formula to Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF). It is worth noting there are no existing automated tools for these three computations integrated into one tool providing a graphical calculator format. Logic Calculator is available at https://sourceforge.net/p/logiccalculator.

Keywords: propositional logic; logical entailment; disjunctive normal form; conjunctive normal form; open source tool

1 Introduction

In this paper we describe a helpful tool developed for the automatic computation of logic operations: the Logic Calculator. We show the development of the calculator and describe its functionality: *(i)* Evaluation of logical formulae. In this mode, it evaluates the basic Boolean operations: negation; conjunction; disjunction; conditional and bi-conditional. The user can insert the logical formula and the Logic Calculator displays the truth table along with the models of the formula. *(ii)* Logical entailment. In this mode, the user can insert a number of premises followed by a number of conclusions, so the Logic Calculator displays the truth table of each premise/conclusion and the result of whether or not these premises logically entail the given conclusions. *(iii)* Conversion to Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF). In this mode, the user inserts a logical formula and the Logic Calculator outputs its representation in both DNF and CNF. The main benefit of the Logic Calculator is the automatic evaluation of complex logic operations where manual calculations may be too difficult and take a considerable time, while there may also be a lack of confidence concerning the logic results obtained by hand.

2 Preliminaries

In this paper we consider the language L of propositional logic formed from the finite set of $\mathcal{P}_{24} = \{a, b, \ldots, t, w, x, y, z\}$ variables, where 24 denotes the maximum available number of variables in the Logic Calculator.

The set $\rho = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ denotes valid logical operators.

A Well Formed Formula (WFF), also called formula or sentence, is a propositional formula ψ containing a subset of variables from $\mathcal{P}_{24}$ and a subset of operators from ρ. Parentheses are used in the traditional way [6].

We can define a WFF as follows:

- An atomic formula (atom or variable) is a formula.
- If ψ is a formula, then $\neg\psi$ is a formula.
- If ψ and υ are formulas, and $\circ$ is a binary connective, then $\psi \circ \upsilon$ is a formula. Where $\circ \in \rho \setminus \{\neg\}$ is a binary connectives.

A truth table is a list of all possible truth assignments (possible worlds) for the set of variables. Such a table typically contains several rows and columns, with the header representing the variables and the WFF. The truth table is composed of one column for each input variable and one final column for all possible results of the logical operation that the table represents [3]. Rows represent every possible interpretation and the truth-values that would occur. Each one of these combinations is called an interpretation, such as an assignment of truth-values to all of the variables in the formula. [5].

A model of a propositional formula ψ is a possible interpretation where ψ is true under the interpretation in the classical truth functional manner. If there is a model for ψ, then ψ is said to be *satisfiable*.

When we want to know whether a set of sentences is a logical conclusion of another set of sentences, this is known as *logical entailment* [4]. A set of sentences Ψ *logically entails* a sentence ψ if and only if every truth assignment that satisfies Ψ also satisfies ψ. That is, the models of Ψ are also models of ψ. This relation is denoted as $\Psi \models \psi$.

A literal is an atom or negation of an atom. A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals. A formula is in Disjunctive Normal Form (DNF) if it is a disjunction of terms, where a term is a conjunction of literals. As in CNF, the only propositional operators in DNF are $\wedge$, $\vee$, and $\neg$. The $\neg$ operator can only appear as part of a literal, which means that it can only precede an atom [3].

3 Implementing the Logic Calculator

In this section we describe the modules and functionality of the Logic Calculator. We explain the main algorithms implemented and show some examples of using the three Logic Calculator modes.

3.1 Logical Formula Evaluator mode

The first mode developed was the evaluation of logical formulae, since this is the basic functionality and is also used by the Logical entailment mode.

The first step is to translate the input infix formula into a Reverse Polish Notation (RPN) (postfix notation) using the Algorithm 1, based on Dijkstra's Shunting-yard algorithm [8]. Postfix notation, while less common than infix notation in written mathematics, is much more explicit in the sense that the operands belonging to a particular operation can be expressed without the need for brackets or operator precedence.

