A Tool for Experimenting with a Theorem Prover

Foteini Grivokostopoulou
School of Engineering Department of Computer Engineering & Informatics
University of Patras
6500 Patras, Hellas (Greece)
grivokwst@ceid.upatras.gr

Ioannis Hatzilygeroudis
School of Engineering Department of Computer Engineering & Informatics
University of Patras
6500 Patras, Hellas (Greece)
ihatz@ceid.upatras.gr

***Abstract*—In this paper, we presented EX-ACT-P, a tool for experimenting with a theorem prover. The heart of EX-ACT-P is ACT-P (A Configurable Theorem-Prover). ACT-P is a resolution-based theorem prover, which has a unique characteristic: allows the user to configure its resolution control regime. EX-ACT-P is an extension of ACT-P that allows user to experiment with theorem proving aspects, such as: configure a suitable resolution control regime, translate and automatically solve problems from the TPTP library, create and prove his/her own problems, display the proof steps in both text-based and graphical way and give information relevant to the proof process. So, EX-ACT-P can be useful to students for learning and to tutors for teaching aspects related to use of logic as a knowledge representation and reasoning language, by creating the right cognitive models, as well as to researchers for experimenting with theorem proving in ACT-P. A small scale evaluation for students has shown promising results.**

Keywords- Automated reasoning; Theorem prover; Resolution control strategies; TPTP library; Teaching logic assistant

I. INTRODUCTION

Logic is one of the fundamental topics taught in computer science and/or engineering departments. In most such departments, logic is taught as a means for constructing formal proofs in a natural deduction style. However, teaching logic, especially first-order logic (FOL), as a knowledge representation and reasoning (KR&R) vehicle is also basic in all introductory artificial intelligence (AI) courses. We have constructed some tools for helping students in learning and tutors in teaching logic as a KR&R language (e.g., tools for translating natural language sentences into FOL ones or for converting FOL sentences into clause form, etc) [6]. However, we haven't constructed a tool for assisting in learning or teaching automated logic-based reasoning. There have been tools assisting students in using logic as a natural deduction tool, usually called proof assistants or proof editors (e.g., like [1, 2, 3, 4, 7, 8]). Most of them give an emphasis on the user interface. Some of them are built on top of an interactive theorem prover, like Isabelle [11]. However, there are no tools for helping students in learning and teachers in teaching automated logic-based reasoning aspects. Even further, there are no tools for experimenting with controlling automated logic-based proof processes.

Theorem proving is a subfield of automated reasoning where logic is used as a KR&R vehicle. Resolution-based reasoning is a fundamental mechanism for logic-based reasoning, namely theorem proving. Many automated theorem provers (ATPs) are based on that mechanism. One of them based on first-order logic (FOL) is Prover9 [9]. So, such an ATP system could be the basis for a system assisting in teaching or generally in experimenting with automated logic-based reasoning.

We introduce here a tool for experimenting with logic-based reasoning via an ATP, namely ACT-P (A Configurable Theorem-Prover). ACT-P was initially introduced in [5] and is configurable in the sense that one can (re)define its resolution control regime. We call our tool EX-ACT-P (EXtended ACT-P), since it extends the interactive and presentation facilities of ACT-P.

The structure of the paper is as follows. Section II presents an overview of ACT-P. In Section III, EX-ACT-P is described. Section IV deals with using EX-ACT-P, whereas Section V briefly refers to system evaluation. Finally, Section VI concludes the paper.

