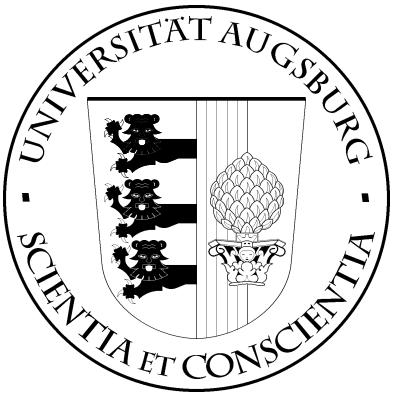
Universität Augsburg

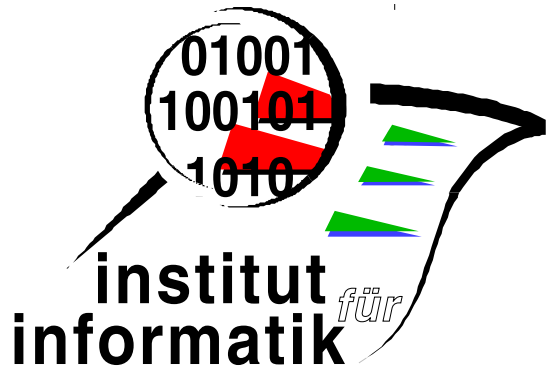


ATPPortal

A User-friendly Webbased Interface for Automated Theorem Provers and for Automatically Generated Proofs

P. Höfner M.E. Müller S. Zeissler

Report 2010-10 September 2010



Institut für Informatik

D-86135 Augsburg

ATPPortal
A User-friendly Web-based Interface for Automatically Theorem Provers and for Automated Generated Proofs

Peter Höfner[1], Martin Eric Müller[2], and Stephan Zeissler[2]

[1] Institut für Informatik, Universität Augsburg
86135 Augsburg, Germany
hoefner@informatik.uni-augsburg.de
[2] Department for Computer Science, University of Applied Sciences Bonn-Rhein-Sieg
53754 St. Augustin, Germany
martin.mueller@h-brs.de, stephan.zeissler@moinz.de

Abstract. This report describes the design, the implementation and the usage of a system for managing different systems for automated theorem proving and automatically generated proofs. In particular, we focus on a user-friendly web-based interface and a structure for collecting and cataloguing proofs in a uniform way. The second point hopefully helps to understand the structure of automatically generated proofs and builds a starting point for new insights for strategies for proof planning.

1 Introduction

Automated theorem proving (ATP) has brought automated reasoning into a wide variety of domains. Examples where significant and important success systems has been achieved with ATP, are software and hardware verification (e.g. [37,13]), software development (e.g. [3]), analysis of network security protocols (e.g. [1,41]) and mathematics (e.g. [52,38]). Full automation (without user interaction) is often only possible by first-order logic. For these tasks ATP systems like Vampire [49] and Prover9 [40] exist. The broad variety of existing ATP systems is due to the multitude of many different proof calculi with many different heuristics guiding proof search. Therefore, ATP systems differ in the derivations found as well as in the way a derivation is constructed. This results in different effective and efficient behaviour when applying theorem provers on concrete problems. However, all off-the-shelf ATP systems have different syntax, different input formats and different outputs.

To overcome these difficulties, `SystemOnTPTP` [53] provides a common language and a common interface for first-order ATP systems. The TPTP library have extensively been used by different users in various case studies covering various areas of sciences (e.g., [9,27,29,30,60]).

Though TPTP with all its components is easy to use for experts, we see two disadvantages:

– For an average user, the TPTP syntax, which is mainly prefix, is difficult to read, to write and to understand. An “unreadable” example is given in

Figure 1, where Back's atomicity refinement law is encoding using Kleene algebra [6,57,29].

```
fof(goals,conjecture,(
    ! [X0,X1,X2,X3,X4,X5] :
      ( ( leq(X0,multiplication(X0,X1))
        & leq(X2,multiplication(X1,X2))
        & multiplication(X1,X3) = zero
        & leq(multiplication(addition(addition(X2,X3),X4),X5),
              multiplication(X5,addition(addition(X2,X3),X4)))
        & leq(multiplication(X1,X5),multiplication(X5,X1))
        & leq(multiplication(X4,X3),multiplication(X3,X4))
        & leq(multiplication(X4,X1),multiplication(X1,X4))
        & star(X4) = strong_iteration(X4) )
     => leq(multiplication(multiplication(X0,
            strong_iteration(addition(addition(addition(X2,X3),
            X4),X5))),X1),multiplication(multiplication(
            multiplication(multiplication(multiplication(X0,
            strong_iteration(X5)),X1),strong_iteration(X4)),X1),
            strong_iteration(multiplication(multiplication(
            multiplication(X2,strong_iteration(X3)),X1),
            strong_iteration(X4))))) ) )).
```

Fig. 1. Back's atomicity refinement law in TPTP

.

