Clausula: A Didactic Tool to Teach First Order Logic

María Virginia Mauco
vmauco@exa.unicen.edu.ar

Enzo Ferrante
eferrante@alumnos.exa.unicen.edu.ar

Facultad Cs. Exactas
Universidad Nacional del Centro de la Pcia. de Buenos Aires

Abstract

Immediate feedback and interactivity are crucial in any learning process. Introductory logic courses exercises are usually performed with paper and pencil approach. Students often have difficulties in handling formalisms and getting familiar with them. For this reason, we developed Clausula, a tool to support students' learning process in some First Order Logic contents.
This paper describes Clausula, an educational, visual, and interactive tool to experiment with arbitrary sets of First Order Logic clauses in order to determine their (un)satisfiability. The tool is easy and intuitive to use, and help students to improve their understanding of logic concepts. Clausula is implemented in C++, and it has been released under a free software license.

Keywords
Software tool, Clauses Satisfiability, Resolution Method, First Order Logic

1. INTRODUCTION

Basic courses on Logic are common in most informatics curricula. In this kind of courses, students have to do a lot of individual work to solve exercises and to gain experience in working with formalisms. In this context, the use of didactic tools which support the learning process, without taking so much time to learn to use them, is really useful.

The Undergraduate Degree Program in Systems Engineering in our career has an introductory course in Propositional Logic and First Order Logic (FOL) that is taught in the first semester of the second year. Learning these subjects requires students' substantial individual work, because they have to solve logic exercises in order to obtain skills in handling formalism. At first, we used lectures and pencil-paper problem solving approach to teach course contents. But as immediate feedback is essential for effective learning, and interactivity and visualization are keys to motivate and improve understanding, we considered the development of a software tool to support students' work. The tool should be very easy and intuitive to use, as it will be devoted to beginner students, and it should follow the same logical notation and approach used in lectures in order to help students get familiar with logical formalisms.

We then developed Clausula, a didactic, visual, and interactive tool that allows teachers and students experiment with arbitrary sets of FOL clauses in order to determine their (un)satisfiability (Ben-Ari, 2001). To achieve this, classic and fundamentals FOL methods were implemented, as for example the resolution method. In addition, Clausula gives the possibility of computing the most general unifier of two literals by using Robinson's

Unification Algorithm, and it allows the application of simplification strategies to the initial set of clauses, such as tautology clauses deletion, elimination of idempotent literals, and simplification of clauses because of the presence of pure literals (Ben-Ari, 2001; Burke, 1996; Cuena, 1985). Concerning the determination of the satisfiability of a set of clauses, the tool distinguishes if the clauses are a Logic Program, a set of Horn clauses or a set of arbitrary clauses in order to apply the corresponding strategy in each case (Burke, 1996).

The tool has been designed in the Object Oriented Paradigm, and implemented using C++, the STL library (STL, 1994) to manage basic structures, and the Qt framework (Qt Reference Documentation, 2009) for the graphic interface. In addition, it has been released under the license GNU GPL v2.0 (GNU, 1991), and thus it is possible to download it for use, study or source code modification from (Ferrante, 2009). This site also includes a complete description of the tool, the user manual, and the history of versions.

Clausula has been developed by an undergraduate third-year student of the career, who is also working as assistant in the course, with teachers' support and experience in introducing these subjects to beginner students.

This paper is structured as follows. In Section 2 Clausula general design is described. Section 3 presents the grammar for defining well-formed clauses. Section 4 describes the General Resolution Method implemented in Clausula. The main functionalities of the tool are shown in Section 5. Section 6 describes the evaluation of the use of Clausula in a logic course. Finally, some conclusions and possible future work are mentioned in Section 7.

2. CLAUSULA: GENERAL DESIGN

Clausula is an educational, didactic, and interactive tool that can be used as an aid in analyzing satisfiability in FOL clauses.

It has been designed in the Object Oriented Paradigm, and it has been implemented in C++. Clausula design is based on a set of Abstract Data Types (ADTs) that represent the different elements of FOL, establishing a correspondence between these elements and the classes which implement them.

