TVT: A SOFTWARE VERIFICATION PACKAGE FOR THE INTERACTIVE LEARNING OF FORMAL PROGRAMMING TECHNIQUES
An Educational Experience

Rafael del Vado Vírseda, Fernando Pérez Morente and Eduardo Berbis González
Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Madrid, Spain

Keywords: Software Tools, Semantic Tableaux, Verification, Debugging, Invariants.

Abstract: While Computational Logic plays an important role in several areas of Computer Science (CS), most educational software developed for teaching logic is not suitable to be used directly in other portions of the CS education domain where the application of logical notions is usually required. In this paper we describe the logic teaching tool *TVT* based on semantic tableaux that has been developed to help the students to use logic as a formal proof technique in other advanced topics of CS, such as the verification of algorithms, the algorithmic debugging of imperative programs, the formal design of invariants, and the design and derivation of algorithms from logical specifications, which are at the basis of the formal learning of programming techniques and good development of software. We present the design, implementation, and results of the evaluation of this tool by means of several educational experiences during the academic courses 2009/2010 and 2010/2011. From the results of these experiences we conclude that the use of the *TVT* tool in the current CS teaching can help our students to understand more advanced CS concepts and to clarify the formal process involved in the design and analysis of correct and efficient imperative programs.

1 INTRODUCTION

Computational Logic is a subject that is taught in the first courses of almost all the Computer Science (CS) universities around the world. The syllabus of the course usually includes the syntax and semantics of propositional and first-order logic, as well as some formal proof techniques such as natural deduction, resolution, or semantic tableaux (Fitting, 1990). In some cases, there are some lectures devoted to explain basic concepts on logic programming and practical work using a *Prolog* interpreter. However, while Computational Logic plays an important role in several areas of CS, most of the educational software developed for teaching logic ignores their possible application in a larger number of subjects of the CS education domain. Many educational tools in this area (e.g., logic inference assistants and proof visualizers) have been developed with different degrees of success, and its utility has been proved by means of several educational experiments and publications. An actual and extensive collection of them can be found at http://www.ucalgary.ca/aslcle/logic-courseware and (Huertas, 2011). Unfortunately, although advanced logical notions are applied in superior courses, this educational logic software is not suitable for being used in these subjects.

The aim of this work is to describe an innovative methodology based on the logic teaching tool *TVT* that uses semantic tableaux to visualize formal proofs on advanced topics in CS, such as the design of correct and efficient algorithms from logical specifications. A *semantic tableau* (Fitting, 1990) is a semantic but systematic method of finding a model for a given set of formulas Γ, usually classified as a refutation system because a theorem φ is proved from Γ by getting its negation $\Gamma \Vdash \neg\varphi$. Our major contribution is the implementation of new tableau methods that provide semantically reach feedback to the students in order to help them to understand the formal reasonings performed in the process of verifying the correctness of algorithms. Moreover, it allows to perform an algorithmic debugging of programs following a classical idea from Shapiro (Shapiro, 1983), that proposes to replace computation traces by computation trees with program fragments attached to their nodes in the debugging process. As novelty in this work, we propose to use semantic tableaux as computation trees to

DOI: 10.5220/0003894400770082
In *Proceedings of the 4th International Conference on Computer Supported Education* (CSEDU-2012), pages 77-82
ISBN: 978-989-8565-07-5

show the students that they can reason on the results of the execution of a program only considering the meaning of the program itself and ignoring complex operational details.

We have tested the *TVT* tool through the academic courses 2009/ 2010 and 2010/2011, performing some educational experiments to estimate the benefits for our students of using the tool as a complement to scheduled regular classes. This evaluation has been carried out by means of several tests, some of them managed in an online platform with open access to the students, and the other ones in a CS laboratory with a controlled group. We show the results of these educational experiments and the benefits of using the *TVT* tool in the teaching of advanced CS concepts involving the formal verification and the algorithmic debugging of imperative programs. We believe that these educational experiences prove that our implementation based on tableaux provides an excellent training to the students in the practical application of advanced logic concepts to perform different CS tasks.

2 THE TVT TOOL

Solving logical exercises is usually done with pen and paper, but educational tools can offer more useful pedagogical possibilities. The role of this educational software is to facilitate the student's grasp of the target procedures of education, and to provide teamwork and communication between teachers and students (van ditmarsch, 2005).

