# The *PIE* Environment for First-Order-Based Proving, Interpolating and Eliminating

Christoph Wernhard

TU Dresden, Dresden, Germany

## Abstract

The *PIE* system aims at providing an environment for creating complex applications of automated first-order theorem proving techniques. It is embedded in Prolog. Beyond actual proving tasks, also interpolation and second-order quantifier elimination are supported. A macro feature and a LaTeX formula pretty-printer facilitate the construction of elaborate formalizations from small, understandable and documented units. For use with interpolation and elimination, preprocessing operations allow to preserve the semantics of chosen predicates. The system comes with a built-in default prover that can compute interpolants.

## 1 Introduction

The *PIE* system aims at providing an environment for creating complex applications of automated first-order theorem proving techniques, in examples, experiments, benchmarks, and as background machinery in other contexts. The system is currently under development, with some major components already functional but others still on the agenda. The state of the art for practically applying first-order proving techniques is perhaps best represented by the the *TPTP* [Sut09, Sut15], which – although designed primarily as a library for testing theorem provers – supports some aspects of creating complex applications: the Prolog-readable *TPTP* formula syntax provides a uniform interface to a variety of systems, proving problems are organized in a way that facilitates repeatability of testing, utilities for preprocessing are provided, and the classification of problems according to useful syntactic characteristics is supported. When creating complex applications becomes the main focus, various further issues arise:

1. **Tasks beyond Proving.** Aside of theorem proving in the literal sense there are various other related tasks which share techniques with first-order theorem proving and have interesting applications. Basic such tasks are model computation, which became popular in knowledge representation as *answer set programming*, the computation of first-order interpolants and second-order quantifier elimination.

   From a certain point of view, the opening of first-order reasoning to second-order extensions seems quite natural: Early technical investigations of first-order logic have been accompanied by investigations of second-order quantifier elimination for formulas with just unary predicates [Lö15, Sko19, Beh22] (see [Cra08]; for a comprehensive presentation of [Beh22] in this context see [Wer15a]). Today there are many known applications of second-order quantifier elimination and variants of it termed uniform interpolation, forgetting and projection. Aside of the study of modal logics, most of these applications are in knowledge representation and include computation of circumscription, representation of nonmonotonic semantics, abduction with



In: P. Fontaine, S. Schulz, J. Urban (eds.): Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016), Coimbra, Portugal, 02-07-2016, published at http://ceur-ws.org

respect to such semantics, forgetting in knowledge bases, and approaches to modularization of knowledge bases based on the notion of conservative extension, e.g., [GO92, LR94, DŁS97, Wer04, GLW06, GSS08, GHKS08, LW11, Wer12, Wer13a, KS13, LK14]. Advanced elimination systems are available for description logics, for example the *LETHE* system [KS15] as well as further implementations of recent techniques for forgetting based on Ackermann's lemma [ZS15, ZS16]. Elimination systems based on first-order logic have been implemented mainly in the late 1990s [Eng96, Gus96].

Interpolation plays an important role in verification [McM05], where it is used mainly on the basis of propositional logic extended with respect to specific theories. An application area on the basis of full first-order logic is database query rewriting [Mar07, NSV10, TW11, BBtC13, BtCT14, HTW15]. It seems that so far there have been only specialized techniques applied in this context, in contrast to general first-order provers. Interpolation for first-order logic is supported by *Princess* [BKRW11b, BKRW11a], as well as in some versions/extensions of *Vampire* [HKV10, HHKV12].

2. **First-Order Logic as Input and Presentation Format.** First-order logic can be considered as close enough to natural language to be well suited for humans to express applications. (This is quite different from propositional logic, where the formulas input to SAT solvers are typically the output of some encoding mechanism of the application problem. Accordingly, in the *DIMACS* format commonly used as input for SAT solvers logic atoms are just represented by natural numbers.) Compared to Prolog notation, readability of first-order formulas can in some cases be improved if functor arguments are not encapsulated by parentheses and not separated by commas, which however requires predeclaration of functor arities. The textbook [EFT84], for example, would write $\mathsf{p}\mathsf{f}xy$ instead of $\mathsf{p}(\mathsf{f}(x), y)$. Pretty-printing, where formula structure is reflected in indentation substantially facilitates readability of longer formulas. LaTeX provides professional formula layout and seems the format of choice for the presentation of formalizations.

   The *TPTP* syntax allows formulas to be read in by Prolog systems, which is very convenient for Prolog implementations of further formula manipulation. The *TPTP* tools also include a pretty-printer for that syntax. There are two subtle mismatches of the *TPTP* syntax and Prolog: *TPTP* formula variables are represented as Prolog variables, although their scoping is different (quantifier arguments vs. clauses); in the ISO Prolog standard unfortunately the character `!` is, unlike `?`, not of type "graphic", effecting that universal and existential *TPTP* quantifier symbols, which are based on `!` and `?`, can not be handled in exact analogy, and that `a != b` has to be read in as `=(!(a),b)`.

3. **Stepwise Construction of Larger Formalizations.** Comprehensibility of even modest-sized formalizations can be improved greatly by using labels, speaking names, for the involved subformulas. In analogy to procedure names, labels are particularly versatile if they can be used with parameters. Such labels also facilitate the re-use of formulas and indicate similarities of related formalizations.

