Logic-Independent Proof Search in Logical Frameworks (extended report)

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

[1] Computer Science, FAU Erlangen-Nürnberg
[2] Department of Computer Science and Engineering, Università di Bologna

Abstract. An important question in research on logical framework is whether they can induce a logic-independent automated theorem prover. While many theorem proving innovations can indeed be generalized to large classes of logics, practically competitive provers usually require logic-specific optimizations.
To investigate this question more deeply, we build a suite of modular logic definitions in a logical framework and generate Prolog-based theorem provers for them. We focus on generality and avoid taking logic-specific knowledge into account. Our generated provers support natural deduction proof search, including backchaining, as well as tableau provers and model generators. Even though these efforts have only just started, we are already able to use the latter in a system for natural language understanding that combines grammatical with semantic processing.

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 untyped first-order logic, or obtained by defining logics as DSLs inside interactive proof assistants like Isabelle [Pau94]. While Isabelle system [Pau94], which was designed as a generic prover [Pau93], is nowadays primarily used specifically to Isabelle/HOL.

On the other hand, there has been an explosion of logical systems, often domain-specific, experimental, or 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 [Rab17b; Rab17a]. 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 [LATINa] 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.

The authors gratefully acknowledge project support by German Research Council (DFG) grants KO 2428/13-1 and RA-18723-1 OAF as well as EU Horizon 2020 grant ERI 676541 OpenDreamKit.

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:

$$\texttt{ND} = \{\textbf{include}\,\texttt{Props}, \textbf{judg}\,\texttt{ded} : \texttt{prop} \to \texttt{type}\}$$
$$\texttt{ConjND} = \{\textbf{include}\ \{\texttt{ND}, \texttt{Conj}\},\ \texttt{andEl} : \Pi_{A,B:\texttt{prop}}\texttt{ded}\ (A \wedge B) \to \texttt{ded}\ A,\ \ldots\}$$
$$\texttt{UnivND} = \{\textbf{include}\ \{\texttt{ND}, \texttt{Univ}\},\ \texttt{univE} : \Pi_P \Pi_{X:\texttt{term}}\texttt{ded}\ \forall P \to \texttt{ded}\ (PX),\ \ldots\}$$

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. [3] 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.

[3] As ELPI (like Prolog) is untyped and does not require names to be declared, translating only rules would already yield a theorem prover for the simple examples of this paper. But in general, our provers also reason about typing, e.g., to synthesize a well-typed term to instantiate a typed universal quantifier. For that, we use a binary ELPI predicate for typing and generate one typing statement for each data type and each data constructor. We omit those here.

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 examples, the following ELPI rules are generated along with a number of helper predicates:

$$\texttt{ded}\ X_2\ F\ \ :-\ \ \texttt{help/andEl}\ X_2\ F\ G\ X_1,\ \ \texttt{ded}\ X_1\ (\texttt{and}\ F\ G).$$
$$\texttt{ded}\ X_2\ X_3\ \ :-\ \ \texttt{help/univE}\ X_2\ X_3\ P\ X\ X_1,\ \ X_3 = P\ X,\ \ \texttt{ded}\ X_1(\texttt{forall}\ P).$$

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:

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

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:

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

Proof Term Tracking In this case, we use the auxiliary predicate to build the proof term by recording the rule application. Moreover, we generate ELPI constructors that take the Cartesian product of the helper predicates. This allows combining proof terms tracking with any other theorem prover. Note that in that case, one part of the product is treated as input (to guard the rule application) while the other is treated as output (to track the proof term). This works without problems due to the Prolog-style behavior.

For example, we can obtain an NDproof for $(A \wedge B) \Rightarrow A$ by combining backchaining with proof term tracking:

$$\texttt{ded}\ (\texttt{prodcert}\ (\texttt{bccert}\ 2)\ (\texttt{ptcert}\ R))\ (\texttt{impl}\ (\texttt{and}\ a\ b)\ a)$$

R contains afterwards the proof term

$$\texttt{impI}\ (\texttt{and}\ a\ b)\ a\ (\texttt{andEl}\ a\ b\ (\texttt{i}\ (\texttt{and}\ a\ b)))$$

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 and0 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.

