Journal of Universal Computer Science, vol. 5, no. 3 (1999), 188-207
submitted: 1/11/98, accepted: 15/3/99, appeared: 28/3/99 © Springer Pub. Co.

Integrating TPS and ΩMEGA

Christoph Benzmüller
Fachbereich Informatik, Universität des Saarlandes, Germany
chris@cs.uni-sb.de

Matthew Bishop[1]
Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, USA
mbishop+@cs.cmu.edu

Volker Sorge
Fachbereich Informatik, Universität des Saarlandes, Germany
sorge@ags.uni-sb.de

Abstract: This paper reports on the integration of the higher-order theorem proving environment TPS [Andrews *et al.*, 1996] into the mathematical assistant ΩMEGA [Benzmüller *et al.*, 1997]. TPS can be called from ΩMEGA either as a black box or as an interactive system. In black box mode, the user has control over the parameters which control proof search in TPS; in interactive mode, all features of the TPS-system are available to the user. If the subproblem which is passed to TPS contains concepts defined in ΩMEGA's database of mathematical theories, these definitions are not instantiated but are also passed to TPS. Using a special theory which contains proof tactics that model the ND-calculus variant of TPS within ΩMEGA, any complete or partial proof generated in TPS can be translated one to one into an ΩMEGA proof plan. Proof transformation is realised by proof plan expansion in ΩMEGA's 3-dimensional proof data structure, and remains transparent to the user.

1 Introduction

Current theorem proving systems, whether automatic or interactive, are usually strong in some domains while lacking reasoning power in others. Furthermore, there are no standardised formats for databases of higher-order problems, as there are for first-order problems [Sutcliffe *et al.*, 1994], and so higher-order theorem provers are generally unable to share databases of problems. In recent years there have been several attempts to combine two or more systems and hence to allow various theorem provers with different proof strategies to cooperate on a problem [Giunchiglia *et al.*, 1996], to allow users of an interactive system to invoke an external automatic system on a subproblem [Slind *et al.*, 1998a; Slind *et al.*, 1998b; Meier, 1997; Dahn *et al.*, 1994] or to avoid duplication of work by sharing databases [Felty and Howe, 1997].

In this paper we describe the integration of the higher-order theorem proving system TPS into the mathematical assistant ΩMEGA, and discuss the benefits that this provides for both systems. For a preliminary report on our work we refer to [Benzmüller and Sorge, 1998b].

[1] Supported by the National Science Foundation under grant CCR-9624683.

1.1 The TPS system

TPS [Andrews *et al.*, 1996] is a higher-order theorem proving system for classical type theory (Church's simply-typed λ-calculus). Proofs in TPS may be constructed automatically using the matings method (connection method) [Andrews, 1981], or interactively using an extended variant of Gentzen's natural deduction calculus [Gentzen, 1935]. Automatic proofs may be translated into natural deduction format [Miller, 1984; Pfenning, 1987], and hence the user may interleave the automatic and interactive proof methods by, for example, invoking the automatic component on a subproblem of a partially-completed interactive proof. This translation between automatic and natural deduction proofs provides the basis for the integration of TPS and ΩMEGA.

There are several built-in automatic search procedures in TPS, each of which is governed by a set of parameters (known as *flags*) which may be adjusted by the user or even automatically by TPS itself. Furthermore TPS can expand definitions using the dual instantiation strategy described in [Bishop and Andrews, 1998]; this provides an effective way to decide which abbreviations to instantiate during a proof. TPS provides a library for storing objects such as theorems, definitions and modes (groups of flag settings), and can also store and retrieve files containing sequences of commands (*work files*) or natural deduction proofs (*proof files*). All of these facilities are also used in the integration of ΩMEGA and TPS.

A more complete description of the capabilities of TPS is provided in [Andrews *et al.*, 1997], or online at `http://www.cs.cmu.edu/~andrews/tps.html`.

1.2 The ΩMEGA system

The ΩMEGA-system [Benzmüller *et al.*, 1997] is designed as an interactive mathematical assistant system, aimed at supporting proof development in mainstream mathematics. It consists of a variety of tools including a proof planner [Huang *et al.*, 1994], a graphical user interface LΩUI [Siekmann *et al.*, 1998], the PROVERB system [Huang and Fiedler, 1997] for translating proofs into natural language, and a variety of external systems such as computer algebra systems [Kerber *et al.*, 1998], automated theorem provers [McCune, 1994; Baumgartner and Furbach, 1994; Weidenbach *et al.*, 1996] and constraint solvers. ΩMEGA also provides the built-in higher-order theorem prover LEO [Benzmüller and Kohlhase, 1998], which specialises in reasoning about higher-order equality and extensionality.

ΩMEGA is, like TPS, a theorem proving system for classical type theory (Church's simply-typed λ-calculus) which uses a ND calculus variant as its basic inference mechanism. However the set of basic ND rules in TPS is larger than that in ΩMEGA, in order to keep TPS proofs concise and readable. Therefore certain rules in TPS abstract over small subproofs (such as *RuleP*, which abstracts over proofs in propositional logic, cf. Section 3). In ΩMEGA, however, the set of basic ND-rules is just large enough to ensure completeness, and all extensions to the basic ND-calculus (e.g. equality substitution) are defined as tactics. Nevertheless, proofs can be both constructed and displayed on several abstract levels by using a 3-dimensional data structure (see Section 2) for representing (partial) proofs. The structure on the one hand enables the user to freely switch back and forth between different abstract levels and on the other hand provides a means for directly integrating results of external reasoners while leaving the expansion to the calculus level to ΩMEGA's tactic mechanism.

Further information and an online version of ΩMEGA are available over the Internet at `http://www.ags.uni-sb.de/~omega/`.

1.3 Benefits of integrating TPS and ΩMEGA

Both TPS and ΩMEGA use a higher-order logic based on Church's simply-typed λ-calculus, and both use a Gentzen-style natural deduction calculus; this makes the integration somewhat easier and more natural than it might otherwise have been. However, the two systems are still different enough for each to benefit considerably from the other.

ΩMEGA is designed to be a mathematical assistant, and so contains a small basic set of natural deduction rules, plus many defined tactics. ΩMEGA provides facilities such as a database of mathematical theories, a proof planner, proof verbalisation, integration of computer algebra systems and first-order theorem provers, and a graphical display in which the level of detail provided may be varied by the user. Since many of the predefined theories contain higher-order concepts, problems formulated in these theories will naturally lie beyond the capabilities of the first-order theorem provers which have already been integrated into ΩMEGA, and so the principal benefit of the integration for ΩMEGA is the addition of a powerful higher-order automated theorem prover as an external reasoning component.

TPS, on the other hand, is designed to be a system for proving theorems in a specific logic (as well as a tool for research into automated theorem proving). TPS must keep its proofs as concise as possible, since it has a command-line interface rather than the graphical interface of ΩMEGA, and so it contains a larger range of natural deduction rules than ΩMEGA. TPS has comparatively few predefined theories, since all but the smallest such theories contain far too many axioms for any of its automatic search procedures to cope with. Furthermore, TPS cannot invoke any external reasoning components. For TPS, then, the principal benefits of integration with ΩMEGA are the addition of a graphical interface, proof verbalisation, and the ability to use external reasoning systems (although the present integration does not allow TPS to call such systems itself, it can in effect call them through ΩMEGA, since ΩMEGA can call both TPS and the other systems, and any proof known to ΩMEGA can be passed to TPS).

2 Natural Deduction Proofs in ΩMEGA

The essential prerequisite for a smooth integration of TPS proofs into ΩMEGA proofs is ΩMEGA's ability to expand abstract inference steps into inferences in its own calculus. This enables the definition of abstract inference methods that can incorporate both decision procedures and partial proofs from other systems. In this section we will elaborate further on this issue by giving an overview of the core of the ΩMEGA system.