   The *TPTP* requires symbolic names as formula identifiers whose scope is the respective proving problem. They typically appear in proof presentations but are not used to specify the composition of problems from formulas. Shared axiom sets can be expressed in the *TPTP* by means on an `include` directive on the file level.

4. **Flexibility of Document Granularity.** Each proving problem in the *TPTP* is represented by a file whose name is formed according to a pattern such that application domain and problem format are indicated. While this is very helpful for an archive and testing library, for the development of formalizations it seems more beneficial to follow the traditional approach of the AI programming languages like Lisp and Prolog, which do not tie document granularity – files – with semantic or abstract structural units. However, support for namespaces and symbol redefinition is in such language systems commonly related to files.

5. **Integration of Natural Language Text.** In many cases it is useful to accompany formalizations with explanatory text. Possibly an interleaved combination of formal and informal material, a form of "literate formalizing", is most adequate for complex formalizations.

   *TPTP* problem files have a structured header that includes an obligatory field for an English one-line description as well as an optional field for a longer description. Also a field for bibliographical references is provided.

6. **Sustainability and Repeatability.** The usability of documents with complex formalizations should not be limited to a narrow context. Involved mechanized tasks must be repeatable. That a proving problem

can be solved not just depends on the choice of the proving system but also on its particular configuration, prone to fail after small changes. This implies that system configurations known to work have to be precisely recorded.

7. **Benchmarks as Side Product.** For the computational tasks involved in a complex formalization, it should be possible to generate benchmark documents that can be evaluated independently and in reproducible ways, for example as *TPTP* problems files.

8. **Programming Language Embedding.** Embedding in a programming language seems useful: Programming is required to express advanced formula manipulations, including preprocessing or generation of formulas according to repeatedly occurring patterns and to specify flows of control concerning the involved proving and related tasks, as well as postprocessing of their outputs. Prolog, the original basis of the *TPTP*, seems well suited, in particular since it supports terms as readable and printable data structures that can be immediately used to represent logic formulas. In addition, specifically the *SWI Prolog* system comes with very good interfaces to XML and RDF, which are common as exchange formats for large fact bases.

9. **Integration of Large Axiom Sets and Fact Bases.** Some applications, in particular in common sense reasoning, include large sets of axioms, such as, for example, the excerpts of *CYC* included in the *TPTP*, or fact bases such as *YAGO*, *DBPedia* and *GND*, which are available in RDF format. Naming of axioms/facts is irrelevant for these. A human user typically sees only a small fraction of them that is exposed in a proof.

   *TPTP* v6.3.0 includes five subsets of *CYC* ranging form 100 up to more than 3 million axioms, where only the second smallest with 8000 axioms was used in the 2015 *CASC-25*. Seemingly that is the one that can be handled without any special mechanisms by most modern first-order provers. With *LTB* the *CASC* also provides a dedicated division for problems with large shared axiom sets, currently extracted from libraries formalized with interactive proving systems. In the competition, provers are allowed to perform offline tuning with the shared axioms and example problems.

These issues are addressed by the *PIE* system with the features and components described in Sect. 2. In the conclusion, Sect. 3, some related work is briefly discussed and an outlook is given. An example that illustrates the use of *PIE* is provided as Appendix A. Development snapshots of the system with some further examples can be obtained under GNU General Public License from

`http://cs.christophwernhard.com/pie/`.

## 2 Features and Components of the *PIE* System

### 2.1 Prolog Embedding and Interfaces to Other Systems

The *PIE* system is the succesor of *ToyElim* [Wer13b], which is based on propositional logic and has been used in experiments with applications of second-order operator elimination, such as forgetting, abduction and nonmonotonic reasoning semantics. Both systems are implemented in *SWI Prolog*. To cope with large formulas, on occasion destructive operations like `setarg` are used but hidden behind an hash table interface.

It is intended to use *PIE* from the *SWI Prolog* environment. Thus, the functionality of the system is provided essentially in form of a library of Prolog predicates. The basic formula syntax of *PIE* differs from the *TPTP* syntax (see also Sect. 1, point 2): A formula with quantifiers is represented by a Prolog ground term. For conjunction and disjunction the operators `,` and `;`, as in Prolog, are used. Implication is represented by `->` (unfortunately this operator has due to its role as "if-then-else" in Prolog a stronger default precedence than `;`). A clausal formula is represented as list of lists of literals, where variables are Prolog variables. Conversions to and from *TPTP* first-order and clausal forms as well as *Otter/Prover9/Mace4* syntax are provided. For propositional logic and quantified Boolean formulas there are conversions to and from *DIMACS* and *QDIMACS*, respectively. Call-interfaces for proving tasks to provers accepting the *TPTP* or the *Prover9* format as well as to SAT and QBF solvers are provided. For propositional elimination tasks there is also a special *DIMACS*-based interface to *Coprocessor* [Man12], a SAT preprocessor that can be configured to perform elimination tasks.

