Integrated Proof Transformation Services

Jürgen Zimmer[1]   Andreas Meier[2]   Geoff Sutcliffe[3]   Yuan Zhang[3]

[1] School of Informatics, University of Edinburgh, Scotland
jzimmer@inf.ed.ac.uk, http://www.mathweb.org/~jzimmer
[2] DFKI GmbH, Saarbrücken, Germany
ameier@dfki.de, http://www.ags.uni-sb.de/~ameier
[3] Department of Computer Science, University of Miami, USA
{geoff@|yuan@mail.}cs.miami.edu, http://www.cs.miami.edu/~geoff

Abstract

In the last few decades a large variety of mathematical reasoning tools, such as computer algebra systems, automated and interactive theorem provers, decision procedures, etc. have been developed and reached considerable strength. It has become clear that no single system is capable of providing all types of mathematical services, and that systems have to be combined for ambitious mathematical applications. Unfortunately, many mathematical reasoning systems use proprietary input and output formats, and the output in these system-specific formats is often incomprehensible to other components and human users. Transformation tools and data-exchange formats are necessary in order to combine systems and to grant common access to mathematical content. This paper describes the integration of several proof transformation tools in a Java agent architecture, their description in a mathematical service description language, and their combination via a brokering mechanism. The applicability of the approach is demonstrated with an example from group theory.

1   Introduction

In the last few decades a large variety of mathematical reasoning tools, such as computer algebra systems, automated and interactive theorem provers, decision procedures, etc. have been developed and reached considerable strength. Diverse repositories of formalized mathematics have also emerged, e.g., [19]. Despite some successful applications of these systems, none of them have scaled up to a mathematical assistant system providing all kinds of mathematical services. The vision of a powerful mathematical assistant environment that provides supports for most tasks of a working mathematician has recently come into focus, stimulating new projects and international research networks across the disciplinary and systems boundaries. Examples are the European CALCULEMUS [12] (integration of computation and deduction) and MKM [4] (mathematical knowledge management) initiatives, and the American QPQ [14] repository of deductive tools. A main goal in these initiatives is to bring together approaches from different directions. It has become clear that no single system is capable of providing

all types of mathematical services, and that systems have to be combined for ambitious mathematical applications. For example, subgoals of one component are commonly delegated to other specialist components, such as automated theorem proving (ATP) systems (e.g., see [27, 6]). Unfortunately, many mathematical reasoning systems use proprietary input and output formats, and the output in these system-specific formats is often incomprehensible to other components and human users. Transformation tools and data-exchange formats are necessary in order to combine systems and to grant common access to mathematical content. This holds, in particular, for the output from ATP systems, because their output often reflects the peculiarities of the internal calculus and proof search procedure.

This paper describes the integration of several proof transformation tools in a Java agent architecture, their description in a mathematical service description language, and their combination via a brokering mechanism. The proof transformation tools provide the following functionality: 1) The Otterfier system translates arbitrary first-order resolution proofs into resolution proofs whose inference steps use only the inference rules of the Otter system [15]. 2) The Tramp system [16] translates problems in full first order form (FOF) logic to equisatisfiable clause normal form (CNF), and translates resolution proofs into proofs in the Natural Deduction (ND) calculus at the assertion level. 3) The *P.rex* system [10] translates ND proofs into natural language. Ideas from Semantic Web research are being adopted to express the functionality of these tools in the service description language MSDL [3]. A brokering mechanism is then used to combine the proof transformation services, to provide customized compound services that are capable of answering queries from other reasoning systems or human users. For example, given a conjecture in classical first-order predicate logic, a proof assistant system could ask for a resolution or ND proof of the conjecture. A human user of the proof assistant may extend the query to request a natural language version of the proof.

The transformation tools and their combination depends heavily on the newly emerging TSTP data-exchange format for problems and proofs [23]. Specifically useful for this work, TSTP defines a syntax for problems in FOF and in CNF. and a format for resolution style derivations. A refutation in TSTP contains *initial clauses*, i.e. the clauses given to an ATP system or produced by its CNF generator, and the *derived clauses* together with the inference rules used to derive them.

This paper is structured as follows: First the various systems involved are introduced in section 2. Section 3 presents the services offered by the systems, and the brokering mechanism that combines these services. The integration approach is explained with a sample application in section 4. Finally, section 5 concludes with some discussion of related and future work.

2   The Systems Involved

2.1   Automated Theorem Proving Systems

This work makes use of the ATP systems Otter and EP. Otter [15] is designed to prove theorems stated in first-order logic with equality. Otter's inference rules are based on resolution and paramodulation, and it includes facilities for term rewriting, term orderings, Knuth-Bendix completion, weighting, and strategies for directing and restricting

searches for proofs. Otter is particularly interesting for our application because the inference rules used in Otter refutation proofs are quite limited, and Otter proofs can therefore readily be used by the proof transformation system Tramp (see section 2.3). A modified version of Otter is also used in Otterfier to transform resolution proofs (see section 2.2).

EP is an equational theorem prover, combining the E system [21] with a proof analysis tool for extracting the required inference steps. The calculus of EP 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. On the one hand, EP is typically much stronger than Otter when proving theorems in automatic mode (cf. CADE system competitions 15 and 16). On the other hand, EP uses rather complex inference rules which makes its proofs hard to process with other systems (e.g., Tramp - see section 2.3).