The entire process of theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, plan execution, and verification, centred around the so-called *Proof Plan Data Structure* ($\mathcal{PDS}$). A $\mathcal{PDS}$ is a hierarchical data structure which represents a (partial) proof at different levels of abstraction

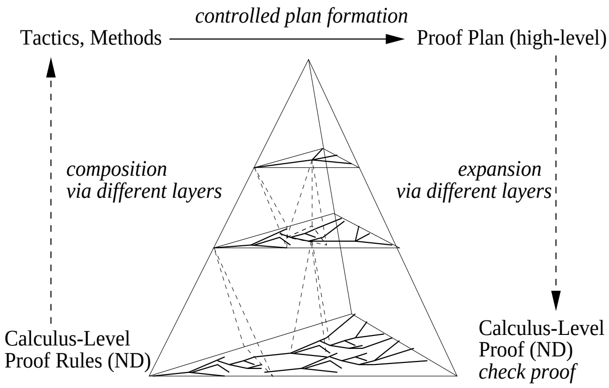

Figure 1: ΩMEGA's 3-dim. $\mathcal{PDS}$

(called *proof plans*). It is represented as a directed acyclic graph, where the nodes are justified by tactics or methods. Conceptually, each justification represents a proof plan (the *expansion* of the justification) at a lower level of abstraction that is computed when the justification is expanded. A proof plan can be recursively expanded until a fully explicit proof on the calculus level (ND) has been reached. In ΩMEGA, the original proof plan is kept in a 3-dimensional expansion hierarchy (cf. Figure 1). Thus the $\mathcal{PDS}$ makes explicit the hierarchical structure of proof plans and retains it for further applications such as proof explanation or analogical transfer of plans.

Once a proof plan is completed, its justifications can successively be expanded to verify the well-formedness of the resulting $\mathcal{PDS}$. When the expansion process is completed, the establishment of correctness of the ND proof relies solely on the correctness of the verifier and the calculus. This approach also provides a basis for a controlled integration of external reasoning components – such as an automated theorem prover or a computer algebra system – if each reasoner's results can (on demand) be transformed into a sub-$\mathcal{PDS}$.

A $\mathcal{PDS}$ can be constructed by automated or mixed-initiative planning, or by pure user interaction. In particular, new pieces of the $\mathcal{PDS}$ can be added by directly calling tactics, by inserting facts from a data base, or by calling some external reasoner.

In order to demonstrate the basic expansion mechanism we consider the ND-rule $\forall_E$ and the simple tactic $\forall_E^*$:

$$\frac{\forall x.A}{[t/x]A}\ \forall_E(t) \qquad \frac{\forall x_1,\ldots,x_n.A}{[t_1/x_1,\ldots,t_n/x_n]A}\ \forall_E^*(t_1,\ldots,t_n)$$

The application of the latter would be on an abstract level in the $\mathcal{PDS}$ and its expansion to ND-level would result in a sequence of applications of the $\forall_E$-rule. Besides providing a means for handling the application and expansion of these rather small abstractions, the $\mathcal{PDS}$ is also the foundation for integrating deductions from external reasoning components into ΩMEGA on a very abstract

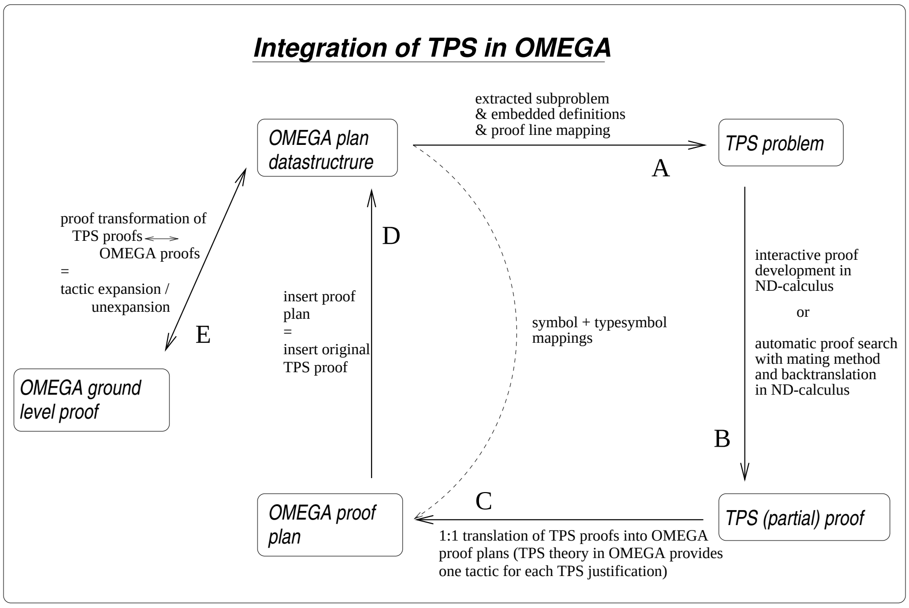

Figure 2: The integration architecture

level. We exploit this possibility for the integration of Tps by specifying three different abstraction levels for Tps's deductions:

1. A single justification expressing that a proof for a particular subproblem has been found by Tps.
2. A second expansion level incorporates the original Tps proof into Ωmega's $\mathcal{PDS}$. On this level the justifications for the respective proof lines contain the justifications of the original Tps proof.
3. A third level where the Tps justifications are mapped to corresponding Ωmega tactics. However, this level does not correspond to a proof on the calculus level, as some of the tactics might need to be expanded even further.

3 The Integration

The general integration approach, as illustrated in Figure 2, is divided into five steps A–E. Currently the integration is still one-directional; Tps can be used from within Ωmega, but Ωmega cannot be used from within Tps. We start with a partial proof plan, on an arbitrary abstraction level in Ωmega, that contains an open subproblem we want to prove with Tps. In step A the focused subproblem is extracted and, together with the relevant concepts from Ωmega's knowledge base, translated into Tps syntax. In step B Tps reads the translated problem and either tries to find a proof automatically (when called in automatic mode from Ωmega) or pops up its command interface for interactive proof development (when called in interactive mode from Ωmega; see the screen-shot in Example 3 in the appendix). The result is a complete or partial proof that is mirrored one-to-one as an ΩMEGA proof plan in step C. In step D, this proof plan is inserted into ΩMEGA's proof data structure ($\mathcal{PDS}$) in order to fill the given gap. Finally, a TPS proof which has been modelled on ΩMEGA's proof tactic level is transformed into a proper proof in ΩMEGA's basic ND-calculus by proof plan expansion in step E. This transformation may require support from the other external reasoners already integrated into ΩMEGA. Since all of the particular expansion steps in the proof transformation are stored in ΩMEGA's 3-dimensional $\mathcal{PDS}$, ΩMEGA's expansion/contraction mechanism for proof tactics allows the user to move freely between the TPS proof on an abstract level, the proof on ΩMEGA's basic ND-calculus level, and all of the intermediate levels of abstraction. This ensures that proof transformation is transparent to the user, and remains so even as the user examines the proof on different levels of abstraction.

In the following we will discuss the particular integration steps in more detail, using the following example as an illustration:

Example 1. (THM136) $\forall r_{o\alpha\alpha}.transitive\ (transitive\text{-}closure\ r)$
This example states that the transitive closure of a relation is transitive[2].

This problem is defined within ΩMEGA's theory *RELATION*, which also provides the recursively entailed defined concepts which are *transitive-closure*, *transitive* and *sub-relation*. These are defined as follows:

$$transitive\text{-}closure := \lambda r_{o\alpha\alpha}.\lambda x_\alpha.\lambda y_\alpha.\forall q_{o\alpha\alpha}.\ (sub\text{-}relation\ r\ q \wedge transitive\ q) \Rightarrow q\ x\ y$$

$$transitive := \lambda r_{o\alpha\alpha}.\forall x_\alpha, y_\alpha, z_\alpha.(r\ x\ y \wedge r\ y\ z) \Rightarrow r\ x\ z$$

$$sub\text{-}relation := \lambda r_{o\alpha\alpha}.\lambda q_{o\alpha\alpha}.\forall x_\alpha, y_\alpha.r\ x\ y \Rightarrow q\ x\ y$$

3.1 A: Calling TPS from ΩMEGA

When calling TPS within ΩMEGA the user specifies the subgoal to be proved, some parameters which specify the proof heuristic to be used by TPS, and a time limit for this proof attempt. Furthermore the user may specify definitions that are entailed in the problem but which are not to be passed to TPS, in order to force TPS to treat them as uninterpreted constants.