The *PIE* system includes the Prolog-based first-order prover *CM*, described further in Sect. 2.4, which is thus immediately available in a lightweight way, without detour through the operation system and without further installation effort. Proofs returned by *CM* can be directly processed with Prolog and allow interpolant extraction. The internals of the prover are easy to modify for experimenting. As a second possibility for interpolation tasks, an interface to *Princess* [BKRW11b] is envisaged.

### 2.2 Macros

A macro facility is provided, where the macros can have parameters and expand to first- or second-order formulas. In this way, a macro name can serve not just as a formula label but, more generally, can be used within another formula in subformula position, possibly with arguments.

Pattern matching is used to choose the effective declaration for expansion, allowing structural recursion in macro declarations. An optional Prolog body in a macro declaration permits expansions involving arbitrary computations. Utility predicates intended for use in these bodies are provided for common tasks, for example, determining the predicates in a formula after expansion. With a macro declaration, properties of its lexical environment, in particular configuration settings that affect the expansion, are recorded.

Macros provide a convenient way to express properties of predicates, e.g., transitivity, abstractly, and also to express more complex application patterns that can not be stated as definitions within second-order logic unless predicates in argument position would be permitted. Various practically useful computations can be considered as elimination of second-order operators, in particular, literal forgetting (a form of predicate quantification that only affects predicate occurrences with negative or positive polarity), projection (which is like predicate quantification, except that the predicates that are not quantified – whose semantics is to be retained – are made explicit) and circumscription. The macro mechanism allows to use dedicated operator notation for these on the user level, whereas on the system level predicate quantification is the only second-order operator that needs to be supported.

### 2.3 LaTeX Document Generation

A formula pretty-printer is available that outputs Prolog-readable syntax as well as LaTeX. In addition to formula indentation, the LaTeX pretty-printer converts numbers and single characters separated by an underscore at the end of Prolog functor names to subscripts in LaTeX. A brief syntax as indicated in Sect. 1, point 2, is optionally available for in- and output of formulas.

Prolog-readable files with macro declarations can be translated to LaTeX documents which contain the pretty-printed declarations, all text within specially marked comments, and all outputs produced by calls wrapped in a special Prolog predicate that indicates evaluation at "printtime". Utility predicates for tasks like proving, interpolation and elimination are provided that pretty-print solutions in LaTeX at "printtime", and in Prolog when invoked normally.

### 2.4 Included Prolog Technology Theorem Prover *CM*

The included prover *CM* was originally written in 1992, based on [Sti88] but as compiler to Prolog instead of Lisp. The underlying calculus can be understood as model elimination, as clausal tableau construction, or as the connection method. The *leanCoP* prover [Ott10], also implemented in Prolog, is based on the same calculus and regularly takes part in the *CASC* competitions. Re-implementations of *leanCoP* have recently found interest to study new techniques [Kal15, KU15]. A conceptual difference to *leanCoP* is that *CM* proceeds in a two-step way, by first compiling the proving problem. This makes the prover less flexible at runtime, but the possibility to inspect the output of the compiler on its own contributes to robustness and facilitates experimenting with modifications. The prover had been revised in 1997 [DW97] and recently in 2015 where it has been ported to *SWI Prolog*, combined with some newly implemented preprocessing operations and improved with the problems of the 2015 *CASC* competition, *CASC-25*, taken as yardstick. For the *CASC-25* problems in the *first-order no equality (FNE)* category it performs better than *leanCop* and *iProver Modulo* but worse than *CVC4*. For the *CASC-25* problems in the *first-order with equality (FEQ)* category it performs significantly worse than *leanCop* but better than *iProver Modulo*. More information on the prover and the evaluation details are available at `http://cs.christophwernhard.com/pie/cmprover/`.

### 2.5 Preprocessing with Semantics Preservation for Chosen Predicates

A number of implemented preprocessing operations is included, most of them currently for clausal inputs. While some of these preserve equivalence, for example conventional CNF transformation (if function introduction through Skolemization is disregarded), others preserve equivalence just with respect to a set of predicates $p_1, \ldots p_n$, that is, inputs $F$ and outputs $G$ satisfy the second-order equivalence

$$\exists q_1 \ldots \exists q_m \, F \equiv \exists q_1 \ldots \exists q_m \, G, \tag{1}$$

where $q_1, \ldots, q_m$ are all involved predicates with exception of $p_1, \ldots p_n$. Such conversions clearly also preserve equi-satisfiability, which corresponds just to the special case where $n = 0$. Keeping track of the predicates $p_1, \ldots, p_n$ whose semantics is to preserve is needed for preprocessing in applications like interpolation and second-order quantifier elimination. (The *Riss* SAT solver and the related preprocessor *Coprocessor* [Man12] also support this.) Some preprocessing conversions allow, moreover, to keep track of predicate semantics with respect to just their occurrences with positive or negative polarity, which is useful for Lyndon interpolation (Sect. 2.6) and literal forgetting (Sect. 2.2).

Implemented preprocessing methods that preserve equivalence with respect to a given set of predicates include a variant of the Plaisted-Greenbaum transformation, a structural hashing transformation that identifies two predicates (or one predicate with the negation of another) in the case where both have equivalent non-recursive and easily detectable "semi-definitions" (the, say, positive "semi-definition" of the predicate $p$ is obtained by taking all clauses containing a positive literal with predicate $p$ and removing these literals, equivalence (4) below can be used to justify this transformation semantically), and elimination of predicates in cases where this does not lead to a substantial increase of the formula size.