II. ACT-P: AN OVERVIEW

ACT-P (A Configurable Theorem-Prover) is the heart of EX-ACT-P. It is a skeleton resolution-based theorem prover, where specific steps are programmable by the user. ACT-P is based on a meta-level architecture. This means that it has an object-level language and a meta-level language. Its object-level language is a classical FOL. Its syntax is based on Cambridge Polish notation, using as connectives {~, &, V, =>} instead of {¬, ∧, ∨, ⇒} respectively and as quantifiers {forall, exists} instead of {∀, ∃} respectively. For example, the following are two well-formed ACT-P expressions:

```
((forall ?x) ((exists ?y) (=> (& (hunter ?x) (animal ?y))
                              (kills ?x ?y))))
(~ ((exists ?x) (V (cat ?x) (~ (kills-mice ?x))))
```

 ISBN: 978-1-61208-108-3

which correspond to the following conventionally notated formulas:

$(\forall x)\ (\exists y)\ (\text{hunter}(x) \wedge \text{animal}(y)) \Rightarrow \text{kills}(x, y)$
$\neg\ (\exists x)\ (\text{cat}(x) \vee \neg\text{kills-mice}(x))$

In the object-level, ACT-P is a classical theorem prover. It can accept a set of FOL sentences (the axioms), which converts into clause form (CF) resulting in a set of clauses. Then it can accept a FOL sentence (the theorem) to be proved from the axioms. To that end, ACT-P negates it, converts it into its CF and tries to prove it using binary resolution refutation. It finally returns the solution (T or variable bindings) or 'no-solution'.

ACT-P internally represents its search space as an OR tree. Each branch of the tree represents a resolution step. Each node of the tree represents the updated set of clauses with the produced resolvent after execution of the incoming branch's step. We store clauses using the discrimination-tree indexing [10], which is an improvement to the version in [5]. This has significantly improved the efficiency of ACT-P.

The meta-level language of ACT-P is Common Lisp extended with a number of meta-primitives. Meta-primitives are predefined Lisp functions that implement a default resolution control regime and help users in implementing other resolution control strategies. That is, the user can redefine the bodies of the meta-primitives, which results in resolution control regime changes. Thus, a large number of different control strategies can be implemented.

III. EX-ACT-P

A. An Overview

We have extended ACT-P into EX-ACT-P (EXtended ACT-P) to be able to (a) experiment with and (b) practice with in a friendly manner. The first direction concerns tutors (even researchers), whereas the second concerns students. So, tutors/researchers can experiment with EX-ACT-P in order to test new possible exercises/problems related to KR&R or try new (combinations of) resolution strategies or test efficiency of them or even ACT-P itself. Students can practice by comparing their hand-made proofs with the ones provided by the system, see the differences when trying different strategies, study the steps of a proof etc. Students can practice with proofs that can't be made by hand, because they take many steps.

More specifically, one can

- Edit a problem in ACT-P FOL language
- Convert a TPTP library problem into ACT-P FOL language
- Determine different combinations of strategies
- Produce an automated proof
- See a linear text-based proof
- See a graphical representation of a proof
- See the number of produced clauses and the time required for a proof

B. System Architecture

The architecture of EX-ACT-P is illustrated in Fig. 1. It consists of six main units: the *user interface* (UI), the ACTRANS, the *problem editor* (PE), the ACT-P, the *problem collection* (PC) and the *control strategies pool* (CSP).

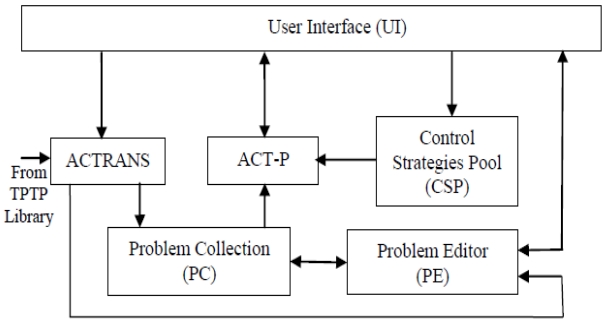

Figure 1. EX-ACT-P Architecture.

PE is used to code a problem in ACT-P FOL language. Then can be added to PC, where it is available for experimentation. ACTRANS (Automated Code TRANSformer) is used for transforming TPTP library problems [12] into ACT-P language. Then the problems can be directly put into PC or be edited first through PE and then stored into PC. From CSP the user can select a proper combination of resolution control strategies to determine the overall control regime of ACT-P.