Firstly, the focused subproblem is extracted from ΩMEGA's $\mathcal{PDS}$, by identifying the open subgoal explicitly mentioned as a parameter and determining its support nodes. Then ΩMEGA computes the set of all defined concepts that are recursively entailed in the extracted subproblem, and eliminates from this set all those concepts which the user has explicitly prohibited from being passed to TPS. Thus for THM136 we get exactly the three definitions shown above, assuming that the user has permitted all definitions to be passed to TPS. In the next step both this subproblem and the selected definitions are translated into TPS syntax. As both systems implement a logic based upon Church's simply-typed λ-calculus, and even their representations of types are very similar, this translation process is rather trivial, and we shall not discuss it in much detail. However, there are some minor considerations to be taken care of:

[2] Information on the syntax: In TPS the type $(\alpha \to \beta) \to \gamma$ is denoted $(\gamma(\beta\alpha))$. In particular, the type $o\alpha\alpha$ (i.e. $((o\alpha)\alpha)$) is the type $\alpha \to \alpha \to o$ of a binary relation on objects of type α.

1. TPS uses a small set of constant symbols with a fixed semantics (e.g. the logical connectives), and these symbols must not be redefined.
2. The polymorphic types which are allowed in ΩMEGA must usually be renamed in order for TPS to interpret them correctly.
3. It is important to maintain a mapping between the initial TPS proof lines in the translated subproblem and their counterparts in ΩMEGA's $\mathcal{PDS}$.

Problems 1 and 2 are solved by setting up hash-tables within ΩMEGA which store the necessary information about renamings of constant symbols (in 1) and the correspondence between the polymorphic type-symbols (in 2). As the line numbering in TPS steadily changes, we can not use another hash-table for solving 3. Fortunately, TPS allows the user to attach arbitrary additional information to each proof line; we use this feature to mark the TPS proof lines in the translated subproblem with the names of their counterparts on the ΩMEGA side.

Apart from the above-mentioned hash-tables, the most important results of phase A are two files containing all the necessary information for TPS. The first file — which we call the *problem-file* — contains the information on the subproblem in focus and the recursively embedded defined concepts. The second file — the *command-file* — contains a sequence of commands to be executed by TPS. These commands tell TPS to read the problem-file, to set the proof tactic as specified by the user and, in the case that TPS is called in automatic mode (see phase B), to invoke TPS's mating-search procedure. The problem-file created by ΩMEGA for our example THM136 is presented in the appendix of this paper (see Example 2).

3.2 B: Automatic or Interactive Proof Search in TPS

TPS can be called from ΩMEGA in either automatic or interactive mode. In the former case the TPS core image is started as a black box and the only information visible to the user is the time resource allocated to TPS's proof attempt. TPS executes only the commands which are specified in the command-file created by ΩMEGA.

When TPS is called in interactive mode, an xterm with TPS's command user interface pops up (see Example 3 in the appendix) and the interactive session is initialised by the commands stored in the command-file. The user can then interactively use all the available features of TPS in order to construct a complete or partial ND-style proof.

TPS's built-in proof transformation procedure [Miller, 1984; Pfenning, 1987] translates mating proofs into ND-calculus such that, in both interactive and automatic modes, the final result of the proof attempt is either a complete or partial proof in TPS's ND-calculus variant. This (partial) proof is then stored in a *tps-output* file[3] and passed back to ΩMEGA.

A very important feature of our approach is that TPS can use its mechanism for dual instantiation [Bishop and Andrews, 1998] within its mating-search procedure. This is possible because we do not expand all defined concepts before

[3] Actually there are two files produced by TPS, one containing the (partial) proof in ASCII format and one containing the same proof in a Lisp-like presentation. The former is only used to present the original TPS proof within ΩMEGA and the latter, which is the more important of the two, is used in phase C to translate the TPS proof to ΩMEGA.

passing the subproblem to TPS, but instead pass these concepts as additional information and leave the subproblem as it is. Thus TPS can decide on its own whether it is necessary to expand particular defined concepts or not. Example 1, above, is a good example of a theorem which cannot be proven by TPS if all the definitions are expanded before the mating-search procedure is called[4]. For a detailed discussion see [Bishop and Andrews, 1998]. The proof generated by TPS for THM136 is presented in Example 3 in the appendix.

3.3 C & D: Representing TPS Proofs as ΩMEGA-Proof Plans and Insertion of Proof Plans

One main idea of our approach is to provide as transparent a translation mechanism as possible, by modelling TPS's ND-calculus variant on ΩMEGA's proof tactic level. We implement this modelling by defining a special theory *TPS* in ΩMEGA's knowledge base. For each possible TPS ND justification, the theory *TPS* introduces a corresponding ΩMEGA-tactic; the expansion contents of some of these tactics are presented in Example 5 in the appendix. There is one additional black box tactic *tps*, which will be used to provide the most abstract view of subproblems proven by TPS. The concrete proof translation proceeds as follows:

1. A proof generated by TPS is mirrored one to one as a proof plan in ΩMEGA by mapping the particular proof justifications in the TPS proof to the corresponding proof tactics provided by the special theory *TPS* in ΩMEGA's knowledge base. In order to guarantee a correct mapping of the entailed constants and type symbols, the translation process uses the hash-tables constructed by ΩMEGA in phase A. Furthermore, the correspondence between the proof lines of the focused ΩMEGA-subproblem and the corresponding TPS proof lines is given as explicit information in the TPS proof. The proof plan we obtain for THM136 is presented as Example 4 in the appendix.
2. The resulting proof plan is then stored in ΩMEGA with a reference to the subproblem on which TPS has been called. Some additional information is also stored, such as the original TPS proof in ASCII format, the proof parameters and some proof statistics.

In phase D the open line itself is first closed and justified by using the special black box tactic *tps*, thereby providing the most abstract view of the proof for our subproblem in focus. By expanding this special tactic the corresponding proof plan is inserted in ΩMEGA's $\mathcal{PDS}$, and the structure of the original TPS proof can be visualised in ΩMEGA's graphical user interface LΩUI [Siekmann *et al.*, 1998]. Example 6 in the appendix presents the proof structure of the original TPS proof for THM136 (see Examples 3 and 4), graphically visualised in LΩUI.

3.4 E: Transparent Proof Transformation by Proof Plan Expansion

It remains to transform the abstract proof plan representing the TPS proof into ΩMEGA's own basic ND-calculus variant. Such a proof transformation is

[4] This theorem is still a challenging problem for current ATP's. Apart from a proof constructed by ΩMEGA's proof planner using very special control information [Sehn, 1995], TPS is the only system known to the authors that can automatically find a proof.

necessary, as ΩMEGA's philosophy on integrated systems is not to trust any externally-produced proof until it can be transformed and proof checked on ΩMEGA's basic ND-calculus level. The transformation problem for TPS proofs has a very simple solution since the ND-calculus variants of both systems are very similar, and the other external reasoners already integrated to ΩMEGA (e.g. OTTER [McCune, 1994]) can fruitfully support the transformation in non-trivial cases.

Proof transformation is realised via tactic expansion. Each proof tactic defined in ΩMEGA's special *TPS* theory contains specific expansion information that maps any concrete application of this particular tactic onto a proof on a lower, more detailed proof level in ΩMEGA's $\mathcal{PDS}$. Thus, by stepwise tactic expansion, the original TPS proof mirrored in ΩMEGA can finally be transformed into ΩMEGA's basic ND-calculus level. A nice side effect of this approach is that the original TPS proof, the corresponding ΩMEGA ND-proof and all intermediate levels of the proof transformation process are permanently stored in ΩMEGA's $\mathcal{PDS}$. Consequently the flexible tactic expansion/contraction mechanism in ΩMEGA allows users to analyse the proof on whatever level interests them. Example 6 in the appendix presents two different layers in ΩMEGA's $\mathcal{PDS}$.

We distinguish four categories of expansion tactics defined in the *TPS* theory, as follows:

I *Simple mapping:* Many rules of the ND-calculus variant of TPS have direct counterparts in ΩMEGA. Examples are presented in Figure 3. Here tactic tps*ForallE is mapped to ΩMEGA's basic ND-calculus rule $\forall_E$ and the tactic tps*Conj is mapped to the tactic $\wedge_E$, which itself expands into the basic ND-calculus rules $\wedge_{E_l}$ and $\wedge_{E_r}$. The expansion content of the tactic tps*ForallE is presented in Example 5 in the appendix.

