

# Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure

GEOFF SUTCLIFFE

University of Miami

and

CHRISTOPH BENZMÜLLER

Articulate Software

---

The Thousands of Problems for Theorem Provers (TPTP) problem library is the basis of a well known and well established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems. The extension of the TPTP from first-order form (FOF) logic to typed higher-order form (THF) logic has provided a basis for new development and application of ATP systems for higher-order logic. Key developments have been the specification of the THF language, the addition of higher-order problems to the TPTP, the development of the TPTP THF infrastructure, several ATP systems for higher-order logic, and the use of higher-order ATP in a range of domains. This paper surveys these developments.

---

## 1. INTRODUCTION

The Thousands of Problems for Theorem Provers (TPTP) problem library [90] is the basis of a well known and well established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems. The infrastructure includes the problem library itself, the TPTP language [92], the SZS ontologies [89], the Thousands of Solutions from Theorem Provers (TSTP) solution library, various tools associated with the libraries [88], and the CADE ATP System Competition (CASC) [93]. All components are freely available online from the TPTP web site <http://www.tptp.org>. This infrastructure has been central to the progress that has been made in the development of high performance ATP systems – most state of the art systems natively read the TPTP language, many produce proofs or models in the TSTP format, much testing and development is done using the TPTP library, and CASC is an annual focal point where developers meet to discuss new ideas and advances in ATP techniques.

The TPTP was originally developed in 1992-3 using the clause normal form (CNF) fragment of first-order logic, and was extended to full first-order form (FOF) in 1997. In 2008-9 the typed higher-order form (THF) was added.<sup>1</sup> The first release of the TPTP problem library with THF problems was in 2009, containing 2729 THF problems. The problems are written in a core subset of the THF language,

---

<sup>1</sup>A typed first-order form (TFF) is now also under development.

---

This research received funding from the European Community's Seventh Framework Programme FP7/2007-2013, under grant agreement PIIF-GA-2008-219982. The second author also received funding from Deutsche Forschungsgemeinschaft (DFG), under grant agreement BE 2501/6-1.

named THF0 [20], based on Church’s simply typed lambda calculus. The full THF language and extended THFX language were also available at the time, but the initial limitation to the THF0 core allowed users to adopt the language without being swamped by the richness of the full THF language. The THF and THFX languages provide a richer type system, the ability to reason about types, more term and type constructs, more connectives, and “syntactic sugar” that is usefully expressive. In conjunction with the addition of THF0 problems to the problem library, the TPTP infrastructure was extended to support THF.

The addition of THF to the TPTP problem library and infrastructure has had an immediate impact on progress in the development of automated reasoning in higher-order logic [91]. The impact has been in terms of higher-order ATP system development, problem encoding in higher-order logic, higher-order ATP system usage, and raised awareness of the growing potential for applications of automated reasoning in higher-order logic.

A major application area of higher-order logic is hardware and software verification. Several interactive higher-order proof assistants put an emphasis on this: the Isabelle/HOL system [73] has been applied in the Verisoft project [45]; the Coq system [22] has been applied in the CompCert verified compiler project [66]; the HOL Light system [54] has been applied to the verification of several floating-point algorithms at Intel [53]; and ACL2 [61] has been used to prove properties of state-of-the-art commercial microprocessors prior to fabrication [26]. Another promising application area for higher-order logic is knowledge based reasoning. Knowledge based projects such as Cyc [68] and SUMO [72] contain a significant fraction of higher-order constructs. A strong argument for adding higher-order ATP to this application area is its demand for natural and human consumable problem and solution representations, which are harder to achieve after translating higher-order content into less expressible frameworks such as first-order logic. Further application areas of higher-order logic include computer-supported mathematics [84, 50, 12, 47], computational linguistics [63], reasoning within and about multimodal logics [18], and logics of access control [35, 13].