$$\begin{aligned}
&\texttt{Tab} = \{\textbf{include}\,\texttt{Props},\ \textbf{judg}\,1 : \texttt{prop} \to \texttt{type},\ \textbf{judg}\,0 : \texttt{prop} \to \texttt{type},\\
&\quad \textbf{judg}\,\bot : \texttt{type},\ \texttt{close} : \Pi_{A:\texttt{prop}} 1\,A \to 0\,A \to \bot\}\\
&\texttt{ConjTab} = \{\textbf{include}\,\texttt{Tab},\ \textbf{include}\,\texttt{Conj},\\
&\quad \texttt{and0} :\ \Pi_{A,B:\texttt{prop}}\ 0\,(A \wedge B) \to (0\,A \to \bot) \to (0\,B \to \bot) \to \bot,\ \ldots\}\\
&\texttt{UnivTab} = \{\ldots,\ \texttt{forall1} : \Pi_P \Pi_{X:\texttt{term}}\ 1(\forall P) \to (1(PX) \to \bot) \to \bot\}
\end{aligned}$$

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 Section 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.

4 Inference for Natural Language Pragmatics

In the previous section we have seen how logics and calculi can be described in MMT and how ELPI provers can be generated from the calculi. Here, we will take a look at a variant of the tableau provers with a different application in mind: model generation for natural language understanding.

Tableau provers start with a negated formula and attempt to close all branches to prove the original formula. Instead, we can also start with a positive formula and look at the branches that remain open when the tableau is fully saturated. These open branches correspond to Herbrand models that satisfy the original formula [LP93].

4.1 Tableaux Machines for Natural Language Understanding

Tableaux machines [KK00] use this mechanism to generate (Herbrand) models for natural language discourse. Due to the ambiguity of natural language, there is a combinatorial explosion in the number of models as the discourse gets longer. Instead of generating every branch (model), the tableaux machine only maintains one model – the "best" one according to some heuristic – and backtracks if it encounters a contradiction and the branch closes. Arguably, this mirrors the mental processes of a human reading a text. It is at least compatible with the psycholinguistic literature [de 95; Sin94; GML87].

Example 1. Now imagine Jane – a researcher in computational linguistics who is well-versed in logic, but has little implementation experience – wants to experiment with this idea. She is specifically interested in modeling anaphor resolution, but rather than doing the experiment on paper, she want to actually implement it in a formal system (after all it is 2020). This prevents her from glossing over details that may be much less trivial than expected and allows her to try out larger examples. To make the experiment as complete as possible, she wants to be able to enter actual English sentences, not just their translation into logical expressions. Her first example is

(1) "*John talks to Mary. Sasha is sad. He loves her.*"

The tableaux machine will have to guess to whom "*he*" and "*her*" refer. The plan of the experiment is to use information from further sentences to force it to re-evaluate these guesses.

Fortunately, with a combination of the technology from Section **??** and previous work by the authors we can offer Jane exactly the system she needs for her experiment. Fig. 1 shows the system architecture/components of the GLIF system (Grammatical/Logical/Inferential Framework) and illustrates the processing pipeline with a simple sentence – we will stick to first-order logic for simplicity.

The first two processing steps make up the Grammatical Logical Framework (GLF) from [KS] that implements a Montague-style semantics construction process [Mon70]. For the syntactic analysis step GLF uses Ranta's Grammatical

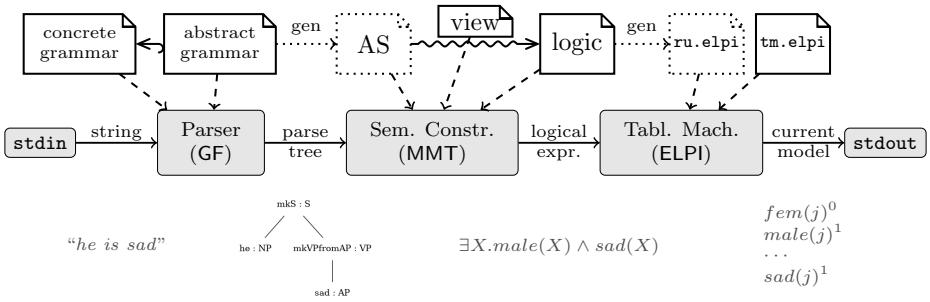

Fig. 1: The GLIF pipeline from the input of a sentence to the output of a model. Note that the tableaux machine is the only component that has to maintain a state, because we update the model incrementally.