- As a consequence of the broad diversity of ATP systems, it is desireable to work with a *multitude* of different ATP systems simultaneously to solve a certain problem when it is unclear to the layman which prover can be expected to deliver best results.
- The generated proofs (if there are any) are not stored and hence cannot be used to learn more about the mathematical structure of the proofs.

The work we present in this report describes a system which tries to fix these issues. We describe the design, the implementation and the usage of a system for managing different systems for automated theorem proving and automatically generated proofs.

The management system for ATP systems allows a common language for all ATP systems (like TPTP). Unlike TPTP a user can define his own format to display formulas and, with small implementation issues, to use it as input and output format. At the moment, formulas can be typed in TPTP-style or in the format of Prover9. Its output is, next to these two formats, MathML, and LaTeX. We hope that the web-based interface enables more users without experience in ATP to use ATP systems.

The produced proofs and some statistics about them are stored in a database in a uniform format. On the one hand, this enables us to transform the proofs in different formats (see above). On the other hand, the stored proofs will help to understand proofs of ATP systems and to analyse proofs. We hope that an analysis yield better ATP systems or an axiom selection system (for details see Section 5).

The report is organised as follows:

In Section 2 we briefly outline the intention and design of the TPTP project. From this, we derive the requirements on which the implemantion of the ATPPortal is based. Section 3 gives a short overview of implementational issues; for details, the reader is asked to consult the appendix. It is followed by a description of some example applications (both in collected data and ATPs integrated) we already have implemented in Section 4. The paper concludes with an extensive outlook of ongoing and future research.

2 Preliminaries

The TPTP World. "The TPTP World is a well known and established infrastructure that supports research, development, and deployment of Automated Theorem Proving systems for classical first-order logics. The TPTP World includes the TPTP problem library, the TSTP solution library, standards for writing ATP problems and reporting ATP solutions, tools for processing ATP problems and solutions, and harnesses for controlling the execution of ATP systems and tools. The TPTP World infrastructure has been deployed in a range of applications, in both academia and industry." [56] The standards within the TPTP World are well established and widely accepted. Moreover the integrated services and tools are a good base for our interface:

`SystemOnTPTP` [53], for example, is a tool that allows to submit problems to a wide range of ATP systems. At the moment we use this utility to connect the ATPPortal with 11 ATP systems. `SystemOnTPTP` itself uses other services provided by the TPTP World like `TPTP2X` or `TPTP4X` [56]. Such tools pre- and postprocess the input and output and give easy feedback.

Requirements. From the point of view of ATP-related research (may it be the development of ATP systems or research using such systems) all we need is a *database* that is organised in a way so we can store all the information we need. But since the ATPPortal is intended to provide an *easy-to-use* interface which hides all the different ATP's pecularities from the user, it has to be a software system that is simple to install and intuitive in its usage.

- User requirements
  1. User Interface
     People have become used to graphical interfaces with point-and-click and drag-and-drop functionality. Theorem provers were (or still are) command-line tools; some are equipped with GUI-Add-Ons. When it

comes down to a quick translation from one syntax into another, the usage of different provers simultaneously, and referring back to aximoatisations that already exist in the ATPPortal-database, one would have to prepare several different input files and write batch scripts to start all involved processes. As a consequence, the ATPPortal shall provide a GUI supporting the point-and-click metaphors for theory and axiom selection.
To ensure that the ATPPortal client can be run on any system, it shall be (and has become) a web-based application.

2. Server-Client architecture
It cannot be taken for sure that every user can install the entire software package on his system. Therefore, it has to be implemented as a Server-Client-System consisting of the client-side GUI, the ATPPortal web server, the underlying database server, and servers hosting all ATP systems that are plugged into the ATPPortal suite.
All parts of the system can be run on arbitrary servers; ranging from one-for-all to one server for each component.

– Implementation requirements
1. Open Source
The ATPPortal is intended for users from various disciplines. Since all of the software (especially all utilised theorem provers) are subject to ongoing research, parts of the system are under continuous development. Also, the ATPPortal is designed as an open system (see below) which benefits from collaboration of many people. Therefore it must not depend on licensed software and, in consequence, is open sofware.
2. OS independency
For the same reason and for universal usability, ATPPortal must be operating system independent. Therefore, all components are chosen so they can be run under virtually all current operating systems (PostgreSQL, Apache Tomcat, etc.). The only exception are the different theorem provers—which, if run under a different operating system are integrated by the client-server-architecture decribed above.
3. JAVA
Again, for the same reason, JAVA is chosen as a programming language. In fact, this is the only "hard" requirement on the system, since everything else is encapsulated in corresponding interfaces and the strict usage of a layered architecture (the difference between using a PostgreSQL-database via a JDBC interface or a plain XML-file as a database is just a matter of a few lines of program code; and the system's functionality does not change at all).

– Maintenance requirements
1. Modularity
The highly modular architecture allows an easy extensibility and maintainability.
2. Interfaces
The ATPPortal lives from its underlying theorem provers which them-

selves—as already mentioned above—are subject to constant development. Therefore, the representation language used within the ATPPortal database is a strict prefix fully bracketed language. In order to add a new theorem prover or change the input/output file format, it just requires the implementation of a new parser specification.