Interactive higher-order proof assistants have been used in all these, and other, applications of higher-order logic. The interactive construction of formal proofs for such applications often requires a large number of user interactions – a resource intensive task that is carried out by highly trained specialists. Often only a few of the interaction steps really require human ingenuity, and many of them could be avoided through better automation support. While there have been several successful integrations of first-order ATP systems with interactive higher-order proof assistants to provide automated support [23, 59, 8], there have been several barriers that have hampered the application of higher-order ATP systems in this sense: (i) the available higher-order ATP systems have not yet been optimized for this task; (ii) typically there are large syntax gaps between the higher-order representation languages of the proof assistants, and the input languages of the higher-order ATP systems; (iii) the results of the higher-order ATP systems have to be correctly interpreted; (iv) their proof objects often need to be translated back to the application domain. The development of the THF infrastructure, and the consequent increased automation in higher-order ATP, provides the way to overcome these problems. This will provide increased possibilities for automated support in

interactive higher-order proof assistants, which will benefit their application and reduce user interaction costs.

This paper surveys the THF logic and language (Sections 2 and 3), the higher-order TPTP and infrastructure (Section 4), some higher-order ATP systems for the THF language (Section 5), and some applications using THF (Section 6).

## 2. THE TPTP'S HIGHER-ORDER LOGIC

### 2.1 Higher-order Logic

There are many quite different frameworks that fall under the general label of “higher-order logic”. The notion reaches back to Frege’s original predicate calculus [41]. Inconsistencies in Frege’s system, caused by the circularity of constructions such as “the set of all sets that do not contain themselves”, made it clear that the expressivity of the language had to be restricted in some way. One line of development, which became the traditional route for mathematical logic, and which is not addressed further here, is the development of axiomatic first-order set theories, e.g. Zermelo-Fraenkel set theory. Russell suggested using type hierarchies, and worked out ramified type theory. Church (inspired by work of Carnap) later introduced simple type theory [34], a higher-order framework built on his simply typed  $\lambda$  calculus, employing types to reduce expressivity and to remedy paradoxes and inconsistencies. Simple type theory is often also called classical higher-order logic.

Via the Curry-Howard isomorphism, typed  $\lambda$ -calculi can also be exploited to encode proofs as types. The simply typed  $\lambda$ -calculus, for example, is sufficient for encoding propositional logic. More expressive logics can be encoded using dependent types and polymorphism [82, 44, 39]. In combination with Martin L  f’s intuitionistic theory of types [67], originally developed for formalizing constructive mathematics, this research led the foundations of modern type theory.

During the last decades various proof assistants have been built for both classical higher-order logic and type theory. Prominent interactive provers for classical higher logic include HOL [49], HOL Light [52], PVS [75], Isabelle/HOL [73], and OMEGA [86]. Prominent interactive type theory provers include the pioneering Automath system [70], Nuprl [1], Lego [79], Matita [7], and Coq [22]. The latter three are based on the calculus of constructions [36]. Further type theory systems are the logical frameworks Elf [76] and Twelf [77].

The work presented in this paper aims at fostering the automation of classical higher-order logic and type theory. The initial focus has been on the former. Automation of classical higher-order logic has been pioneered by the work of Andrews on resolution in type theory [2], by Huet’s pre-unification algorithm [58] and his constrained resolution calculus [57], and by Jensen and Pietrowski’s [78] work. More recently extensionality and equality reasoning in Church’s type theory has been studied [11, 16, 28]. The TPS system [5], which is based on a higher-order mating calculus, is a pioneering ATP system for Church’s simple type theory. Higher-order ATP systems based on extensional higher-order resolution are LEO [17] and LEO-II [19], and a system based on higher-order extensional tableaux is Satallax [9]. Otter- $\lambda$  [10] is an extension of the first-order system Otter to an untyped variant of Church’s type theory. Church’s simple type theory has been a common and simple subset of the logics supported by such systems, and thus motivated using it

as the starting point for the THF logic, in the form of THF0. THF0 is therefore synonymous with Church’s simple type theory.

THF0 is the starting point for the development of more expressive languages in the THF family, in order to serve and support type theory provers in the future. Features beyond THF0 that are already in the full THF language are described in Section 3.3. The inclusion of these has been motivated by their use in existing reasoning tools for logics beyond Church’s simple type theory. For just a few examples, subtyping is supported in Nuprl, Isabelle, and Matita; binders for choice and/or description are supported in Isabelle, HOL, NuPrL, and Satallax; dependent product and sum types are supported in Automath, Lego, and Coq; sequents are supported in the OMEGA system [86].

