JProver: Integrating Connection-based Theorem Proving into Interactive Proof Assistants

Stephan Schmitt[1], Lori Lorigo[2], Christoph Kreitz[2], Aleksey Nogin[2]

[1] Department of Sciences & Engineering, Saint Louis University, Madrid, Spain
schmitts@spmail.slu.edu
[2] Department of Computer Science, Cornell-University, Ithaca, NY, U.S.A.
{lolorigo,kreitz,nogin}@cs.cornell.edu

Abstract. JProver is a first-order intuitionistic theorem prover that creates sequent-style proof objects and can serve as a proof engine in interactive proof assistants with expressive constructive logics. This paper gives a brief overview of JProver's proof technique, the generation of proof objects, and its integration into the Nuprl proof development system.

1 Introduction

In large scale applications of automated reasoning, interactive proof assistants such as Coq, HOL, Isabelle, Nuprl, and PVS are the tools of choice. Because of their expressive logics, they are more generally applicable than first-order tools, yet at a much lesser degree of automation.

JProver was developed in an effort to combine the expressive power of interactive proof assistants with the automatic capabilities of first-order theorem proving, both for reasoning about mathematics and for reasoning about programs. It provides a theorem prover for first-order intuitionistic and classical logic based on the connection method [3,10], a tool for generating proof objects in the style of sequent proofs [11], and is coupled with mechanisms for integrating the prover into the Nuprl proof/program development system [4,1] and the MetaPRL proof environment [8,9]. These components enable a user to invoke the automatic prover on proof goals that can be solved by first-order reasoning while using the expressive logic of the proof assistant for the more demanding proof parts. Furthermore, the proof information returned by JProver enables the proof assistant to build a valid proof in its own calculus.

As an example, Figure 1 describes the link between JProver and Nuprl, which is described in detail in Section 3. JProver is a stand-alone prover that communicates with a proof assistant through a logic module. Invoking JProver on a Nuprl subgoal sequent causes this sequent to be sent to JProver. The proof-search method in JProver will then generate a matrix proof from the corresponding formula tree (provided the sequent is valid), which then will be converted into a list of sequent rules that expresses a sequent proof for the formula. Upon receiving this list, Nuprl will build a sequent proof for the original goal sequent, thus confirming that the proof found is valid. Information about the relation between this sequent and the formula proven by JProver will be used during that step.

R. Gore, A. Leitsch, T. Nipkow, eds. *International Joint Conference on Automated Reasoning (IJCAR 2001)*, LNAI 2083, pp. 421–426, ©Springer Verlag, 2001.

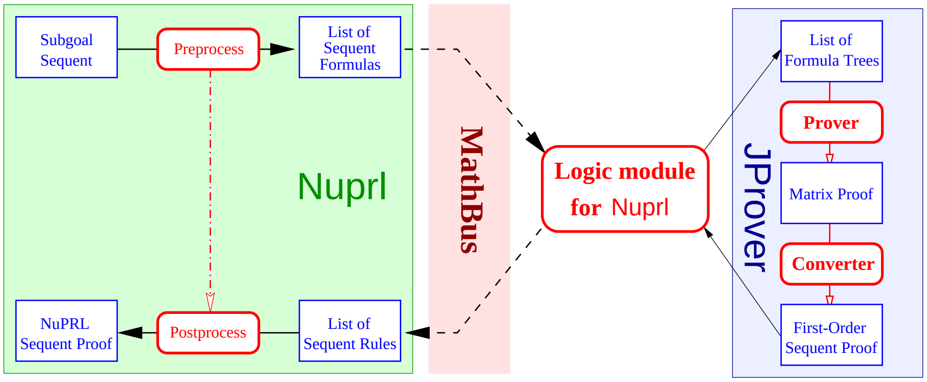


Fig. 1. Architecture of JProver in connection with Nuprl

Over the past years there have been various approaches to combining interactive proof assistants with automatic proof tools [2,17]. Our application differs from these in that we provided a fully automatic theorem prover for classical *and* intuitionistic first-order logic with a very compact search space. A user may trust its results or expand them in order to inspect the proof. Furthermore, JProver supports constructive logic and is thus well suited for reasoning about programs.