### 2.6 Support for Interpolation

A Craig interpolant for two given first-order formulas $F$ and $G$ such that $(F \rightarrow G)$ is valid is a first-order formula $H$ such that $(F \rightarrow H)$ and $(H \rightarrow G)$ are valid and $H$ contains only symbols (predicates, functions, constants, free variables) that occur in both $F$ and $G$. Craig's interpolation theorem ensures the existence of such interpolants. The *PIE* system supports the computation of Craig interpolants from closed clausal tableaux with a variant of the tableau-based interpolant construction method from [Smu95, Fit95]. (It seems that this method has not been reviewed in the recent literature on interpolation for first-order logic in automated systems, e.g., [HKV10, BKRW11a, BJ15].)

Suitable clausal tableaux are constructed by the *CM* prover. Interpolant construction can there be done entirely by pre- and postprocessing, without requiring alterations to the core prover. The preprocessing includes simplifications as outlined in Sect. 2.5. Also the *Hyper* hypertableau prover [PW07] produces proof terms that can be converted to tableaux, but this is not fully worked out in the current implementation.

The constructed interpolants strengthen the requirements for Craig interpolants in that they are actually Lyndon interpolants, that is, predicates occur in them only in polarities in which they occur in both of the two input formulas. Symmetric interpolation, a generalization of interpolation described in [Cra57, Lemma 2] and called *symmetric*, e.g., in [McM05, Sect. 5], is supported, implemented by computing a conventional interpolant for each of the input formulas, corresponding to the induction suggested in [Cra57]. Handling of functions (with exception of individual constants) is not supported in the current implementation. It seems possible to implement this by a relational translation of the functions that is reversed on the output.

### 2.7 Support for Second-Order Extensions to First-Order Logic

In some cases second-order formalizations are convertible to first-order logic but more concise than their first-order counterparts. This suggests to use preprocessors that produce first-order outputs to handle such second-order inputs. The following equivalence is analogous to the familiar effect of Skolemization of individual variables quantified by outermost quantifiers but applies to predicate variables:

$$\exists p_1 \ldots \exists p_n \, F[p_1, \ldots, p_n] \models \forall q_1 \ldots \forall q_m \, G[q_1 \ldots, q_m] \quad \text{iff} \quad F[p'_1, \ldots, p'_n] \models G[q'_1 \ldots, q'_m], \tag{2}$$

where $F[p'_1, \ldots, p'_n]$ is obtained from $F[p_1, \ldots, p_n]$ by replacing all occurrences of predicates $p_i$ for $1 \leq i \leq n$ with fresh predicates $p'_i$ and, analogously, $G[q'_1 \ldots, q'_m]$ from $G[q_1 \ldots, q_m]$ by replacing $q_i$ for $1 \leq i \leq m$ with fresh $q'_i$. The left side of (2) is a second-order entailment, the right side a first-order entailment. This pattern occurs in the characterization of definability and computation of definientia by interpolation (see Appendix A for an example), where advantages of the second-order formulation are that predicate renaming is hidden by quantification and that a relation to the strongest necessary and weakest sufficient condition [Lin01, DŁS01, Wer12], which are essentially second-order concepts, is made explicit.

Moving second-order quantifiers outward can be necessary to achieve a form matching the left side of (2). Aside of the equivalence preserving quantifier movement operations familiar from first-order quantifiers, the following second-order equivalence, "Ackermann's quantifier switching", due to [Ack35b] can be useful there:

$$\forall x \exists p \, F[p\boldsymbol{t}_1, \ldots, p\boldsymbol{t}_m] \quad \equiv \quad \exists q \forall x \, F[qx\boldsymbol{t}_1, \ldots, qx\boldsymbol{t}_m], \tag{3}$$

where $F[p\boldsymbol{t}_1, \ldots, p\boldsymbol{t}_m]$ is a second-order formula in which $x$ only occurs free and $p$ is a predicate of arity $n \geq 0$ with exactly the $m$ indicated occurrences, instantiated with $n$-ary argument sequences $\boldsymbol{t}_1$ to $\boldsymbol{t}_m$ and $F[qx\boldsymbol{t}_1, \ldots, qx\boldsymbol{t}_m]$ is like $F[p\boldsymbol{t}_1, \ldots, p\boldsymbol{t}_m]$ except that the occurrences of $p$ are replaced by occurrences of a fresh predicate $q$ of arity $n+1$, applied on the original argument sequences $\boldsymbol{t}_i$ prepended by $x$. Applied in the converse direction than displayed here, the equivalence (3) can help to facilitate second-order quantifier elimination by simplifying the argument formula of the predicate quantifier, leading for example to an elimination-based decision method for the description logic $\mathcal{ALC}$ [Wer15b]. The current implementation of *PIE* includes automatic handling of second-order quantifiers in proving tasks according to (2). More advanced preprocessing is planned.

### 2.8 Support for Second-Order Quantifier Elimination

Second-order quantifier elimination means to compute for a given formula with second-order quantifiers, that is, quantifiers upon predicate or function symbols, an equivalent first-order formula. (For *functions* an arity $> 0$, in contrast to individual constants, is assumed here.)

