Logic-Independent Proof Search in Logical Frameworks (Short Paper)

Michael Kohlhase[1], Florian Rabe[1], Claudio Sacerdoti Coen[2], and Jan Frederik Schaefer[1(✉)]

[1] Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany
jan.frederik.schaefer@fau.de
[2] Department of Computer Science and Engineering, Università di Bologna, Bologna, Italy

Abstract. Logical frameworks like LF allow to specify the syntax and (natural deduction) inference rules for syntax/proof-checking a wide variety of logical systems. A crucial feature that is missing for prototyping logics is a way to specify basic proof automation. We try to alleviate this problem by generating λProlog (ELPI) inference predicates from logic specifications and controlling them by logic-independent helper predicates that encapsulate the prover characteristics. We show the feasibility of the approach with three experiments: We directly automate ND calculi, we generate tableau theorem provers and model generators.

1 Introduction and Related Work

Logical frameworks like LF [HHP93] and λProlog [Mil] enable prototyping and analyzing logical systems, using high-level declarative logic definitions based on higher-order abstract syntax. Building theorem provers automatically from declarative logic definitions has been a long-standing research goal. But currently, logical framework-induced fully logic-independent proof support is generally limited to proof checking and simple search.

Competitive proof support, on the other hand, is either highly optimized for very specific logics, most importantly propositional logics or untyped first-order logic. Generic approaches have so far been successful for logics without binding (and thus quantifiers) such as the tableaux prover generation in MetTeL2 [TSK]. Logical frameworks shine when applied to logics with binding, for which specifying syntax and calculus is substantially more difficult. However, while the Isabelle framework was designed as such a generic prover [Pau93], it is nowadays primarily used for one specific logic (Isabelle/HOL). On the other hand, there has been an explosion of logical systems, often domain-specific, experimental, or

The authors gratefully acknowledge project support by German Research Council (DFG) grants KO 2428/13-1 and RA-1872/3-1 OAF as well as EU Horizon 2020 grant ERI 676541 OpenDreamKit.

N. Peltier and V. Sofronie-Stokkermans (Eds.): IJCAR 2020, LNAI 12166, pp. 395–401, 2020.
https://doi.org/10.1007/978-3-030-51074-9_22

otherwise restricted to small user communities that cannot sustain the development of a practical theorem prover. To gain theorem proving support for such logics, proof obligations can be shipped to existing provers via one of the TPTP languages, or the logics may be defined as DSLs inside existing provers as is commonly done using Coq [Coq15], Isabelle [Pau94], or Leo [Ben+08]. If applicable, these approaches are very successful. But they are also limited by the language and proof strategy of the host system, which can preclude fully exploring the design space for logics and prover.

We investigate this question by combining the advantages of two logical frameworks: To define logics, we use the implementation of LF in Mmt [Rab17,Rab18]. Mmt is optimized for specifying and prototyping logics, providing in particular type reconstruction, module system, and graphical user interface. Then we generate ELPI theorem provers from these logic definitions. ELPI [SCT15] is an extension of λProlog with constraint programming via user-defined rules, macros, type abbreviations, optional polymorphic typing and more. ELPI is optimized for fast execution of logical algorithms such as type inference, unification, or proof search, and it allows prototyping such systems much more rapidly than traditional imperative or functional languages. Both Mmt and ELPI were designed to be flexible and easy to integrate with other systems. Our approach is logic-independent and applicable to any logic defined in LF. Concretely, we evaluate our systems by generating provers for the highly modular suite of logic definitions in the LATIN atlas [Cod+11], which includes e.g. first- and higher-order and modal logics and various dependent type theories. These logic definitions can be found at [LATIN] and the generated ELPI provers in [GEP].

We follow the approach proposed by Miller et al. in the ProofCert project [CMR13] but generalize it to non-focused logics. The key idea is to translate each rule R of the deduction system to an ELPI clause for the provability predicate, whose premises correspond to the premises of R. The provability predicate has an additional argument that represents a proof certificate and each clause has a new premise that is a predicate, called its *helper predicate*, that relates the proof certificates of the premises to the one of the conclusion. Following [CMR13], the definitions of the certificates and the helper predicates are initially left open, and by providing different instances we can implement different theorem provers. In the simplest case, the additional premise acts as a guard that determines if and when the theorem prover should use a rule. It can also suggest which formulas to use during proof search when the rule is not analytic. This allows implementing strategies such as iterative deepening or backchaining. Alternatively, the helper predicates can be used to track information in order to return information such as the found proof. These can be combined modularly with minimal additional work, e.g., to return the proof term found by a backchaining prover or to run a second prover on a branch where the first one failed.