Our *Tableaux Verification Tool*, named *TVT* (see the current version at gpd.sip.ucm.es/CSEDU2012/TVT.zip), is an educational application based on first-order semantic tableaux with equality and unification (Fitting, 1990) used as a support for the teaching of deductive reasoning at an elementary university level for Computer Science students. The tool helps our students to learn how to build semantic tableaux, and to understand the philosophy of this proof device using it not only to establish consistency/inconsistency or to draw conclusions from a given set of premises, but also for verification and debugging purposes as we propose in this paper. Our first year students have learnt tableau calculus in the classroom and this software has helped them to understand advanced CS concepts visualizing and producing their own proof trees.

The tool consists of two main parts: one that produces first-order tableaux, and another one based on this tableaux methodology for verification and debugging of algorithms. In both cases, the application possesses a drawing window where trees will be graphically displayed. The major functional interface of the *TVT* tool is shown in Figure 1. The user interacts with the prover through this graphical interface. In the following sections we describe the use of the tool and their main features by means of a running example.

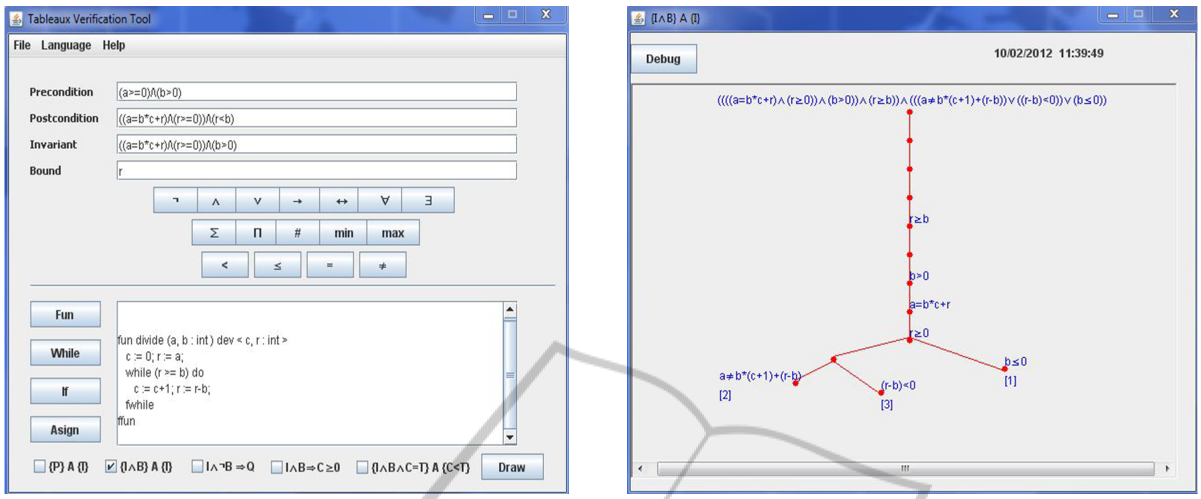
3 FORMAL VERIFICATION

The main novelty of the *TVT* tool is to train our students in the art and science of specifying correctness properties of algorithms and proving them correct. For this purpose, we use the classical approach developed by Edsger W. Dijkstra and others during the 1970s (Dijkstra, 1976). We use a guarded command language to denote our algorithms A, represented by functions `fun` A `ffun` that may contain variables (x, y, z, etc.), value expressions (e) and boolean expressions (B). The code of an algorithm is built out of the skip (`skip`) and assignment statements ($x := e$) using sequential composition ($S_1 ; S_2$), conditional branching (`if` B `then` S_1 `else` S_2 `fif`), and `while`-loops (`while` B `do` S `fwhile`). This language is quite modest but rich enough to represent sequential algorithms in a succinct and elegant way. As an illustrative example, we consider a simple algorithm *divide* to compute the positive integer (*int*) division between a and b with quotient c and remainder r (represented in *TVT* on the left and bottom of Figure 1):

```
fun $divide\ (a, b : int)$ dev $< c, r : int >$
   $c := 0;\ \ r := a;$
  while $r \geq b$ do
          $c := c+1;\ \ r := r-b$
  fwhile
ffun
```