## 2.2 Church’s Simple Type Theory

Church’s simple type theory is based on the simply typed  $\lambda$  calculus. The set of simple types is freely generated from basic types  $\iota$  and  $o$ , and possibly further base types using the function type constructor  $\rightarrow$ . Higher-order terms are built up from simply typed variables ( $X_\alpha$ ), simply typed constants ( $c_\alpha$ ),  $\lambda$ -abstraction, and application. It is assumed that sufficiently many special logical constants are available, so that all other logical connectives can be defined. For example, it is assumed that  $\neg_{o \rightarrow o}$ ,  $\vee_{o \rightarrow o \rightarrow o}$ , and  $\Pi_{(\alpha \rightarrow o) \rightarrow o}$  (for all simple types  $\alpha$ ) are given, where  $\Pi_{\dots}$  is used to encode universal quantification. The semantics of these logical symbols is fixed according to their intuitive meaning.

Well-formed (simply typed) higher-order terms are defined simultaneously for all simple types  $\alpha$ . Variables and constants of type  $\alpha$  are well-formed terms of type  $\alpha$ . Given a variable  $X$  of type  $\alpha$  and a term  $T$  of type  $\beta$ , the abstraction term  $\lambda X.T$  is well-formed and of type  $\alpha \rightarrow \beta$ . Given terms  $S$  of type  $\alpha \rightarrow \beta$  and  $T$  of type  $\alpha$ , the application term  $(S\ T)$  is well-formed and of type  $\beta$ .

The choice of semantics for higher-order logic is of interest, as, unlike the first-order case, there are different options [15, 16]. The semantics for THF is Henkin semantics with choice (Henkin semantics by definition also includes Boolean and functional extensionality) [16, 55].<sup>2</sup> However, there is no intention to limit the semantics, e.g., to fully extensional semantics only. The THF language is designed to express problems syntactically, with the semantics being specified separately, as illustrated in Section 3.2.

## 3. THE THF LANGUAGE

One of the keys to the success of the TPTP and related infrastructure is the consistent use of the TPTP language. The TPTP language is a human-readable, easily machine-parsable, flexible and extensible language suitable for writing both ATP problems and solutions. The THF language is a syntactically conservative extension of the untyped first-order TPTP language, adding the syntax for higher-order logic. Maintaining a consistent style between the first-order and higher-order languages facilitates reuse of infrastructure for processing TPTP format data, e.g., parsing tools, formatting tools, system testing, and result analysis (see Section 4).

<sup>2</sup>Initially Henkin semantics without choice was used, but choice was added very soon after the first TPTP THF release, due to its common use in higher-order ATP systems.

A particular feature of the TPTP language, which has been maintained in THF, is Prolog compatibility. This allows an annotated formula to be read with a single Prolog `read/1` call, in the context of appropriate operator definitions. There are good reasons for maintaining Prolog compatibility [92], including the ATP community's familiarity with the generic style of Prolog syntax, fast prototyping and development of TPTP compliant ATP systems and software (without the need to write IO routines), and easy reuse or extension of existing TPTP tools (e.g., GDV and IDV, as mentioned in Section 4.3.3). Reuse and upgrade of internal support tools in the TPTP was a key to the rapid development of the THF part of the TPTP.

### 3.1 General Features of the TPTP Language

The top level building blocks of the TPTP language are *annotated formulae*, *include directives*, and *comments*. An annotated formula has the form:

*language*(*name*, *role*, *formula*, [*source*, [*useful\_info*]]).

An example annotated first-order formula, supplied from a file, is:

```
fof(union,axiom,
  ( ! [X,A,B] :
    ( member(X,union(A,B))
      <=> ( member(X,A)
          | member(X,B) ) )
    file('SET006+0.ax',union),
    [description('Definition of union'), relevance(0.9)]).
```

The *languages* supported are clause normal form (*cnf*), first-order form (*fof*), and typed higher-order form (*thf*). The *role*, e.g., *axiom*, *lemma*, *conjecture*, defines the use of the formula in an ATP system. The forms of identifiers for uninterpreted functions, predicates, and variables follow Prolog conventions, i.e., functions and predicates start with a lowercase letter, variables start with an uppercase letter, and all contain only alphanumeric characters and underscore. The TPTP language also supports interpreted symbols, which either start with a \$, or are composed of non-alphanumeric characters. The basic logical connectives are `!`, `?`, `~`, `|`, `&`, `=>`, `<=`, `<=>`, and `<^>`, for  $\forall$ ,  $\exists$ ,  $\neg$ ,  $\vee$ ,  $\wedge$ ,  $\Rightarrow$ ,  $\Leftarrow$ ,  $\Leftrightarrow$ , and  $\oplus$  respectively. Quantified variables follow the quantifier in square brackets, with a colon to separate the quantification from the logical formula. The *source* is an optional term describing where the formula came from, e.g., an input file or an inference. The *useful\_info* is an optional list of terms from user applications.