2.2   Otterfier - A CNF Derivation Transformer

The derivations (typically refutations) output by contemporary CNF based ATP systems are built from inference steps, which have one or more parent clauses and one resultant inferred clause. The inference rules that create the steps vary depending on the ATP system, ranging from simple binary resolution through to complex rules such as superposition [2]. In almost all cases the inferred clauses are logical consequences of their parent clauses, the most common exception being clauses resulting from the various forms of splitting that have been implemented in ATP systems such as Vampire [18], E, and SPASS [24]. While a wider range and complexity of inference rules typically improves the performance of ATP systems, it is impractical to require proof postprocessing tools to be able to process inference steps created by all the various rules (and new ones that may be invented in the future). It is therefore desirable to be able to transform derivations so that each inference step uses one of a limited selection of inference rules that are amenable to a range of postprocessing operations. The Otterfier system is a transformation tool that transforms a *source derivation* containing *source inference steps* of logical consequence, to a derivation whose inference steps use only selected inference rules available in Otter. The transformation is independent of the inference rules used in the source inference steps, relying only on the inferred clauses being logical consequences of their parent clauses.

Otterfier uses a modified version of Otter. The standard Otter system includes the *hints* strategy. Hints are normally used as a heuristic for guiding the search, in particular, in selecting the given clauses and in deciding whether to keep derived clauses. The fast version of the hints strategy, called *hints2* in Otter, allows the user to specify a set of clauses against which newly inferred clauses are tested for subsumption. For Otterfier the hints strategy has been modified so that when a newly inferred clause is equal to or subsumes a hint, the search is halted and the derivation of the newly inferred clause is output. This modified strategy is called the *target* strategy.

The basic mechanism of Otterfier is to place the parents of a source inference step into Otter's set-of-support list, and the inferred clause into Otter's hint list. The inferred source clause is called the *target clause* in this context. Otter is then run with a complete selection of inference rules, e.g., binary resolution, factoring, and paramodula-

3

tion. As Otter derives the logical consequences of the parent clauses, the target strategy checks each logical consequence against the target clause in the hints list. When the target clause is derived or subsumed, the derivation output by Otter provides a transformed version of the source inference step, using only Otter's inference rules. A source derivation is transformed by performing this transformation on each source inference step, and the combined transformed steps form a complete transformed derivation.

The search strategy of the modified Otter is the default strategy of the normal Otter, i.e., aimed at finding a refutation of the input clauses. If the parent clauses of a source inference step are satisfiable, as are the parents of a source inference step in most cases, then no refutation can be found (see below for the case when the parent clauses are unsatisfiable). In this situation Otter derives clauses with a focus on clauses with lower symbol count. As the number of clauses with a given symbol count is finite, Otter derives longer and longer clauses as its search progresses, eventually deriving clauses whose length is that of the target clause. By that stage the target clause can be derived or subsumed. While Otter can be configured to be refutation complete, it is not known to be deduction complete, i.e., it is possible that the target clause may never be derived or subsumed. In practice this possibility has not yet been encountered, and if it does occur it will merely be a cause of incompleteness of the transformation process.

There are several special outcomes from the target strategy that allow Otterfier to optimize the transformed derivation. First, if the clause derived by Otter subsumes the target clause (rather than being only equal to the target clause), Otter's derived clause replaces the corresponding parent clause in subsequent source inference steps. This maintains the coherency of the transformed derivation. Second, the clause derived by Otter might subsume the inferred clause of a subsequent source inference step of which the current target clause is a parent or ancestor. In this situation the subsequent inference step is removed from the source derivation, and Otter's derived clause replaces the inferred clause of the subsequent inference step. Third, if the source derivation is a refutation, i.e., ends with an empty clause, and Otter derives the empty clause while searching for a non-empty target clause, then a transformed refutation is created, consisting of only the transformed inference steps that end at Otter's derived empty clause. Both the second and third special cases allow Otterfier to produce a transformed derivation that is, in some sense, shorter than the source derivation. The discovery of such derivations is of interest in it's own right [25].

2.3   Tramp - A Natural Deduction Proof Generator

The Tramp system [16] can transform the output of several automated theorem provers into natural deduction proofs at the assertion level [13]. The assertion level allows for human-oriented macro-steps justified by the application of theorems, lemmas, or definitions, which are collectively called *assertions*. For instance, the assertion level step

$$\frac{F \subset G \qquad c \in F}{c \in G} \subset \text{DEF}$$

derives the conclusion $c \in G$ by an application of the subset definition $\subset$DEF — formalized by $\forall S_1.\forall S_2.(S_1 \subset S_2 \Leftrightarrow \forall x.(x \in S_1 \Rightarrow x \in S_2))$— from the premises $c \in F$

and $F \subset G$. A corresponding base ND proof, including the expansion of the subset definition, consists of a lengthy sequence of ND steps.

Tramp consists of a set of transformation procedures. First, there are transformation procedures that take a proof output from an ATP system and transform it into an internal resolution proof object. In its current distribution Tramp is able to process the output of the ATP systems SPASS, Bliksem, Otter, Waldmeister, ProTeIn, and EQP. At the heart of Tramp is a transformation procedure that creates an ND proof at the assertion level. Finally, the resulting ND proof at the assertion level can be further processed. In particular, each assertion application can be expanded such that the resulting proof is a pure ND proof without assertion application steps. Tramp can output its proofs in LaTeX format as well as in the languages $\mathcal{POST}$ and Twega (cf. section 2.4).

The original Tramp system could take only one input: a description of a FOF problem in $\mathcal{POST}$ syntax. Tramp then translated the FOF problem into its equivalent CNF and called one of the supported ATP systems. The result of the ATP system was then converted into Tramp's internal format and an ND proof for the original conjecture was created. The reason for the CNF creation within Tramp is that, in order to create a ND proof, Tramp has to establish a connection between the literals of a resolution proof and the corresponding literal sub-formulae in the original FOF problem.