3 Implementation

In [16], Evans presents the method of "Domain Driven (Software) Design". The most important aspect is that any kind of software project can be sliced into four levels resulting in a *layered architecture*, see Figure 2.

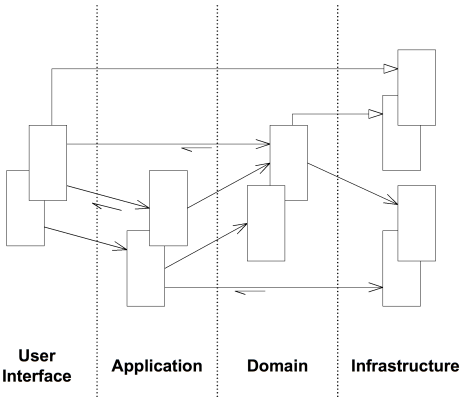


Fig. 2. Layered Architecture Model
%fboxDas Bild ist echt übel; haben wir das in einer besseren Qualität?

At the very bottom, the infra-structure layer contains all the base functionality of the system; these are components and modules that actually handle the control and data flow. Examples are utility or fascade classes that provide interfaces to external code libraries or services.

The second level is called the *domain*-knowledge. It contains (or rather implements) all domain-dependent program logic. The clear distinction between program logic and control results in abstract and adaptable source code.

On top, the *application layer* contains all application-dependent logic. In our context, the domain layer contains everything we need to represent and process theories and proofs; the application layer implements the functionality of the ATPPortal itself (with, e.g., postgres being the concrete database whereas the domain layer only contains an abstract definition of a formula "repository".

Finally, the fourth level is the *user interface level*. It implements the visualisation and interaction that is required for the domain model to be manipulated by the user.

3.1 Used Standards, Models, and Software

ATPPortal Implementation. For the implementation of ATPPortal v.1.0, we have built on several different technologies we list in the following:

- *Java Platform, Enterprise Edition (JEE)* (e.g. [19]) is a Java platform for server programming that differs from the Java Standard Edition Platform (Java SE) in that it provides libraries for the development for fault-tolerant, distributed, multi-tier Java software, based on modular components running on an application server. Especially the components for the application server are used ATPPortal. In our implemention we use the *JEE 6* platform.
- The *Java Servlet.*[3] is a Standard Extension to the Java platform that provides web application developers with a simple consistent mechanism for extending the functionality of a web server. A Servlet is a Java class which conforms a protocol by which a Java class may respond to HTTP requests. It can be used to add *dynamic content* to a Web server using the Java platform. In ATPPortal we use *Java Servlet API 2.5*.
- *Java Server Pages (JSP)* is the Java platform technology for building applications containing dynamic Web content such as HTML, DHTML, XHTML and XML. The JavaServer Pages technology allows to create dynamic content. In ATPPortal we use *JSP 2.1*.
- *Apache Tomcat version 6.0.24* (e.g. [10]) is an open source implementation of the Java Servlet and the JavaServer Pages specifications.
- Since the ATPPortal collects, manages and catalogues data (theorems, axioms and proofs), a database is necessary. For our implementation we use *PostgreSQL, Version 8.4.2,* [47] , an object-relational database management system developed at the University of California at Berkeley Computer Science Department. It is an open-source descendant of this original Berkeley code and supports a large part of the SQL standard.
- *jQuery 1.3.2* [32] is a cross-browser JavaScript library designed to simplify the client-side scripting of HTML, in particular with respect to database connections and queries.
- To provide a simple mechanism to build the open source JAVA-based project, we use *Apache Ant*[4]. It is a software tool for automating software build processes similar to Make but is implemented using the Java language.
- *Apache Commons* [20] provides reusable, open source Java software. The Commons is composed of three parts: proper, sandbox, and dormant.

Theorem Provers. As mentioned in the introduction, one of the main purposes of ATPPortal is to provide a uniform user interface for ATP systems. At the moment one theorem prover (Prover9) is fully connected to our portal, i.e., the original output of Prover9 is parsed and the proofs are stored in our database. Next to that, we integrated 10 ATP systems via the TPTP project.

[3] `http://java.sun.com/products/servlet/`
[4] `http://ant.apache.org/`

This mechanism allows an easy and quick possibility to connect ATP systems with ATPPortal. Theorems can be proved, but a proof is not given with this procedure. Hence we cannot store a proof.

We briefly list the integrated theorem prover systems. The systems description are mainly from [54] and [55].