ACT-P is used to produce the proof for a selected problem from PC. Finally, UI is the means of interacting with the rest units of the system and presenting proof results.

C. ACTRANS

ACTRANS (Automated Code TRANSformer) is a very useful tool of EX-ACT-P. It takes as input a TPTP problem file (e.g., puz006-1.p), through a direct link to TPTP, and creates two Lisp files. The first (PUZ006-1.lsp) contains whatever the TPTP file contains except that the problem formulas are converted into ACT-P language. The second (PUZ006-1PR.lsp) contains the query of the problem, i.e. the theorem to be proved.

Let see an example. Following are the axioms and the theorem representing problem "puz002-1" from Puzzle category in the TPTP library:

```
cnf(only_cats_in_house,axiom,
    ( ~ in_house(Cat)
    | cat(Cat) )).
cnf(gazers_are_suitable_pets,axiom,
    ( ~ gazer(Gazer)
    | suitable_pet(Gazer) )).
```

```
cnf(avoid_detested,axiom,
    ( ~ detested(Detested)
    | avoided(Detested) )).
cnf(carnivores_are_prowlers,axiom,
    ( ~ carnivore(Carnivore)
    | prowler(Carnivore) )).
cnf(cats_are_mice_killers,axiom,
    ( ~ cat(Cat)
    | mouse_killer(Cat) )).
cnf(in_house_if_takes_to_me,axiom,
    ( ~ takes_to_me(Taken_animal)
    | in_house(Taken_animal) )).
cnf(kangaroos_are_not_pets,axiom,
    ( ~ kangaroo(Kangaroo)
    | ~ suitable_pet(Kangaroo) )).
cnf(mouse_killers_are_carnivores,axiom
,
    ( ~ mouse_killer(Killer)
    | carnivore(Killer) )).
cnf(takes_to_me_or_detested,axiom,
    ( takes_to_me(Animal)
    | detested(Animal) )).
cnf(prowlers_are_gazers,axiom,
    ( ~ prowler(Prowler)
    | gazer(Prowler) )).
cnf(kangaroo_is_a_kangaroo,axiom,
    ( kangaroo(the_kangaroo) )).
cnf(avoid_kangaroo,negated_conjecture,
( ~ avoided(the_kangaroo) ))
```

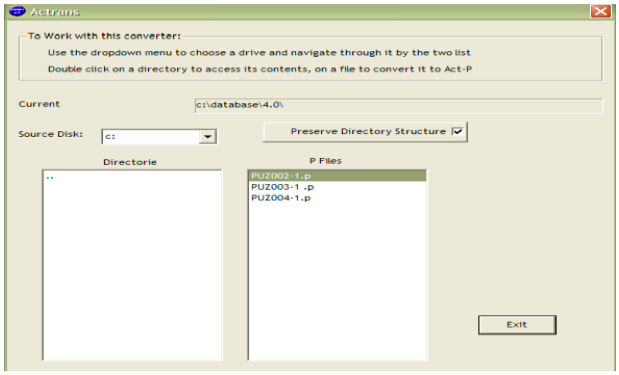

Figure 2. ACTRANS user interface.

After passing through ACTRANS, they are converted to the following ACT-P sentences:

```
(premise '(V (~(in_house ?Cat)) (cat
?Cat)))
(premise '(V (~(gazer ?Gazer))
(suitable_pet ?Gazer)))
(premise '(V (~(detested ?Detested))
(avoided ?Detested)))
(premise '(V (~(carnivore ?Carnivore))
(prowler ?Carnivore)))
(premise '(V (~(cat ?Cat)) (mouse_killer
?Cat)))
(premise'(V (~(takes_to_me ?Taken_animal))
(in_house ?Taken_animal)))
(premise '(V (~(kangaroo ?Kangaroo))
(~(suitable_pet ?Kangaroo))))
(premise '(V (~(mouse_killer ?Killer))
(carnivore ?Killer)))
(premise '(V (takes_to_me ?Animal)
(detested ?Animal)))
(premise '(V (~(prowler ?Prowler)) (gazer
?Prowler)))
(premise '(kangaroo the_kangaroo))
(setf query '(~(~(avoided the_kangaroo))))
```

All above sentences, except the last one, are included in the PUZ002-1.lsp file, while the last one is included in the PUZ002-1PR.lsp file. Figure 2 depicts the user interface of ACTRANS.