Algorithm 1: *Shunting-yard* algorithm adapted for logical operators.

```
   input  : WFF
   output: RPN WFF
 1 foreach token in turn in the input WFF do
 2 |   if token is a variable α ∈ P_24 then
 3 |   |   Append α to the postfix output
 4 |   else if token is '(' then
 5 |   |   push '(' onto the stack
 6 |   else if token is an operator Δ ∈ ρ then
 7 |   |   while there is an operator Λ of higher or equal precedence than Δ at the top of
   |   |     the stack do
 8 |   |   |   Pop Λ off the stack
 9 |   |   |   Append Λ to the postfix output
10 |   |   end
11 |   |   Push Δ onto the stack
12 |   else if token is ')' then
13 |   |   repeat
14 |   |   |   Pop token x off the stack
15 |   |   |   if  x ∈ ρ then
16 |   |   |   |   Append x to the postfix output
17 |   |   |   end
18 |   |   until  x is '(';
19 |   else
20 |   |   throw ConvertException
21 |   end
22 end
23 while there are operators in the stack do
24 |   Pop the operator Δ on the top of the stack
25 |   Append Δ to the postfix output
26 end
```

To parse the expression, the operator precedence must also be defined. The list of valid operators in order of precedence (highest to lowest) is: [1] Negation ($\neg$), [2] Conjunction ($\wedge$), [3] Disjunction ($\vee$), [4] Implication ($\rightarrow$) and [5] Bi-implication ($\leftrightarrow$). Where operators have equal precedence, their associativity indicates in which order they are applied.

For example, the WFF 1 is processed by Algorithm 1, producing the RPN WFF 2.

$$p \vee q \leftrightarrow (q \rightarrow r) \wedge \neg s \tag{1}$$

$$p\ q \vee q\ r \rightarrow s\ \neg \wedge \leftrightarrow \tag{2}$$

The second step consists in setting up the truth table for the formula variables. If we consider the case where n is the number of distinct variables in the formula, then we need to set up a truth table with n columns and 2^n rows, as follows: column under the first variable should alternate 2^{n-1} *true*'s with 2^{n-1} *false*'s; column under the second variable should alternate 2^{n-2} *true*'s with 2^{n-2} *false*'s; continuing until we reach the last column variable.

The third step consists of evaluating the RPN WFF for each interpretation of the generated truth table. The explicitness of the RPN WFF obtained by Algorithm 1 allows us to implement the algorithm for evaluating postfix notation in a straightforward manner. Algorithm 2 shows the RPN parser developed using a stack as a helper data structure.

Algorithm 2: RPN parser.

```
   input  : RPN WFF, Interpretation Boolean values
   output: Interpretation Boolean result
 1 foreach token in turn in the RPN WFF do
 2     if token is a variable α ∈ P_24 then
 3         Push α Boolean value in the stack
 4     else if token is an operator Δ ∈ ρ then
 5         if Δ ≡ ¬ then
 6             Pop Boolean value β from the stack
 7             Apply negation to β
 8             Push β in the stack
 9         else
10             Pop Boolean value β from the stack
11             Pop Boolean value γ from the stack
12             Apply δ = βΔγ
13             Push δ in the stack
14         end
15     end
16 end
17 Pop Boolean result from the stack
```

For example given the RPN Formula 2 and the interpretation $p = 1$, $q = 0$, $r = 1$ and $s = 0$, Algorithm 2 outputs 1 as the result of the evaluation. There are 2^4 calls to Algorithm 2 since there are 4 variables in this formula.

3.2 Logical Entailment mode

In this mode we ca determine whether a set of premises logically entails a conclusion (or set of conclusions). The logical symbol $\models$ allows us to represent a logical entailment, separating the premises from the conclusion, with associativity from left to right. Premises are separated by commas (,). Parentheses are used in the classical way, usually to override the normal precedence of operators. Premises and conclusions are WFFs.

First step is to save the premises in a list and store the conclusion in separate lists. After evaluating each WFF from the premises part (using the modules developed for the Logical formula evaluator mode) and evaluating each WFF from the conclusion, a final truth table is used to determine whether the logical entailment holds (see Algorithm 3).

Algorithm 3: Logical entailment processor.

```
   input  : Premises list, conclusions list
   output: Whether logical entailment holds
 1 foreach WFF ψ in premises list do
 2 |   Evaluate truth table of ψ
 3 |   Add truth table to final table Γ
 4 end
 5 foreach WFF ψ in conclusions list do
 6 |   Evaluate truth table of ψ
 7 |   Add truth table to final table Γ
 8 end
 9 Evaluate Γ
```

For example, given the following case:

$$p \vee q, \quad q \rightarrow r, \quad \neg s \quad \models \quad q \vee r \tag{3}$$

We have the problem of determining whether $\{q \vee r\}$ (the conclusion) is logically entailed from $\{p \vee q, q \rightarrow r, \neg s\}$ (the premises). So, the Logic Calculator prints the final truth table and the result, in this case: *"Premises do not logically entail the conclusion"*.