For Boolean quantifier elimination, that is, second-order quantifier elimination on the basis of propositional logic, several methods were already available in the *ToyElim* system: expanding subformulas with the Shannon expansion followed by simplifications (a more advanced version of this method is described in [Wer09]), and for clausal formulas a resolution-based method as well and a call-interface to *Coprocessor* [Man12], a SAT preprocessor that can be configured to perform resolution-based elimination. For first-order logic, the *PIE* system currently includes a variant of the *DLS* algorithm [DŁS97], which is based on Ackermann's Lemma, the following equivalence due to [Ack35a]:

$$\exists p\,(\forall x\, px \rightarrow G[x]) \wedge F[pt_1, \ldots, pt_n] \quad \equiv \quad F[G[t_1], \ldots, G[t_n]], \tag{4}$$

where $p$ does not occur in $G[x]$ and does occur in $F$ in exactly the $n$ indicated occurrences which all have positive polarity, and $F[G[t_1], \ldots, G[t_n]]$ is obtained from $F[pt_1, \ldots, pt_n]$ by replacing all occurrences of $p$ with occurrences of $G[x]$ in which $x$ is instantiated to the respective argument $t_i$ of $p$. (It is assumed that $x$ does not occur bound in $G[x]$.) The lemma obviously also holds in a dual version, where the implication is $(\forall x\, px \leftarrow G[x])$ and the occurences in $F$ all have negative polarity. It can be straightforwardly generalized to predicates $p$ with more than a single argument.

The improvement of the second-order quantifier elimination techniques and their integration with the other supported tasks belong to the major issues for the next development steps.

## 3 Conclusion

Concerning related work based on first-order proving, the TPTP had already been mentioned in many specific contexts. For developing certain kinds of advanced and complex mathematical proofs in a human-assisted way, some first-order systems, notably *Otter* and *Prover9/Mace4*, alone seem suitable by providing control features such as weighting and hints. The most ambitious approach for a working environment around first-order provers so far seemed to be the *ILF* project in the 1990s, targeted at human users and mathematical applications. It was centered around an advanced proof conversion with LaTeX presentation in textbook style [DGH+94, Dah98]. Further goals were a mathematical library based on *Mizar* and an embedding into the *Mathematica* environment [DW97, DHHW98]. The goals of *PIE* are much more modest. The main focus is not on proofs but on formulas, possibly reduced by elimination and normalized in certain ways. In the conception of the *ILF Mathematical Library* there was a somewhat ad hoc pragmatic break between syntactic preselection of possibly relevant axioms and their semantic processing by provers. It is the hope that second-order techniques justify a more thoroughly semantic approach to modularization and retrieval from formalized knowledge bases.

It is planned to use *PIE* embedded into another Prolog-based system that provides knowledge based support for scholarly editing [KW16a, KW16b]. The first envisaged application there is to control the preprocessing of RDF fact bases through definienda obtained by interpolation.

As already indicated in Sect. 2.8, the extension of the facilities for second-order quantifier elimination will be a main issue for the next development steps of *PIE*. Aside of *DLS*, the second major approach to second-order quantifier elimination is the resolution-based *SCAN* algorithm. A variant of this has also been already implemented in *PIE* but mainly for abstract investigations, without expectation to compete with possible implementations based on dedicated resolution provers. *DLS* is known to be complete (i.e. terminate successfully) exactly for inputs that satisfy a certain syntactic condition [Con06]. Orthogonal to that condition, there are

further cases with known terminating methods, for example, formulas with only unary predicates [Lö15, Beh22] (see also [Wer15b]), "semi-monadic" formulas [Wer15b], and the case where explicitly given ground atoms are "forgotten" [LR94]. Various elimination techniques have thus to be combined for practical application, in a way that ensures completeness for known cases.

A related topic for future research is the exploration of relationships between interpolation and elimination. For example, the objective of one of the preprocessing steps of *DLS* is to decompose a given formula into a conjunction $A \wedge B$ where $A$ contains the predicate $p$ to eliminate just in occurrences with negative polarity and $B$ just with positive polarity, approximately matching the conjunction in the left side of (4). Symmetric interpolation can be used to solve this step in a complete way, that is, whenever there is a decomposition into such $A$ and $B$, then one is found. Other parallels between interpolation and elimination concern the notion of conservative extension, a relationship between formulas that can be expressed as entailment of an existential second-order formula. For the special case of definitional extension (see e.g., [Hod97]), also if the extension is not given in the syntactic form of explicit definitions, the corresponding task can be expressed by first-order reasoning based on interpolation.

To sum up, the *PIE* system tries to supplement what is needed to create applications with techniques of automated first-order proving techniques, not solely in the sense of mathematical theorem proving but considered as general tools of information processing. The main focus are formulas, as constituents of complex formalizations and as results of elimination and interpolation. Special emphasis is made on utilizing some natural relationships between first- and second-order logics. Key ingredients of the system are macros, LaTeX prettyprinting and integration into a programming environment.

