SATOULOUSE: the computational power of propositional logic shown to beginners

Olivier Gasquet, François Schwarzentruber, and Martin Strecker

IRIT (Institut de Recherche en Informatique de Toulouse)
Université de Toulouse
Toulouse, France

Abstract. We present a tool (SATOULOUSE) that can help teachers, particularly in computer science, to convince undergraduate students that logic may be powerful. It is to be used very early in a logic course, in order to enhance students' motivation to learn propositional logic. SATOULOUSE simply consists of a friendly interface that offers several syntactic facilities and which is connected with a sufficiently powerful SAT-prover (namely SAT4J) allowing to automatically solve big instances of difficult problems (such as time-tables or Sudokus).

Key words: Teaching logic in computer science; SAT solvers; constraint solving

1 Introduction

The authors of this paper have been teaching logic to second-year undergraduate students in computer science for several years.

From our experience, most of our students are not sensitive to the "magic of logic", only a few are, and we found it could be useful to be helped in convincing them that logic is "at least" useful for computer scientists and that computer science does not only consist in hacking C-code or JAVA or PHP. But this has to be done at a time they do not know a lot of logic of course, the aim being to show them the "power of logic" in order to improve their motivation to study it.

Up to now, we used to motivate logic by abstract examples about its application in computer science (program verification, knowledge representation, planning, circuits design) and also by solving toy problems on a blackboard. But we began to think that it would be preferable to show and not only tell them that with only little knowledge, logic can be used to solve difficult problems whose size prevents humans from solving them by hand easily or would require rather complex programming. As we said, this has to be done very early during their course if motivation is in question. A SAT-solver can do that, but it turned out that none of the existing tools fitted our needs.

Of course, there are loads of logic tools (provers, proof assistants, truth table editors,...) on the Internet, even PROLOG could have been used, but none fits our requirements which are:

- the tool must be very easy to install and to use, with no complex syntax;

– the prover can be used as a black box without knowing how it works;
– no normal forming, ordering on clauses, or PROLOG cut must be needed;
– only little knowledge in logic should be necessary.

As we could not find an existing tool fulfilling these requirements, we started to implement ours, and we came to the idea of just developing an interface that allows to very comfortably use a powerful SAT-prover (namely SAT4J, details in section 3): the whole tool being called SATOULOUSE. With this tool, students can experiment by themselves that a logical language is not only descriptive but may lead to computations that solve real-life problems. In particular, with SATOULOUSE, they solve Sudokus very easily, as well as many other combinatorial problems (Time-table, Map coloring, Electronic circuits,...). Since they are also asked in their programming course to implement a Sudoku solver in C, they can compare both approaches with many advantages for the logical one. SATOULOUSE has a real impact on students: they can compare the time needed to implement a C-program that solves Sudokus with no real warranty about its correctness, with the time needed to encode Sudokus in propositional logic. This comparison is neatly in favor of the logical approach. Of course, we also give them contextual elements about research on the SAT problem: conferences, SAT-solvers competitions, and benchmarks... Our aim is to make our students understand that being a computer scientist does not only consist in hacking C-code, but may only require them to tackle problems more abstractly so that they can re-use existing and efficient tools.

Here are the main facilities that SATOULOUSE offers:

– Input formulas need not to be in clausal form and arbitrary connectives may be used, normal forming is done dynamically during keyboarding of the user;
– Big conjunctions and disjunctions facilities are offered like in:

$$\bigwedge_{i\in\{1..9\}} \bigvee_{j\in\{1..9\}} \bigwedge_{n\in\{1..9\}} \bigwedge_{m\in\{1..9\},m\neq n} (p_{i,j,n} \rightarrow \neg p_{i,j,m})$$

– Running the solver only consists in clicking a button;
– The tool displays a model in the syntax of the input formula.