It becomes obvious that neither tracing nor testing can guarantee the absence of errors in algorithms. To be sure of the correctness of an algorithm one has to prove that it meets its *specification* (Dijkstra, 1976). A specification of an algorithm A consists of the definition of a *state* space (a set of program variables), a *precondition* P and a *postcondition* Q (both predicates expressing properties of the values of variables), denoted as $\{P\}\ A\ \{Q\}$. Such a triple means that Q holds in any state reached by executing A from an initial state in which P holds. For example, a formal specification for the *divide* function (represented in *TVT* on the left and top of Figure 1) is:

```
$\{P\ :\ a \geq 0 \wedge b > 0\}$
  fun $divide\ (a, b : int)$ dev $< c, r : int >$
$\{Q\ :\ a = b*c+r \wedge r \geq 0 \wedge r < b\}$
```



Precondition (a>=0)/\(b>0)
Postcondition ((a=b*c+r)/\(r>=0))/\(r<b)
Invariant ((a=b*c+r)/\(r>=0))/\(b>0)
Bound r

```
fun divide (a, b : int) dev < c, r : int >
  c := 0; r := a;
  while (r >= b) do
    c := c+1; r := r-b;
  fwhile
ffun
```

☐ {P} A {I} ☑ {I∧B} A {I} ☐ I∧¬B ⇒Q ☐ I∧B⇒C≥0 ☐ {I∧B∧C=T} A {C<T}

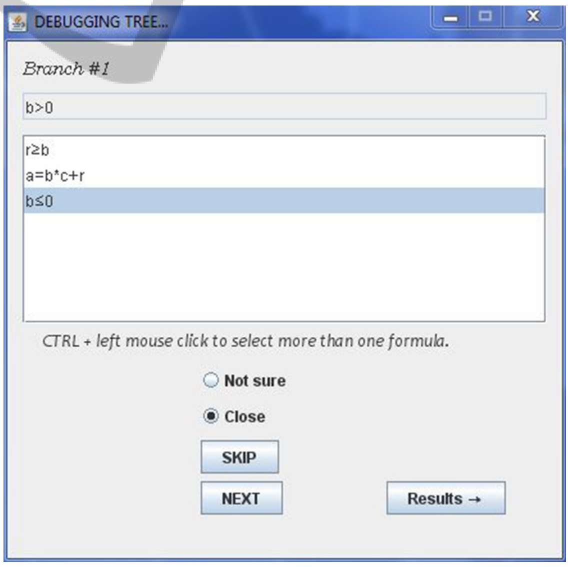
Figure 1: The *TVT* tool for the formal learning of programmming techniques.

An algorithm together with its specification is viewed as a theorem. The theorem expresses that the program satisfies the specification. Hence, all algorithms require proofs (as theorems do). Following (Kaldewaij, 1990), the verification is based on a *loop invariant* I supplied by the designer or by some invariant-finding tool (in our example $a = b*c+r \wedge r \geq 0 \wedge b > 0$), a *bound function* C for termination (in our example r), and the following five proofs (see again Figure 1):

- $\{P\}\ c := 0; r := a\ \{I\}$.
- $\{I \wedge r \geq b\}\ c := c+1; r := r-b\ \{I\}$.
- $I \wedge r < b \Rightarrow Q$.
- $I \wedge r \geq b \Rightarrow C \geq 0$.
- $\{I \wedge r \geq b \wedge C = T\}\ c := c+1; r := r-b\ \{C < T\}$.

The *TVT* tool verifies algorithms A according to their specification $\{P\}\ A\ \{Q\}$ in a constructive way based on semantic tableaux $P \Vdash \neg wp(A,Q)$, where $\neg wp(A,Q)$ is the negation of the *weakest precondition* of A with respect to Q, which is the 'weakest' predicate that ensures that if a state satisfies it then after executing A the predicate Q holds (see (Kaldewaij, 1990) for more details). For example, we have the following tableau proof (graphically displayed by the *TVT* tool in the right side of Figure 1) to verify the proof corresponding to the preservation of the invariant I in the body of the loop: $\{I \wedge r \geq b\}$ $c := c+1; r := r-b\ \{I\} \Leftrightarrow I \wedge r \geq b \Vdash \neg wp(c := c+1; r := r-b, I) \Leftrightarrow I \wedge r \geq b \Vdash \neg(I^{c+1,r-b}_{c,r})$, where $I^{c+1,r-b}_{c,r}$ represents the predicate I in which c and r are replaced by $c+1$ and $r-b$, respectively (see the root of the tree in Fig. 1).