II *Case Distinction:* Some tactics of the *TPS* theory need case distinctions in their expansion mapping. For example, the tactic tps*Neg justifies applications of the push negation as well as the pull negation principle; see Figure 3. ΩMEGA provides the corresponding tactics Pushneg and Pullneg, and thus the expansion of tps*Neg simply analyses the situation and maps to either Pushneg or Pullneg, as appropriate. Both Pushneg and Pullneg are tactics that expand with case distinction mappings to a lower level in ΩMEGA's $\mathcal{PDS}$. By subsequent tactic expansion we finally get a medium-sized derivation in ΩMEGA's basic ND-calculus. The definition of the tactic tps*Neg is presented in Example 5 in the appendix.

III *Restructuring:* Existential quantification elimination in TPS (the particular rule in TPS is called RuleC) structures a proof slightly differently from the corresponding rule $\exists_E$ in ΩMEGA; see Figure 3. Consequently the expansion of the tactic tps*RuleC into rule $\exists_E$ requires some simple restructuring of the proof with respect to the dependencies between some proof lines.

IV *External Reasoners:* TPS abbreviates pure propositional logic derivations in a complex ND proof with a single-step justification, called RuleP, and hides the boring details from the user. Thus both RuleP and the ΩMEGA-tactic tps*RuleP mean that a particular proof line follows from some premise lines by propositional logic. We need a way to expand this rather general justification, with so little detailed information available, into a concrete derivation in ΩMEGA's basic ND-calculus. An extravagant solution would be to imple-

Cat.	TPS tactic in ΩMEGA	Expansion Mapping	ΩMEGA's ND-calculus
I	$\dfrac{\forall x.A}{[x \leftarrow a]A}$ tps*Foralle(a)	$\dfrac{\forall x.A}{[x \leftarrow a]A}\ \forall_E(a)$	$\dfrac{\forall x.A}{[x \leftarrow a]A}\ \forall_E(a)$
I	$\dfrac{A \wedge B}{A \quad B}$ tps*Conj	$\dfrac{A \wedge B}{A \quad B}\ \wedge_E$	$\dfrac{A \wedge B}{A}\ \wedge_{E_l}$ $\dfrac{A \wedge B}{B}\ \wedge_{E_r}$
II	$\dfrac{\neg A}{A'}$ tps*Neg $\dfrac{A'}{\neg A}$ tps*Neg	$\dfrac{\neg A}{A'}$ Pushneg $\dfrac{A'}{\neg A}$ Pullneg	$\neg A$ $\boxed{\text{derivation D1}}$ A' A' $\boxed{\text{derivation D2}}$ $\neg A$
III	$\dfrac{\exists x.A}{[[x \leftarrow a]A]^1}$ tps*Choose(a) $\dfrac{B}{B}$ tps*RuleC^1	$\dfrac{\exists x.A \qquad \begin{array}{c}[[x \leftarrow a]A]^1 \\ \vdots \\ B\end{array}}{B}\ \exists_E{}^1$	$\dfrac{\exists x.A \qquad \begin{array}{c}[[x \leftarrow a]A]^1 \\ \vdots \\ B\end{array}}{B}\ \exists_E{}^1$
IV	$\dfrac{A}{A'}$ tps*RuleP	$\dfrac{A}{A'}$ call-PL-ATP	A $\boxed{\text{derivation D3}}$ A'

Figure 3: Transparent transformation of TPS proofs into ΩMEGA proofs, as realised by ΩMEGA's tactic expansion mechanism.

ment a propositional logic prover in ΩMEGA and to employ this prover in the expansion of tps*RuleP. Fortunately there are already several systems integrated to ΩMEGA, such as the first-order provers OTTER [McCune, 1994], SPASS [Weidenbach *et al.*, 1996] or PROTEIN [Baumgartner and Furbach, 1994], which can be used instead. In fact, TPS itself also provides a special propositional logic mode that can be used to construct detailed propositional logic proofs. Hence no additional implementation effort with respect to the expansion of tps*RuleP is necessary; we simply map tps*RuleP to a recursive call of an arbitrary system, already integrated to ΩMEGA, that is able to construct propositional logic derivations (see Figure 3). In the first implementation we used OTTER in connection with a special mapping from higher-order to propositional logic. We can also map tps*RuleP back to a call of TPS in propositional logic mode. Then, by expanding tps*RuleP, ΩMEGA's tactic mechanism automatically performs a recursive call to TPS. The definition of the tactic tps*RuleP is presented in Example 5 in the appendix.

4 Examples

Our integration approach does not restrict the set of examples that can be proved by TPS. If one introduces the necessary definitions in ΩMEGA's knowledge base then generally all the theorems provable by TPS alone should be provable by calling TPS from ΩMEGA as well. Among the TPS examples that have already been proven by calling TPS from ΩMEGA (where they can be fully expanded and proof checked) are[5]:

Cantor's theorem: $\forall g_{o\alpha}.g <_{card} (\mathcal{P}\ g)$

The cardinality of the powerset of a set g is greater than the cardinality of g.

THM15b: $\forall f_{\iota\iota}.(\exists g_{\iota\iota}.(\mathit{iteratep+}\ f\ g) \wedge (\exists x_{\iota}.(g\ x) = x \wedge (\forall z_{\iota}.(g\ z) = z \Rightarrow z = x) \Rightarrow (\exists y_{\iota}.((f\ y) = y))$

This theorem is discussed in detail in [Andrews *et al.*, 1996]. It states that if some positive iterate of f has a unique fixed point, then f has a fixed point.

THM48: $\forall f_{\iota\iota}.\forall g_{\iota\iota}.(\mathit{injectivep}\ f) \wedge (\mathit{injectivep}\ g) \Rightarrow (\mathit{injectivep}\ (f \circ g))$

The composition of injective functions is injective.

THM134: $\forall z_{\iota}.\forall g_{\iota\iota}.(\mathit{iteratep+}\ (\lambda x_{\iota}.z)\ g) \Rightarrow (\forall x_{\iota}.(g\ x) = z)$

The only positive iterate of a constant function is that function.

THM135: $\forall f_{\iota\iota}.\forall g^1_{\iota\iota}.\forall g^2_{\iota\iota}.(\mathit{iteratep}\ f\ g^1) \wedge (\mathit{iteratep}\ f\ g^2) \Rightarrow (\mathit{iteratep}\ f\ (g^1 \circ g^2))$

The composition of two iterates of a function f is an iterate of f.

[5] These examples from the TPS library are also discussed in [Andrews *et al.*, 1996].The definitions occurring in the above examples are defined in ΩMEGA's knowledgebase (analogously to TPS's library) as follows:
$<_{card} := \lambda g_{o\alpha}.\lambda h_{o\beta}.\neg\exists f_{\beta\alpha}.(\mathit{surjective}\ g\ h\ f)$
$\mathit{surjective} := \lambda f_{o\alpha}.\lambda g_{o\beta}.\lambda h_{\beta\alpha}.\forall x_{\beta}.(gx) \Rightarrow (\exists y_{\alpha}.(fy) \wedge (x = (hy)))$
$\mathcal{P} := \mathit{superset},\ \mathit{superset} := \lambda u_{o\alpha}.\lambda v_{o\alpha}.\forall x_{\alpha}(u\ x) \Rightarrow (v\ x)$
$\mathit{iteratep} := \lambda f_{\alpha\alpha}.\lambda g_{\alpha\alpha}.\forall p_{o\alpha\alpha}.(p\ (\lambda u_{\alpha}.u) \wedge (\forall j_{\alpha\alpha}.(p\ j) \Rightarrow (p\ (f \circ j)))) \Rightarrow (p\ g)$
$\mathit{iteratep+} := \lambda f_{\alpha\alpha}.\lambda g_{\alpha\alpha}.\forall p_{o\alpha\alpha}.(p\ f) \wedge (\forall j_{\alpha\alpha}.(p\ j) \Rightarrow (p\ (f \circ j))) \Rightarrow (p\ g)$
$\mathit{injectivep} := \lambda f_{\gamma\beta}.\forall x_{\beta}.\forall y_{\beta}.((f\ x) = (f\ y)) \Rightarrow (x = y)$
$\circ := \lambda f_{\gamma\delta}.\lambda g_{\beta\gamma}.\lambda x_{\delta}.g\ (f\ x)$

$$\begin{aligned} \textit{THM270:}\ \forall f_{\beta\alpha}.\forall g_{\gamma\alpha}.\forall h_{\gamma\beta}.\ & (\forall x_\alpha.h\ (f\ x) = g\ x) \wedge (\forall y_\beta.\exists x_\alpha.f\ x = y) \\ & \wedge (\forall x_\alpha.\forall y_\alpha.f\ (x *^1 y) = (f\ x) *^2 (f\ y)) \\ & \wedge (\forall x_\alpha.\forall y_\alpha.g\ (x *^1 y) = (g\ x) *^3 (g\ y)) \\ & \Rightarrow (\forall x_\beta.\forall y_\beta.h\ (x *^2 y) = (h\ x) *^3 (h\ y)) \end{aligned}$$

If f is a surjective homomorphism, g is a homomorphism, and h is any function such that for all x, $h\ (f\ x) = g\ x$, then h is a homomorphism.