Then it is possible to show the power of propositional logic to students that have been trained a couple of hours to formalize sentences in logic and have acquired basic notions of validity and satisfiability. First, as an exercise, they formalize the rules of Sudoku on paper, as well as data corresponding to the content of already filled cells. Then they are asked to run SATOULOUSE and to solve the Sudoku (in fact, formulas are available in a menu, so they don't have to type them in). But this is not the whole story, since the same SAT-solver may be used for solving many other combinatorial problems as easily as they just did for Sudokus: they just have to formalize the constraints. Our students are asked to do so for: time-table, map coloring,... The use of SATOULOUSE is very recent and has concerned about 80 students, but it seems that for quite many of them SATOULOUSE had a real impact on the representation they had of logic, in the

sense that they were more numerous than usual to say that they believe logic to be useful for computer scientists, and it seems to us that they got better marks than usual when passing their exam. But we need to find out how to evaluate this more objectively.

SATOULOUSE is publicly available for download from the following site

`http://www.irit.fr/satoulouse/`

2 Interaction with SAToulouse

Launching SATOULOUSE displays a formula editor that allows to enter a set of formulae to be tested for satisfiability. Formulae can either be hand-written in a Lisp-like syntax, or introduced in a sort of syntax-directed editor, by progressively refining the syntax tree.

The formulae of SATOULOUSE are the traditional formulae of propositional logic, plus indexed conjunction and disjunction, which are of the forms:

- $\bigwedge_{i \in E} P_i$, where E is an enumeration of natural numbers.
  For example, $\bigwedge_{i \in \{1,2,3\}} P_i$ represents the conjunction $P_1 \wedge P_2 \wedge P_3$. It is written `(bigand j (1 2 3) (P i))` in textual syntax.
- $\bigwedge_{i \in E} P_i$, where E is a range of natural numbers (for example `(bigand i (1 .. 9) (P i))` in textual syntax).
- $\bigwedge_{i \in E | i \neq j} P_{i,j}$, where E is as above and $i \neq j$ is an inequality constraint (for example `(bigand i (1 .. 9) (diff i j) (P i j))` in textual syntax).

Corresponding indexed connectors exist for disjunction. On writing a formula in textual syntax, SATOULOUSE immediately renders it in a LaTeX-style.

2.1 Be familiar with the notion of satisfiability

During our course, we give to the students easy exercises of formalization like:

"A robbery happened last night. Inspector Lestrade is on the premises. The thief has been seen: he is small and he left footprints of size 40[1]. There are three suspects: A, B or C. A is tall and takes size 43 shoes. B is small and takes size 40 shoes. C is small and takes size 43 shoes. Who is the thief?" (NB: they will have to elicit the fact that no one who takes size 43 wears 40).

SATOULOUSE is used to enter such formalization into the computer and solve this kind of problems in order to accustom students with it, but such examples can be treated by hand. Hence, we consider far bigger problems...

2.2 Logic programming

Let's take a look at an example, the coding of a particularly simple form of Sudoku (9×9 cells) for the sake of clarity. A screenshot is shown in Figure 1.

[1] French size...