In order to employ Tramp in the integrated proof transformation system, it was necessary to extend Tramp in three ways: (1) A new input module for TSTP resolution proofs was developed. Currently, this TSTP module exists in parallel with the transformation procedures for the ATP systems supported by Tramp. However, TSTP will eventually become the only necessary input format of Tramp. (2) Tramp now accepts two inputs: the FOF description of the original conjecture and a TSTP resolution proof of the conjecture. Tramp tries to map the literals of the initial clauses in the resolution proof to the corresponding literal sub-formulae of the original first-order formulae. However, Tramp can compute this mapping only if the initial clauses of the resolution proof comply with Tramp's clause normal form (CNF) algorithm (see also section 3). If this is the case Tramp produces an ND proof of the FOF problem. (3) A FOF problem generating procedure was added. This is activated if no FOF problem is provided as input, or if Tramp cannot compute the relationship between the literals of the resolution proof and the literal sub-formulae of the FOF problem. The procedure computes a FOF problem description that corresponds to the initial clauses in a given resolution proof, by creating a disjunction of the literals in each clause and universally quantifying all variables. Since the transformation procedure cannot distinguish between Skolem functions and other functions, it interprets every function symbol as a function of the input signature, and does not create any existential quantifications.

The extensions are important for the use of Tramp in the integrated transformation system: The first allows Tramp to work on TSTP proofs and to be coupled with other reasoning systems, such as Otterfier. The second makes Tramp independent of the ATP system used. The third reduces the necessary input and enables a broader application of Tramp.

2.4 *P.rex* - A Natural Language Proof Generator

P.rex [10] is an interactive natural language proof explanation system for machine-generated proofs. It adapts its explanations to the user and can interact with the user by questions and requests [9]. In the context of the integrated transformation system the full functionality of *P.rex* is not exploited – it is used only to obtain a single natural language explanation of a proof.

P.rex is based on a logical framework and uses the formal language TWEGA for inputting proofs. Since Tramp can output TWEGA format, Tramp's ND proofs can be further processed by *P.rex*. A proof is handled by *P.rex* in two steps: First, a dialog plan is created, and this is then passed to a presentation component that creates a natural language presentation [10]. *P.rex* can provide its natural language explanation in ASCII, LaTeX, and a markup language similar to HTML.

The (quality of the) output of *P.rex* depends on the availability of linguistic knowledge. Linguistic knowledge is stored in a database that is structured in theories. In order to make use of the linguistic knowledge the theory to which the problem belongs has to be provided as the second input to *P.rex*. This feature is not yet used in the integrated transformation system.

3   Combination of Transformation Services

A framework for describing the capabilities of deduction systems in a formal service description language was introduced in [26]. These "semantic" descriptions can be used for service discovery and brokering, i.e., the search for services suitable for tackling a given problem. A *broker* can also use the service descriptions to dynamically combine systems to solve a given problem. Human users or reasoning systems can simply send queries to a broker and wait for a result. Following this idea, some of the functionality of the systems described in section 2 have been captured in the service description language MSDL [3]. This section briefly describes the ontology underlying the service descriptions, the descriptions themselves, and the brokering mechanism.

3.1   An Ontology for Service Descriptions

Semantic descriptions of services offered by deductive components depend on a commonly agreed ontology. We are currently developing such an ontology in the Web Ontology Language (OWL) [8]. Only a small fragment of the ontology, as needed to describe the services offered by the systems of section 2, is described here - to increase readability most properties are not shown. Figure 1 shows the "is-a" (subclass) relationship between some concepts of the ontology as solid lines with arrows. Properties and their cardinality restrictions are denoted with dashed lines. Individuals (instances) are connected to their concepts by dotted lines.

The concept Proving-Problem is crucial for this paper. In addition to the axioms and the conjecture that ought to be proven, a Proving-Problem contains information such as the logic in which the problem is defined and resource limits. A Proving-Problem can be further specialized to a FO-Proving-Problem which is formulated in first-order predicate logic, and then to TSTP problems in CNF or FOF syntax [23].
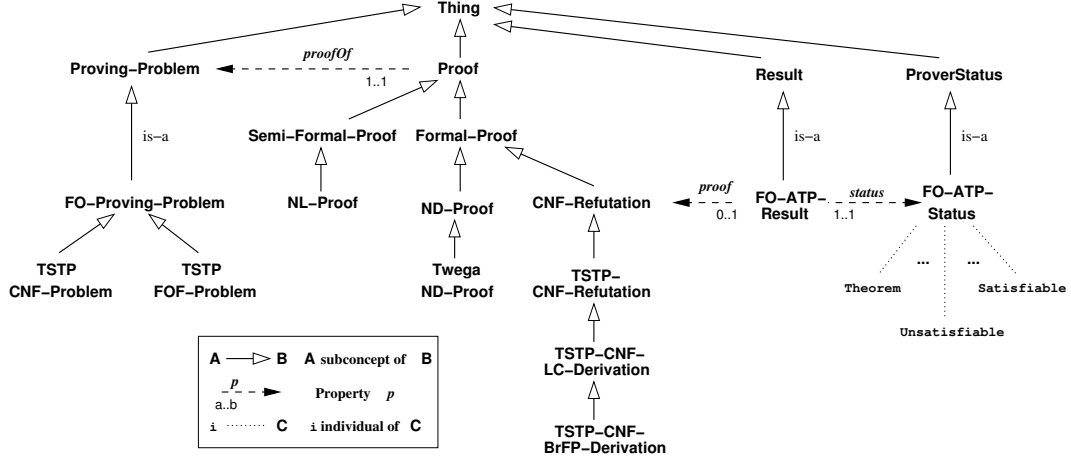
Figure 1: A fragment of the ontology for deduction services

Another crucial concept for this paper is FO-ATP-Result, which denotes results of first-order ATP systems. The *status* of a FO-ATP-Result always contains one of the valid statuses of first-order ATP systems as described in [23]. This status describes unambiguously what an ATP system has established about the given problem, e.g., the status `Unsatisfiable` means that the system has established that the given set of clauses is unsatisfiable. A FO-ATP-Result can also have a *proof* property, which can contain at most one proof of a given conjecture (the range of the property *proofOf* is the concept Proof). Certain domain-specific consistency rules apply such as, for instance, that a FO-ATP-Result must not contain a *proof* if its *status* is `Unknown`.