In the following we present two examples, which are not automatically provable in either TPS or ΩMEGA alone, and which motivate a cooperation between the two systems.[6]

$$\begin{aligned} \textit{THM262:}\quad & \forall p_{o(o\iota)}.\textit{partition}\ p \\ & \Rightarrow \exists q_{(o\iota\iota)}.\textit{equivalence-rel}\ q \wedge (\textit{equivalence-classes}\ q) = p \end{aligned}$$

This states that if p is a partition, then there is an equivalence relation q whose equivalence classes are exactly the elements of p. We now demonstrate how a partly interactive and partly automatic proof can be constructed, and show how the integration of TPS and ΩMEGA can help with this task.

Suppose that the user begins by providing the appropriate instantiation for q (namely $\lambda x_\iota.\lambda y_\iota.\exists s_{o\iota}.p\ s \wedge s\ x \wedge s\ y$).This reduces the problem to two subgoals: proving that this lambda-term defines an equivalence relation, and proving that the equivalence classes of this relation are exactly p. In both cases, we have the hypothesis that p is a partition. The former subgoal can be proven automatically by TPS in about 35 seconds. The latter subgoal is harder for TPS; however, by using the interactive tactics for extensionality and universal generalisation, the user can reduce it to $(\textit{equivalence-classes}\ (\lambda x_\iota.\lambda y_\iota.\exists s.p\ s \wedge s\ x \wedge s\ y)\ b_{o\iota}) \equiv p\ b$. This equivalence can in turn be reduced interactively to a pair of implications, of which one (the right-to-left direction) can be proven automatically by TPS in about 30 seconds. This leaves the left-to-right direction of the equivalence as the only remaining subgoal to be proven. The automatic procedures of TPS cannot produce a proof of this subgoal, due to the complexity of the equality reasoning which is required, and so a user constructing this proof from within TPS would have to complete the proof interactively. The proof of this subgoal is non-trivial, and requires a significant amount of work on the part of the user.

However, with the integrated system, the user can begin proving THM262 in ΩMEGA, exactly as above, calling TPS to complete two of the three subgoals (none of the other systems integrated to ΩMEGA is known to be able to complete either subproof). For the remaining subgoal, instead of laboriously constructing an interactive proof, the user now has the additional option of invoking one of the other automated provers which are integrated to ΩMEGA or to call ΩMEGA's proof planner. It is very likely that an improved version of ΩMEGA$'s$ own higher-order theorem prover LEO, which specialises in reasoning about equality and extensionality, will be able to find an automatic proof of this subgoal.[7]

---

[6] The definitions used in this examples are as follows:
$\textit{partition} := \lambda s_{o(o\iota)}.(\forall p_{o\iota}.s\ p \Rightarrow (\exists.z_\iota p\ z)) \wedge (\forall x_\iota.\exists p.s\ p \wedge p\ x \wedge (\forall q_{o\iota}.s\ q \wedge q\ x \Rightarrow q = p))$
$\textit{equivalence-rel} := \lambda r_{o\iota\iota}.\textit{reflexive}\ r \wedge \textit{symmetric}\ r \wedge \textit{transitive}\ r$
$\textit{equivalence-classes} := \lambda r_{o\iota\iota}.\lambda s_{o\iota}.(\exists z_\iota.s\ z) \wedge (\forall x_\iota.s\ x \Rightarrow (\forall y_\iota.s\ y \equiv r\ x\ y))$
$\textit{reflexive} := \lambda r_{o\iota\iota}.\forall x_\iota.r\ x\ x \qquad \textit{symmetric} := \lambda r_{o\iota\iota}.\forall x_\iota.\forall y_\iota.r\ x\ y \Rightarrow r\ y\ x$
$\textit{transitive} := \lambda r_{o\iota\iota}.\forall x_\iota.\forall y_\iota.\forall z_\iota.r\ x\ y \wedge r\ y\ z \Rightarrow r\ x\ z \qquad \emptyset := \lambda X_\alpha.\bot$

[7] In principle LEO provides exactly the required extensionality treatment to solve this subgoal, but due to its prototypical implementation LEO can still handle only small search spaces; the search space defined by this problem is rather large because many free predicate variables are involved. A technically improved and heuristically better-

The following statement (which we admit is rather contrived) serves to illustrate some of the strengths and weaknesses of TPS and LEO, as it is only provable when both systems cooperate.[8]

$$(\exists \circ_{ooo} . \top \circ \top \wedge \neg(\bot \circ \bot) \wedge \neg(\bot \circ \top) \wedge \neg(\top \circ \bot)) \wedge (\forall m_{oo} . \top \in m \equiv (\top \Rightarrow \top) \in m)$$

The first conjunct claims the existence of the logical connective $\wedge$ which is specified by its truth table. In order to prove this statement primitive substitution[9] has to be employed, which is strongly supported in TPS but widely avoided in LEO. For the proof of the second statement, on the other hand, the unification of $\top \in m$ and $(\top \Rightarrow \top) \in m$ requires a recursive call to the higher-order theorem prover from within higher-order unification. This most general form of extensionality treatment is supported in LEO but not in TPS. Hence this conjunction is provable in the combined system with three straightforward interactions.

Both examples illustrate that the integrated system of TPS and ΩMEGA allows the user to complete some proofs in much fewer interactions than would be required by either system alone. In fact, the few interactions which are required are already supported by the suggestion mechanism in ΩMEGA [Benzmüller and Sorge, 1998a]. While this in itself is already a major benefit to the user, it also suggests that it should be possible to use the built-in proof planner of ΩMEGA to oversee the cooperation of the various external systems, and to produce proofs such as the one above without the necessity of user interaction.

5 Conclusion

Our objective was to integrate the two knowledge-based higher-order theorem proving environments TPS and ΩMEGA in a way that would be as transparent to the user as possible. We believe that the approach to integration described above, although designed specifically for these two systems, provides some generally interesting and elegant ideas.

Our work (see also [Benzmüller and Sorge, 1998b]) is closely related to, and was developed simultaneously with, the approach for integrating the proof planner CLAM and the interactive theorem prover HOL [Slind *et al.*, 1998a; Slind *et al.*, 1998b]. Although we must admit that our work was simplified by the fact that ΩMEGA and TPS are much more similar than are HOL and CLAM, we believe that our approach provides some additional features, e.g. the communication of definitions between the two systems, and a more transparent proof transformation process.

In conclusion, we now summarise some of the more interesting general properties of our integration method.

- The integration of ΩMEGA and TPS also includes the communication of system-specific knowledge defined in the systems' knowledge bases. TPS and

guided version of LEO, which is currently being re-implemented, will most likely be able to find the proof.

[8] Although this example looks rather trivial at first glance, to the knowledge of the authors it is currently not automatically provable by any system.

[9] The primitive substitution principle guesses instantiations for free predicate variables. In this case the prover has to guess the instantiation $\wedge$ for $\circ$ and then to verify the conditions specified by the truth table.