[CMR13] and later works by the authors use focusing as a preliminary requirement. While focusing was the source of inspiration of the whole technique and brings great benefits by reduction of search space, we show here that focusing is not really a requirement and that we can achieve comparable reductions in search space by designing certificates that impose the two phases of proof search

in focused calculi a posteriori. By not using focusing, our work makes the approach much more accessible as focusing is largely unknown in many communities (e.g., in linguistics). Moreover, [CMR13] was motivated by applications to one specific logic: classical/intuitionist focused first-order logic. While it is clear in theory that the same methodology can be applied to other logics, in practice we need to build tools to test and benchmark the approach in those logics. Here the possibility of generating many different provers for LF-defined logics in a completely uniform way is an important novelty of our work.

This paper is a short version of [Koh+20] which contains additional details.

2 Natural Deduction Provers

Logic Definitions in MMT/LF. While our approach is motivated by and applicable to very complex logics, including e.g. dependent type theories, it is easier to present our ideas by using a very simple running example. Concretely, we will use the language features of conjunction and untyped universal quantification. Their formalized syntax is shown below relative to base theories for propositions and terms.

$$\texttt{Props} = \{\texttt{prop} : \texttt{type}\} \quad \texttt{Terms} = \{\texttt{term} : \texttt{type}\}$$
$$\texttt{Conj} = \{\textbf{include}\,\texttt{Props},\ \texttt{and} : \texttt{prop} \to \texttt{prop} \to \texttt{prop}\}$$
$$\texttt{Univ} = \{\textbf{include}\,\texttt{Props},\ \textbf{include}\,\texttt{Terms},\ \texttt{univ} : (\texttt{term} \to \texttt{prop}) \to \texttt{prop}\}$$

Below we extend these with the respective natural deduction rules relative to a theory ND that introduces a judgment to map a proposition $F : \texttt{prop}$ to the type $\texttt{ded}\ F$ of proofs of F:

$$\texttt{ND} = \{\textbf{include}\,\texttt{Props},\ \textbf{judg}\,\texttt{ded} : \texttt{prop} \to \texttt{type}\}$$
$$\texttt{ConjND} = \{\textbf{include}\ \{\texttt{ND}, \texttt{Conj}\},\ \texttt{andEl} : \Pi_{A,B:\texttt{prop}}\texttt{ded}\ (A \wedge B) \to \texttt{ded}\ A,\ \ldots\}$$

For brevity, we only give some of the rules and use the usual notations for the constants and and univ. Note that **judg** tags ded as a judgment: while this is irrelevant for LF type checking, it allows our theorem provers to distinguish the data from the judgment level. Concretely, type declarations are divided into *data* types (such as prop and term) and *judgments* (such as ded). And term declarations are divided, by looking at their return type, into *data constructors* (such as and and univ) and *rules* (such as andEl and univE).

Generating ELPI Provers. Our LF-based formalizations of logics define the well-formed proofs, but implementations of LF usually do not offer proof search control that would allow for automation. Therefore, we systematically translate every LF theory into an ELPI file. ELPI is similarly expressive as LF so that a naive approach could simply translate the andEl rule to the ELPI statement

$$\texttt{ded}\ A \ \ :-\ \ \texttt{ded}\ (\texttt{and}\ A\ B)$$

Note how the Π-bound variables of the LF rule (which correspond to implicit arguments that LF implementations reconstruct) simply become free variables

for ELPI's Prolog engine to instantiate. However, this would not yield a useful theorem prover at all — instead, the depth-first search behavior of Prolog would easily lead to divergence. Therefore, to control proof search, we introduce additional arguments as follows:

- An n-ary judgment like `ded` becomes a $(1+n)$-ary predicate in ELPI. The new argument, called a *proof certificate*, can record information about the choice of rules and arguments to be used during proof search.
- A rule r with n premises (i.e., with n arguments remaining after discarding the implicit ones) becomes an ELPI statement with $1+n$ premises.

The additional premise is a predicate named r_{help}, i.e., we introduce one helper predicate for each rule; it receives all certificates and formulas of the rule as input. A typical use for r_{help} is to disable or enable r according to the certificate provided in the input and, if enabled, extract from this certificate those for the recursive calls. An alternative use is to synthesize a certificate in output from the output certificates of the recursive calls, which allows recording a successful proof search attempt. This is possible because λProlog is based on relations, and it is not fixed a priori which arguments are to be used as input and which as output. The two uses can even be combined by providing half-instantiated certificates in input that become fully instantiated in output.