The concept of Proofs subsumes Semi-Formal-Proofs, e.g., natural language proofs, and Formal-Proofs in different logical calculi, e.g., ND or resolution calculus. Twega-ND-Proofs are special ND-Proofs in the TWEGA language. TSTP-CNF-Refutations are refutation proofs in TSTP format. A TSTP-CNF-LC-Refutation employs only inference rules that produce logical consequences. The latter can be further restricted to a TSTP-CNF-BrFP-Refutation which employs only binary resolution, paramodulation, and factoring.

3.2 Proof Transformation Services

The projects *MathBroker* [20] and MONET [5] have developed the Mathematical Service Description Language (MSDL) [3] to semantically describe reasoning services on the Semantic Web. Although MSDL aims at describing all kinds of mathematical services, the two projects have, so far, investigated only the description of symbolic and numeric computation services. We are using our expertise in deduction systems to extend the use of MSDL to deduction services.

An MSDL document describes many different facets of a service. In what follows we present only the facet important for this paper, namely the abstract mathematical problem that can be solved.[1] As an example a generic first-order theorem proving

---

[1]Those parts of the MSDL description needed for further classification of the service and for the

service, GenericATP, is presented. GenericATP is provided by an ATP system such as EP. To increase readability, the service description is presented in a table rather than in the XML syntax of MSDL.

Service: GenericATP	
input parameters:	*problem*::TSTP-CNF-Problem
output parameters:	*result*::FO-ATP-Result
pre-conditions:	⊤
post-conditions:	*proof(?result, ?proof)* ⇒ *type(?proof, TSTP-CNF-Refutation)*

GenericATP has one input, a clause normal form of a conjecture in TSTP format (TSTP-CNF-Problem), and one output, a FO-ATP-Result. A '::' is used to separate the name of a parameter (e.g., *result*) from the RDF type information in the MSDL description (e.g., FO-ATP-Result). It is important to note that GenericATP always delivers a FO-ATP-Result after the given time resource[2] is used up. However, the result might not contain a proof. The **pre-conditions** of an MSDL service state service-specific conditions the input parameters have to fulfill. The **post-conditions** can give further information about the output parameters and can relate input and output parameters. At the moment, pre- and post-conditions may contain RDF triples on concept properties, conjunctions of triples, and Horn clauses. GenericATP has no pre-conditions - they are simply set to ⊤. Its post-conditions say that if the result contains a proof then it is a CNF refutation proof in TSTP format.

The following paragraphs describe the services provided by the systems introduced in section 2.

The Otterfier Service. OtterfierService takes the result of any ATP system and tries to transform the refutation proof in it, if existent, into a TSTP-CNF-BrFP-Refutation which contains only applications of binary resolution, factoring and paramodulation (the BrFP calculus). The fact that Otterfier is based on a modified Otter justifies that the service returns a FO-ATP-Result:

Service: OtterfierService	
input parameters:	*oldResult*::FO-ATP-Result
output parameters:	*newResult*::FO-ATP-Result
pre-conditions:	⊤
post-conditions:	*proof(oldResult, ?oldProof)* ∧ *proof(newResult, ?newProof)* ∧ *type(?newProof, TSTP-CNF-BrFP-Refutation)* ∧ *altProof(?newProof, ?oldProof)*

The post-conditions of OtterfierService express that the newly generated proof is a TSTP-CNF-BrFP-Refutation, and is an alternative refutation proof of the same conjecture.[3]

---

service grounding, i.e. low-level details about how to invoke the service, are omitted.

[2] The time resource is a property of the concept Proving-Problem and, hence, also of the TSTP-CNF-Problem input of the service.

[3] As a side effect, this equivalence provides a semantic verification of the original refutation.

Services offered by Tramp. Since GenericATP accepts only CNF problems, a clause normalization service is needed for problems in FOF format. This is provided by Tramp, which ensures that the resulting CNF is compatible with Tramp's ND proof generation routines. Thus, the first service of Tramp transforms a FOF problem into a CNF:

Service: ClauseNormalizer	
input parameters:	*fofProblem*::TSTP-FOF-Problem
output parameters:	*cnfProblem*::TSTP-CNF-Problem
pre-conditions:	⊤
post-conditions:	*sat-equiv*(fofProblem, cnfProblem)

The fact that the new CNF problem is satisfiability-equivalent to the initial FOF problem is expressed in the post-conditions of ClauseNormalizer.

The second service of Tramp expects two inputs: a FOF problem in TSTP format, and the result of an ATP system. Furthermore, the result of the ATP system should contain a TSTP-CNF-BrFP-Refutation proof. If Tramp can match the literals in the refutation's initial clauses with literal sub-formulae in the FOF problem, then the service returns an ND proof for the FOF problem. The service fails if the initial clauses of the refutation proof are not compatible with Tramp's CNF generator. It is very difficult though to express this constraint in the pre-conditions of the service. It is therefore kept implicit.