An *include* directive may include an entire file, or may specify the names of the annotated formulae that are to be included from the file. Comments in the TPTP language extend from a `%` character to the end of the line, or may be block comments within `/* ...*/` bracketing.

Figure 1 presents an example problem file, `SET171^3.p`, written in the core THF0 language. A selection of relevant formulae from the *included* axiom file, `SET008^0.ax`, is shown in Figure 2. The example is from the domain of set theory, stating the distributivity of union and intersection. The first section of each TPTP problem file, as shown in Figure 1, is a header that contains information for the user. This information is not for use by ATP systems. It is divided into four parts. The first part identifies and describes the problem, the second part provides

information about occurrences of the problem in the literature and elsewhere, the third part gives the problem's status as an SZS ontology value [89] and a table of syntactic measurements made on the problem, and the last part contains general comments about the problem. For THF problems the status value is for the default semantics – Henkin semantics with choice. If the status is known to be different for other semantics, e.g., without extensionality, this is provided on subsequent lines, with the modified semantics noted, e.g.,

```
% Status      : Theorem
%              : Without Boolean extensionality : CounterSatisfiable
%              : Without xi extensionality : CounterSatisfiable
```

Following the header are any necessary `include` directives (typically for axiom files containing axiom annotated formulae), and then the annotated formulae for the problem, as shown in Figure 1.

### 3.2 The THF0 Language

The example in Figures 1 and 2 is a higher-order version of the first-order TPTP problem SET171+3, which employs first-order set theory to achieve a first-order encoding suitable for first-order theorem provers. The encoding in THF0 exploits the fact that higher-order logic provides a naturally built-in set theory, based on the idea of identifying sets with their characteristic functions. As a result the higher-order encoding of a set theory problem can often be solved more efficiently than the corresponding first-order encoding [21].

The first and third annotated formulae of Figure 2 are *type declarations* that declare the type signatures of  $\cup$  and  $\cap$ . A simple type declaration has the form *constant\_symbol:signature*. For example, the type declaration

```
thf(union_decl,type,(
  union: ( $i > $o ) > ( $i > $o ) > $i > $o )).
```

declares the symbol `union` (for  $\cup$ ), to be of type  $(\iota \rightarrow o) \rightarrow (\iota \rightarrow o) \rightarrow \iota \rightarrow o$ . The mapping arrow is right associative. Thus, `union` subsequently expects two sets as its arguments, and returns a set. Note the use of the TPTP interpreted symbols `$i` and `$o` for the standard atomic types  $\iota$  and  $o$ . In addition to `$i` and `$o`, THF defines `$tType` to denote the collection of all atomic types. `$tType` is used to introduce further atomic types on the fly. For example, the following introduces the atomic type  $u$  together with a corresponding constant symbol `in_u` (for  $\in$ ) of type  $u \rightarrow (u \rightarrow o) \rightarrow o$ .

```
thf(type_u,type,(
  u: $tType)).

thf(const_in_u,type,(
  in_u: ( u > ( u > $o ) > $o ) ).
```

In a THF problem, all new atomic types must be declared before use, and all uninterpreted symbols must have their signatures declared before use. THF0 does not support polymorphism, product types or dependent types – such language constructs are addressed in the full THF language (see Section 3.3).