Although this paper focuses on the integration of JProver into Nuprl and MetaPRL, the underlying mechanisms are quite general and might easily be adapted to integrate JProver into other proof assistants for constructive and classical logics. In the rest of this paper we shall briefly discuss JProver's proof search procedure, the tool for generating proof objects, and the mechanisms for integrating JProver into the Nuprl and MetaPRL proof development systems.

2 JProver: Proof Search and Transformation

JProver implements a full first-order theorem prover for classical and intuitionistic logic that realizes the connection-based proof procedure presented in [10]. It transforms a set of first-order sequent formulas into a set of formula trees, that will be annotated by tableau *types*, *polarities*, and so-called *prefixes*. During the proof process, JProver identifies *connections* between pairs of atoms and checks whether each *path* through the formulas contains such a connection. The formula is valid if each of these connections is *complementary*, that is if the connected atomic formulas can be unified by a global *term substitution* and – for intuitionistic validity – if their prefixes can be unified. To compute the *prefix substitution*, we use a specialized string unification algorithm based on [14]. The resulting *matrix proof* is a *reduction ordering* that consists of the original formula trees together with the connections and non-permutability constraints induced by the substitutions.

JProver's converter component uses the algorithms described in [11,15] to reconstruct a first-order sequent proof from the classical or intuitionistic matrix proof. It essentially transforms the reduction ordering into a linear order and constructs a sequent rule for each node, using the term substitution to instantiate quantified variables. Since additional proof knowledge from the matrix proof is exploited, proof reconstruction can be done without search [15,16].

The selection of the target sequent calculus for proof reconstruction depends on the calculus underlying the connected proof assistant. For the intuitionistic case, JProver first generates a *multiple-conclusioned* sequent proof [6] because of its proof-theoretical closeness to the matrix proof. If needed, this proof can further be transformed into a *single-conclusioned* sequent proof [7] using a second conversion step as described in [5]. Nuprl, for instance, requires a single-conclusioned proof whereas MetaPRL does not. The resulting sequent proofs can be used to generate proof objects in order to validate, check, or guide proof construction in the interactive proof assistants.

JProver is implemented in OCaml as a stand-alone theorem prover. However, it is embedded into the MetaPRL environment [9], which allows it to use MetaPRL's quantifier unification algorithm as well as its module system for communicating with interactive proof assistants.

3 Integration into Interactive Proof Assistants

JProver is implemented on top of the MetaPRL core, using MetaPRL as a *toolkit* that provides the basic functionality — term structure, substitution, unification, etc. JProver takes as its input a small JLogic module that represents the logic of the proof assistant with which JProver will cooperate. The JLogic module describes which terms implement logical connectives, how to access subterms from those connectives, and how to convert JProver's generic representation of a sequent proof into the internal data structures of the proof assistant.

In order to be able to call JProver from some proof assistant, one would need to write a *logic module* that consists of two components: a piece of OCaml code for communicating with that proof assistant (using whatever communication protocol developers would choose) and a JLogic module capable of decoding the sequent received from that communication code and of encoding JProver's response into a form the communication code expects.

Currently we have integrated JProver into the MetaPRL and Nuprl systems. The technical integration of JProver into MetaPRL is straightforward, as JProver is a module in MetaPRL's code base. MetaPRL can communicate with it simply by making a function call. The logical module of the MetaPRL type theory passes its formulas directly to JProver and the JLogic module for MetaPRL converts JProver's sequent proof into a MetaPRL tactic, which will generate a MetaPRL proof for the proof goal.

The integration into Nuprl (Figure 1) is not as straightforward. Calling JProver from a Nuprl sequent requires Nuprl to preprocess the goal and the list of hypotheses and to send them to a MetaPRL process running JProver. The preprocessing accounts for differences in the representation of variables and applications of terms, and also addresses differences in the type theory semantics.

For example, JProver, as a first-order intuitionistic prover, cannot understand type information contained in Nuprl's sequents. We can, however, encode the type information as a logical predicate which is understood, and then later reinterpret JProver's results to fit the original sequent. In most cases, however, the logical proof does not depend on type information. We simply discard it if the sequent mentions only a single type.