- *Prover9 0908* [40] is a saturation-based theorem prover for first-order equational logic. It implements an ordered resolution and paramodulation calculus and, by its treatment of equality by rewriting rules and Knuth-Bendix completion.
- *Waldmeister C09a (via TPTP)* [21] is a system for unit equational deduction. Its theoretical basis is unfailing completion in the sense of [5] with refinements towards ordered completion (cf. [4]). The system saturates the input axiomatization, distinguishing active facts, which induce a rewrite relation, and passive facts, which are the one-step conclusions of the active ones up to redundancy. The saturation process is parameterized by a reduction ordering and a heuristic assessment of passive facts [22].
- *iProver 0.7 (via TPTP)* [34] is an automated theorem prover based on an instantiation calculus Inst-Gen [17,35] which is complete for first-order logic. One of the distinctive features of iProver is a modular combination of first-order reasoning with ground reasoning. In particular, iProver currently integrates MiniSat [15] for reasoning with ground abstractions of first-order clauses. In addition to instantiation, iProver implements ordered resolution calculus and a combination of instantiation and ordered resolution, see [35] for the implementation details. The saturation process is implemented as a modification of a given clause algorithm.
- *E 1.1(via TPTP)* [51,50] is a purely equational theorem prover. The core proof procedure operates on formulas in clause normal form, using a calculus that combines superposition (with selection of negative literals) and rewriting. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution. The basic calculus is extended with rules for AC redundancy elimination, some contextual simplification, and pseudo-splitting. The latest versions of E also supports simultaneous paramodulation, either for all inferences or for selected inferences.
- *EP 1.1 (via TPTP)* is just a combination of E 0.999 in verbose mode and a proof analysis tool extracting the used inference steps.
- *Vampire 11.0 (via TPTP)* [49] is an automatic theorem prover for first-order classical logic. It consists of a shell and a kernel. The kernel implements the calculi of ordered binary resolution and superposition for handling equality. The splitting rule in kernel adds propositional parts to clauses, which are manipulated using binary decision diagrams (BDDs). A number of standard redundancy criteria and simplification techniques are used for pruning the search space: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The reduction ordering is the Knuth-Bendix Ordering.

– *Otter 3.3 (via TPTP)* [39] is an ATP system for statements in first-order (unsorted) logic with equality. Otter is based on resolution and paramodulation applied to clauses. An Otter search uses the "given clause algorithm", and typically involves a large database of clauses; subsumption and demodulation play an important rôle.
– *SPASS 3.5f (via TPTP) and SPASS 3.01 (via TPTP)* [59] is an automated theorem prover for full first-order logic with equality and a number of non-classical logics. It is a saturation based prover employing superposition, sorts and splitting [58]
– *Prover9 0908 (via TPTP)* For a description see above. The only difference is that this one uses `SystemOnTPTP`.

3.2 System Architecture

The object model on which the ATPPortal implementation is based is mainly motivated by the underlying entity-relationship model. The focus of our work lies in storage, intuitive access and processing of theories, axiomatisations and proofs. Therefore, it is exactly this domain knowledge (c.f. layered architecture model in Section 3.2) that motivates the database design and, thus, the object model. We first describe the database model.

E/R Model. The E/R-diagram of the underlying database is shown in Figure 3. According to the main purpose of the ATPPortal, the entire design is grouped around three basic concepts:

1. Algebras
2. Formulas
3. Proofs

Algebras are defined in terms of *axioms*, which in turn are just a special kind of formulas. A *proof* consists of a sequence of *theorems*, which are (non-axiomatic) formulas or *goals* (however, goals are not explicitly labelled as such for one goal of a proof can be a lemma in another). The three basic types of entities are stored in tables `algebra`, `formula`, and `proof`. `algebra` consists of an `id`, a `name` (e.g. "HA-HCR"), and a `comment` ("Heyting Algebra; derived from a Horn clause representation (Prolog)"). It is connected to the table `formula` via an $m \times n$-relation `algebra_formula` which defines the set of formulas that belong to an algebra by relating their corresponding `id`s. In addition to this, it contains a boolean flag that is used to mark formulas as an `axiom`.

The table `formula` provides an `id`, a `name`, a `comment`, and of course the formula itself (`formula_text`). Additional information (currently references to publications only) are stored in a separate table `formula_reference` (these are optional; but for compatibility reasons it is desireable to be able to import all the information stored in the TPTP system).

A `proof` does not have names; they only have a running `id` and a `timestamp` which originates from the date when the proof was triggered. Since a proof