The second and fourth annotated formulae of Figure 2 are *axioms* that specify the meanings of  $\cup$  and  $\cap$ . A THF0 logical formula can use the basic TPTP connectives,

```

%-----
% File      : SET171^3 : TPTP v4.0.1. Released v3.6.0.
% Domain    : Set Theory
% Problem    : Union distributes over intersection
% Version    : [BS+08] axioms.
% English    : The union of X and (the intersection of Y and Z) is the
%              intersection of (the union of X and Y) and (the union of X and Z).

% Refs      : [BS+05] Benzmueller et al. (2005), Can a Higher-Order and a Fi
%              : [BS+08] Benzmueller et al. (2008), Combined Reasoning by Autom
%              : [Ben08] Benzmueller (2008), Email to Geoff Sutcliffe
% Source     : [Ben08]
% Names      :

% Status     : Theorem
% Rating      : 0.00 v4.0.1, 0.33 v3.7.0
% Syntax      : Number of formulae      : 29 ( 0 unit; 14 type; 14 defn)
%              Number of atoms          : 219 ( 19 equality; 53 variable)
%              Maximal formula depth    : 9 ( 6 average)
%              Number of connectives    : 46 ( 5 ~; 3 |; 6 &; 31 @)
%              ( 0 <=>; 1 ==>; 0 <=; 0 <~>)
%              ( 0 ~|; 0 ~&; 0 !!; 0 ??)
%              Number of type conns     : 73 ( 73 >; 0 *; 0 +; 0 <<)
%              Number of symbols        : 18 ( 14 :; 0 :=)
%              Number of variables      : 38 ( 1 sgn; 4 !; 2 ?; 32 ^)
%              ( 38 :; 0 :=; 0 !>; 0 ?*)
%              ( 0 @-; 0 @+)

% Comments   : THF0 syntax
%-----
%----Basic set theory definitions
include('Axioms/SET008^0.ax').
%-----
thf(union_distributes_over_intersection,conjecture,(
  ! [A: $i > $o,B: $i > $o,C: $i > $o] :
    ( ( union @ A @ ( intersection @ B @ C ) )
      = ( intersection @ ( union @ A @ B ) @ ( union @ A @ C ) ) ) ).

%-----

```

Fig. 1. The TPTP problem file SET171^3.p

$\lambda$ -abstraction using the  $\wedge$  quantifier followed by a list of typed  $\lambda$ -bound variables in square brackets, and function application using the  $@$  connective. Function application is left associative. (The explicit function application operator  $@$  is necessary for parsing function application expressions in Prolog.) Additionally, universally and existentially quantified variables must be typed. For example, the axiom

```

thf(union,definition,
  ( union
    = ( ^ [X: $i > $o,Y: $i > $o,U: $i] :
        ( ( X @ U )
          | ( Y @ U ) ) ) ) ).

```

specifies the meaning of  $\cup$  by equating it to  $\lambda X_{\iota \rightarrow o}, Y_{\iota \rightarrow o}, U_{\iota}.((X U) \vee (Y U))$ . Thus

```

%-----
thf(union_decl,type,(
  union: ( $i > $o ) > ( $i > $o ) > $i > $o )).

thf(union,definition,(
  ( union
    = ( ^ [X: $i > $o,Y: $i > $o,U: $i] :
      ( ( X @ U )
        | ( Y @ U ) ) ) ) ).

thf(intersection_decl,type,(
  intersection: ( $i > $o ) > ( $i > $o ) > $i > $o )).

thf(intersection,definition,(
  ( intersection
    = ( ^ [X: $i > $o,Y: $i > $o,U: $i] :
      ( ( X @ U )
        & ( Y @ U ) ) ) ) ).
%-----

```

Fig. 2. Extract from the TPTP axiom file SET008~0.p

the union of two sets  $X$  and  $Y$  is reduced to the disjunction of applying the sets  $X$  and  $Y$  (seen as their characteristic functions).

The annotated formula of Figure 1 is the *conjecture* to be proved.

```

thf(union_distributes_over_intersection,conjecture,(
  ! [A: $i > $o,B: $i > $o,C: $i > $o] :
    ( ( union @ A @ ( intersection @ B @ C ) )
      = ( intersection @ ( union @ A @ B ) @ ( union @ A @ C ) ) ) ).

```

This encodes  $\forall A_{l \rightarrow o}. \forall B_{l \rightarrow o}. \forall C_{l \rightarrow o}. (A \cup (B \cap C)) = ((A \cup B) \cap (A \cup C))$ .