ΩMEGA, which are both based on classical higher-order logic, do not need to agree on common definitions, rules or other logical concepts (apart from the logical connectives which are in any case identical in both systems), as is necessary for the integration of, for example, CL^{A}M and HOL [Slind *et al.*, 1998a; Slind *et al.*, 1998b]. Instead, ΩMEGA need only communicate to TPS all of the potentially important definitions and concepts belonging to the the specific subproblem to be solved. Most importantly, ΩMEGA does not expand any definition in the focused subproblem, but leaves the decision as to whether this is useful or necessary to TPS, which can use its mechanism for selectively instantiating definitions [Bishop and Andrews, 1998]. The user may even actively prevent some defined concepts from being passed to TPS.
- TPS is not only integrated as a fully automated black box system, but can also be called as an interactive theorem prover. Thus ΩMEGA, with its hierarchically structured knowledge base, can be seen in the integrated system as a second user interface to the TPS system, with its own knowledge base. As an automated black box system, TPS can be called from ΩMEGA either alone or concurrently with other integrated theorem provers such as the first-order systems OTTER, SPASS and PROTEIN.
- The ΩMEGA system models the particular ND-calculus variant used by TPS by providing corresponding tactics in a special theory *TPS* which introduces one ΩMEGA tactic for each TPS justification. Hence any TPS proof can be translated one to one into a corresponding ΩMEGA proof plan using the tactics from theory *TPS*. As the structure of the resulting proof plans can be visualised graphically in ΩMEGA's graphical user interface LΩUI [Siekmann *et al.*, 1998] TPS thereby gains a visualisation tool and graphical interface for free.
- Proof transformation of TPS proofs (mirrored as proof plans in ΩMEGA) into proofs in ΩMEGA's basic ND-calculus is realised by tactic expansion. As ΩMEGA's 3-dimensional proof data structure ($\mathcal{PDS}$) permanently stores all different abstraction levels of a proof (the ΩMEGA basic ND-level proof at the bottom layer, the mirrored TPS proof at an abstract level, and all intermediate abstraction levels between those), proof transformation becomes and remains transparent to the user, who can freely move between different levels of abstraction in the proof.
- Non-trivial tactic expansions (such as the one for RuleP) are supported by other external reasoners that are already integrated to ΩMEGA, or even by TPS itself. This saves us from having to define and implement complicated tactic expansions from scratch. Indeed, this can serve as a general approach for a tactic-based proof transformation within a system like ΩMEGA that already provides other integrated systems: as soon as a particular expansion step seems overly complicated, one can recursively call other integrated systems that are suited to support this particular expansion step.
- The reuse of mirrored TPS proof plans within an analogy-based theorem proving approach [Melis and Carbonell, 1998] is supported by our integration, as these proof plans are explicitly stored and thus available in ΩMEGA's $\mathcal{PDS}$. They can also be stored in ΩMEGA's knowledge base.

We are currently investigating whether TPS, ΩMEGA's own higher-order theorem prover LEO (which is specialised in reasoning about extensionality) and the various first-order theorem provers which have been integrated with ΩMEGA can fruitfully cooperate. We hope to use ΩMEGA's $\mathcal{PDS}$ as the central data structure for the necessary information exchange between the cooperating systems, and ΩMEGA's planning mechanism to guide the cooperation between them.

References

[Andrews *et al.*, 1996] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[Andrews *et al.*, 1997] P. B. Andrews, S. Issar, D. Nesmith, F. Pfenning, H. Xi, and M. Bishop. *TPS3 Facilities Guide for Programmers and Users*, 1997. 207+viii pp.

[Andrews, 1981] P. B. Andrews. Theorem Proving via General Matings. *Journal of the Association for Computing Machinery*, 28(2):193–214, 1981.

[Baumgartner and Furbach, 1994] P. Baumgartner and U. Furbach. PROTEIN: A PROver with a Theory Extension INterface. In Bundy [1994], pages 769–773.

[Benzmüller and Kohlhase, 1998] C. Benzmüller and M. Kohlhase. LEO, a higher-order theorem prover. In Kirchner and Kirchner [1998], pages 139–143.

[Benzmüller and Sorge, 1998a] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In F. Giunchiglia, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 1480 in LNAI, pages 102–114, Sozopol, Bulgaria, 1998. Springer Verlag.

[Benzmüller and Sorge, 1998b] C. Benzmüller and V. Sorge. Integrating TPS with ΩMEGA. In J. Grundy and M. Newey, editors, *Theorem Proving in Higher Order Logics: Emerging Trends*, Technical Report 98-08, Department of Computer Science, pages 1–19, Canberra, Australia, 1998. The Australian National University.

[Benzmüller *et al.*, 1997] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In McCune [1997].

[Bishop and Andrews, 1998] M. Bishop and P. B. Andrews. Selectively Instantiating Definitions. In Kirchner and Kirchner [1998], pages 365–380.

[Bundy, 1994] A. Bundy, editor. *Proceedings of CADE–12*, volume 814 of *LNAI*. Springer, Berlin, Germany, 1994.

[Dahn *et al.*, 1994] B. I. Dahn, J. Gehne, T. Honigmann, L. Walther, and A. Wolf. Integrating Logical Functions with ILF. Technical Report 94-10, Institut für Mathematik, Humboldt Universität zu Berlin, Germany, 1994.

[Felty and Howe, 1997] A. P. Felty and D. J. Howe. Hybrid Interactive Theorem Proving Using Nuprl and HOL. In McCune [1997], pages 351–365.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[Giunchiglia *et al.*, 1996] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories – Towards an Architecture for Open Mechanized Reasoning Systems. In F. Baader and K. U. Schulz, editors, *Frontiers of combining systems*, volume 3 of *Applied logic series*, pages 157 – 174, Dordrecht, The Netherlands, 1996. Kluwer Academic Publishers.

[Huang and Fiedler, 1997] X. Huang and A. Fiedler. Proof Verbalization in *PROVERB*. In *Proceedings of the First International Workshop on Proof Transformation and Presentation*, pages 35–36, Schloss Dagstuhl, Germany, 1997.

[Huang *et al.*, 1994] X. Huang, M. Kerber, J. Richts, and A. Sehn. Planning Mathematical Proofs with Methods. *Journal of Information Processing and Cybernetics (formerly: EIK)*, 30(5/6):277–291, 1994.

[Kerber *et al.*, 1998] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[Kirchner and Kirchner, 1998] C. Kirchner and H. Kirchner, editors. *Proceedings of CADE–15*, volume 1421 of *LNAI*. Springer, Berlin, Germany, 1998.

[McCune, 1994] W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[McCune, 1997] W. McCune, editor. *Proceedings of CADE–14*, volume 1249 of *LNAI*. Springer, Berlin, Germany, 1997.

[Meier, 1997] A. Meier. Übersetzung automatisch erzeugter Beweise auf Faktenebene. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[Melis and Carbonell, 1998] E. Melis and J .G. Carbonell. An argument for derivational analogy. *Advances in Analogy and Research*, 1998.

[Miller, 1984] D. Miller. Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs. In R.E. Shostak, editor, *Proceedings of CADE–7*, volume 170 of *LNCS*, pages 375–303. Springer, Berlin, Germany, 1984.

[Pfenning, 1987] F. Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, Pittsburgh Pa., 1987.

[Sehn, 1995] A. Sehn. DECLAME – eine deklarative Sprache zur Repräsentation von Methoden. Master's thesis, Computer Science Department, Universität des Saarlandes, 1995.

[Siekmann *et al.*, 1998] J. Siekmann, S. M. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, M. Kohlhase, K. Konrad, E. Melis, A. Meier, and V. Sorge. LΩUI: A Distributed Graphical User Interface for the Omega Proof System. International Workshop on User Interfaces for Theorem Provers, 1998.

[Slind *et al.*, 1998a] K. Slind, M. Gordon, R. Boulton, and A. Bundy. An Interface between CLAM and HOL. In Kirchner and Kirchner [1998], pages 134–138.

[Slind *et al.*, 1998b] K. Slind, M. Gordon, R. Boulton, and A. Bundy. An Interface between CLAM and HOL. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume 1479 of *LNCS*, pages 87–104. Springer, Berlin, Germany, 1998.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP Problem Library. In Bundy [1994], pages 252–266.

[Weidenbach *et al.*, 1996] Ch. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER, version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of CADE–13*, volume 1104 of *LNAI*. Springer, Berlin, Germany, 1996.

In the appendix we illustrate the integration architecture by presenting some concrete information on the interaction between ΩMEGA and TPS when proving THM136 (see Example 1).