Service: NDforFOF	
input parameters:	*fofProblem*::TSTP-FOF-Problem
	atpResult::FO-ATP-Result
output parameters:	*ndProof*::Twega-ND-Proof
pre-conditions:	*proof(atpResult, ?proof)* ∧
	type(?proof, TSTP-CNF-BrFP-Refutation)
post-conditions:	*proofOf(ndProof, fofProblem)*

Tramp's third service takes the result of an ATP system, which should contain a TSTP-CNF-BrFP-Refutation proof. Tramp internally creates a first-order problem from the initial clauses in the proof, and transforms the refutation proof into a ND proof for this problem. The service returns the ND proof as well as the newly generated FOF problem:

Service: NDforCNF	
input parameters:	*atpResult*::FO-ATP-Result
output parameters:	*ndProof*::Twega-ND-Proof
	fofProblem::TSTP-FOF-Problem
pre-conditions:	*proof(atpResult, ?proof)* ∧
	type(?proof, TSTP-CNF-BrFP-Refutation)
post-conditions:	*proofOf(ndProof, fofProblem)*

The P.rex Service. As mentioned above, only some of the functionality of *P.rex* is used, to produce a natural language proof in LaTeX. The *P.rex* service gets a TWEGA proof in the ND calculus and provides a natural language proof in LaTeX format:

Service: PrexND2NL	
input parameters:	*ndProof*::Twega-ND-Proof
output parameters:	*nlProof*::NL-Proof
pre-conditions:	$\top$
post-conditions:	*proofOf(ndProof, ?p)* $\wedge$ *informalProofOf(nlProof, ?p)*

The post-conditions state that the result is an informal natural language proof of the conjecture proved by the ND proof.

3.3   Brokering of Proof Transformation Services

Using the service descriptions introduced above the broker can combine them to obtain customized compound services. The broker currently translates MSDL service descriptions into plan operators. Queries are translated into an initial state of a modified partial order planner. The planner uses the plan operators to come up with a suitable combination of services that might answer the given query. These partially ordered plans are linearized and translated into an execution protocol. For instance, if a proof assistance system comes up with an open conjecture in first-order logic, it can encode it in TSTP FOF format and ask the broker to deliver an ND proof for the conjecture. Assuming some ATP service (GenericATP), the OtterfierService, and the two Tramp services are available, the broker can then simply combine the four services. Figure 2 shows the resulting combination. Some post-conditions of the services are also shown with dashed arrows.
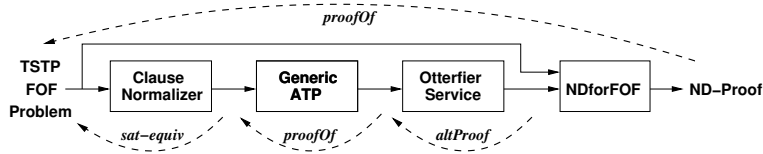

Figure 2: A combination of four services

4   Example Scenario

In this section some possible combinations of the transformation services are demonstrated with an example. Imagine that there are three users of our system, Peter, Susan and Mary. They all want to prove the following problem from group theory:[4]

> *Let F be a group and U a subset of F. Moreover, for U the so-called subgroup-criterion holds: if X, Y belong to U, then $X \circ Y^{-1}$ belongs to U (where $\circ$ is the operation and $^{-1}$ is the inverse function of the group F). Then, U is closed wrt. to the inverse function of F, i.e., U contains X^{-1} whenever it contains X.*

All three users managed to formalize the problem as FOF formulae. They could then use more advanced tools like the Java OPENMATH editor (JOME) [7] to input the

---

[4]Problem GRP006 in the TPTP library [22], although a slightly different formalization is used here.

formulas typing a Maple like syntax. The resulting OpenMath formulas could then be translated automatically into TSTP format. The TSTP problem descriptions could, among others, contain the following formulae:

```
fof(subset,axiom, ( ! [SG,G] :
            ( subset(SG,G)
                <=> ! [X] : ( member(X,SG) => member(X,G) )))).
fof(inverse,axiom, ( ! [G,X] :
        ( ( group(G) & member(X,G) )
            => ( member(inverse(G,X),G)
                & equal(multiply(G,X,inverse(G,X)),identity(G))
                & equal(multiply(G,inverse(G,X),X),identity(G)))))).
                    . . .
fof(f_group,hypothesis, ( group(f) )).
fof(u_subset,hypothesis, ( subset(u,f) )).
fof(subgroupcriterion,hypothesis, ( ! [X,Y] :
            ( (member(X,u) & member(Y,u))
              => member (multiply(f,X,inverse(f,Y)),u) ))).
fof(prove_this,conjecture, ( ! [V] :
            ( member(V,u)
              => member(inverse(f,V),u) ))).
```

In this formalization the function *inverse* as well as the operation *multiply* are parameterized w.r.t. the group structure to which they belong. *identity* is a function whose value is the unit element of the group.

Proving the Theorem
Peter, the first user of the system, is an expert in automated deduction and can understand resolution proofs. Thus he sends the FOF formalization of the problem to the broker asking for a CNF Refutation in TSTP format. The broker simply combines Tramp's ClauseNormalizer service (to create the corresponding CNF problem) and the GenericATP service offered by the EP system. The resulting refutation proof in TSTP format contains, among others, the following clauses:

```
cnf(1,axiom,( member(X1,X2) | ~member(X1,X3) | ~subset(X3,X2))).
  . . .
cnf(10,axiom,( equal(multiply(X1,X2,inverse(X1,X2)),identity(X1))
            | ~group(X1) | ~member(X2,X1))).
cnf(12,axiom,(group(f))).
cnf(13,axiom,(subset(u,f))).
cnf(14,axiom,( member(multiply(f,X1,inverse(f,X2)),u)
            | ~member(X1,u) | ~member(X2,u))).
cnf(15,conjecture,(member(sk2,u))).
  . . .
cnf(31,derived,( member(identity(f),u)
                | ~member(X1,u) | ~group(f) | ~member(X1,f)),
        inference(pm,[status(thm)],[14,10,theory(equality)])).
  . . .
cnf(273,derived,(~member(sk2,f)),
    inference(rw,[status(thm)],[270,15,theory(equality)])).
```

```
cnf(274,derived,(false),
    inference(rw,[status(thm)],[273,51,theory(equality)]])).
```

The clauses marked as `axiom` and `conjecture` are derived by clause normalization from the FOF problem formalization, e.g., clause 1 is derived from axiom `subset`. Derived clauses, such as clauses 31, 273, and 274, have a justification listing the inference rule used to derive the clause and the parent clause names. For example, clause 273 was derived from clauses 270 and 15 by EP's `rw` rule, which implicitly uses axioms of equality.