D. Control Strategies

We have categorized resolution control strategies according to the heuristics they use to reduce the search space and control resolution refutation process. That categorization helped us to design the reasoning cycle of ACT-P. We distinguish three major classes of heuristics. The first major class, *resolution restricting* strategies, concern generation of the search space and are used to increase efficiency by restricting the size of the search space. Resolution restricting strategies comprise two subtypes, namely *parent selection* strategies and *clause elimination* strategies. Parent selection strategies are based on the observation that not all possible resolvents have to be constructed to be able to derive the empty clause. They therefore impose restrictions on the clauses to be selected for resolution. Parent selection strategies have also been called 'refinement strategies' or 'restriction strategies'. The second type of resolution restricting strategies, clause elimination strategies, aim to eliminate clauses that will not be useful in further search. They are also called 'simplification strategies' or 'deletion strategies'.

The second main class of resolution control heuristics, *resolution search* strategies, concern the way the search space is searched. We distinguish between general search strategies and resolution ordering strategies. *General search* strategies traverse the search space in a blind way, without taking into account any resolution or domain or problem specific knowledge. Unlike general search strategies, *resolution ordering* strategies, aim to increase the efficiency of the theorem proving process by judiciously ordering potential resolutions. Clearly, such ordering strategies presuppose some ordering criterion. Best-first type strategies belong to this class.

Whereas resolution search strategies concern the way in which the search space is searched, our third main class of heuristics, *process oriented* strategies, concern the way in

 ISBN: 978-1-61208-108-3

which resolutions are performed. They are typically used in conjunction with a specific parent selection strategy. We draw a distinction between *resolving preparation* and *resolving operation* strategies. The former concern operations on clauses before the actual application of the resolution rule. Resolving operation heuristics are used during actual resolution.

We have implemented a number of strategies from each category and stored in CSP. A user can select an appropriate combination of strategies from some or all the categories, through UI, to determine the overall control regime of ACT-P. A parent selection strategy should be first selected, since it will be the core strategy.

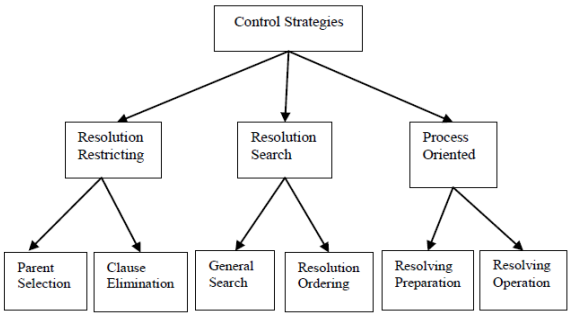

Figure 3. Classification of resolution control heuristics.

E. Implementation Issues

EX-ACT-P has been developed in Lispworks 4.4. The choice of Lispworks as the programming environment was made on the basis of its features, such as its support for projects that are complex or need rapid prototyping and delivery. The user interface of ACTRANS has been developed in MS Visual Studio, an environment for constructing graphical user interface

IV. USING EX-ACT-P

The main scenario for using EX-ACT-P is as follows:

1. Connect to TPTP and choose a problem file
2. Use ACTRANS to transform it into corresponding ACT-P files
3. Alternatively, choose a problem form PC or create a new problem via PE
4. Choose a combination of resolution strategies
5. Call ACT-P to activate proof process and produce the proof file
6. See the displayed proof information
7. If satisfied, stop; otherwise, go to any of steps 1, 3 or 4.