An advantage of higher-order logic over first-order logic is that the  $\forall$  and  $\exists$  quantifiers can be encoded [34]: Quantification is expressed using the logical constant symbols  $\Pi$  and  $\Sigma$  in connection with  $\lambda$ -abstraction. Higher-order systems typically make use of this feature in order to avoid introducing and supporting binders in addition to  $\lambda$ . THF0 provides the logical constants  $!!$  and  $??$  for  $\Pi$  and  $\Sigma$  respectively, and leaves use of the universal and existential quantifiers open to the user. Here is an encoding of the conjecture from Figure 1, using  $!!$  instead of  $!$ .

```

thf(union_distributes_over_intersection,conjecture,(
  !! ( ^ [A: $i > $o] :
    !! ( ^ [B: $i > $o] :
      !! ( ^ [C: $i > $o] :
        ( ( union @ A @ ( intersection @ B @ C ) )
          = ( intersection @ ( union @ A @ B )
            @ ( union @ A @ C ) ) ) ) ) ).

```

This encodes  $\Pi \lambda A_{l \rightarrow o}. \Pi \lambda B_{l \rightarrow o}. \Pi \lambda C_{l \rightarrow o}. (A \cup (B \cap C)) = ((A \cup B) \cap (A \cup C))$ .

A final feature of THF0, not shown in Figures 1 and 2, is the use of logical connectives as terms. For example,

```

thf(and_false,definition,(
  ( ( & @ $false )
    = ( ^ [Q: $o] : $false ) ) ).

```



This is quite straightforward in the logic, because all logical connectives (in Henkin semantics) are themselves equivalent to corresponding lambda abstractions, e.g., conjunction is equivalent to  $\lambda P_o, Q_o. (P \wedge Q)$ .

Figure 3 presents the annotated formulae from another example problem encoded in THF0. The example is from the domain of puzzles, and it encodes the following ‘Knights and Knaves Puzzle’: *A very special island is inhabited only by knights and knaves. Knights always tell the truth. Knaves always lie. You meet two inhabitants: Zoey and Mel. Zoey tells you that Mel is a knave. Mel says, ‘Neither Zoey nor I are knaves.’ Can you determine who is a knight and who is a knave?*<sup>3</sup> Puzzles of this kind have been discussed extensively in the AI literature. This illustrates the intuitive and straightforward encoding that can be achieved in THF0. This encoding embeds formulae (terms of Boolean type) in terms and quantifies over variables of Boolean type; see for instance the `knights_tell_truth` axiom.

### 3.3 The THF and THFX Languages

The first release of the TPTP problem library contained problems in only the core THF0 fragment of the THF language. As higher-order ATP system developers and users adopt the THF language, the demand for the richer features of the full THF language and the extended THFX language will emerge. The THF and THFX languages have been defined with this in mind, with a rich set of syntactic constructs. They provide a richer type system, the ability to reason about types, more term and type constructs, more connectives, and “syntactic sugar” that is usefully expressive. The TPTP infrastructure is capable of processing THF and THFX formulae. The higher-order TPTP is thus able to meet the expectations of the community, hence continuing uptake of the THF language and use of the TPTP as a common basis for research and developments of higher-order ATP.

THF constructs, which are not shown in Figures 1 and 2, are:

- `<<` as the subtype operator. For example, with `fruit` and `food` having been declared as atomic types, the following declares that `fruit` is a subtype of `food`:  
`fruit << food`
- `@+` and `@-` as the binders for choice (indefinite description) and definite description. For example, if all elements of set `p` (of type `$i > $o`) are equal to element `y` (of type `$i`), then definite description applied to the set returns this single element:  

$$\begin{aligned} & ( ! [Z: \$i] : ((p @ Z) => (y = Z)) ) \\ => & ( ( @-[X: \$i] : (p @ X) ) = y ) \end{aligned}$$
- `!>` and `?*` for  $\Pi$  (dependent product) and  $\Sigma$  (sum) types. For example, to define a list constructor to be a function from a list of length `N` to a list of length `N + 1`:  
`cons: !> [N:nat] : ($i > (list @ N) > (list @ (succ @ N)))`  
 Another (more complex) example is motivated by functors in category theory.  
`func: ?* [F: obj > obj] : !> [X,Y: obj] :`  

$$((\text{hom} @ X @ Y) > (\text{hom} @ (F @ X) @ (F @ Y)))$$
  
 Here `func` is declared to be of a dependent sum type, and so it is a pair  $(F, F')$ .