For our running example, the following ELPI rule is generated along with a number of helper predicates:

$$\texttt{ded}\ X_2\ F\ \ :-\ \texttt{help/andEl}\ X_2\ F\ G\ X_1,\ \ \texttt{ded}\ X_1\ (\texttt{and}\ F\ G).$$

Iterative Deepening. Iterative deepening is a very simple mechanism to control the proof search and avoid divergence. Here the certificate simply contains an integer indicating the remaining search depth. A top-level loop (not shown here) just repeats proof search with increasing initial depth. Due to its simplicity, we can easily generate the necessary helper predicate automatically:

$$\texttt{help/andEl}\ (\texttt{idcert}\ X_3)\ F\ G\ (\texttt{idcert}\ X_2)\ \ :-\ \ X_3 > 0,\ \ X_2\ \texttt{is}\ X_3 - 1.$$

Backchaining. Here, the idea is to be more cautious when trying to use non-analytic elimination rules like `andEl`, whose premises contain a sub-formula not present in the conclusion. To avoid wrongly guessing these, Miller [CMR13] employs a focused logic where forward and backward search steps are alternated. We reproduce a similar behavior for our simpler unfocused logic by programming the helper to trigger forward reasoning search and by automatically generating forward reasoning clauses for some of our rules:

$$\begin{array}{l}\texttt{help/andEl}\ (\texttt{bccert}\ X_3)\ F\ G\ (\texttt{bccert}\ (\texttt{bc/fwdLocked}\ X_2))\\ \quad :-\ \texttt{bc/val}\ X_3\ X_4,\ \ X_4 > 0,\ \ X_2\ \texttt{is}\ X_4 - 1,\ \ \texttt{bc/fwdable}\ (\texttt{and}\ F\ G).\\ \texttt{bc/fwdable}\ \ :-\ \texttt{ded/hyp}\ \_\ T,\ \ \texttt{bc/aux}\ T\ A.\end{array}$$

Here we use two predicates that are defined once and for all, i.e., logic-independently: `bc/fwdable` (`and` F G) asks for a forward reasoning proof of (`and` F G); and `ded/hyp` _ T recovers an available hypothesis T.

Finally, `bc/aux` T A proves A from T using forward reasoning steps. Its definition picks up on annotations in LF that mark forward rules, and if `andEl` is annotated accordingly, we automatically generate the forward-reasoning clause below, which says that X_5 is provable from (`and` F G) if it is provable from F:

$$\texttt{bc/aux}\ (\texttt{and}\ F\ G)\ X_5\ \ :-\ \ \texttt{bc/aux}\ F\ X_5.$$

3 Tableau Provers

Logic Definitions in MMT/LF. The formalizations of the tableau rules for our running example are given below. The general idea is to represent each branch of a tableau as an LF context; the unary judgments $1\,A$ and $0\,A$ represent the presence of the signed formula A on the branch, and the judgment $\bot$ represents its closability. Thus, the type $0\,A \to \bot$ represents that A can be proved. For example, the rule `and`0 below states: if $0\,(A \wedge B)$ is on a branch, then that branch can be closed if the two branches extending it with $0\,A$ resp. $0\,B$ can.

$$\begin{aligned}
&\texttt{Tab} = \{\textbf{include}\,\texttt{Props},\ \textbf{judg}\,1 : \texttt{prop} \to \texttt{type},\ \textbf{judg}\,0 : \texttt{prop} \to \texttt{type},\\
&\quad \textbf{judg}\,\bot : \texttt{type},\ \texttt{close} : \Pi_{A:\texttt{prop}} 1\,A \to 0\,A \to \bot\}\\
&\texttt{ConjTab} = \{\textbf{include}\,\texttt{Tab},\ \textbf{include}\,\texttt{Conj},\\
&\quad \texttt{and}0 :\ \Pi_{A,B:\texttt{prop}}\ 0\,(A \wedge B) \to (0\,A \to \bot) \to (0\,B \to \bot) \to \bot,\ \ldots\}\\
&\texttt{UnivTab} = \{\ldots,\ \texttt{forall1} : \Pi_P \Pi_{X:\texttt{term}}\ 1(\forall P) \to (1(PX) \to \bot) \to \bot\}
\end{aligned}$$

Generating ELPI Provers. We use the same principle to generate ELPI statements, i.e., every LF-judgment receives an additional argument and every LF-rule an additional premise.

To generate a **tableau prover**, we use the additional arguments to track the current branch. This allows recording how often a rule has been applied in order to prioritize rule applications. For first-order logic, this is only needed to allow applying the relevant quantifier rules more than once.

For theorem proving, branches that are abandoned when the depth-limit is reached represent failed proof attempts. But we can just as well use the prover as a **model generator**: here we modify the helper predicates in such a way that abandoning an unclosed branch returns that branch. Thus, the overall run returns the list of open branches, i.e., the potential models.

Note that the ND theorem prover from Sect. 2 is strong enough to prove the tableau rules admissible for the logics we experimented with. If this holds up, it makes prototyping proof support for logic experiments much more convenient.