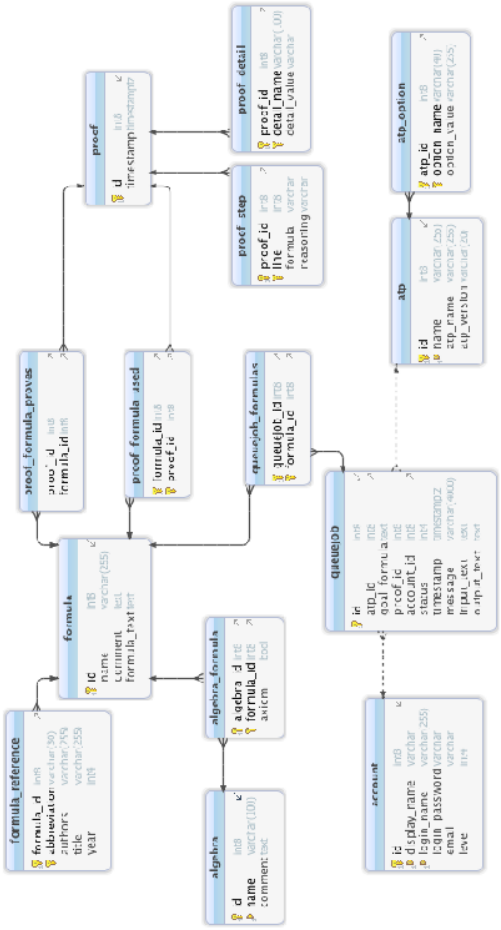


Fig. 3. The ATPPortal database structure

consist of a sequence of formulas, the output of an ATP is parsed line by line where each line is assumed to represent one single step. The results are stored in the tabe `proof_step`. The ATP's output undergoes a rudimentary parsing procedure that results in the `formula` occuring/used in this step and the kind of `reasoning` rule that has been applied. By referring to the `line` counter, proofs can be reconstructed from the database. Each proof has attached possibly several `proof_details`.

The set of formulas one provides when triggering a proof is stored in the `prof_formula_proves`. However, not all the formulas provided are actually *used* within a proof. Therefore, in addition to `proof_step` and `proof_formula_proves`, the table `proof_formula_used` holds information about which formula actually occurs in a proof.

The rest of the database contents mainly concerns administrative tasks:

- The above mentioned problems with varied use of operator names is solved using a "lexicon" of operator names and their proper usage. Internally, the formulas only use standardised, automatically generated functor symbols which, via the tables `operator` and `operator_syntax_format`, are translated (back) into the desired notation (these tables are not shown in the E/R-diagram in Figure).
- The interfacing to the ATPs is also managed by the database. The tables `atp` and `atp_option`. The latter contains information about syntactical pecularities, binary names, execution paths (or remote servers), command-line parameters and/or configuration files for the different ATPs.
- Whenever ATPPortal is used as a frontend to perform proofs, a `queuejob` is started. The user submits a query which by ATPPortal is translated and forwarded to the according thorem prover. The proof requests are listed in a queue which allows to sequentially process a set of requests from possibly many users. Proofs that take longer remain in the queue until finished or cancelled; i.e. the result can be retrieved later in a separate session. Also, a parameter specifying the maximum number of parallel child processes helps to avoid server unresponsiveness due to server overload.
- The `account`-table finally contains all the information required for user administration.

Object Model. The object model of the system implementation reflects the domain model, and, hence, the database structure (see Figure 4):

Every table representing basic concepts (such as algebras, formulas, proofs, etc) corresponds to an entity–class. The relations between such objects (realised as tables in the database) are implemented as queries in repository classes. The advantage is that loading of objects from memory is always explicitely realised by repository classes. This helps to avoid obfuscated loading by way of getter-methods in entity objects.

All repository classes are implemented as abstract as possible; every repository class resides in a package of its own (see the upmost row or level in the UML diagram in Figure).

The advantages are the clear distinction between domain logic and infrastructure (here, e.g., SQL code) and the possibility of using alternative storage solutions (e.g. a flat XML-file storage rather than a postgres database). Every single repository offers a getInstance method, to create/retrieve an instance from the repository. This instance is stored (as a *Singleton-Pattern*), where the instance is generated by an initial call of the RepositoryFactory.

This RepositoryFactory is an abstract *Singleton*. The front end instantiates an implementation of the RepositoryFactory and saves it by the setInstance method in the singleton. By reimplementaion of factories one can add further implementations of repositories.

The class Atp is used to store configuration data for ATPs. In addition to this, we created a factory AtpFactory which upon initialisation also checks the

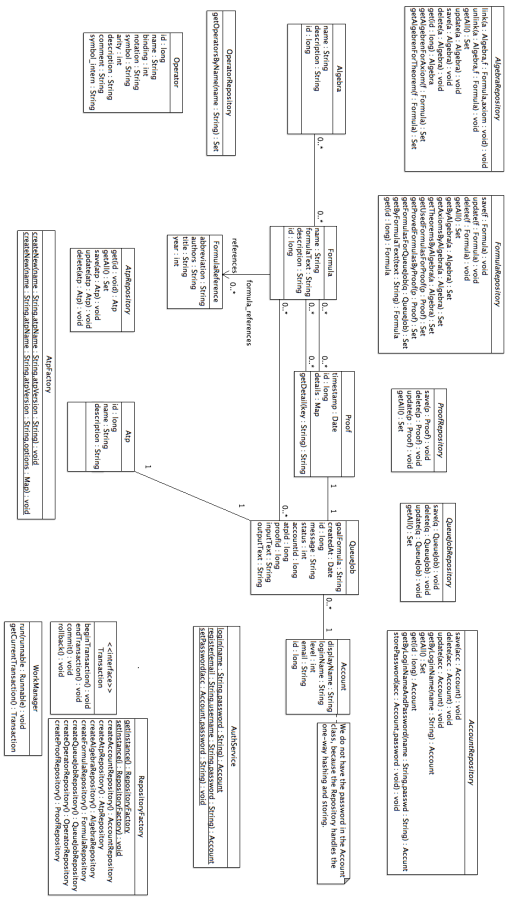


Fig. 4. UML Class diagram

validity of paramaters passed over to the theorem prover. Similarly, AuthService provides methods for user authentication.