Generating an ND Proof.
Susan, the second user has some knowledge of first-order logic, but knows nothing about clauses and the resolution calculus. She also submits the problem to the broker but asks for a ND proof instead. The broker comes up with the combination of services shown in Figure 2. The OtterfierService transforms the refutation shown above into a new refutation containing, among others, the following clauses:

```
cnf(1,initial,(member(A,B)|~member(A,C)|~subset(C,B))).
  ...
cnf(31,derived,(member(identity(f),u)|~member(A,u)|~group(f)|~member(A,f)),
    inference(factor_simp,[status(thm)],[
        inference(para_from,[status(thm)],[10,14,theory(equality)])]])).
  ...
cnf(273,derived,(~member(sk2,f)),inference(binary,[status(thm)],[270,15])).
cnf(274,derived,(false),inference(binary,[status(thm)],[51,273])).
```

Note how Otterfier transformed EP's single `pm` inference step (from clauses 10 and 14 to clause 31) to two inferences using Otter's `para_from` and `factor_simp` inference rules. In some cases a fully separate TSTP inference step containing the intermediate inferred formula (rather than a single TSTP step containing two Otter inferences as here) may be generated by Otterfier. Altogether, the resolution proof output by Otterfier consists of 23 clauses, 9 initial and 14 derived.

Finally, the NDforFOF service is invoked with the FOF problem formalization of the problem and the FO-ATP-Result of the OtterfierService. The underlying Tramp creates an ND proof in linearized style as introduced in [1]. The lines of the proof are of the form $L.\ \Delta \vdash F\ (\mathcal{R})$, where L is a unique label, $\Delta \vdash F$ a sequent denoting that the formula F can be derived from the set of hypotheses Δ, and $(\mathcal{R})$ is a justification expressing how the line was derived. Tramp starts with an initial open ND proof that consists of the axioms of the FOF problem and the conjecture. Each axiom becomes an initial hypothesis (justified by Hyp), the conjecture is the initial goal (justified by $Open$). The initial ND proof for the problem is as follows:[5]

---

[5]Variables are now written with lower case letters, and constants are capitalized.

12

$\subset DEF.$	$\subset DEF$	$\vdash \forall s', s\text{.}subset(s', s) \Leftrightarrow$	(Hyp)
		$\qquad \forall x\text{.}(member(x, s') \Rightarrow member(x, s))$	
$UnitAx.$	$UnitAx$	$\vdash \forall g\text{.}group(g) \Rightarrow$	(Hyp)
		$\qquad (member(identity(g), g) \wedge$	
		$\qquad\qquad \forall x\text{.}(member(x, g) \Rightarrow$	
		$\qquad\qquad\qquad (multiply(g, x, identity(g)) = x$	
		$\qquad\qquad\qquad \wedge\ multiply(g, identity(g), x) = x)))$	
$InvAx.$	$InvAx$	$\vdash \forall g, x\text{.}(group(g) \wedge member(x, g)) \Rightarrow$	(Hyp)
		$\qquad (member(inverse(g, x), g)$	
		$\qquad\qquad \wedge\ multiply(g, x, inverse(g, x)) = identity(g)$	
		$\qquad\qquad \wedge\ multiply(g, inverse(g, x), x) = identity(g))$	
$Criterion.$	$Criterion$	$\vdash \forall x, y\text{.}(member(x, U) \wedge member(y, U)) \Rightarrow$	(Hyp)
		$\qquad member(multiply(F, x, inverse(F, y)), U)$	
$FGroup.$	$FGroup$	$\vdash group(F)$	(Hyp)
$U \subset.$	$U \subset$	$\vdash subset(U, F)$	(Hyp)
$Conj.$	$\mathcal{H}$	$\vdash \forall x\text{.}member(x, U) \Rightarrow member(inverse(F, x), U)$	$(Open)$

$$\mathcal{H} = \subset DEF, UnitAx, InvAx, Criterion, FGroup, U \subset$$

During the transformation of the resolution proof Tramp adds justification steps and
nodes to the ND proof until all nodes are justified. The complete ND-proof at assertion
level created by Tramp is (only the new lines and justifications):

$L2.$	$L2$	$\vdash member(C, U)$	(Hyp)
$L4.$	$\mathcal{H}_1$	$\vdash member(multiply(F, C, inverse(F, C)), U)$	$(Criterion\ L2)$
$L5.$	$\mathcal{H}_2$	$\vdash member(C, F)$	$(\subset DEF\ U \subset\ L2)$
$L6.$	$\mathcal{H}_3$	$\vdash multiply(F, C, inverse(F, C)) = identity(F)$	$(InvAx\ FGroup\ L5)$
$L7.$	$\mathcal{H}_4$	$\vdash member(identity(F), U)$	$(=Subst\ L4\ L6)$
$L8.$	$\mathcal{H}_4$	$\vdash member(multiply(F, identity(F), inverse(F, C)), U)$	$(Criterion\ L7\ L2)$
$L9.$	$\mathcal{H}_3$	$\vdash member(inverse(F, C), F)$	$(InvAx\ FGroup\ L5)$
$L10.$	$\mathcal{H}_5$	$\vdash multiply(F, identity(F), inverse(F, C))$	$(UnitAx\ FGroup\ L9)$
		$\qquad\qquad\qquad = inverse(F, C)$	
$L3.$	$\mathcal{H}, L2$	$\vdash member(inverse(F, C), U)$	$(=Subst\ L8\ L10)$
$L1.$	$\mathcal{H}$	$\vdash member(C, U) \Rightarrow member(inverse(F, C), U)$	$(\Rightarrow I\ L3)$
$Conj.$	$\mathcal{H}$	$\vdash \forall x\text{.}member(x, U) \Rightarrow member(inverse(F, x), U)$	$(\forall I\ L1)$