3.3 Normal Form Converter mode

The conversion of a WFF to its DNF and CNF is made in two steps. First, a binary tree is created from parsing the WFF. Operator $\rightarrow$ is used to build the tree from the minor priority operator. For example, the binary tree shown in Figure 1 is created from Formula 4 with respect to the priority of operators.

$$a \rightarrow b \wedge c \vee d \rightarrow e \tag{4}$$

Second, the WFF conversion to CNF (resp. DNF) is based on the laws governing CNF and DNF conversion [5].

For example, walking through the binary tree t and generating the tree shown in Figure 2, resulting in the CNF shown in Formula 5. Similarly, the tree shown in Figure 3 corresponds to the DNF formula shown in Formula 6.

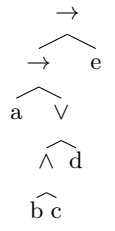

Fig. 1. Binary tree t representing the Formula 4.

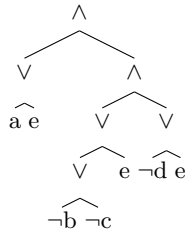

Fig. 2. Binary tree corresponding to CNF formula.

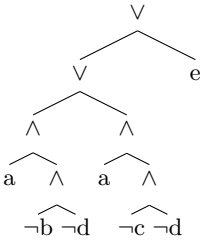

Fig. 3. Binary tree corresponding to DNF formula.

In both Formula 5 and 6 we have a formula translated so there are no negated parentheses and no other connectives apart from ¬, ∨ and ∧.

$$(a \vee e) \wedge (\neg b \vee \neg c \vee e) \wedge (\neg d \vee e) \tag{5}$$

$$(a \wedge \neg b \wedge \neg d) \vee (a \wedge \neg c \wedge \neg d) \vee e \tag{6}$$

It is worth clarifying that the Logic Calculator converts to an equivalent CNF formula rather than an equisatisfiable CNF formula (via the insertion of new variables [2]). The equisatisfiability technique is more common in terms of existing tools. Source code of this module is based on the JavaScript code from the Theorem Proving in Propositional Logic (WFFs) web site [1].

3.4 Logic Calculator GUI

One of the goals of the Logic Calculator is to provide a user-friendly User Interface (UI) similar to an arithmetic calculator. Figure 4 shows the GUI of the Logic Calculator in Logical formula evaluator mode. The output box is a text area for convenience, allowing easy copying and pasting of formulae. Debug area outputs debug details and log information.

4 Performance tests

We stress-tested the Logic Calculator on an Alienware M17x laptop, with an Intel Core i7-2670QM CPU @2.20GHz×8, 8 GB in RAM, 500 GB in HDD, Ubuntu Linux 12.04 64-bit and OpenJDK 7 64-bit.

Table 1 (column Op1) shows processing time plus printing time of the truth table. Memory is an issue when printing a truth table of 23 variables because $2^{23} = 8388608$ interpretations, so the Logic Calculator runs out of memory at printing 4206949 lines. This runtime exception is thrown depending on the amount of virtual memory that the JVM has access to. But we can perform operations with up to 24 variables without memory problems. For example, Table 1 (column Op2) shows processing time but only printing the number of models without printing the truth table, and Table 1 (column Op3) shows time performing conversions of WFF to their CNF and DNF. We can see that computing formulae conversions is quite quick, even for 24 variables formula.

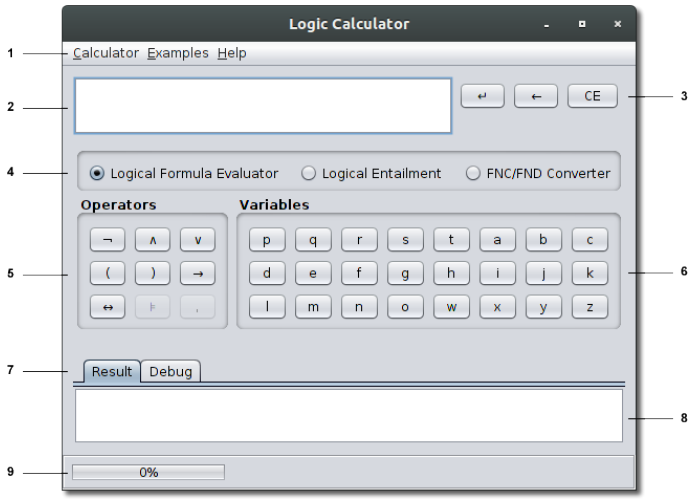

Fig. 4. Logic Calculator GUI components. (1) Main menu: Calculator, Examples & Help options (2) Text area for formula diplay (3) Input controls: Go, Backspace & Clear buttons (4) Calculator modes: Logical formula evaluator, Logical entailment evaluator & Normal Form converter (5) Available operators, depending of the selected mode (6) Up to 24 variables (7) Output & Debug tabs (8) Output text area for displaying results (9) Status bar.