The class WorkManager offers an API that allows to change a domain model even during transactions and wich at the same time hides error handling from the user.

```
WorkManager.run(new Runnable() {
  public void run() {
     // Code in here is execute inside of a Transaction
                    }          }
            );
```

The programmer is still able to manually perform a `commit()` or `rollback()` by way of `getCurrentTransaction()`.

Postgres Repositories Similarly, there exists a repository for database interaction. Every repository contains a set of standard methods (save, update, delete und get) corresponding to the functionality of database commands `INSERT`, `UPDATE`, `DELETE` and `SELECT`. There exists a number of additional methods for ATPPortal-specific queries to the proof-data repository; for example, the method `getFormulasForProof` returns all formulas of a proof. The implementation of these methods reside in their own Java-Package so as to force a strict distinction between data model and SQL-query code. The ConnectionFactory manages connections which can be requested by reqositories via `open` (reopsitories also have to properly `close` connections).

```
Connection con = null;
try {
 con = factory.open();  // use the connection

 // ..

} finally {
 if ( con != null ) factory.close(con);
}
```

Connection pooling speeds up database access.

4 Current Applications

Though the design and the implementation of ATPPortal is most flexible and allows any application based on (first-order) ATP systems, the momentarily main focus lies on algebraic and formal methods with applications in computer science.

This is due to the following reasons:

- Simple, first-order algebraic structures, like Boolean algebras, Heyting algebras, relation algebra or Kleene algebras are particular suitable for ATP systems. This has been shown in several case studies [27,28].
- These structures and variants of them have been emerged as fundamental structures in computing. They have been used in various applications ranging from concurrency control [11,23,24] over program analysis and semantics, like Hoare logic [36,42] and the wp-calculus of Dijkstra [43] to hybrid systems. [25]

For the purpose of the paper we present two examples: The first present an algebra for feature-oriented software development, the second implements Heyting algebra and derives properties. Heyting algebras are special partially ordered sets that play a crucial rôle for intuitionistic logic pointless topology.

Feature Algebra. *Feature-Oriented Software Development (FOSD)* is a paradigm that provides formalisms, methods, languages, and tools for building variable, customizable, and extensible software. A *feature* reflects a stakeholder's requirement and is typically an increment in functionality; features are used to distinguish between different variants of a program or software system [33].

Research along different lines has been undertaken to realize the vision of FOSD [33,48,7,12,31,2]. While there are the common notions of a feature and feature composition, present approaches use different techniques, representations, and formalisms. *Feature algebra* [3,26] is such a framework for FOSD. The algebra itself abstracts from details of different programming languages and environments; alternative design decisions in the algebra reflect variants and alternatives in concrete programming language mechanisms. It is based on simple first-order logic and is therefore predestinated to be integrated into ATPPortal.

Heyting Algebra. Boolean algebras (as used for algebraic models of classical, two–valued propositional or first-order logic) are a special case of the more general concept of Heyting algebras (HA). In short, the law of the excluded middle does not generally hold in Heyting algebras which weakens complementation to relative pseudo-complementation. This is a desirable property for algebraic models of intuitionistic logic (IL). The study of IL focuses on the notion of *derivability* rather than provability in the sense that while in full logic it either holds that ϕ or $\neg\phi$, in IL we assume that ϕ is true if $\vdash \phi$ and $\neg\phi$ otherwise. This is closely related to negation as failure as it is utilised in Prolog [18].

Our interest in IL is motivated by reasoning with rough logic, [14,45,46]In the course of theory refinement in machine learning one seeks to find a set H of formulas ("hypothesis") which together with a given theory Φ ("background knowledge") entails a set of positive examples (positive literals) and which does not entail a set of negative examples (negative literals). Then, the most general hypothesis is the relative pseudocomplement of negative examples with respect to Φ while the most specific hypothesis is the relative pseudocomplent of the negated positive examples, [44] The fact that neither $\Phi \vdash \phi$ nor $\Phi \nvdash \neg\phi$ finally expresses that we cannot give a definite answer concerning the validity of ϕ by way of $\vdash$. What then in rough set theoy corresponds to an element in a boundary region is in rough logic a formula that requires a third truth value (which is why the semantics of rough logic is defined in Łukasciewicz algebras).

5 Conclusion and Outlook

ATPPortal is a long term project for maintaining different ATP systems. Its main focus is on a user-friendly interface for arbitrary first-order systems. The design is made such flexible that the integration of new ATP systems as well as new input or output language is easy.