$$
\begin{aligned}
\mathcal{H} &= \subset DEF, UnitAx, InvAx, Criterion, FGroup, U \subset \\
\mathcal{H}_1 &= Criterion, L2 \\
\mathcal{H}_2 &= \subset DEF, U \subset, L2 \\
\mathcal{H}_3 &= InvAx, FGroup, \subset DEF, U \subset, L2 \\
\mathcal{H}_4 &= Criterion, InvAx, FGroup, \subset DEF, U \subset, L2 \\
\mathcal{H}_5 &= UnitAx, InvAx, FGroup, \subset DEF, U \subset, L2
\end{aligned}
$$

Here the justifications $\forall I$ and $\Rightarrow I$ of the nodes $L1$ and $Conj$ are the basic ND rules
introduction of universal quantification and introduction of implication. $=Subst$, which
is used in the justifications of node $L3$ and $L7$, is the ND rule for equality substitu-
tion. All other justifications are assertion applications. For instance, the justification
$(\subset DEF\ U \subset\ L2)$ of node $L5$ is the application of assertion $\subset DEF$ to the premises
$U \subset$ and $L2$. Altogether the resulting ND proof at the assertion level consists of 17
nodes and 6 assertion steps. When all complex steps are expanded, then the resulting
basic level ND proof consists of 54 nodes.

The main – clearly comprehensible – steps in the direct ND proof are: First, assume
that an arbitrary C is in U (in $L2$). Then, use the subgroup-criterion to derive that the

13

identity of F is in U (in $L7$). Finally, use again the subgroup-criterion to derive that the inverse of C is in U.

From ND to NL.
Mary knows only mathematical proofs as they are presented in textbooks. She asks the broker to return a natural language proof for the problem. The broker therefore extends the service sequence in Figure 2 with the PrexND2NL service. The *P.rex* system underlying this service can access basic linguistic knowledge about first-order logic connectives but doesn't have any knowledge about the particular domain or the problem.

From the ND proof at the assertion level, *P.rex* creates a natural language proof. The relevant parts of the Postscript version of the proof are shown in Figure 3. In the verbalization of assertion applications, this natural language proof refers to the axioms of the ND proof, e.g., $\subset DEF$ and *Criterion*. In the complete verbalization, which we skip here, these axioms are introduced and verbalized as well.

[...]   Let $member(C, U)$.   Then $member(C, F)$ because $subset(U, F)$ by $\subset DEF$.   Thus $member(inverse(F, C), F)$ because $group(F)$ by $InvAx$.   That implies that $multiply(F, identity(F), inverse(F, C)) = inverse(F, C)$ by $UnitAx$ since $group(F)$.   That implies that $member(multiply(F, C, inverse(F, C)), U)$ by $Criterion$.   That leads to $multiply(F, C, inverse(F, C)) = identity(F)$ by $InvAx$ because $group(F)$.   That implies that $member(identity(F), U)$.   Therefore $member(multiply(F, identity(F), inverse(F, C)), U)$ by $Criterion$.   That implies that $member(inverse(F, C), U)$.   Therefore $member(C, U)$ implies that $member(inverse(F, C), U)$.   That implies that $member(x, U)$ implies that $member(inverse(F, x), U)$ for all x.

Figure 3: Fragment of the *P.rex* proof verbalization with basic linguistic knowledge

Mary still has some problems understanding the proof because *P.rex* was not given any linguistic knowledge about the domain. Simple facts, such as the expression $subset(t, t')$, should be written as $t \subset t'$. Futhermore, $member(x, s)$ should be written as $x \in s$. This would considerably improve the readability of the proof.

It is important to note that our brokering mechanism acts as a black box and Mary does, for instance, not have to work with theorem provers on the level of clauses and resolution proofs.

Figure 4 shows the length of the four different proofs as well as the time for finding/transforming the proofs involved. The run times were measured on a Pentium IV 2GHz machine. The time of the Otterfier service is CPU time while all other times are wall clock times. Both resolution proofs contain 9 initial clauses, all other clauses are derived. The ND proof contains 6 assertion level steps and 4 ND calculus steps. *P.rex* does not use any domain-specific linguistic knowledge while translating the ND proof into natural language.

Proof	Proof Length	Time (secs)	Format/Calculus
EP proof	19 clauses	0.031	EP
Otterfier proof	30 clauses	15	BrFP
Tramp proof	10 ND steps	4	ND
P.rex proof	10 sentences	54	NL

Figure 4: Lengths and proving times for the different proofs

5   Conclusion and Future Work

Ongoing work on integrated proof transformation services and their dynamic combination has been presented. By using the TSTP data-exchange format, and by defining the notion of a resolution proof in a restricted calculus, it has been possible to combine several independently developed systems that could not previously interact with each other. The extension and integration of the systems into a Java framework has been completed, and a prototypical version of the broker has been implemented. Efforts are underway to make the services available as web services. An execution framework for the service sequences planned by the broker will be implemented in the near future. The outcome is an integrated service that can support mathematical reasoning from problem specification in first-order logic through to proof presentation in natural language. Our system absolves the users from the need to know details of system specific data representations, low-level reasoning processes, and possible tool combinations. It is worth mentioning that the brokering mechanism is domain-independent in the sense that the only domain-specific knowledge is encoded in our ontology. There is no knowledge about proof systems hardwired in our broker. Furthermore, our approach is not limited to a first-order logic domain. With an appropriate extension of our ontology we hope to be able to describe also higher-order theorem provers.