To communicate the processed sequent, the Nuprl/JProver link takes advantage of the Nuprl Logical Programming Environment's [1] open architecture, which supports communication with external proof tools by sending terms in MathBus format [13] over an INET socket. Since most of the terms in the sequent are left unchanged, the common MathBus format is valuable in communicating and understanding contrasting syntax of the linked systems. Once the sequent is sent, the JLogic module for Nuprl describes how JProver can access the semantical information of its terms and also how to convert JProver's resulting sequent proof into a list of sequent rules with parameters, that Nuprl can then interpret. From this list of rules, Nuprl then builds a proof tree for the original sequent in a depth-first, left-to-right fashion.

Neither MetaPRL nor Nuprl rely on the correctness of JProver or the processing. Instead, JProver's output provides these systems with a proof strategy, which is then executed on the original sequent in the respective environment.

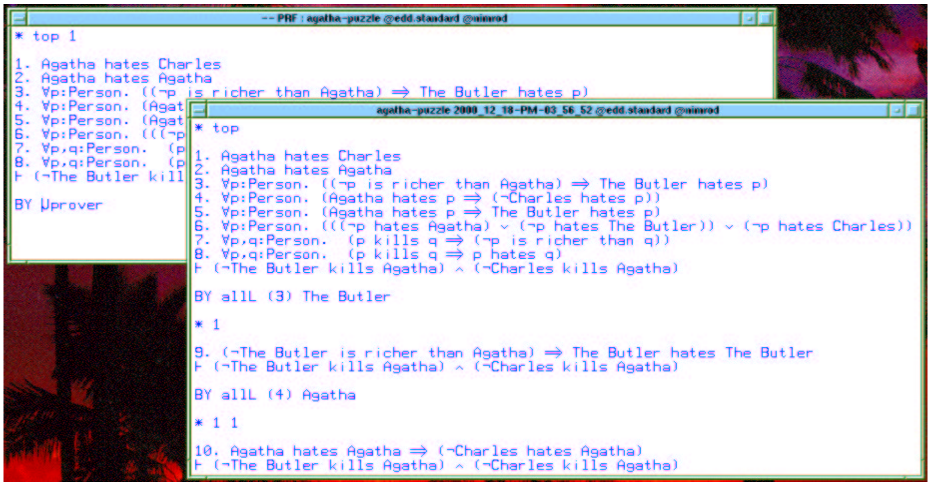
4 Progress and Availability

The connection between JProver and Nuprl is an example in which *hybrid proofs*, i.e. proofs created by multiple provers with different formalisms, have been successfully and *verifiably* generated. It gives a user the full expressive power of the proof assistant when dealing with complex proofs and verifications, while at the same time taking advantage of well-understood and efficient proof techniques for subproblems that only depend on first-order reasoning.

A snapshot from a proof of the "Agatha Murder Puzzle" is depicted in Figure 2 and illustrates the cooperation of JProver with Nuprl. After the first step the user invokes JProver through a Nuprl tactic, which completely proves the goal (left window). To inspect proof details, the user may request the complete sequent proof with elementary rules to be displayed (right window). Experience has shown that this option has considerable educational value.

It should be noted that JProver is not restricted to the syntax of first-order logic: unknown terms are simply treated as uninterpreted function or predicate symbols. This allows us to apply JProver to proof problems that are usually outside the range of first-order provers and to combine it with other proof techniques that are available to proof assistants.

In the future we intend to extend JProver's capabilities by coupling it with Nuprl tactics and decision procedures. We also intend to strengthen the prover component by adding mechanisms for inductive theorem proving described in [12] and modules for handling modal logics and fragments of linear logic [10,11].



```
-- PRF : agatha-puzzle @edd.standard @nimrod
* top 1

1. Agatha hates Charles
2. Agatha hates Agatha
3. ∀p:Person. ((¬p is richer than Agatha) ⇒ The Butler hates p)
4. ∀p:Person. (Agat
5. ∀p:Person. (Agat
6. ∀p:Person. (((¬p
7. ∀p,q:Person.  (p
8. ∀p,q:Person.  (p
⊢ (¬The Butler kill

BY Jprover
```

```
agatha-puzzle 2000_12_18-PM-03_56_52 @edd.standard @nimrod
* top

1. Agatha hates Charles
2. Agatha hates Agatha
3. ∀p:Person. ((¬p is richer than Agatha) ⇒ The Butler hates p)
4. ∀p:Person. (Agatha hates p ⇒ (¬Charles hates p))
5. ∀p:Person. (Agatha hates p ⇒ The Butler hates p)
6. ∀p:Person. (((¬p hates Agatha) ∨ (¬p hates The Butler)) ∨ (¬p hates Charles))
7. ∀p,q:Person.  (p kills q ⇒ (¬p is richer than q))
8. ∀p,q:Person.  (p kills q ⇒ p hates q)
⊢ (¬The Butler kills Agatha) ∧ (¬Charles kills Agatha)

BY allL (3) The Butler

* 1

9. (¬The Butler is richer than Agatha) ⇒ The Butler hates The Butler
⊢ (¬The Butler kills Agatha) ∧ (¬Charles kills Agatha)

BY allL (4) Agatha

* 1 1

10. Agatha hates Agatha ⇒ (¬Charles hates Agatha)
⊢ (¬The Butler kills Agatha) ∧ (¬Charles kills Agatha)
```