As future work we plan to extend the project in different directions:

- Prolog is not just a programming language but a theorem prover itself. In a first step, we shall integrate Prolog into ATPPortal. Second, Prolog is an

ideal programming environment to implement theorem provers by adapting and controlling Prolog's internal stack-based resolution machine.
- Next to the integration of new theorem provers, there are also engines for counterexample search (e.g., Mace4). To complement the ATP systems integrated in ATPPortal, these systems should be integrated to. Due to the modularity of our systems this should be an easy task.
- A much more complicated task is the integration of higher-order theorem provers. To make them usable for "non-experts", they have to be set up in a fully automated mode. Moreover a new user-intuitive input language should be developed
- A huge collection of proofs of different theorems in different algebras using different theorem provers allows us to compare instances of each of the three sets: If two theorems share a common scheme of proof in all provers and/or algebras, it appears they share a common property. If a certain axiom or theorem occurs several times for different proofs, then this formula can be considered an important axiom. If a certain set of problems can be solved efficiently with the same provers (and fail on another set of provers) then the problems obviously require a re-formulation for different proof methods. And finally, every theorem prover can be configured and fine-tuned by a multitude of different parameters.
  The idea behind applying a proof analysis is to identify similarities behind theories/theorems, proof patterns and to induce heuristics that an be used to adapt provers to problems using according parameters. So far there are only a few systems like SRASS that try to select the "important" axioms.
- Verification and testing Verification of software, protocols, hardware and process specification (workflow) are of increasing importance. One method is "exhaustive testing", also known as model checking. In the context of abstract process specification (may it be software processes or enterprise processes including software, hardware and human staff), an algebraic or logic specification is rather uncommon for two reasons: they are hard to understand (and maintain) and they need to be verified using suitabe proof methods (which, in the business world, are rather uncommon as well). Therefore many such specifications are given in prose, by use of some graphical specification language which lack a proper semantics, or by abstract machine semantics (that is, a rather procedureal semantics rather than a declarative one). Standardisation committees (DIN, IEC, ISO) prefer the latter approaches: there are dozens of modeling languages for dozens of different specific purposes. Since a clear semantics is missing but required by a proper verification, the committees also favour procedural semantic approaches. Finally, there is one simple reason why automated verification is a hard business: There are many theorem provers around—and for nearly all problems there is a suitable one. The problem is that it is hard to tell which prover is the right one for a certain problem. The ATPPortal provides a very simple testbed by which one could give it a try and endeavour a verification by parallel execution of many provers—hoping that atleast one will return an answer soon enough (c.f. [8]).

Acknowledgements: We are grateful to Han-Hing Dang and Sarah Edenhofer, Roland Glück and Markus Teufelhart for implementing parts of ATPPortal and to Bernhard Möller for many fruitful remarks. We would also like to thank Geoff Sutcliffe for his permanent support over the last years concerning TPTP.

References

1. M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM*, 2004.
2. S. Apel, T. Leich, and G. Saake. Aspectual feature modules. *IEEE Transactions on Software Engineering*, 34(2):162–180, 2008.
3. S. Apel, C. Lengauer, B. Möller, and C. Kästner. An algebraic foundation for automatic feature-based program synthesis and architectural metaprogramming. *Science of Computer Programming*, 2009. (to appear).
4. J. Avenhaus, T. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *Journal of Symbolic Computation*, 36(1-2):217–233, 2003.
5. L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, pages 1–30. Academic Press, 1989.
6. R.-J. Back. A method for refining atomicity in parallel algorithms. In E. Odijk, M. Rem, and J.-C. Syr, editors, *Parallel Architectures and Languages Europe*, volume 366 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 1989.
7. D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 187–197. Proceedings of the IEEE, 2003.
8. E. Börger and B. Thalheim. Modeling workflows, interaction patterns, web services and business processes: The asm-based approach. In *Proceedings of the 1st international conference on Abstract State Machines, B and Z*, number 5238 in Lecture Notes in Computer Science, pages 24–38. Springer, 2008.
9. J. Bos. Applied Theorem Proving - Natural Language Testsuite. http://www.coli.uni-sb.de/ bos/atp/, 2000.
10. J. Brittain and I. Darwin. *Tomcat: The definitive guide, 2nd edition.* O'Reilly, 2007.
11. E. Cohen. Using Kleene algebra to reason about concurrency control. Technical report, Telcordia, 1994.
12. K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000.
13. M. Das. Formal specifications on industrial-strength code - from myth to reality. In T. Ball and R. Jones, editors, *Computer Aided Verification*, number 4144 in Lecture Notes in Computer Science, page 1. Springer, 2006.
14. I. Düntsch. A logic for rough sets. *Theoretical Computer Science*, 179(1-2):427–436, 1997.
15. N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Conference on Theory and Applications of Satisfiability Testing*, number 2919 in Lecture Notes in Computer Science, pages 502–518. Springer, 2004.