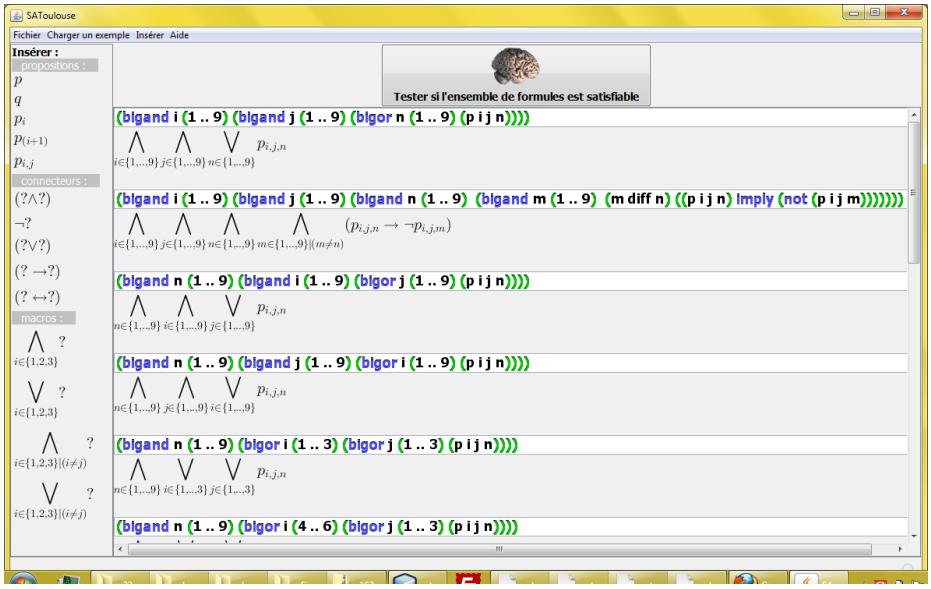

Fig. 1. Input of the Sudoku 9×9 problem in SATOULOUSE

Remember that in a Sudoku, the player is requested to fill in an 9×9 grid with numbers ranging from 1 to 9, such that some conditions are satisfied. To express the fact that at position (i, j) on the grid, there is a unique number 9, we use a number of constraints, for example (see screenshot):

- In each cell at position i, j there is at least one $n \in \{1, \ldots, 9\}$
- each cell (i, j) contains at most one n
- Each line and column contains each n of $\{1, \ldots, 9\}$ exactly once
- The game starts with already filled cells, this initial setting being described by a conjunction like $p_{1,2,3} \wedge p_{1,6,1} \wedge p_{2,3,6} \wedge \ldots$ (there is 3 in cell (1, 2), there is 1 in cell (1, 6), there is 6 in cell (2, 3), etc.)

Pressing the "brain" button transforms these formulae into the input format required by the SAT solver, and passes it the transformed input formula. SATOULOUSE then reports an outcome: Either the formula is satisfiable and then SATOULOUSE returns a validating assignment. Or SATOULOUSE returns a warning saying that the formula is unsatisfiable. For our example, SATOULOUSE returns the valuation depicted on Figure 2.

There are several other problems which can be easily implemented into SATOULOUSE (some of them are available in the menus). For instance:

Map coloring; Cryptograms;
Planning; 8 queens;
Pentominoes; Failure in a electric circuit;
Configuration of an avionics network, etc.