Application to Natural Language Understanding. Part of the motivation for the work reported here was to add an inference component — a tableau machine for natural language pragmatics — to our Grammatical/Logical Framework [KS19], which combines syntactic processing via the LF-based GF [Ran04] with MMT to obtain an NL interpretation pipeline that is parametric in the target logic. A variant of the model generator discussed above—where we extend the helper

predicate to choose a currently preferred branch when a resource bound in saturation is reached—yields an NL understanding framework that combines anaphor resolution, world-knowledge, and belief revision as in [KK03] with support for changing and experimenting with the target logic. A demo of the envisioned system can be found at [GD].

4 Conclusion and Future Work

We have revisited the question of generating theorem provers from declarative logic definitions in logical frameworks. We believe that, after studying such frameworks for the last few decades, the community has now understood them well enough and implemented them maturely enough to have a serious chance at succeeding. The resulting provers will never be competitive with existing state-of-the-art provers optimized for a particular logic, but the expressivity and flexibility of these frameworks allows building practically relevant proof support for logics that would otherwise have no support at all.

Our infrastructure already scales well to large collections of logics and multiple prover strategies, and we have already used it successfully to rapidly prototype a theorem prover in a concrete natural language understanding application. In the future, we will develop stronger proof strategies, in particular better support for equational reasoning and advanced type systems. We will also integrate the ELPI-based theorem provers as a backend for Mmt/LF in order to provide both automated and interactive proof support directly in the graphical user interface. A key question will be how the customization of the theorem prover can be integrated with the logic definitions (as we already did by annotating forward rules) without losing the declarative flavor of LF.

References

[Ben+08] Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 162–170. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_14

[CMR13] Chihani, Z., Miller, D., Renaud, F.: Checking foundational proof certificates for first-order logic. In: Blanchette, J., Urban, J. (eds.) Proof Exchange for Theorem Proving. EasyChair, pp. 58–66 (2013)

[Cod+11] Codescu, M., Horozal, F., Kohlhase, M., Mossakowski, T., Rabe, F.: Project Abstract: Logic Atlas and Integrator (LATIN). In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) CICM 2011. LNCS (LNAI), vol. 6824, pp. 289–291. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_24

[Coq15] Coq Development Team: The Coq proof assistant: reference manual. Technical report INRIA (2015)

[GD] GLIF Demo. https://gl.kwarc.info/COMMA/glif-demo-ijcar-2020. Accessed 27 Jan 2020

[GEP] Generated ELPI Provers. https://gl.mathhub.info/MMT/LATIN2/tree/devel/elpi. Accessed 26 Jan 2020

[HHP93] Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. J. Assoc. Comput. Mach. **40**(1), 143–184 (1993)

[KK03] Kohlhase, M., Koller, A.: Resource-adaptive model generation as a performance model. Log. J. IGPL **11**(4), 435–456 (2003). http://jigpal.oxfordjournals.org/cgi/content/abstract/11/4/435

[Koh+20] Kohlhase, M., et al.: Logic-independent proof search in logical frameworks (extended report). Extended Report of Conference Submission (2020). https://kwarc.info/kohlhase/submit/mmtelpi.pdf

[KS19] Kohlhase, M., Schaefer, J.F.: GF + MMT = GLF - from language to semantics through LF. In: Miller, D., Scagnetto, I., (eds.) Proceedings of the Fourteenth Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2019). Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 307, pp. 24–39 (2019). https://doi.org/10.4204/EPTCS.307.4

[LATIN] LATIN2 - Logic Atlas Version 2. https://gl.mathhub.info/MMT/LATIN2. Accessed 02 June 2017

[Mil] Miller, D: λProlog. http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/

[Pau93] Paulson, L.C.: Isabelle: The Next 700 Theorem Provers. In: arXiv CoRR cs.LO/9301106 (1993). https://arxiv.org/abs/cs/9301106

[Pau94] Paulson, L.C. (ed.): Isabelle: A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0030541

[Rab17] Rabe, F.: How to identify, translate, and combine logics? J. Log. Comput. **27**(6), 1753–1798 (2017)

[Rab18] Rabe, F.: A modular type reconstruction algorithm. ACM Trans. Comput. Log. **19**(4), 1–43 (2018)

[Ran04] Ranta, A.: Grammatical framework, a type-theoretical grammar formalism. J. Funct. Programm. **14**(2), 145–189 (2004)

[SCT15] Coen, C.S., Tassi, E.: The ELPI system (2015). https://github.com/LPCIC/elpi

[TSK] Tishkovsky, D., Schmidt, R.A., Khodadadi, M.: MetTeL2: towards a tableau prover generation platform. In: Proceedings of the Third Workshop on Practical As- pects of Automated Reasoning (PAAR-2012), p. 149 (2012)