Of course, one can start from step 3, by choosing a problem from PC or creating his/her own problem files.

In step 3, the user can determine a combination of strategies from some of the drop-down menus, illustrated in Table I.

A combination of strategies can include, for example, a parent selection, a clause elimination and a search strategies, or any two of them. Another type of combination can include a process oriented strategy and a search strategy etc.

The screenshot in Fig. 4 presents the situation after completion of step 5. In the main window, the problem description file in ACT-P language is presented. Also, the proved theorem and proof information (number of axioms, number of produced clauses, number of used clauses and the CPU time required) are also displayed. At the right-hand side the selected problem and the chosen strategies are also displayed.

TABLE I. RESOLUTION CONTROL STRATEGIES IN EX-ACT-P

Resolution Control Strategies			
Parent Selection	***Clause Elimination***	***Search***	***Process Oriented***
Input Resolution Linear Resolution Linear Input Resolution Set of Support Resolution Unit Resolution P1 Resolution N1 Resolution Hyperresolution	Tautology Elimination Pure Literal Elimination Backward Subsumption Forward Subsumption	Fewest Literals Preference Unit Preference	Ordered Resolution OI-Resolution Lock Resolution Model Elimination

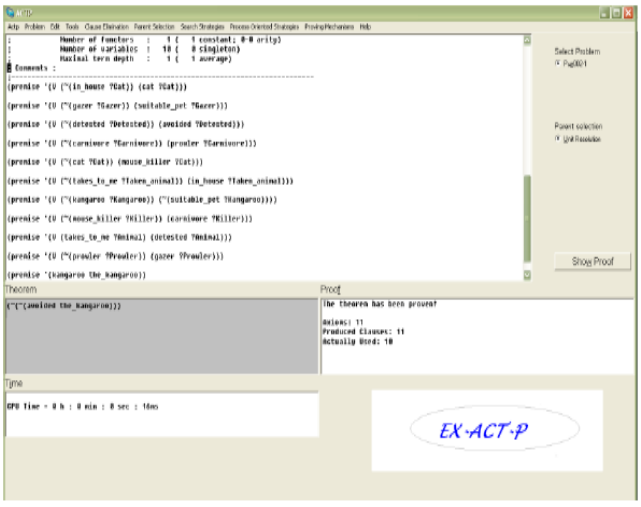

Figure 4. A screenshot of EX-ACT-P in use.

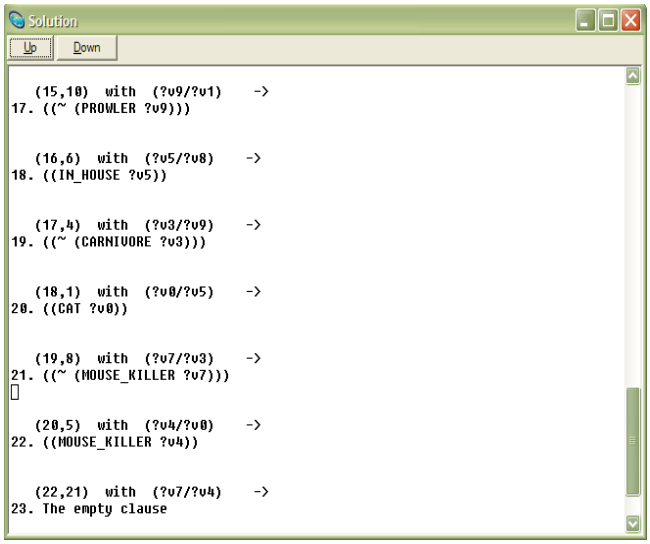

Figure 5. A screenshot of the Proof window.

By clicking on the "Show Proof" button, two windows, namely the Proof and the Proof Graph, appear. In Figure 5, a snapshot of the Proof window is presented. The Proof window contains step by step the proof process in a text-base style.