A: Translating from ΩMEGA to TPS

Example 2 Problem-file. This is the content of the *problem-file* for THM136 generated by ΩMEGA and passed to TPS. The line with keyword ASSERTION defines the theorem to be proved and the line with keyword LINES introduces the initial partial proof to be completed by TPS, which here consists only of one line. A reference to the corresponding open proof line in ΩMEGA (the entry "(OMEGA-LABEL THM136)") and some further information belonging to ΩMEGA can be found at the end of this proof line. Note that the defined concepts *transitive*, *transitive-closure* and *sub-relation* are not expanded in this initial partial proof; they are passed to TPS as defined abbreviations (in the three lines with keyword DEF-ABBREV).

```
(DEFSAVEDPROOF OMEGA-SUBPROBLEM-THM136 (1998 9 30)
 (ASSERTION
  "[FORALL R(OaA)[TRANSITIVE(O(OAA))[TRANSITIVE-CLOSURE(OAA(OAA))R(OAA)] ] ] ")
 (NEXT-PLAN-NO 2) (PLANS ((1)))
 (LINES
  (1 NIL "[FORALL R(OAA)[TRANSITIVE [TRANSITIVE-CLOSURE R(OAA)] ] ] "
```

```
  PLAN1 NIL NIL "((OMEGA-LABEL THM136) (OMEGA-JUSTIFICATION OPEN))"))
0
((DEF-ABBREV TRANSITIVE (TYPE "O(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE TRANSITIVE) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-50(OAA)
     [FORALL DC-51(A)
      [FORALL DC-52(A)
       [FORALL DC-53(A)
        [IMPLIES [AND [DC-50(OAA)DC-51(A)DC-52(A)] [DC-50(OAA)DC-52(A)DC-53(A)]]
                 [DC-50(OAA)DC-51(A)DC-53(A)]]]]]]")
  (MHELP
   "Definition of the predicate for transitivity. (transitive R) is true, iff Rxy and Ryz imply Rxz. "))
 (DEF-ABBREV SUB-RELATION (TYPE "O(OAA)(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE SUB-RELATION) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-54(OAA)
     [LAMBDA DC-55(OAA)
      [FORALL DC-56(A)
       [FORALL DC-57(A)[IMPLIES [DC-54(OAA)DC-56(A)DC-57(A)] [DC-55(OAA)DC-56(A)DC-57(A)]]]]]]")
  (MHELP
 "Definition of the predicate for sub-relations. (sub-relation R R') is true, iff Rxy implies R'xy. "))
 (DEF-ABBREV TRANSITIVE-CLOSURE (TYPE "OAA(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE TRANSITIVE-CLOSURE) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-58(OAA)
     [LAMBDA DC-59(A)
      [LAMBDA DC-60(A)
       [FORALL DC-61(OAA)
        [IMPLIES [AND [SUB-RELATION(O(OAA)(OAA))DC-58(OAA)DC-61(OAA)] [TRANSITIVE DC-61(OAA)]]
                 [DC-61(OAA)DC-59(A)DC-60(A)]]]]]]")
  (MHELP "Definition of the transitive closure as in TPS. ")))
(COMMENT "OMEGA proof (report problems to the OMEGA group)")
(LOCKED (1)))
```

B: Proof Construction in TPS

Example 3 TPS *Proof.* Figure 4 presents a screenshot of the TPS interface displaying the TPS proof for THM136. This proof is discussed in detail in [Bishop and Andrews, 1998].

C & D: Translating from TPS to ΩMEGA and Inserting the Proof Plan

Example 4 ΩMEGA *Proof Plan.* ΩMEGA's special theory *TPS* provides one proof tactic for each TPS justification. Thus the proof presented in Example 3 can be translated one to one into a proof plan using the proof tactics of this theory. Tactics defined in this special theory have the prefix "TPS". The structure of this proof plan can be graphically visualised in ΩMEGA's graphical user interface LΩUI, as presented in Example 6.

```
THM136 ()         ! (FORALL [R:(O BB BB)]                               TPS*UGEN: (R) (L23)
                     (TRANSITIVE (TRANSITIVE-CLOSURE R)))
L23    ()         ! (TRANSITIVE (TRANSITIVE-CLOSURE R))                 TPS*EQUIVWFFS: (L22)
L22    ()         ! (FORALL [DC-51:BB,DC-52:BB,DC-53:BB]              TPS*UGEN: (DC-51) (L21)
                     (IMPLIES
                      (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                           (TRANSITIVE-CLOSURE R DC-52 DC-53))
                      (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L21    ()         ! (FORALL [DC-52:BB,DC-53:BB]                       TPS*UGEN: (DC-52) (L20)
                     (IMPLIES
                      (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                           (TRANSITIVE-CLOSURE R DC-52 DC-53))
                      (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L20    ()         ! (FORALL [DC-53:BB]                                TPS*UGEN: (DC-53) (L19)
                     (IMPLIES
                      (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                           (TRANSITIVE-CLOSURE R DC-52 DC-53))
                      (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L19    ()         ! (IMPLIES                                              TPS*DEDUCT: (L18)
                    (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                         (TRANSITIVE-CLOSURE R DC-52 DC-53))
                    (TRANSITIVE-CLOSURE R DC-51 DC-53))
```