Concerning the design model, three distinct layers can be distinguished:

- Main Structures Layer:

This layer contains all the ADTs that are the basis for the algorithms implemented in the upper layer. These ADTs provide an interface to manipulate FOL elements, and implement some basic methods to obtain potentially useful information, for example the ADT set of clauses can be asked if it is a set of Horn clauses. The correspondence between each FOL element and the classes and methods that implement them makes this layer the richest one concerning the design of structures. It is important that each class structure ensures an easy way to access the data it stores, because all the algorithms depend on them.

The modular way used to design the structures promotes software reuse (for example, structures may be used in other programs as external libraries).

- Algorithms Layer:

This layer includes the algorithms that are clients of the structures defined in the previous layer. These algorithms process the data represented by the ADTs in order to get useful information for the users. This layer has the methods of highest computational cost, as some of the algorithms are implementations of classic methods of FOL, such as Robinson's Unification Algorithm (Cuena, 1985), and the General Resolution Method (Ben-Ari, 2001).

- Graphic Layer:

This layer contains all the code for the Graphical User Interface (GUI) which provides easiness and agility in using the program. For this layer's implementation, the Qt framework (Qt Reference Documentation, 2009) was selected because it simplifies the design of forms and the portability of the application in different Operative Systems. Version 0.45 of Clausula is available for MS Windows as well as GNU/Linux, in order to let students select the platform to work with.

3. WELL-DEFINED CLAUSES

Clausula works with a subset of FOL formulas: a set of clauses (Ben-Ari, 2001). Therefore, it was necessary to define a way to check that all the strings typed by users of Clausula belonged to the language of FOL well-defined clauses. Then, a context-free grammar was defined in order to recognize strings that are FOL well-defined clauses; the BNF for the grammar is shown in Figure 1. This grammar was implemented using the free tools Flex, for lexical analysis (Paxson, 1995), and Bison, for syntactic analysis (Donnelly, 1995). In this way, we obtained a module that detects if a string is a well-defined clause or not.

It is important to remark that, in case of error, Clausula reports the type of mistake the user has done in order s/he could detect and correct it easily. This is important from a didactic point of view as the users are not only warned about the error but they also get some clues to correct it.

4. GENERAL RESOLUTION METHOD

The General Resolution Method is the main part in Clausula. This procedure will try to determine if a set of FOL clauses is satisfiable or unsatisfiable. Satisfiability may not be always determined because clauses satisfiability in FOL is undecidible (Ben-Ari, 2001). Then, if the set of clauses is unsatisfiable Clausula will always give an answer; however, if it is satisfiable an answer may or may not be given. For this reason, what Clausula implements is a procedure but not an algorithm, as we cannot assure it will finish for any input set of clauses (Lewis, 1998). Nevertheless, Clausula will always give a response in case the input set of clauses corresponds to a Logic Program as it has been demonstrated it is always satisfiable (Ben-Ari, 2001).

Resolution Method uses only one rule to deduce new clauses: the resolution rule or resolvent definition. This rule takes two clauses C_1 and C_2 (which do not share variables), and literals $l_1 \in C_1$ and $l_2 \in C_2$, such that l_1 and the complementary of l_2 can be unified by the most general unifier u, and defines the resolvent of C_1 and C_2 as the clause

$$Res(C_1, C_2) = (C_1u - \{l_1u\}) \cup (C_2u - \{l_2u\})$$

Before the application of this rule to any pair of clauses, variables must be renamed (so that clauses do not share variables), and the literals involved must be unified.

The basic idea behind the General Resolution Procedure implemented is to work with two lists of clauses L_c and L_t. L_c contains potentially combinable clauses (a clause is potentially combinable if it is not equivalent to any other clause in the set of resolvents, and it has a literal whose complementary one may be found in another clause), and L_t groups all the clauses used in the process (the original ones plus the ones obtained applying the resolution rule). Then, in each iteration a clause C from L_c is combined with all the possible ones in L_t (exploring breadth all possible solutions). Once all the combinations are made, C is deleted from L_c (but it remains in L_t). To determine if two clauses may be combined, it should be verified if they have complementary literals. The clause obtained from the combination may be a potentially combinable clause (and in this case it is added to both lists) or not (it is only added to L_t). It is important to remark that before adding a clause to any of the lists, the strategies of deletion of tautological clauses and simplification of clauses by idempotent literals (Kelly, 1997) are applied. Any time the empty clause is obtained as resolvent, the procedure finishes and Clausula informs the user the input set of clauses is unsatisfiable. If the L_c list becomes empty, the tool answers the input set is satisfiable.