To verify and check that this tableau is closed, the student must press the "*Debug*" button. The tool then begins to traverse the tree and tries to locate, for each of its three branches, a pair of atomic predicates that are contradictory with the interactive help of the student. Beginning with the first branch, the student finds that $b > 0$ and $b \leq 0$ are contradictory and marks the branch as "*Close*":



Branch #1

b>0

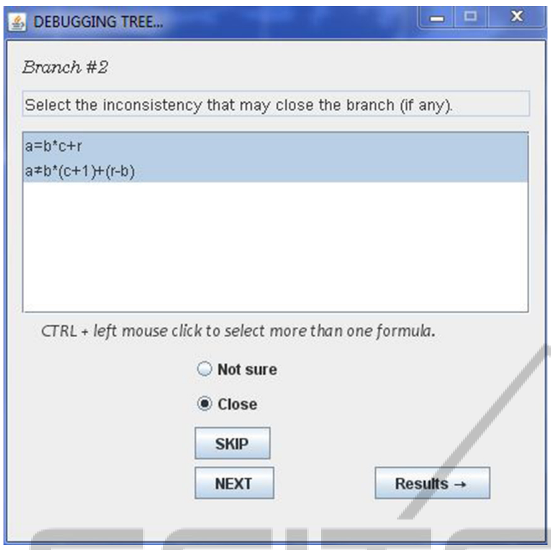
r≥b
a=b*c+r
b≤0

CTRL + left mouse click to select more than one formula.

○ Not sure
◉ Close

Figure 2.

After pressing the button "*Next*", the tool finds in the second branch two possible atomic predicates that are contradictory: $a = b*c+r$ and $a \neq b*(c+1)+(r-b)$. The student marks the option "*Close*":



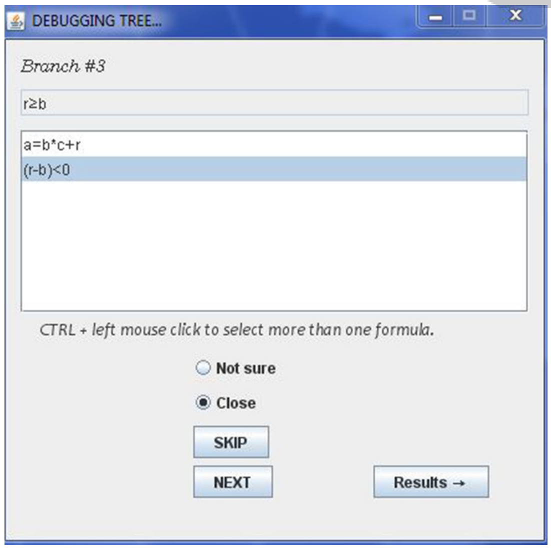
Select the inconsistency that may close the branch (if any).

a=b*c+r
a≠b*(c+1)+(r-b)

CTRL + left mouse click to select more than one formula.

Figure 3.

The tool analyzes the third branch and finds two atomic predicates that are contradictory: $r \geq b$ and $(r-b) < 0$. So, the student marks again the option "*Close*":



r≥b

a=b*c+r
(r-b)<0

CTRL + left mouse click to select more than one formula.

Figure 4.

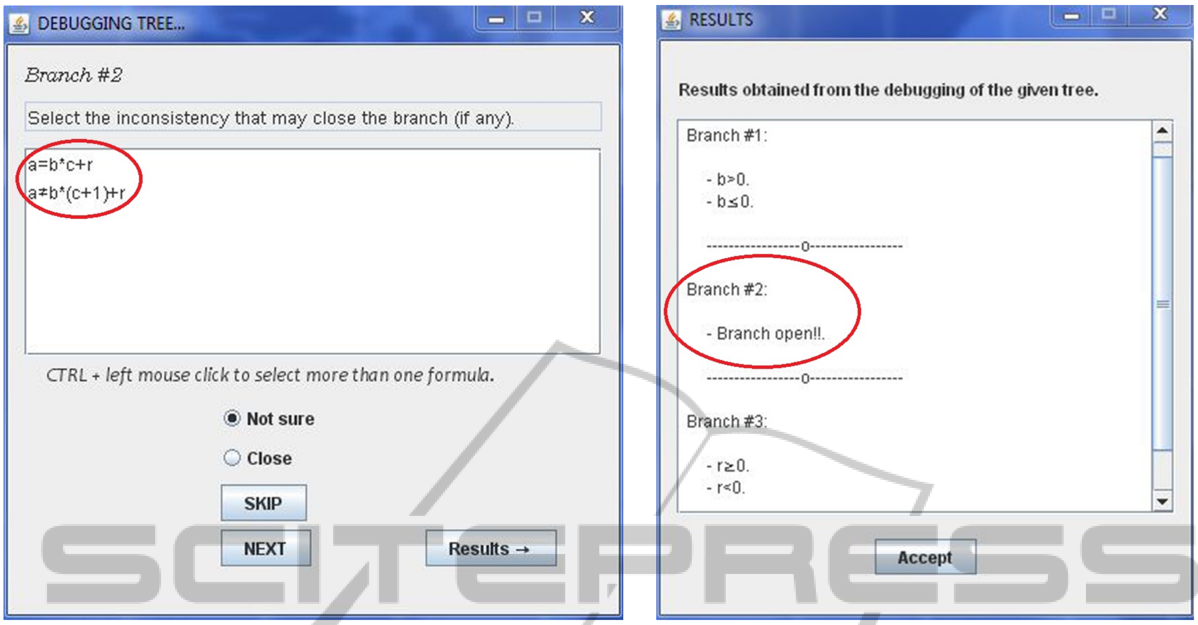
Finally, after pressing the button "Results", the tool summarizes the conflicting atomic predicates that have found for each of the tableau's branches. The formal verification session has finished.