16. E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.
17. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In P. Kolaitis, editor, *IEEE Symposium on Logic in Computer Science*, pages 55–64, 2003.
18. A. Gomolko. Negation as inconsistency in prolog via intuitionistic logic. In E. Börger, Y. Gurevich, and K. Meinke, editors, *7th Workshop on Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 128 – 138. Springer, 1993.
19. A. Goncalves. *Beginning Java EE 6 Platform with GlassFish 3: From Novice to Professional*. Apress, 2009.
20. V. Goyal. *Using the Jakarta Commons, Part 1*. O'Reilly, 2003.
21. T. Hillenbrand. Citius altius fortius: Lessons learned from the theorem prover waldmeister. In I. Dahn and L. Vigneron, editors, *Proceedings of the 4th International Workshop on First-Order Theorem Proving*, number 86.1 in Electronic Notes in Theoretical Computer Science, 2003.
22. T. Hillenbrand, A. Jaeger, and B. Löchner. Waldmeister - improvements in performance and ease of use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 232–236. Springer, 1999.
23. C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In M. Bravetti and G. Zavattaro, editors, *CONCUR 09 — Concurrency Theory*, volume 5710 of *Lecture Notes in Computer Science*, pages 399–414. Springer, 2009.
24. C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Foundations of concurrent Kleene algebra. In R. Berghammer, A. Jaoua, and B. Möller, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *Lecture Notes in Computer Science*. Springer, 2009.
25. P. Höfner. *Algebraic Calculi for Hybrid Systems*. Books on Demand GmbH, 2009.
26. P. Höfner and B. Möller. An algebra of hybrid systems. *Journal of Logic and Algebraic Programming*, 78:74–97, 2009.
27. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig, editor, *Automated Deduction — CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 279–294. Springer, 2007.
28. P. Höfner and G. Struth. On automating the calculus of relations. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning (IJCAR 2008)*, volume 5159 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2008.
29. P. Höfner, G. Struth, and G. Sutcliffe. Automated verification of refinement laws. *Annals of Mathematics and Artificial Intelligence, Special Issue on First-order Theorem Proving*, pages 35–62, 2008.
30. A. Hommersom, P. Lucas, and P. van Bommel. Automated Theorem Proving for Quality-checking Medical Guidelines. In G. Sutcliffe, B. Fischer, and S. Schulz, editors, *Workshop on Empirically Successful Classical Automated Reasoning*, 2005.
31. D. Hutchins. Eliminating distinctions of class: Using prototypes to model virtual classes. In P. L. Tarr and W. R. Cook, editors, *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006*, pages 1–20. ACM Press, 2006.
32. jQuery. `http://docs.jquery.com/Release:jQuery_1.3.2`.
33. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University Software Engineering Institute, 1990.

34. K. Korovin. Implementing an instantiation-based theorem prover for first-order logic. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 63–63, 2006.
35. K. Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *4th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence. Springer, 2008.
36. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000.
37. W. Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall, 2005.
38. W. McCune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.
39. W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.
40. W. W. McCune. Prover9 and Mace4. <`http://www.cs.unm.edu/~mccune/prover9`>. (accessed October 31, 2010).
41. J. Mitchell. Security analysis of network protocols: Logical and computational methods. In P. Barahona and A. Felty, editors, *Principles and Practice of Declarative Programming*, ACM SIGPLAN Notices, pages 151–152, 2005.
42. B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.
43. B. Möller and G. Struth. WP is WLP. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science*, volume 3929 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2006.
44. M. E. Müller. Modalities, relations, and learning. In R. Berghammer, B. Möller, and A. Jaoua, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2009.
45. A. Nakamura. A rough logic based on incomplete information and its application a rough logic based on incomplete information and its application. *International Journal of Approximate Reasoning*, 15:367–378, 1996.
46. E. Orlowska. Reasoning with incomplete information: Rough set based information logics. In *Proceedings of the SOFTEKS Workshop on Incompleteness and Uncertainty in Information Systems*, pages 16–33, 1993.
47. The PostgreSQL Global Development Group, http://www.postgresql.org/files/documentation/pdf/8.4/postgresql-8.4.5-A4.pdf. *PostgreSQL 8.4.5 Documentation*, 2009.
48. C. Prehofer. Feature-oriented programming: A fresh look at objects. In *European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 419–443. Springer, 1997.
49. A. Riazanov and A. Voronkov. The design and implementation of vampire. *AI Communications*, 15(2-3):91–110, 2002.
50. S. Schulz. A comparison of different techniques for grounding near-propositional cnf formulae. In S. Haller and G. Simmons, editors, *15th International FLAIRS Conference*, pages 72–76. AAAI Press, 2002.
51. S. Schulz. E: A brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002.
52. J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers and Mathematics with Applications*, 29(2):115–132, 1995.

53. G. Sutcliffe. System description: SystemOnTPTP. In D. McAllester, editor, *Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 406–410. Springer, 2000.
54. G. Sutcliffe. Proceedings of the CADE-16 ATP System Competition, 2008.
55. G. Sutcliffe. Proceedings of the CADE-17 ATP System Competition, 2009.
56. G. Sutcliffe. The TPTP World — Infrastructure for automated reasoning. In *Logic for Programming and Automated Reasoning*, Lecture Notes in Artificial Intelligence. Springer, 2010. (to appear).
57. J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1-2):23–45, 2004.
58. C. Weidenbach. Combining superposition, sorts and splitting. *Handbook of Automated Reasoning*, pages 1965–2013, 2001.
59. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS Version 3.5. In R. Schmidt, editor, *Automated Deduction*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.
60. A. Wojcik. Formal Design Verification of Digital Systems. In *20th Design Automation Conference*, 1983.