5. CLAUSULA MAIN FEATURES

As Clausula has been designed with the purpose of having a didactic tool, it is important to highlight the functionalities it includes as regards FOL teaching. During FOL learning process, there are a number of different reasons that make students analyze satisfiability in a set of clauses:

- to learn the concept of satisfiability in order to check exercises;

- to determine if a conclusion follows from a set of hypothesis (logical consequence notion, (Ben-Ari, 2001));

- to obtain automatically a correct answer substitution from a specification written in FOL (Ben-Ari, 2001).

In all these cases Clausula offers didactic support to verify exercises, giving confidence to students about the correctness of their results.

The UML activity diagram in Figure 2 shows (in a general way) data flow in the tool, and allows one to understand how the modules implementing each functionality of the tool are related and interact.

Below, a more detailed description of each functionality is provided.

Resolution in a Set of Clauses

As it has been mentioned previously, the main idea behind Clausula is to provide an easy implementation of the Resolution Method in FOL. As regards computability theory, the problem of determining if a set of FOL clauses is or is not satisfiable is undecidible (Ben-Ari, 2001). This means there is no algorithm to decide if any arbitrary set of clauses is satisfiable or not. However, there are set of clauses that have some singularities that may be used to define some heuristics or refinement methods which help to reduce processing times.

Clausula has implemented two of such refinements in order to try to overcome some of the limitations inherent to the Resolution Method: detection of Horn Clauses and detection of a Logic Program. In case these refinements cannot be applied, the General Resolution Method is be used.

- Horn Clauses

A Horn clause is a clause with at most one positive literal. When Clausula detects the input set of clauses corresponds to a set of Horn clauses, it applies Unit Resolution, a complete strategy when working with Horn clauses (Burke, 1996). This refinement forces one of the clauses to be used in the calculation of a resolvent to be a unit clause, that is a literal. In this way, the number of possible combinations of clauses to calculate is reduced considerably.

- Logic Program

A Horn clause may be of any of three types: rule (one positive literal and the rest negative ones), fact (unit clauses with one positive literal), or goal (no positive literal, all are negative ones). A set of rules and facts defines a Logic Program. It has been demonstrated that every Logic Program is satisfiable (Ben-Ari, 2001). Then, when detecting the input set of clauses is a Logic Program, Clausula informs the user the input set of clauses is satisfiable without calculating any resolvent.

- General Case

In case the input set of clauses is not a set of Horn clauses or a Logic Program, Clausula applies the General Resolution Method. This gives robustness to the tool in the sense it tries to determine the satisfiability of any arbitrary set of clauses.

It is important to remark that the tool always tells the user which was the strategy followed to get a result. In this way, users will know the theoretical concept applied to solve the problem, and thus they may compare it with their own results. In addition, Clausula shows the resolvents that were calculated by the resolution process applied, thus giving students more elements to understand the problem and its solution. Figure 3 shows an example set of clauses with all the computations performed by the tool to conclude the set is unsatisfiable. In addition to the desired set of clauses, the user must indicate which symbols must be considered as constants. In this case, we can see that, before performing the resolution process, the tool has simplified the original set of clauses using pure literal elimination strategy.

Unification

Before applying the FOL resolution rule, the pair of clauses involved must be unified. Clausula implements a reduced version of Robinson's Unification Algorithm as it works with two literals (instead of n literals as Robinson's Algorithm does) (Cuena, 1985).

In addition of the use of this algorithm as part of resolution process, Clausula allows students make use of it in an option of the menu especially devoted to unification. When this option is selected, given two literals the tool will find the most general unifier (mgu) and it will show the result; in case the literals are not unifiable, Clausula will inform this to users. In this way, students could check their exercises to find out if their results are correct. Figure 4

shows the result when two unifiable literals are input.

Simplification Methods

One effective way of reducing the number of combinations of clauses to perform and thus improving the procedure performance is the application of simplification methods before and after resolution process.