4 ALGORITHMIC DEBUGGING

Debugging is one of the essentials phases of the software development cycle and a practical need for helping our students to understand why their programs do not work as intended. In this section we apply the ideas of *algorithmic debugging* (Naish, 1997) as an alternative to conventional approaches to debugging for imperative programs. The major advantage of algorithmic debugging compared to conventional debugging is that allows our students to work on a higher level of abstraction. In particular, we have successfully applied our *TVT* tool based on semantic tableaux for the algorithmic debugging of simple programs to show how one can reason about such programs without operational arguments. Following a seminal idea from Shapiro (Shapiro, 1983), algorithmic debugging proposes to replace computation traces by *computation trees* with program fragments attached to the nodes. As novelty, in this work we propose to use semantic tableaux as computation trees.

As an example, we alter the code of the previous algorithm with a simple mistake: we omit the instruction $r := r - b$. If we try to verify this erroneous algorithm, we can execute again the *TVT* tool. If the student presses the "*Debug*" button to start a debugging session on the proof tree of Figure 1, the *TVT* tool tries to close automatically all the possible branches from the stored information entered in the previous session. However, since it is not possible to close the second branch, the tool shows to the student the predicates $a = b*c+r$ and $a \neq b*(c+1)+r$ as possible opposite atoms that should be closed to close the branch and the proof tree (see the left side of Figure 5). The user checks that $a = b*c+r$ and $a \neq b*c+(r+b)$ are not necessarily contradictory for any value of the divisor b. Therefore, the student marks the option "*Not sure*", and the tool confirms that this branch cannot be finally closed and the proof tree remains open (see the right side of Figure 5).

What failed to close this branch and how it could be solved? If the student compares again the atomic predicates previously showed by the tool, $a = b*c+r$ and $a \neq b*c+(r+b)$, easily concludes that the expression $r+b$ should be r for every b. But, how can this be done? Only if the code of the algorithm executes the instruction $r := r-b$. By examining the code, the student discovers that this assignment has been omitted, and introduces this piece of code into the body of the loop, because the tool is actually examining the proof tree $\{I \wedge B\}\ A\ \{I\}$ corresponding to this part of the code. After running again the *TVT* tool, the student sees that all the proof trees remain closed. Then, the algorithmic debugging session finishes.



Branch #2

Select the inconsistency that may close the branch (if any).

a=b*c+r
a≠b*(c+1)+r

CTRL + left mouse click to select more than one formula.

(● Not sure) ○ Close

Results obtained from the debugging of the given tree.

Branch #1:

- b>0.
- b≤0.

----------------0----------------

Branch #2:

- Branch open!!.

----------------0----------------

Branch #3:

- r≥0.
- r<0.

Figure 5: Algorithmic debugging with the *TVT* tool.

5 EXPERIENCES AND RESULTS

The educational tool *TVT* is available for the students of the topics *Computational Logic* and *Design of Algorithms* in the Computer Science Faculty of our University through an online educational platform called *Virtual Campus*. The following results are based on the statistics from the 186 students who took the course in 2009/2010 and 2010/2011.

5.1 Design of the Experiences

We have carried out two educational experiences:

- One **non-controlled** experience: All the students may access the Virtual Campus and participate freely in the experience: download and use the *TVT* tool, and answer different tests.
- One **controlled** experience: Two groups of students must answer a test limited in time and access to material.