**Acknowledgments**

This work was supported by *DFG grant WE 5641/1-1.*

## References

[Ack35a] Wilhelm Ackermann. Untersuchungen über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ack35b] Wilhelm Ackermann. Zum Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.

[BBtC13] Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting guarded negation queries. In *MFCS 2013*, volume 8087 of *LNCS*, pages 98–110. Springer, 2013.

[Beh22] Heinrich Behmann. Beiträge zur Algebra der Logik, insbesondere zum Entscheidungsproblem. *Mathematische Annalen*, 86(3–4):163–229, 1922.

[BJ15] Maria Paola Bonacina and Moa Johansson. Interpolation systems for ground proofs in automated deduction: a survey. *Journal of Automated Reasoning*, 54(4):353–390, 2015.

[BKRW11a] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. Beyond quantifier-free interpolation in extensions of Presburger arithmetic. In *VMCAI 2011*, volume 6538 of *LNCS*, pages 88–102. Springer, 2011.

[BKRW11b] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. An interpolating sequent calculus for quantifier-free Presburger arithmetic. *Journal of Automated Reasoning*, 47(4):341–367, 2011.

[BtCT14] Michael Benedikt, Balder ten Cate, and Efthymia Tsamoura. Generating low-cost plans from proofs. In *PODS'14*, pages 200–211. ACM, 2014.

[Con06] Willem Conradie. On the strength and scope of DLS. *Journal of Applied Non-Classsical Logics*, 16(3–4):279–296, 2006.

[Cra57] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.

[Cra08] William Craig. Elimination problems in logic: A brief history. *Synthese*, (164):321–332, 2008.

[Dah98] Bernd Ingo Dahn. Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem. *Journal of Algebra*, 208(2):526–532, 1998.

[DGH+94] Bernd Ingo Dahn, Jürgen Gehne, Thomas Honigmann, Lutz Walther, and Andreas Wolf. Integrating logical functions with ILF. Technical Report Preprints aus dem Institut für Mathematik 10, Humboldt Universität zu Berlin, Institut für Mathematik, 1994.

[DHHW98] Bernd Ingo Dahn, Andreas Haida, Thomas Honigmann, and Christoph Wernhard. Using Mathematica and automated theorem provers to access a mathematical library. In *CADE-15 Workshop on Integration of Deductive Systems*, pages 36–43, 1998.

[DŁS97] Patrick Doherty, Witold Łukaszewicz, and Andrzej Szałas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–338, 1997.

[DŁS01] Patrick Doherty, Witold Łukaszewicz, and Andrzej Szałas. Computing strongest necessary and weakest sufficient conditions of first-order formulas. In *IJCAI-01*, pages 145–151. Morgan Kaufmann, 2001.

[DW97] Ingo Dahn and Christoph Wernhard. First order proof problems extracted from an article in the Mizar mathematical library. In *FTP'97*, RISC-Linz Report Series No. 97–50, pages 58–62. Johannes Kepler Universität, Linz, 1997.

[EFT84] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. Springer, 1984.

[Eng96] Thorsten Engel. Quantifier elimination in second-order predicate logic. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, 1996.

[Fit95] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 2nd edition, 1995.

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, 31(1):273–318, 2008.

[GLW06] Silvio Ghilardi, Carsten Lutz, and Frank Wolter. Did I damage my ontology? A case for conservative extensions in description logics. In *KR 06*, pages 187–197. AAAI Press, 2006.

[GO92] Dov Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second-order predicate logic. In *KR'92*, pages 425–435. Morgan Kaufmann, 1992.

[GSS08] Dov M. Gabbay, Renate A. Schmidt, and Andrzej Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, 2008.

[Gus96] Joakim Gustafsson. An implementation and optimization of an algorithm for reducing formulae in second-order logic. Technical Report LiTH-MAT-R-96-04, Univ. Linköping, 1996.

[HHKV12] Kryštof Hoder, Andreas Holzer, Laura Kovács, and Andrei Voronkov. Vinter: A Vampire-based tool for interpolation. In *APLAS 2012*, volume 7705 of *LNCS*, pages 148–156. Springer, 2012.

[HKV10] Krystof Hoder, Laura Kovács, and Andrei Voronkov. Interpolation and symbol elimination in Vampire. In *IJCAR 2010*, volume 6173 of *LNCS*, pages 188–195. Springer, 2010.

[Hod97] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, Cambridge, 1997.

[HTW15] Alexander Hudek, David Toman, and Grant Wedell. On enumerating query plans using analytic tableau. In *TABLEAUX 2015*, volume 9323 of *LNCS (LNAI)*, pages 339–354. Springer, 2015.

[Kal15] Cezary Kaliszyk. Efficient low-level connection tableaux. In *TABLEAUX 2015*, volume 9323 of *LNCS (LNAI)*, pages 102–111. Springer, 2015.

[KS13] Patrick Koopmann and Renate A. Schmidt. Uniform interpolation of $\mathcal{ALC}$-ontologies using fixpoints. In *FroCoS 2013*, volume 8152 of *LNCS (LNAI)*, pages 87–102. Springer, 2013.

[KS15] Patrick Koopmann and Renate Schmidt. LETHE: Saturation-based reasoning for non-standard reasoning tasks. In *ORE 2015*. CEUR-WS.org, 2015.