Framework (GF) [Ran11] that modularly represents grammars as LF-based theories and abstract syntax trees as corresponding expressions. GLF passes these to MMT, which executes a **semantics-construction view**[1] to obtain logical expressions. We can feed the resulting logical expressions into the tableaux machine implemented as an ELPI program `tm.elpi`. This generates Herbrand models from the logical expressions. EdN:1

As the ELPI-based tableaux machine is the main contribution to GLIF, we will now look at this in more detail. Fig. 2 illustrates an example run of the tableaux machine. Usually, the tableaux machine needs some background knowledge. In this case, we provide some information about genders. Then we can start feeding it sentences. The first sentence is "*John talks to Mary*", which GLF translates to $talkto(j, m)$. This simply adds one entry to the model in the tableaux machine. "*Sasha is sad*" also just adds an entry. The next sentence – "*He loves her*" – is more interesting. The tableaux machine tries to use the most recently mentioned people as referents for "*he*" and "*her*". In this case, Sasha was mentioned most recently. Since Sasha can't be both male and female at the same time, that branch closes. The next best guess is that "*he*" refers to "*Sasha*" and that "*her*" refers to "*Mary*". This doesn't result in a closed branch, so it is picked as the new model. We can force the tableaux machine to backtrack on this decision by entering another sentence stating that "*Sasha is a woman*", i.e. that "*he*" can refer to "*Sasha*". The real-life equivalent of this is a misunderstanding that gets clarified.

We had already some first successes using generated ELPI code (Section 3) for the tableau machine, but more work is needed to steer the handling of quantifiers. A demo with hand-written rules can be found at [GD].

[1] EdNote: MK: introduce views in Section ??

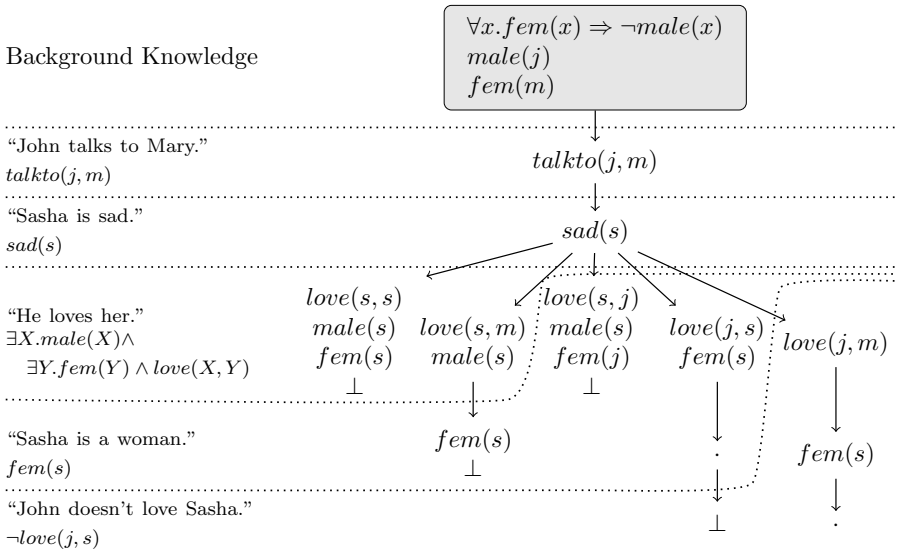

Fig. 2: Example tableau for a few sentences. For conciseness, only atomic statements that are marked true are shown.

4.2 Experimenting with GLIF: The Method of Fragments

GLF is a direct implementation of Richard Montague's "Method of Fragments" [Mon70] which postulated that to model a linguistic phenomenon one should first precisely delineate a language fragment that includes it via a grammar G, then provide a set of (compositional) translation rules from G-induced (abstract syntax) trees to logical formulae, which would then represent the meaning of an utterance. The system in Fig. 1 extends this with a configurable tableaux machine; an inferential component that allows to model the influence of context e.g. in anaphor and ambiguity resolution, conversational implicatures, etc. – linguistic pragmatics. Thus the processing pipeline in Fig. 1 represents a full Natural Language Understanding (NLU) system that starts with NL utterances and ends with a logical representation of the information conveyed.

Note that the systems in the pipeline are fully general and Jane's linguistic model only comes in from the resources in the pipeline (on the top in Fig. 1). Note furthermore that all the resources are both declarative and logic-based, which allows Jane to experiment with them easily. Thus we can think of GLIF as a general **NLU** experimentation framework; in fact we anticipate that the relative ease of experimentation has the chance to re-invigorate logic-based semantics in general and the method of fragments in computational linguistics. The latter has mostly been seen as a methodological in linguistics and language philosophy so far, not a practical research tool; GLIF can change that. [2]

EdN:2

[2] EdNote: MK: maybe save some of these arguments for the conclusion.