Clausula allow users to choose which method or set of methods to be applied to the input set of clauses. Following, we briefly describe each method implemented by the tool.

- *Equivalent Clauses*

Clausula will compare each clause in the set to all the rest trying to find an equivalent one. If two clauses are found equivalent, one of them is deleted from the set. Figure 5 shows the application of this method to input set of clauses shown in Clauses of Figure 3.

- *Idempotent Literals*

This simplification method uses idempotence property (Cuena, 1985) to simplify literals inside a clause. The algorithm is simple: for each clause in the set to simplify, each literal is compared to all the rest to verify if it is equivalent or not to any literal in the clause; if it is equivalent, it is deleted from the clause.

- *Tautological Clauses*

The simplification by tautological clauses is very simple, and it consists in asking each clause if it is a tautology or it is not; in case the clause is a tautology, it is deleted from the set, as it will not contribute to deduce the empty clause. The method associated to each clause to determine if it is a tautology or not, takes each literal in the clause and compares it to the rest trying to find its complementary literal (if the complementary is found, the clause is considered a tautology). Figure 6 displays the new set of clauses after eliminating tautological clause $\neg C(u) \vee C(u)$ from the input set of clauses that appear in Clauses of Figure 3.

- *Pure Literals*

A literal l in a clause C is defined as pure for the set of clauses S ($C \subseteq S$) if and only if the complementary literal of l cannot be found in any clause of S different from C (Arenas, 1996).

Considering this definition, the simplification method will delete clauses containing one or more pure literal, as these clauses will never contribute determining the unsatisfiability of the set. Figure 7 shows the new set of clauses after deleting $\neg A(x, y) \vee \neg B(y) \vee D(x, f(x))$ from the input set of clauses that appear in Clauses of Figure 3 as it contains pure literal D(x, f(x)).

6. EVALUATION

Clausula has been successfully class-tested during the first semester of this year. It has been used in lectures, to introduce FOL resolution method, and for homework assignments. Students report positive experiences with its use (we have approximately 150 students per year). They found Clausula easy to install and very intuitive to use, and so they were quite enthusiastic in working with the tool on practical exercises. Students also appreciated the assistance provided by Clausula to correct mistakes when calculating the most general unifier or when trying to find out if a set of clauses is satisfiable or not. As Clausula checks the input set of clauses by using the context-free grammar defined in Section 3, students could correct mistakes when defining FOL clauses. As another advantage, they mentioned the possibility of using simplification methods as desired in order to contrast the results obtained using paper and pencil. One limitation remarked by many students was the fact that Clausula does not work with arbitrary FOL formulas, it only accepts clauses. Then, they suggested including a module to convert any FOL formula to a set of FOL clauses.

7. CONCLUSIONS AND FUTURE WORK

Clausula is a didactic, interactive and easy to use educational tool integrated in an undergraduate course in FOL. It was developed by an undergraduate third-year student, who is also an assistant in the course, considering contents and teaching methodology followed in the course. Clausula has been and will be used as support software in the course. The experience of its introduction in the logic course was completely positive, as according to students and teachers opinions the tool is easy and intuitive to use, and allow students to apply the learnt concepts and to verify

their results, thus participating more actively in the learning process. In this way, FOL concepts remain in the center with the tool collaborating as an assistant. Besides, we want to highlight that the development of educational tools by students in order to be used by other students, is an additional motivation when students have to choose the projects to implement in the different courses of the career.

Additionally to the fact that Clausula is a didactic tool, and all the functionalities it provides to analyze FOL clauses satisfiability, the tool has an important advantage for teaching: it is free software (Stallman, 2002). When free software is used in teaching/learning processes students have the possibility of using and sharing, without restrictions, all the resources it offers. But more importantly, students can analyze concrete implementations of algorithms and thus, besides using them as support in their learning processes, they can participate in the development of their own tools. For example, one of the students who took the course and experimented with Clausula is finishing the implementation of a new functionality to add to the tool: a translator from arbitrary FOL formulas to formulas in clausal form (also known as Skolem normal form) (Burke and Foxley, 1997). In this way, Clausula will extend its functionality to work with arbitrary sets of FOL formula.