Fig. 2. The Nuprl/JProver link: proving the "Agatha Murder Puzzle"

These modules will make JProver valuable for a variety of other proof assistants. We plan to build the corresponding interfaces as well.

Although JProver's main emphasis is not high-performance but bringing the advantages of connection-based theorem proving such as complete and efficient search into tactic-based proof assistants, we plan to incorporate well-known techniques for speeding up automated theorem provers in order to improve JProver's performance as a stand-alone prover.

JProver is a part of the MetaPRL code base and can be downloaded from MetaPRL's home page [9]. An executable copy of Nuprl running under Linux is available at `http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl5/index.html`

References

1. S. Allen, R. Constable, R. Eaton, C. Kreitz & L. Lorigo. The Nuprl open logical environment. In D. McAllester, ed., *CADE-17*, LNAI 1831, pages 170–176. Springer Verlag, 2000.
2. C. Benzmüller et. al. *Ω*mega: Towards a mathematical assistant. In W. McCune, ed., *CADE-14*, LNAI 1249, pages 252–256. Springer Verlag, 1997.
3. W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, 1987.
4. R. L. Constable et. al. *Implementing Mathematics with the* Nuprl *proof development system.* Prentice Hall, 1986.
5. U. Egly & S. Schmitt. On intuitionistic proof transformations, their complexity, and application to constructive program synthesis. *Fundamenta Informaticae* 39(1–2):59–83, 1999.
6. M. C. Fitting. *Intuitionistic logic, model theory and forcing.* Studies in logic and the foundations of mathematics. North–Holland, 1969.
7. G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

8. J. Hickey & A. Nogin. Fast tactic-based theorem proving. In J. Harrison & M. Aagaard, eds., *TPHOLs 2000*, LNCS 1869, pages 252–266. Springer,2000.
9. Jason J. Hickey, Aleksey Nogin, et al. MetaPRL home page. `http://metaprl.org/`.
10. C. Kreitz & J. Otten. Connection-based theorem proving in classical and non-classical logics. *Journal for Universal Computer Science* 5(3):88–112, 1999.
11. C. Kreitz & S. Schmitt. A uniform procedure for converting matrix proofs into sequent-style systems. *Journal of Information and Computation* 162(1–2): 226–254, 2000.
12. C. Kreitz & B. Pientka. Matrix-based inductive theorem proving. In R. Dyckhoff, ed., *TABLEAUX-2000*, LNAI 1847, pages 294–308. Springer Verlag, 2000.
13. The MathBus Term Structure.
`http://www.cs.cornell.edu/simlab/papers/mathbus/mathTerm.htm`
14. J. Otten & C. Kreitz. T-string-unification: unifying prefixes in non-classical proof methods. In U. Moscato, ed., *TABLEAUX'96*, LNAI 1071, pages 244–260. Springer Verlag, 1996.
15. S. Schmitt. *Proof reconstruction in classcial and non-classical logics*, *Dissertationen zur Künstlichen Intelligenz 239*. Infix, 2000.
16. S. Schmitt. A tableau-like representation framework for efficient proof reconstruction. In R. Dyckhoff, ed., *TABLEAUX-2000*, LNAI 1847, pages 398–414. Springer Verlag, 2000.
17. K. Slind, M. Gordon, R. Boulton & A. Bundy. An interface between CLAM and HOL. In C. Kirchner & H. Kirchner, eds., *CADE-15*, LNAI 1421, pages 134–138. Springer Verlag, 1998.