Table 1. Logic Calculator time (in seconds) evaluating 8 expressions: printing their truth tables (Op1), without printing their truth tables (Op2), converting the expressions to both CNF and DNF (Op3). Column 1 indicates the number of variables in each formula. `*` indicates a `java.lang.OutOfMemoryError`. – indicates a non-tested expression.

Vars.	Formula	Op1	Op2	Op3
2	$a \to b$	0.040	0.029	0.032
5	$a \to b \wedge c \vee d \to e$	0.202	0.030	0.032
10	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j))$	0.643	0.082	0.033
20	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j)) \to k \wedge l \vee \neg m \vee n \wedge o \to p \vee q \to r \wedge s \to t$	124.427	1.929	0.034
21	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j)) \to k \wedge l \vee \neg m \vee n \wedge o \to p \vee q \to r \wedge s \to (t \wedge w)$	337.289	3.295	0.034
22	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j)) \to k \wedge l \vee \neg m \vee n \wedge o \to p \vee q \to r \wedge s \to (t \wedge w \wedge x)$	765.596	7.846	0.034
23	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j)) \to k \wedge l \vee \neg m \vee n \wedge o \to p \vee q \to r \wedge s \to (t \wedge w \wedge x \to y)$	*	20.481	0.042
24	$a \to b \wedge c \vee d \to e \to (f \wedge g \to h \wedge (i \vee j)) \to k \wedge l \vee \neg m \vee n \wedge o \to p \vee q \to r \wedge s \to (t \wedge w \wedge x \to y \wedge \neg z)$	–	40.408	0.052

5 Conclusion

The main benefit of the Logic Calculator is the automatic evaluation of complex logic operations. For example, to evaluate complex logical formulae with at most 24 variables and any number of parentheses, or to determine if a set of highly elaborate premises logically entails a set of highly elaborate conclusions, or to translate complex logical formulae to CNF or DNF, are highly complex tasks. Currently there are no tools which can solve these problems effectively, in a user-friend manner. Logic Calculator addresses these issues, so it can be an effective tool for new methods of reasoning.

The Logic Calculator was developed under the Open Java Development Kit (OpenJDK) 7 but source code using Java 5 as target, therefore it needs Java Runtime Environment version 5 or higher. OpenJDK platform has the same attributes of the official JDK, plus the benefits of being free software [7].

Multiplatform support is also an advantage, thus it can be executed over recent versions of Microsoft Windows, most Linux distributions, most UNIX variants and modern versions of Mac OS. Future work considers improve the efficiency of algorithms, manage more variables and a possible implementation in a hardware device.

Logic Calculator source code is downloadable from Sourceforge repositories at `https://sourceforge.net/p/logiccalculator`. A number of downloads up to 2016 is available at `https://sourceforge.net/projects/logiccalculator/files/stats/timeline?dates=2013-09-27+to+2016-12-31`.

Acknowledgments

To CONACYT for supporting the doctoral program in Computer Science at the Universidad Juárez Autónoma de Tabasco.

References

1. Allison, L. Theorem Proving in Propositional Logic (WFFs). `http://www.allisons.org/ll/Logic/Propositional/Wff/`, 2015.
2. Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer Publishing, USA, 1st edition, 2007.
3. H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, USA, 2nd edition, 2001.
4. M.R. Genesereth and E. Kao. *Introduction to Logic.* Morgan & Claypool Publishers. Synthesis Lectures on Computer Science, USA, 2nd edition, 2013.
5. M. Hazewinkel. *Encyclopedia of Mathematics*, volume 10. Springer Publishing, USA, 2002.
6. E. Mendelson. *Introduction to Mathematical Logic.* CRC Press, USA, 5th edition, 2009.
7. J. Russell and R. Cohn. *OpenJDK.* Book on Demand, USA, 1st edition, 2012.
8. R. Wilson. *An Introduction to Dynamic Data Structures.* McGraw Hill, USA, 1st edition, 1988.