Clausula interface was initially developed for Spanish speakers by using the Qt framework. One of the additional advantages of having used this library is that Clausula interface could be easily translated to English using a set of tools provided by Qt.

We continue evaluating, improving, and extending Clausula as an integrated educational tool, on the basis of our teaching experience and students' suggestions. Concerning the tool implementation, we will work on reducing the computational cost of some functions involved in the resolution process. Regarding the addition of new functionalities, we plan to extend the tool to work with the notion of logical consequence. Another extension could be the inclusion of a module to allow users to guide themselves the resolution process, i.e. selecting which pair of clauses will be taken to calculate next resolvent. Moreover, it would be really useful that the tool could show the resolution tree graphically. We also plan to study in depth Human-Computer Interaction factors (HCI, 2009) that influence students' learning process success in order to adjust Clausula interface.

REFERENCES

Arenas, A. (1996) Lógica Formal para Informáticos (Formal Logic for Informatics), Editorial Diaz de Santos.

Ben-Ari, M. (2001) Mathematical Logic for Computer Science, Prentice Hall, Series in Computer Science.

Burke, E. and E. Foxley (1996) Logic and its Applications, Prentice Hall, Series in Computer Science.

Cuena, J. (1985) Lógica Informática (Informatic Logic), Alianza Editorial.

Donnelly, C. and R. Stallman (1995) Bison - Version 1.25: The YACC-compatible Parser Generator http://dinosaur.compilertools.net/bison/index.html (Accessed May 2009).

Ferrante, Enzo (2009) Clausula: Resolución en Lógica de Primer Orden. http://clausula.sourceforge.net/

GNU General Public License, version 2 (1991) http://www.gnu.org/licenses/gpl-2.0.html (Accessed August 2009).

HCI Bibliography: Human - Computer Interaction Resources (2009) http://hcibib.org/ (Accessed October 2009).

Kelly, J. (1997) The Essence of Logic, Prentice Hall.

Lewis, H. and C. Papadimitriou (1998) Elements of the Theory of Computation, Prentice Hall, Second Edition.

Paxson, V. (1995) Flex - Version 2.5: A Fast Scanner Generator. http://dinosaur.compilertools.net/flex/flex.ps. (Accessed May 2009).

Qt Reference Documentation (Open Source Edition) http://doc.trolltech.com/4.5/ (Accessed May 2009).

STL: Standard Template Library Programmer's Guide (1994) http://www.sgi.com/tech/stl/ (Accessed May 2009).

Stallman, R. (2002) Free Software Free Society: selected essays of Richard M. Stallman. http://www.gnu.org/philosophy/fsfs/rms-essays.pdf (Accessed May 2009).

Appendix

BNF: FOL Clauses Language

<CLAUSE> ::= <CLAUSE> v <LITERAL> | <LITERAL>
<LITERAL> ::= ~<ATOMICFORMULA> | <ATOMICFORMULA>
<ATOMICFORMULA> ::= <literalsymbol> (<TERMLIST>)
<TERMLIST> ::= <TERMLIST>, <TERM> | <TERM>
<TERM> ::= <termsymbol> | <termsymbol> (<TERMLIST>)
<termsymbol> ::= a | b | c | ... | z
<literalsymbol> ::= A | B | C | ... | Z