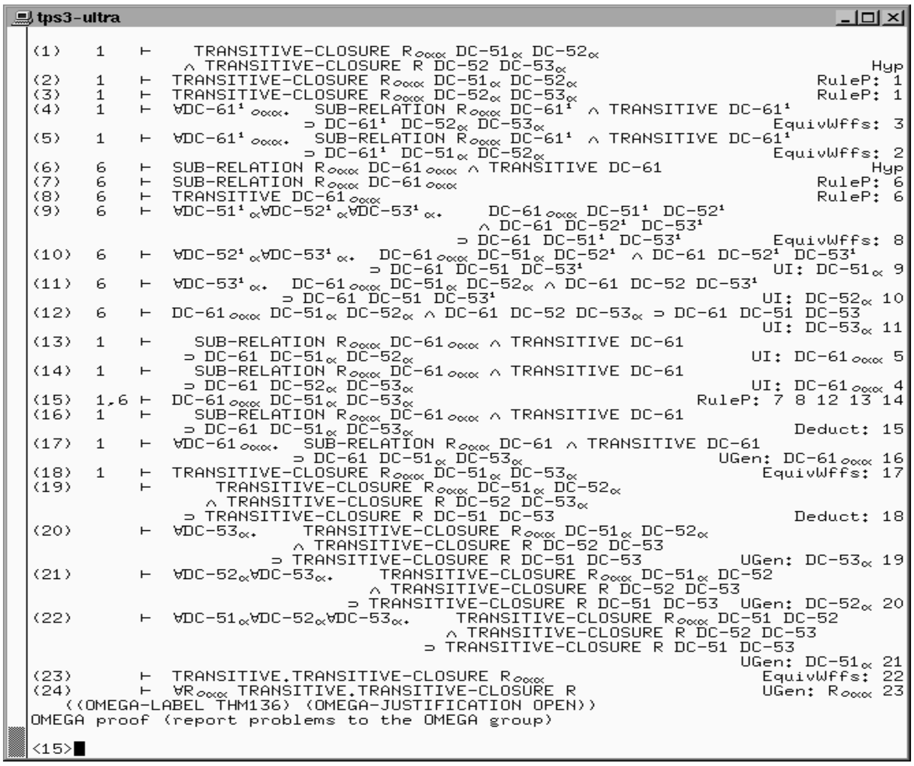

Figure 4: TPS-Xterm with the proof of THM136

```
L18   (L1)     ! (TRANSITIVE-CLOSURE R DC-51 DC-53)                    TPS*EQUIVWFFS: (L17)
L17   (L1)     ! (FORALL [DC-61:(O BB BB)]                            TPS*UGEN: (DC-61) (L16)
                  (IMPLIES
                   (AND (SUB-RELATION R DC-61)
                        (TRANSITIVE DC-61))
                   (DC-61 DC-51 DC-53)))
L16   (L1)     ! (IMPLIES                                                  TPS*DEDUCT: (L15)
                  (AND (SUB-RELATION R DC-61)
                       (TRANSITIVE DC-61))
                  (DC-61 DC-51 DC-53))
L15   (L1 L6)  ! (DC-61 DC-51 DC-53)                        TPS*RULEP: (L7 L8 L12 L13 L14)
L14   (L1)     ! (IMPLIES                                              TPS*UI: (DC-61) (L4)
                  (AND (SUB-RELATION R DC-61)
                       (TRANSITIVE DC-61))
                  (DC-61 DC-52 DC-53))
L13   (L1)     ! (IMPLIES                                              TPS*UI: (DC-61) (L5)
                  (AND (SUB-RELATION R DC-61)
                       (TRANSITIVE DC-61))
                  (DC-61 DC-51 DC-52))
L12   (L6)     ! (IMPLIES                                             TPS*UI: (DC-53) (L11)
                  (AND (DC-61 DC-51 DC-52) (DC-61 DC-52 DC-53))
                  (DC-61 DC-51 DC-53))
L11   (L6)     ! (FORALL [DC-53^1:BB]                                 TPS*UI: (DC-52) (L10)
                  (IMPLIES
                   (AND (DC-61 DC-51 DC-52)
                        (DC-61 DC-52 DC-53^1))
                   (DC-61 DC-51 DC-53^1)))
L10   (L6)     ! (FORALL [DC-52^1:BB,DC-53^1:BB]                       TPS*UI: (DC-51) (L9)
                  (IMPLIES
                   (AND (DC-61 DC-51 DC-52^1)
                        (DC-61 DC-52^1 DC-53^1))
                   (DC-61 DC-51 DC-53^1)))
```

```
L9     (L6)      ! (FORALL [DC-51^1:BB,DC-52^1:BB,DC-53^1:BB]                    TPS*EQUIVWFFS: (L8)
                    (IMPLIES
                     (AND (DC-61 DC-51^1 DC-52^1)
                          (DC-61 DC-52^1 DC-53^1))
                     (DC-61 DC-51^1 DC-53^1)))
L8     (L6)      ! (TRANSITIVE DC-61)                                                TPS*RULEP: (L6)
L7     (L6)      ! (SUB-RELATION R DC-61)                                            TPS*RULEP: (L6)
L6     (L6)      ! (AND (SUB-RELATION R DC-61) (TRANSITIVE DC-61))                           TPS*HYP
L5     (L1)      ! (FORALL [DC-61^1:(O BB BB)]                                   TPS*EQUIVWFFS: (L2)
                    (IMPLIES
                     (AND (SUB-RELATION R DC-61^1)
                          (TRANSITIVE DC-61^1))
                     (DC-61^1 DC-51 DC-52)))
L4     (L1)      ! (FORALL [DC-61^1:(O BB BB)]                                   TPS*EQUIVWFFS: (L3)
                    (IMPLIES
                     (AND (SUB-RELATION R DC-61^1)
                          (TRANSITIVE DC-61^1))
                     (DC-61^1 DC-52 DC-53)))
L3     (L1)      ! (TRANSITIVE-CLOSURE R DC-52 DC-53)                                TPS*RULEP: (L1)
L2     (L1)      ! (TRANSITIVE-CLOSURE R DC-51 DC-52)                                TPS*RULEP: (L1)
L1     (L1)      ! (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)                                   TPS*HYP
                        (TRANSITIVE-CLOSURE R DC-52 DC-53))
```

E: Transparent Proof Transformation by Proof Plan Expansion

Example 5 Modelling TPS*'s calculus in* ΩMEGA*'s theory TPS .* The tactics in ΩMEGA's special theory TPS contain expansion information that allows proof plans constructed in this theory to be mapped to ΩMEGA proofs on a lower abstraction level. We present some sample expansions here. The simplest is tps*Conj, which is simply mapped to the ΩMEGA tactic ande. The expansion of tps*Neg first analyses the given situation and then maps either to Pushneg or Pullneg. tps*RuleP recursively invokes an external propositional logic prover integrated to ΩMEGA.

```
(defun tpstac=expand-tps*Conj (outline parameters)
  (tacl~init outline)
   (tacl~apply 'ande outline nil)
  (tacl~end))

(defun tpstac=expand-tps*Neg (outline parameters)
  (tacl~init outline)
  (cond ((tpstac=pushneg-a-p (node~formula (car outline)) (node~formula (cadr outline)))
         (tacl~apply 'pushneg outline nil))
        ((tpstac=pullneg-a-p (node~formula (cadr outline)) (node~formula (car outline)))
         (tacl~apply 'pullneg outline nil))
        (t (warn "Something went wrong while expanding justification tps*Neg")))
  (tacl~end))

(defun tpstac=expand-tps*RuleP (outline parameters)
  (declare (ignore parameters))
  (let* ((node (car outline))
         (premises (just~premises (node~justification node))))
    (tacl~init outline)
    (tpstac=call-external-atp node premises)
    (tacl~end)
    (setf (pdsj~status (node~justification node)) "untested")))
```

Example 6 ΩMEGA*-proof.* Finally, we present in figure 5 the visualization of the original TPS proof (as a proof plan) in ΩMEGA's graphical user interface LΩUI. By expanding all nodes exactly one step, we reach another layer in ΩMEGA's 3-dimensional $\mathcal{PDS}$ which is visualized in the second screenshot. Here the squares represent the recursive calls to a propositional theorem prover which are obtained by the expansion of tactic *tps*RuleP*.

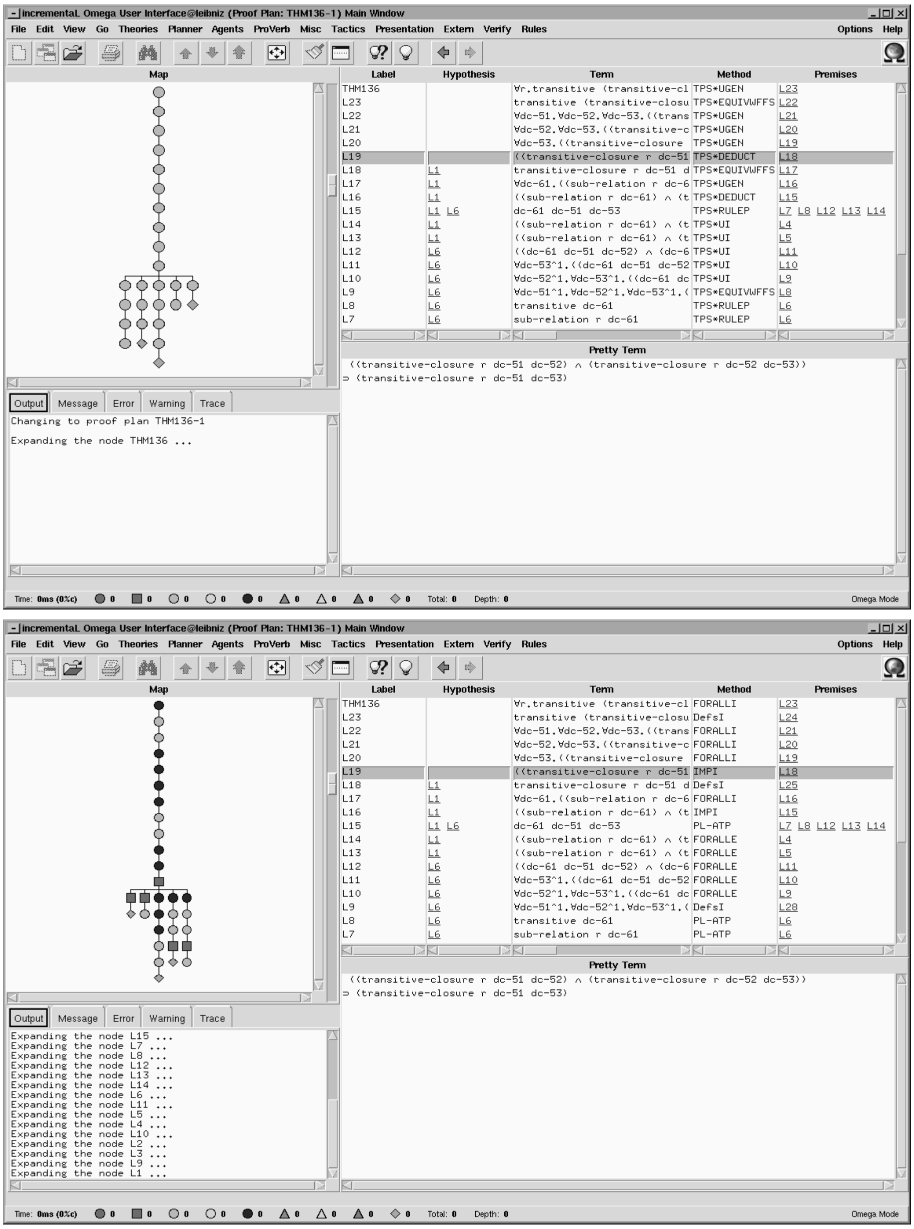

Figure 5: Transparent proof transformation within ΩMEGA's 3-dimensional $\mathcal{PDS}$.