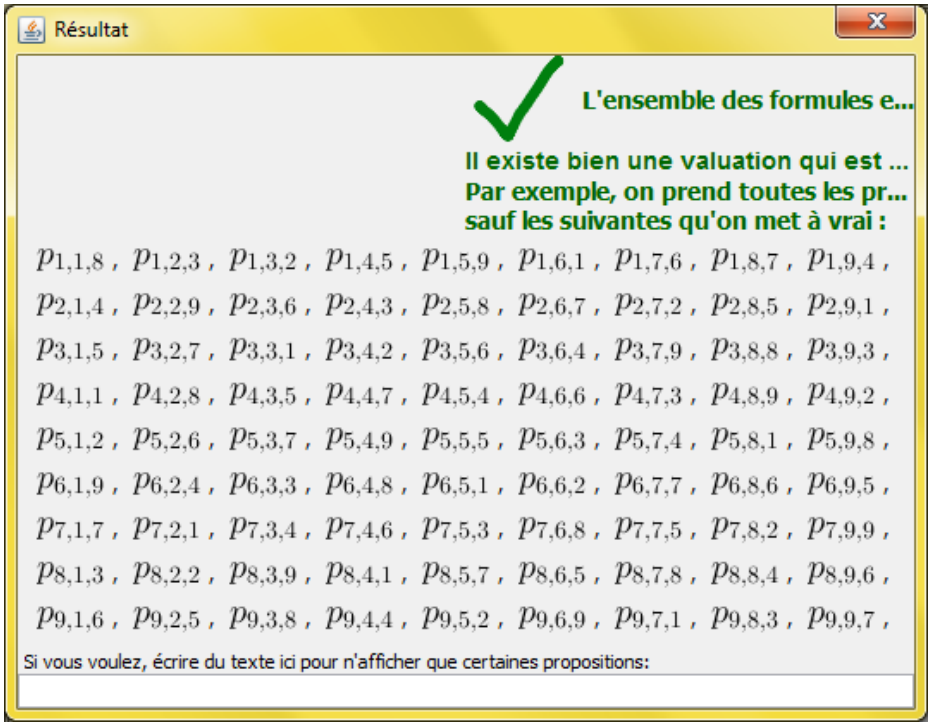

Fig. 2. Example of a valuation returned for the Sudoku problem of Figure 1

2.3 An application to genetics

The classical examples of the previous section are closely related to computer science. We want also our students to be open-minded and to be conscious that computer science, and especially logic, has applications in other domain, for instance in genetics.

A classical problem problem in genetics is the problem of haplotype inference by pure parsimony (HIPP) [3][4]:

- input: n genotypes, that is to say n sequences of the form $\begin{pmatrix} hz_{i,1} & \dots & hz_{i,m} \\ g_{i,1} & & g_{i,m} \end{pmatrix}$ of m sites which can be either homozygous ($hz_{i,j}$) or heterozygous ($\neg hz_{i,j}$). In the case where the site is homozygous we specify whether the site is wild ($g_{i,j}$) or mutant ($\neg g_{i,j}$) where i ranges over $\{1, \dots, n\}$ and j ranges over $\{1 \dots, m\}$;
- output: the smaller set of haplotypes that explains the genotypes. More precisely we have to find r sequences of the form $(h_{k,1} \dots h_{k,m})$ such that for all $i \in \{1, \dots n\}$ there exists $k, l \in \{1, \dots r\}$ such that:
  - if $hz_{i,j}$, then $h_{k,j} = h_{l,j} = g_{i,j}$;
  - if $\neg hz_{i,j}$ then $h_{k,j} \neq h_{l,j}$.

2.4 Highlighting the difficulty of SAT

Students may think that SAT is an easy problem: indeed the syntax and the semantics is so simple that it is surprising that this problem is difficult. In our course, we try to convince our students that the SAT problem is difficult in terms of complexity: indeed it is NP-complete [5]. In order to make our students aware of this difficulty, we show them that the current research in this area is prolific. We show them some big formulas of some benchmarks like on the Web site `http://www.satlib.org/` and treat them in SATOULOUSE.

3 Behind the scenes

The piece of software SATOULOUSE is an open-source project globally written in JAVA so that it can be used on many different platforms.

3.1 The graphical user interface

The graphical user interface (GUI) enables the user to write formulas and to click on the button "check if the set of formulas is satisfiable". The GUI is written in JAVA in order to be able to use the suitable library SWING. In particular, as this library is object oriented, it was really simple to create our own widget to enter a formula of propositional logic. Formulas are also displayed in LaTeX style thanks to the library jLatexMath[2] which does not require any additional software (in particular it does not require a real LaTeX installation).

3.2 The SAT solver

The SAT solver used as backend of SATOULOUSE is SAT4J [3]. The first advantage of this solver is that it is written in JAVA so it is easy to call it from the GUI. The second one is its good efficiency [1], and even if it is not the best SAT solver in the world (see SAT Race 2008), it is one of the best.

3.3 Connection between the SAT solver and the graphical user interface

The translation from formulas written by the user to formulas for the SAT solver SAT4J is divided in two parts:

- expanding formulas, that is to say, rewrite macros into expanded conjunction normal forms. For instance, we expand the formula $\bigwedge_{i\in\{1..4\}} p_i$ into $(((p_1 \wedge p_2) \wedge p_3) \wedge p_4)$. Expanding formulas is done in Scheme code intepreted with the JAVA library kawa [2]. Scheme is a suitable language to perform this translation because parsing of formulas is directly done in this language: formulas are directly s-expressions.