<sup>3</sup>This puzzle was auto-generated by a computer program, written by Zac Ernst.

```

%-----
%----Type declarations
thf(mel_type,type,(
    mel: $i )).

thf(zoey_type,type,(
    zoey: $i )).

thf(knight_type,type,(
    knight: $i > $o )).

thf(knave_type,type,(
    knave: $i > $o )).

thf(says_type,type,(
    says: $i > $o > $o )).

%----A very special island is inhabited only by knights and knaves.
thf(knights_xor_knaves,axiom,(
    ! [P: $i] :
        ( ( knight @ P )
          <~> ( knave @ P ) ) )).

%----Knights always tell the truth.
thf(knights_tell_truth,axiom,(
    ! [P: $i,S: $o] :
        ( ( ( knight @ P )
          & ( says @ P @ S ) )
          => S ) )).

%----Knaves always lie.
thf(knaves_lie,axiom,(
    ! [P: $i,S: $o] :
        ( ( ( knave @ P )
          & ( says @ P @ S ) )
          => ~ ( S ) ) )).

%----Zoey says 'Mel is a knave'
thf(zoey_speaks,axiom,
    ( says @ zoey @ ( knave @ mel ) ) ).

%----Mel says 'Neither Zoey nor I are knaves'
thf(mel_speaks,axiom,
    ( says @ mel
      @ ( ~ ( knave @ zoey )
        & ~ ( knave @ mel ) ) ) ).

%----What are Zoey and Mel?
thf(who_is_knight_and_knave,conjecture,(
    ? [Knight: $i,Knave: $i] :
        ( ( knight @ Knight )
          & ( knave @ Knave ) ) ) ).
%-----

```

Fig. 3. The annotated formulae of TPTP problem file PUZ081~3

The first component  $F$  is a function mapping objects to objects. The second component  $F'$  has a more complicated type that depends on  $F$ .  $F'$  is a function that takes three arguments. The first two arguments of  $F'$  are objects  $X$  and  $Y$ . The third argument of  $F'$  has type  $(\text{hom } X \ Y)$ . Note that this type depends on the first two arguments. The return type of  $F'$  is  $(\text{hom } (F \ X) \ (F \ Y))$ . Note that this type depends on the first component of the pair as well as the first two arguments of  $F'$ .

- $*$  and  $+$  for simple product and sum (disjoint union) types. For example, this defines the type of a function that returns the roots of a quadratic, either two roots, a single root, or *undefined* in the case of no roots:

```
roots: quadratic > (( $\$real * \$real$ ) +  $\$real$  + undef)
```

THFX constructs, which are not shown in Figures 1 and 2, are:

- $[ \ ]$  for tuples, used to represent lists. For example, a lambda term that takes three individuals and forms a list would be:

```
make.triple =  $\wedge$  [X:  $\$i$ ,Y:  $\$i$ ,Z:  $\$i$ ] : [X,Y,Z]
```

- $-->$  as the sequent connective, which takes tuples as its left and right operands. For example, this sequent asserts that if the result of applying the conjunction connective to a conjunction is *false*, then at least one of the conjuncts must be *false*.

```
[(& @ (a & b)) =  $\wedge$  [X:  $\$o$ ] :  $\$false$ ] --> [a =  $\$false$ ,b =  $\$false$ ]
```

THF also provides some support for *type variables* of type  $\$tType$  and polymorphism. If a type declaration is prefixed by universally quantified type variables, these act as type parameters. Type parameters are supplied with the application operator  $@$ . For example, the set membership ( $\in$ ) constant for the user type  $u$ , declared and defined by

```
thf(type_u,type,(
  u:  $\$tType$  )).

thf(const_in_u,type,(
  in_u: u > ( u >  $\$o$  ) >  $\$o$  )).

thf(ax_in_u,axiom,
  ( in_u
    = (  $\wedge$  [X: u,S: u >  $\$o$ ] :
      ( S @ X ) ) )).
```

is generalized by

```
thf(const_in_all,type,(
  ! [U:  $\$tType$ ] :
    ( ( in_all @ U ): U > ( U >  $\$o$  ) >  $\$o$  ) )).

thf(ax_in_all,axiom,
  ( in_all
    = (  $\wedge$  [U:  $\$tType$ ,X: U,S: U >  $\$o$ ] :
      ( S @ U @ X ) ) )).
```

#### 4. THE HIGHER-ORDER TPTP AND INFRASTRUCTURE

This section describes the higher-order components of the TPTP, noted in Section 1 as having had an impact on the development of automated reasoning for higher-order logic.