Figure 1 – Context-free grammar for FOL Clauses Language

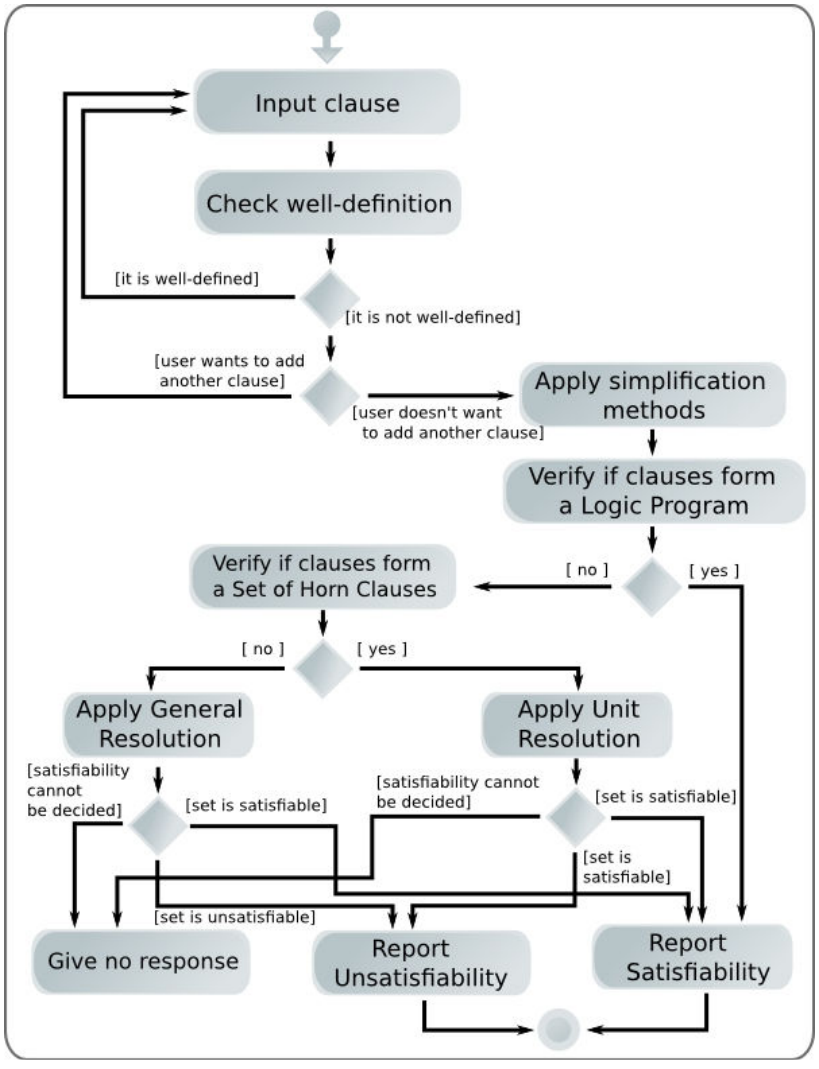

Figure 2 – Clausula Activity Diagram

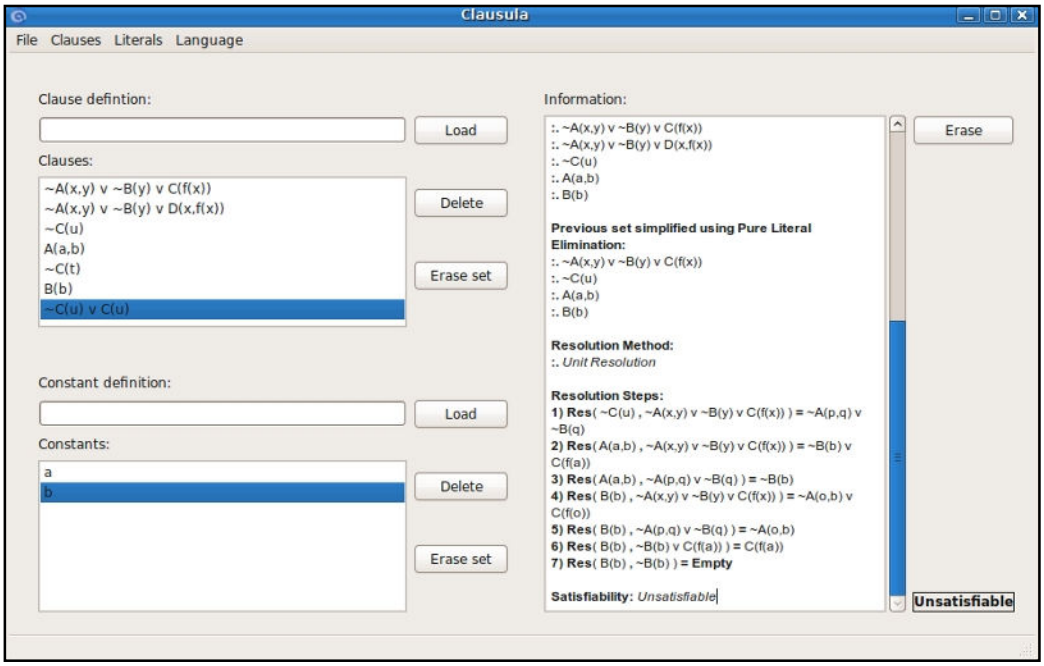

Figure 3 – General Resolution Method Application

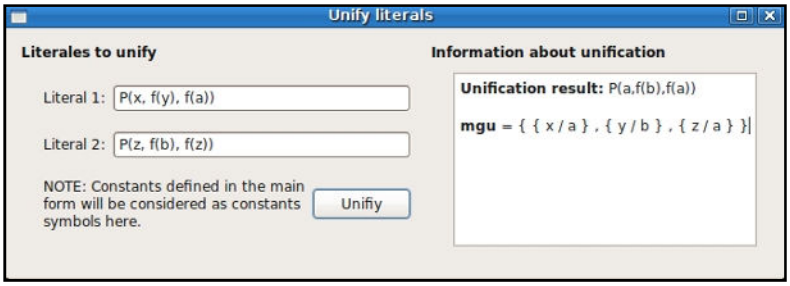

Figure 4 – Most General Unifier Computation