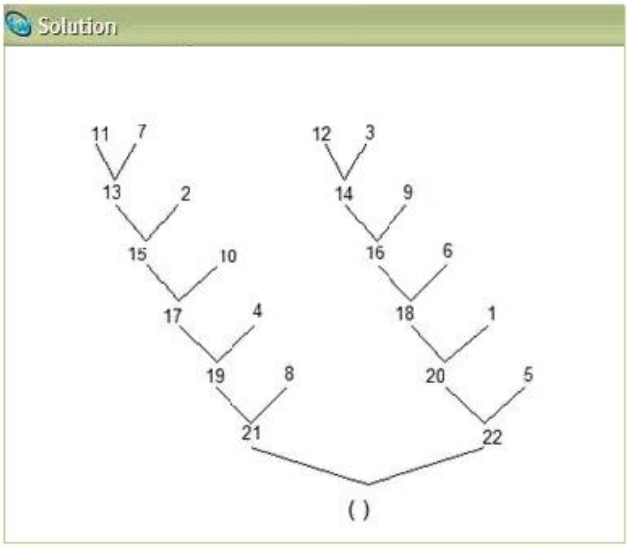

Figure 6. A Screenshot of the Proof Graph window using "Unit Resolution"

Figure 6 and Figure 7 present two snapshots of the Proof Graph window, after having used two different control strategies. Figure 6 presents a proof graph of the problem "puz002-1" after having used the "Unit Resolution" strategy, whereas Figure 7 the one after having used "P1-resolution'. The user can now have a better view for understanding and comparing proofs. In this case, ACT-P solves the problem with fewer steps by using "P1 Resolution" than by using "Unit Resolution". The numbers in Figures 6 and 7 represent corresponding clauses (which are resolved in each step).

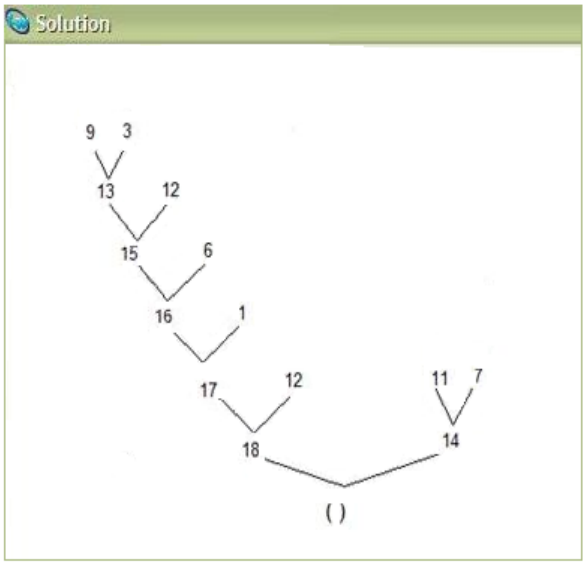

Figure 7. A Screenshot of the Proof Graph window using "P1 Resolution"

V. SYSTEM EVALUATION

The system was used by the class of the Artificial Intelligence course, in our Department, which consisted of twenty-five senior computer engineering students. The students had been taught about logic as a knowledge representation and automated reasoning during the course lectures. They were instructed to use the system. Then, they were asked to fill in a questionnaire including questions for evaluating usability and learning.

TABLE II. QUESTIONNAIRE RESULTS

Q	Questionnaire					
	Questions	*Answers (%) Total Students 25*				
		1	2	3	4	5
1	How much did the system help you to learn automated reasoning ?	0	0	20	36	44
2	How much did the system help you to learn about resolution control strategies?	0	4	24	32	40
3	Will you suggest the system to next year students?	0	0	16	32	52
4	Will you use the system in the future to prove problems?	0	4	8	42	44
5	Did you find the interface easy to use?	0	4	32	48	16