[KU15] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *LPAR-20*, volume 9450 of *LNCS*, pages 88–96. Springer, 2015.

[KW16a] Jana Kittelmann and Christoph Wernhard. Knowledge-based support for scholarly editing and text processing. In *DHd 2016 – Konferenzabstracts*, pages 176–179. nisaba verlag, 2016.

[KW16b] Jana Kittelmann and Christoph Wernhard. Towards knowledge-based assistance for scholarly editing. In *AITP 2016 (Book of Abstracts)*, 2016.

[Lin01] Fangzhen Lin. On strongest necessary and weakest sufficient conditions. *Artificial Intelligence*, 128:143–159, 2001.

[LK14] Michel Ludwig and Boris Konev. Practical uniform interpolation and forgetting for $\mathcal{ALC}$ TBoxes with applications to logical difference. In *KR'14*. AAAI Press, 2014.

[LR94] Fangzhen Lin and Raymond Reiter. Forget It! In *Working Notes, AAAI Fall Symp. on Relevance*, pages 154–159, 1994.

[LW11] Carsten Lutz and Frank Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *IJCAI-11*, pages 989–995. AAAI Press, 2011.

[Lö15] Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915.

[Man12] Norbert Manthey. Coprocessor 2.0 – a flexible CNF simplifier. In *SAT '12*, volume 7317 of *LNCS*, pages 436–441. Springer, 2012.

[Mar07] Maarten Marx. Queries determined by views: Pack your views. In *PODS '07*, pages 23–30. ACM, 2007.

[McM05] Kenneth L. McMillan. Applications of Craig interpolants in model checking. In *TACAS 2005*, volume 3440 of *LNCS*, pages 1–12, Berlin, Heidelberg, 2005. Springer.

[NSV10] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3), 2010.

[Ott10] Jens Otten. Restricting backtracking in connection calculi. *AI Communications*, 23(2-3):159–182, 2010.

[PW07] Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In *CADE-21*, volume 4603 of *LNCS (LNAI)*, pages 503–513. Springer, 2007.

[Sko19] Thoralf Skolem. Untersuchungen über die Axiome des Klassenkalküls und über Produktations- und Summationsprobleme welche gewisse Klassen von Aussagen betreffen. *Videnskapsselskapets Skrifter*, I. Mat.-Nat. Klasse(3), 1919.

[Smu95] Raymond M. Smullyan. *First-Order Logic*. Dover publications, New York, 1995. Corrected republication of the original edition by Springer-Verlag, New York, 1968.

[Sti88] Mark E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, 1988.

[Sut09] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[Sut15] Geoff Sutcliffe. The TPTP Problem Library – TPTP v6.0.0, 2015. `http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml`, accessed 26 April 2016.

[TW11] David Toman and Grant Weddell. *Fundamentals of Physical Design and Query Compilation*. Morgan and Claypool, 2011.

[Wer04] Christoph Wernhard. Semantic knowledge partitioning. In *JELIA 04*, volume 3229 of *LNCS (LNAI)*, pages 552–564. Springer, 2004.

[Wer09] Christoph Wernhard. Tableaux for projection computation and knowledge compilation. In *TABLEAUX 2009*, volume 5607 of *LNCS (LNAI)*, pages 325–340, 2009.

[Wer12] Christoph Wernhard. Projection and scope-determined circumscription. *Journal of Symbolic Computation*, 47:1089–1108, 2012.

[Wer13a] Christoph Wernhard. Abduction in logic programming as second-order quantifier elimination. In *FroCoS 2013*, volume 8152 of *LNCS (LNAI)*, pages 103–119. Springer, 2013.

[Wer13b] Christoph Wernhard. Computing with logic as operator elimination: The ToyElim system. In *INAP 2011/WLP 2011*, volume 7773 of *LNCS (LNAI)*, pages 289–296. Springer, 2013.

[Wer15a] Christoph Wernhard. Heinrich Behmann's contributions to second-order quantifier elimination from the view of computational logic. Technical Report Knowledge Representation and Reasoning 15-05, TU Dresden, 2015.

[Wer15b] Christoph Wernhard. Second-order quantifier elimination on relational monadic formulas – A basic method and some less expected applications. In *TABLEAUX 2015*, volume 9323 of *LNCS (LNAI)*, pages 249–265. Springer, 2015.

[ZS15] Yizheng Zhao and Renate A. Schmidt. Concept forgetting in $\mathcal{ALCOI}$-ontologies using an Ackermann approach. In *ISWC 2015, Part I*, volume 9366 of *LNCS*, pages 587–602. Springer, 2015.

[ZS16] Yizheng Zhao and Renate A. Schmidt. Forgetting concept and role symbols in $\mathcal{ALCOIH}\mu^+(\nabla, \sqcap)$-ontologies. In *IJCAI 2016*. AAAI Press, 2016. To appear.

# A Example Document

The following pages illustrate the user view on the *PIE* system with the rendered LaTeX output and the corresponding Prolog-readable input document of a short example. Further examples are included with the system distribution and can be inspected on the system Web page.