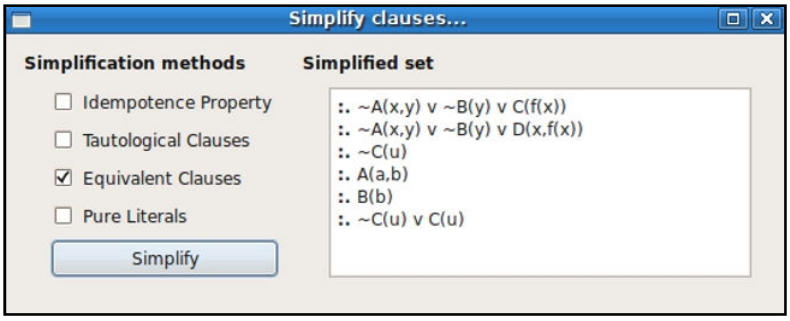

Figure 5 – Simplification by Equivalent Clauses

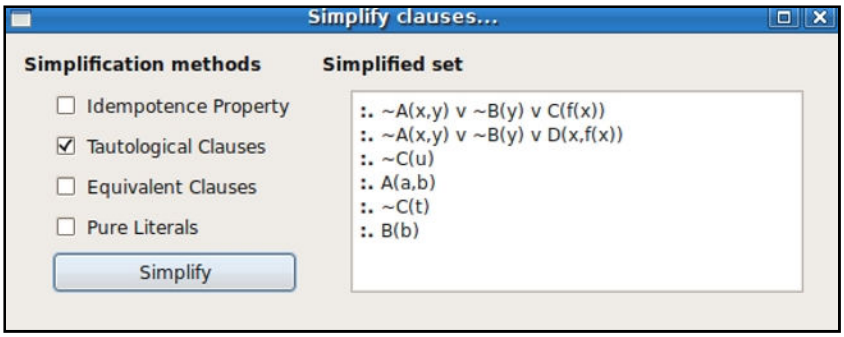

Figure 6 – Simplification by Tautological Clauses

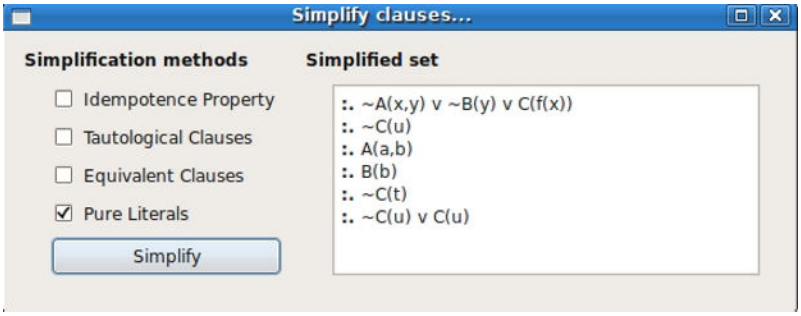

Figure 7 – Simplification by Pure Literals Strategy