[2] `http://forge.scilab.org/index.php/p/jlatexmath/`
[3] `http://www.sat4j.org/`

– translating the expanded conjunctive normal forms into a rigorous SAT4J input. This part is done in Java because we need to run SAT4J which is coded in JAVA. We construct a map between the propositions of the GUI (strings) and strict positive integers used by SAT4J and then build corresponding arrays of integer for SAT4J.

3.4 Loading and saving a problem

As we can give explicit names to propositions (and not only integers as for SAT4J), as we have macros (like $\bigwedge_{i\in\{1..4\}} switch_i$), SATOULOUSE has its own file format: we simply write s-expressions representing formulas. Nevertheless, SATOULOUSE is also able to load standard DIMACS cnf format files as depicted below.

```
c A sample .cnf file.
p cnf 3 2
1 -3 0
2 3 -1 0
1 2 -3 0
```

$((p_1 \vee \neg p_3)\wedge$
$(p_2 \vee p_3 \vee \neg p_1)\wedge$
$(p_1 \vee p_2 \vee \neg p_3))$s

The standard DIMACS cnf format

4 Evaluation and further work

As far as we are aware, there is no other tool targeted at the same public as SATOULOUSE, since existing pedagogical tools (either implementation of truth-tables or semantic tableaux) that could do the job of searching a model cannot efficiently handle big problems, and real tools able to deal with them are definitely not designed to be used by beginners in logic.

The tool which comes closest to SATOULOUSE is Limboole[4], which also has the purpose of being used in class and is based on standard propositional logic (and not only clausal) input format. However, it does not offer the quantifiers over finite domains (indexed conjunction / disjunction) that allow for a concise statement of problems over structured domains such as Sudokus, map coloring etc. On the other end of the spectrum, there are highly developed input languages like of the Chaff [5] solver, which have a much too steep learning curve for 2nd year students.

The planning and configuration problems mentioned above can also be solved by constraint programming languages like Mozart [6], which often use dedicated solvers. The Alloy language and tool [7] allow to describe configuration problems in an "object-oriented" style and to state desired properties in an OCL-like syntax, which are then translated to SAT problems. These tools have clear merits

[4] http://fmv.jku.at/limboole
[5] http://www.cs.nyu.edu/acsys/cvc3/
[6] http://www.mozart-oz.org
[7] http://alloy.mit.edu/

for professional computer scientists, but, on the downside, have a steep learning curve and are therefore less appropriate for beginning computer science students.

We plan, in the future, to ask students to solve other problems that can be expressed in propositional logic, but to this end, we need to enrich the language of SATOULOUSE used in big conjunctions. Improvements we intend to introduce in SATOULOUSE are:

- Optimization of the translation of formulas into the internal format of SAT4J (by compiling them dynamically);
- Enriching the language of SATOULOUSE, e.g. by adding cardinality conditions for use in big conjunctions and disjunctions. For instance `(exact 3 i (1 .. 40)` ϕ_i`)` (`(atmost 3 i (1 .. 40)` ϕ_i`)`, `(atleast 3 i (1 .. 40)` ϕ_i`)`) for specifying that exactly (at most, at least) three formulas of the ϕ_i are true.

References

1. Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
2. Per Bothner. Kawa: compiling dynamic languages to the java vm. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '98, 1998.
3. D. Gusfield and S. Orzach. Handbook on Computational Molecular Biology, volume 9 of, 2005.
4. J. Marques-Silva. Practical applications of boolean satisfiability. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 74–80. IEEE, 2008.
5. C. H. Papadimitriou. *Computational complexity.* Addison Wesley, 1994.