## A.1 Output of the Example

*PIE* Example Document

## 1 Projection and Definientia

---

*project*$(S, F)$

---

Defined as

$$\exists S_1\, F,$$

where

$$S_2 := \mathsf{free\_predicates}(F),$$
$$S_1 := S_2 \setminus S.$$

---

*definiens*$(G, F, S)$

---

Defined as

$$\mathit{project}(S, (F \wedge G)) \rightarrow \neg \mathit{project}(S, (F \wedge \neg G)).$$

## 2 Obtaining a Definiens with Interpolation

We specify a background knowledge base:

---

$kb_1$

---

Defined as

$$\forall x\, (\mathsf{q}x \rightarrow \mathsf{p}x) \wedge (\mathsf{p}x \rightarrow \mathsf{r}x) \wedge (\mathsf{r}x \rightarrow \mathsf{q}x).$$

To obtain a definiens of $\exists x\, \mathsf{p}x$ in terms $\{q, r\}$ within $kb_1$ we specify an implication with the *definiens* macro:

1

*ex₁*

Defined as

$$definiens((\exists x\, \mathsf{p}x), kb_1, [\mathsf{q}, \mathsf{r}]).$$

We now invoke a utility predicate that computes and prints for a given valid implication an interpolant of its antecedent and consequent:
Input: *ex₁*.
Result of interpolation:

$$\exists x\, \mathsf{q}x.$$

Before we leave that example, we take a look at the expansion of $ex_1$:

$$\begin{array}{l} (\exists p\, (\forall x\, (\mathsf{q}x \to px) \wedge (px \to \mathsf{r}x) \wedge (\mathsf{r}x \to \mathsf{q}x)) \quad \wedge \\ \quad (\exists x\, px)) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to \\ \neg(\exists p\, (\forall x\, (\mathsf{q}x \to px) \wedge (px \to \mathsf{r}x) \wedge (\mathsf{r}x \to \mathsf{q}x)) \wedge \\ \quad \neg(\exists x\, px)). \end{array}$$

And we invoke an external prover (the $E$ prover) to validate it:
This formula is valid: *ex₁*.

## 3 Obtaining the Strongest Definiens with Elimination

The antecedent of the implication in the expansion of *definiens* specifies the strongest necessary condition of $G$ on $S$ within $F$. In case definability holds (that is, the implication is valid), this antecedent denotes the strongest definiens. In the example it has a first-order equivalent that can be computed by second-order quantifier elimination.

*ex₂*

Defined as

$$project([\mathsf{q}, \mathsf{r}], (kb_1 \wedge (\exists x\, \mathsf{p}x))).$$

Input: *ex₂*.
Result of elimination:

$$\begin{array}{l} (\forall x\, \neg\mathsf{r}x \vee \mathsf{q}x) \wedge \\ (\forall x\, \neg\mathsf{q}x \vee \mathsf{r}x) \wedge \\ (\exists x\, \mathsf{r}x). \end{array}$$

## Index

### A.2 Source of the Example

```
/** \title{\textit{PIE} Example Document} \date{} \maketitle */

:- use_module(folelim(support_scratch)).
:- set_info_verbosity(50).
:- ppl_set_verbosity(7).

/** \section{Projection and Definientia} */

def(project(S,F)) :: ex2(S1, F) ::-
        mac_free_predicates(F, S2),
        mac_subtract(S2, S, S1).

/* mac_free_predicates/2 and mac_subtract/2 are utility functions
   for use in macro definitions
*/

def(definiens(G,F,S)) ::
project(S, (F , G)) -> ~project(S, (F, ~(G))).

/** \section{Obtaining a Definiens with Interpolation} */

/**
We specify a background knowledge base:
*/

def(kb_1, [syntax=brief]) ::
all(x, ((qx->px), (px->rx), (rx->qx))).

/**
\bigskip

To obtain a definiens of $\exists x\, \f{p}x$ in terms $\{q, r\}$ within
$\mathit{kb_1}$ we specify an implication with the $\mathit{definiens}$ macro:
*/

def(ex_1, [syntax=brief]) ::
definiens(ex(x, px), kb_1, [q,r]).

/**
\bigskip

We now invoke a utility predicate that computes and prints for a given valid
implication an interpolant of its antecedent and consequent:
*/

:- ppl_printtime(ppl_ipol(ex_1)).

/**
\bigskip

Before we leave that example, we take a look at the expansion of
$\mathit{ex}_1$:
*/
```

```
:- ppl_printtime(ppl_form(ex_1, [expand=true])).

/**
And we invoke an external prover (the \textit{E} prover) to validate it:

*/

:- ppl_printtime(ppl_valid(ex_1, [prover=tptp(eprover)])).

/** \section{Obtaining the Strongest Definiens with Elimination} */

/**
The antecedent of the implication in the expansion of $\mathit{definiens}$
specifies the strongest necessary condition of $G$ on $S$ within $F$.  In case
definability holds (that is, the implication is valid), this antecedent denotes
the strongest definiens. In the example it has a first-order equivalent
that can be computed by second-order quantifier elimination.
*/

def(ex_2, [syntax=brief]) ::
project([q,r], (kb_1, ex(x, px))).

:- ppl_printtime(ppl_elim(ex_2)).

:- ppl_set_source.
:- mac_install.
```