With respect to the **non-controlled** experience, the students may freely access the Virtual Campus without any restriction of time or material (slides, bibliography, and the tool) and answer the questions of several tests. For each of the explained topics in Computer Science we have provided a test that evaluates the knowledge of our students applying different kinds of semantic tableaux. The students may use these tests to verify their understanding of the different concepts. The questions are structured in three blocks: *propositional and predicate logic*, *specification and verification* of algorithms, and *debugging and derivation* of imperative programs. The resolution of the tests by the students is controlled by the Virtual Campus with the help of an interactive tutoring system. In the **controlled** experience we try to evaluate more objectively the usefulness of the tool. In particular we have chosen the application of *TVT* for the verification and debugging of simple searching and sorting algorithms (Kaldewaij, 1990). We have chosen two groups of students answering the same questions: approximately half of the students works only with the slides of the course and the books at class; and the other half works only with the tool at a Computer Laboratory.

5.2 Results

5.2.1 Non-controlled Experience

We outline here the main conclusions from the results of the **non-controlled** experience. With respect to the material the students used to study, as long as the exercises were more complicated the use of the tool (simulations, cases execution, and tool help) increased considerably. Better results were obtained in the verification and debugging of searching and sorting problems (linear and binary search, insertion and

	correct		errors		don't knows	
	mean	σ	mean	σ	mean	σ
slides/books	9.36	2.35	6.23	2.37	3.21	2.82
TVT tool	12.77	3.71	4.81	2.10	1.22	1.73

Figure 6: Means and standard deviations (σ).

selection sort). The tool helped our students to visualize array manipulations in array assignments. In the rest of the algorithms (slope search and advanced sorting algorithms) they used only the class material or bibliography. When answering the tests questions, the students were also asked whether they needed additional help to answer them. In the case of linear and binary search they used the tool as much as the class material, which means that visualization of their own proof tableaux were a useful educational complement. We can conclude that our students consider the tool as an interesting resource and have used it to complement the rest of the available material.

5.2.2 Controlled Experience

The **controlled** experience was carried out with 59 students. We gave 32 of them only the slides of the course and the books of the bibliography (Fitting, 1990; Kaldewaij, 1990). The rest were taken to a Computer Laboratory, where they could execute the *TVT* tool. We gave the same test to both groups, consisting of 18 questions, 12 of them on specification aspects of the algorithms (inference of invariants and bound functions), and the rest on their verification and debugging from the code. In Figure 6 we provide the means and the standard deviations of correct, errors, and *don't knows* answers. First, we observe that students using *TVT* answer more questions than the other ones. In addition, they make less errors than the others. This is due to the fact that most of the students of the *TVT tool* group performs the analysis of the algorithms directly from the corresponding semantic tableau displayed by the tool, while the *slides/book* group has to hardly deduce it directly from the code. All the students who used *TVT* indicated the benefits of using tableaux to understand the code of the algorithms from their specifications. Therefore, we can conclude that the methodology proposed in this work constitutes a good complement to facilitate the comprehension of the design and analysis of imperative programs. In addition, the methodology based on tableaux has helped us to detect in the students difficulties applying the formal techniques to derive correct and efficient imperative programs from specifications.

6 CONCLUSIONS

We have presented the educational and interactive tool *TVT* based on semantic tableaux for a specification language on predicate logic to perform verification and debugging of algorithms. This is a first step towards the development of a practical teaching technology for the formal learning of programming techniques.

We have systematically evaluated the proposed methodology to confirm that *TVT* is a good complement to both the class explanations and material, making easier the visualization of proofs in the reasoning needed for the design of correct and efficient programs. We look forward to making good use of what we have learned from this evaluation to improve tool's usefulness in Computer Science Education.

REFERENCES

Dijkstra, E. (1976). *A Discipline of Programming*. Prentice Hall.

Fitting, M. (1990). *First-Order Logic and Automated Theorem Proving*. Springer, Graduate Texts in Computer Science.

Huertas, A. (2011). Ten years of computer-based tutors for teaching logic 2000-2010: Lessons learned. In *TICTTL*.

Kaldewaij, A. (1990). *Programming: The Derivation of Algorithms*. Prentice-Hall International Series in Computer Science.

Naish, L. (1997). A declarative debugging scheme. *Journal of Functional and Logic Programming, vol. 3*.

Shapiro, E. (1983). *Algorithmic Program Debugging*. MIT.

van ditmarsch, H. (2005). *Logic software and logic education*.