The questionnaire included eight questions. The questions 1-5 were based on Likert scale (1: not at all, 5:

very much). The result is shown as Table 1. Finally, questions 5-8 were of open type and concerned strong and weak points or problems faced in using the system. Twenty five students filled in the questionnaire. Their answers showed that the students in general were helped in learning automated reasoning with the system (Table II). Also, they found that the user interface is easy to use. On the other hand, 72% the students agreed that the system helped them in learning the resolution control strategies.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present EX-ACT-P system, a tool for experimenting with an ATP system, namely ACT-P. Experimentation can be at student, tutor or even researcher level. A student can try to solve as many problems as required, compare hand-made solutions with automatic ones, see graphical representations of solutions and even try his/her own problems. A tutor can test any candidate exercises related to FOL based reasoning, check the complexity of an exercise, gain experience from using different strategies combinations etc. Finally, a researcher can try any problem from the TPTP library, even difficult ones, experiment with different combinations of strategies and see which combinations are more appropriate for which category of problems etc. We have used ACT-P in a number of difficult problems from the TPTP library. The results are more than promising, but we want to re-check it.

EX-ACT-P can be improved in a number of ways. A first direction is the enhancement of the CSP, both in number of implemented strategies and its structure. At the moment, the structure of the CSP and corresponding interface menus do not absolutely reflect the categorization of the strategies of Fig. 3. Furthermore, the categorization of Fig. 3 is not final. It can be more sophisticated (like the one in [5]).

A more interesting direction is to construct an intelligent system advising the user about which combinations of strategies have or have no meaning. Finally, a large scale experimentation with problems of various categories from TPTP problem library targeting at comparing various combinations of strategies, could create a guide for determining control regimes in resolution-based ATP systems.

ACKNOWLEDGMENT

This work was supported by the Research Committee of the University of Patras, Greece, Program “Karatheodoris”, project No C901.

REFERENCES

[1] A. Asperti, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. User interaction with the Matita proof assistant. Journal of Automated Reasoning, 2007. Special Issue on User Interfaces for Theorem Proving.

[2] D. Aspinall. Proof General: A generic tool for proof development. In Susanne Graf and Michael Schwartzbach, editors, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1785, pages 38–42. Springer, 2000

[3] W. Billingsley and P. Robinson. Student Proof Exercises Using MathsTiles and Isabelle/HOL in an Intelligent Book. Journal of Automated Reasoning, 39:181–218, 2007.

[4] R. Bornat and B. Sufrin.. Jape: A calculator for animating proof-on-paper. In Mc-Cune, W. (ed.) Automated Deduction - CADE-14. LNCS, vol. 1249, pp. 412–415, 1997. Springer, Heidelberg

[5] I. Hatzilygeroudis and H. Reichgelt. ACT-P: A Configurable Theorem Prover, Data and Knowledge Engineering 12 (1994) 277-296.

[6] I. Hatzilygeroudis, C. Giannoulis and C. Koutsojannis. A Web Based Education System for Predicate Logic. Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT’04), 106-110.

[7] C. Kaliszyk, F. Wiedijk, M. Hendriks and F. van Raamsdonk. Teaching logic using a state-of-the-art proof assistant, in: H. Geuvers and P. Courtieu, editors, International Workshop on Proof Assistants and Types in Education (PATE’07), 2007.

[8] C. Kaliszyk, F. van Raamsdonk, F. Wiedijk, H. Wupper, M. Hendriks & R. de Vrijer, "Deduction using the ProofWeb system", report ICIS-R08016, Radboud University Nijmegen, 2008

[9] W.McCune. Prover9: Automated theorem prover for first-order and equational logic, 2008. http://www.cs.unm.edu/~mccune/mace4/ manual-examples.html.

[10] W. McCune. Experiments with discrimination-Tree and Path Indexing for Term Retrieval. Journal of Automated Reasoning 9:147-167, 1992

[11] T. Nipkow, L. C. Paulson and M. Wenzel. Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science 2283, Springer, 2002.

[12] G. Sutcliffe and C. B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning, 21(2): 177-203, 1998.

 ISBN: 978-1-61208-108-3