Several frameworks for the integration of reasoning systems have been developed. The MATHWEB Software Bus [27] (MATHWEB-SB) integrates heterogeneous reasoning systems at the system level. However, the user of the MATHWEB-SB needed to have quite a lot of knowledge of the systems involved. Furthermore, a dynamic combination of systems by the MATHWEB-SB is not possible. The question of how theorem proving components can be easily combined in a single environment has led to the concept of Open Mechanized Reasoning Systems [11] (OMRS). In OMRS, systems are described on three different levels: the control, the logic, and the interface level. These descriptions are far more fine-grained than the service descriptions used in this work, because they aim at a low-level corporation of systems. OMRS has been studied mainly for the combination of theorem provers and decision procedures.

Future work includes a refinement of the existing services, the incorporation of new services and a more sophisticated brokering mechanism. At this stage Otterfier, for instance, is capable of transforming a derivation only if all the inference steps produce a logical consequence. As indicated in Section 2.2, many modern ATP systems use some form of satisfiability preserving splitting rule, which is useful in the context of a search for a refutation. Otterfier cannot transform derivations containing applications of splitting. In the future it is planned to build a transformation tool that will remove

splitting steps from a derivation, by "glueing" together the parts of the derivation that contain clauses inferred by splitting.

So far, our focus has been on first order theorem provers and proof transformation services. In the future it would be desirable to integrate higher-order proving systems, model generators, and decision procedures into the framework. The formalization of systems' logics and calculi in the Logical Framework (LF), implemented in the Twelf [17] system, is being considered.

The brokering of services can be improved in several ways. Reasoning on our ontology during plan formation, for instance, could improve the flexibility of our broker. The use of several plans, conditional plans, and re-planning, could improve the broker's behavior in case plan execution fails.

Acknowledgments

References

[1] P.B. Andrews. Transforming matings into natural deduction proofs. In *Proc. of CADE-5*, pages 281–292, 1980.

[2] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[3] O. Caprotti and W. Schreiner. Towards a mathematical services description language. In *Proc. of the International Congress of Mathematical Software, ICMS 2002*, Beijing, China, August 2002.

[4] The MKM Consortium. Mathematical Knowledge Management Network. http://monet.nag.co.uk/mkm.

[5] The MONET Consortium. The MONET Project. http://monet.nag.co.uk/cocoon/monet/index.html, April 2002.

[6] L. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and Melham T. The PROSPER Toolkit. *International Journal on Software Tools for Technology Transfer*, 4(2):189–210, 2000.

[7] L. Dirat. Jome: The java openmath editor. `http://mainline.essi.fr/wiki/bin/view/Jome/WebHome`.

[8] S. Bechhofer et al. OWL – Web Ontology Language Reference, February 2004. Available at http://www.w3.org/TR/owl-ref/.

[9] A. Fiedler. Using a cognitive architecture to plan dialogs for the adaptive explanation of proofs. In Thomas Dean, editor, *Proc. of the 16th International Joint*

Conference on Artificial Intelligence (IJCAI), pages 358–363, Stockholm, Sweden, 1999. Morgan Kaufmann.

[10] A. Fiedler. *P.rex*: An interactive proof explainer. In Rejeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI, pages 416–420, Siena, Italy, 2001. Springer Verlag.

[11] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning theories – towards an architecture for open mechanized reasoning systems. IRST-Technical Report 9409-15, IRST, Trento, Italy, Juni 1994.

[12] The Calculemus Interest Group. The CALCULEMUS Project. `http://www.eurice.de/calculemus/`.

[13] X. Huang. Reconstructing proofs at the assertion level. In *Proc. of CADE-12*, pages 738–752, 1994.

[14] SRI International Computer Science Laboratory. QED Pro Quo. `http://www.qpq.org`.

[15] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[16] A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In *Proc. of CADE–17*, volume 1831 of *LNAI*, pages 460–464. Springer, 2000.

[17] F. Pfenning and C. Schürmann. System description: Twelf — A meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proc. of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, 1999. Springer-Verlag LNAI 1632.

[18] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[19] P. Rudnicki. An Overview of the Mizar Project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pages 311–332, 1992.

[20] W. Schreiner and O. Caprotti. The MathBroker Project. http://poseidon.risc.uni-linz.ac.at:8080/index.html, October 2001.

[21] S. Schulz. System Abstract: E 0.61. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proc. of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 370–375. Springer-Verlag, 2001.

[22] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Alan Bundy, editor, *12th International Conference on Automated Deduction, CADE-12*, volume 814 of *LNAI*, pages 252–266, Nancy, France, Juni 1994. Springer Verlag, Berlin.

[23] G. Sutcliffe, J. Zimmer, and S. Schulz. Communication Standards for Automated Theorem Proving Tools. In V. Sorge, S. Colton, M. Fisher, and J. Gow, editors, *Proc. of the Workshop on Agents and Automated Reasoning, 18th International Joint Conference on Artificial Intelligence*, 2003.

[24] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Tpoic. System Description: SPASS Version 1.0.0. In H. Ganzinger, editor, *Proc. of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 378–382. Springer-Verlag, 1999.

[25] L. Wos and G. Pieper. *Automated Reasoning and the Discovery of Missing and Elegant Proofs*. Rinton Press, 2003.

[26] J. Zimmer. A Framework for Agent-based Brokering of Reasoning Services. In Raul Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Juan Humberto Sossa Azuela, editors, *MICAI*, volume 2972 of *Lecture Notes in Computer Science*. Springer, 2004.

[27] J. Zimmer and M. Kohlhase. System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. In A. Voronkov, editor, *Proc. of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 139–143. Springer-Verlag, 2002.