One extended experiment might be to integrate propositional attitudes into the linguistic model, e.g. by adding epistemic modalities in the grammar and – correspondingly – extending the logic from first-order logic to some first-order version of KD45. This is relatively easy, since GF provides a **resource grammar** [GFR] that incorporates modalities for most of its ca. 35 languages, and the LATIN atlas [Cod+11; LATINb] provides MMT theories for many modal logics. So, the only real work left for Jane is to extend the model generation rules to epistemic modal logic, e.g. following [GN08]. Other experiments can follow the same lines.

5 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] C. Benzmüller et al. “LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description)”. In: *Automated Reasoning*. Ed. by A. Armando, P. Baumgartner, and G. Dowek. Springer, 2008, pp. 162–170.

[CMR13] Z. Chihani, D. Miller, and F. Renaud. “Checking Foundational Proof Certificates for First-Order Logic”. In: *Proof Exchange for Theorem Proving*. Ed. by J. Blanchette and J. Urban. EasyChair, 2013, pp. 58–66.

[Cod+11] Mihai Codescu et al. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 289–291. URL: https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf.

[Coq15] Coq Development Team. *The Coq Proof Assistant: Reference Manual*. Tech. rep. INRIA, 2015.

[de 95] M. de Vega. "Backward updating of mental models during continuous reading of narratives". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 21 (1995), pp. 373–385.

[GD] *GLIF Demo*. URL: https://gl.kwarc.info/COMMA/glif-demo-ijcar-2020 (visited on 01/27/2020).

[GEP] *Generated ELPI Provers*. URL: https://gl.mathhub.info/MMT/LATIN2/tree/devel/elpi (visited on 01/26/2020).

[GFR] B. Bringert, T. Hallgren, and A. Ranta. *GF Resource Grammar Library: Synopsis*. URL: http://www.grammaticalframework.org/lib/doc/synopsis.html (visited on 09/27/2017).

[GML87] A. M. Glenberg, M. Meyer, and K. Lindem. "Mental models contribute to foregrounding during text comprehension". In: *Journal of Memory and Language* 26 (1987), pp. 69–83.

[GN08] R. Gore and L. Nguyen. "Analytic cut-free Tableaux for regular modal logics of agent beliefs". In: *Computational Logic in Multi-Agent Systems*. Ed. by F. Sadri and K. Satoh. Springer, 2008, pp. 268–287.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. "A framework for defining logics". In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.

[KK00] Michael Kohlhase and Alexander Koller. "Towards A Tableaux Machine for Language Understanding". In: *Proceedings of Inference in Computational Semantics ICoS-2*. Ed. by Johan Bos and Michael Kohlhase. Computational Linguistics, Saarland University, 2000, pp. 57–88.

[KS] Michael Kohlhase and Jan Frederik Schaefer. "GF + MMT = GLF – From Language to Semantics through LF". In: *LFMTP 2019, Proceedings*. Electronic Proceedings in Theoretical Computer Science (EPTCS). URL: https://kwarc.info/kohlhase/submit/lfmtp-19.pdf.

[LATINa] *LATIN2 – Logic Atlas Version 2*. URL: https://gl.mathhub.info/MMT/LATIN2 (visited on 06/02/2017).

[LATINb] *The LATIN Logic Atlas*. URL: https://gl.mathhub.info/MMT/LATIN (visited on 06/02/2017).

[LP93] Shie-Jue Lee and David A. Plaisted. "Problem solving by searching for models with a theorem prover". In: *Artificial Intelligence* 69.1-2 (1993), pp. 205–233.

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

[Mon70] R. Montague. "English as a Formal Language". In: Reprinted in [Tho74], 188–221. Edizioni di Communita, Milan, 1970. Chap. Linguaggi nella Societa e nella Tecnica, B. Visentini et al eds, pp. 189–224.

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

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover*. Vol. 828. Lecture Notes in Computer Science. Springer, 1994.

[Rab17a] Florian Rabe. "A Modular Type Reconstruction Algorithm". In: *ACM Transactions on Computational Logic* (2017). accepted pending minor revision; see https://kwarc.info/people/frabe/Research/rabe_recon_17.pdf.

[Rab17b] Florian Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* 27.6 (2017), pp. 1753–1798.

[Ran11] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth). Stanford: CSLI Publications, 2011.

[SCT15] Claudio Sacerdoti Coen and Enrico Tassi. *The ELPI system*. 2015. URL: https://github.com/LPCIC/elpi.

[Sin94] M. Singer. "Discourse Inference Processes". In: *Handbook of Psycholinguistics*. Ed. by M. A. Gernsbacher. Academic Press, 1994, pp. 479–515.

[Tho74] R. Thomason, ed. *Formal Philosophy: selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.