##### 4.1 The TPTP Problem Library

The first release of the TPTP problem library with THF problems was v4.0.0, released on 4th July 2009. It contained 2729 problems written in THF0, stemming from 852 abstract problems in nine domains.

- ALG** - 80 problems. Fifty of these are problems concerning higher-order abstract syntax, encoded in higher-order logic [29]. The rest are mostly problems exported from the TPS test suite by Chad Brown.<sup>4</sup>
- COM** - 1 problem. This is a very naive version of the recursion theorem.
- GRA** - 93 problems. These are problems about Ramsey numbers, some of which are open in the mathematics community.
- GRP** - 1 problem. This problem proves that a group is Abelian iff every element has order 2.
- LCL** - 107 problems. These are mostly intuitionistic and modal logic problems that have been encoded in higher-order logic, using the technique described in [18]. There are also some problems exported from the TPS test suite.
- MSC** - 2 problems. These are two problems exported from the TPS test suite.
- NUM** - 212 problems. These are mostly theorems from the well known Landau book [65]. The problems originate from Jutting's Automath formalization [96], from which they have been automatically converted into THF0. There are also a few problems on Church numerals, which test higher-order ATP operations such as  $\beta$ -normalization and unification of function variables.
- PUZ** - 51 problems. The first group of these are classic puzzles, e.g., “knights and knaves” problems, and the wise men puzzle, encoded in modal logic and hence in higher-order logic. The second group of these are “checker board” problems, exported from the TPS test suite.
- SET** and **SEU** and **SEV** - 1407 problems. Many of these are “standard” problems in set theory that have TPTP versions in first-order logic. There is also a significant group of problems in dependently typed set theory [27], a group of interesting problems about binary relations, and a large group of problems exported from the TPS test suite.
- SWV** - 37 problems. These are security problems in authorization logics that can be converted via modal logics to THF0 [13, 42].
- SYN** and **SY0** - 738 problems. These are simple problems designed to test properties of higher-order ATP systems. They have been extracted from a range of sources [46, 15, 31]. There are also some syntactic intuitionistic logic problems taken from the ILTP library [81], encoded in higher-order logic using the approach from [18].

<sup>4</sup>The original TPS suite is available at <http://gtps.math.cmu.edu/tps.html>.

Table I. Statistics for THF problems in TPTP v4.0.0

	Min	Max	Avg	Median
Number of formulae	1	749	65	8
% of unit formulae	0%	99%	24%	25%
Number of atoms	2	7624	641	87
% of equality atoms	0%	33%	5%	6%
in problems with equality	1%	33%	7%	7%
% of variable atoms	0%	84%	34%	33%
Avg atoms per formula	1.5	998.0	32.5	8.1
Number of symbols	1	390	37	8
Number of variables	0	1189	100	16
$\lambda$ quantified	0	175	17	2
$\forall$ quantified	0	1067	76	8
$\exists$ quantified	0	64	6	2
Number of connectives	0	4591	348	44
Number of type connectives	0	477	53	10
Maximal formula depth	2	359	38	11
Average formula depth	2	350	11	6

2009 of the problems (73%) contain equality. 2098 of the problems (76%) are known or believed to be theorems, 317 (11%) are known or believed to be non-theorems, and the remaining 314 problems (13%) have unknown status. As is noted in Section 5, non-theorems are useful for testing the soundness of provers, and conversely theorem are useful for testing the soundness of model finders. Table I provides some further detailed statistics about the problems.

#### 4.2 The TSTP Solution Library

The Thousands of Solutions from Theorem Provers (TSTP) solution library, the “flip side” of the TPTP, is a corpus of contemporary ATP systems’ solutions to the TPTP problems. A major use of the TSTP is for ATP system developers to examine solutions to problems, and thus understand how they can be solved.

Table II summarizes the TSTP results from the higher-order ATP systems described in Section 5, for the 2729 THF problems in TPTP v4.0.0. The systems are (in historical order of development) Tps 3.080227G1, LEO-II 1.1, three automated versions of Isabelle 2009-1 (one - IsabelleP - trying to prove theorems, and two - IsabelleM and IsabelleN - trying to find (counter-)models), and Satallax 1.1 (which can prove theorems and find some (counter-)models). The “Any” and “All” rows are with respect to the theorem provers and model finders respectively. The “None” row shows the number of problems for which no result has been established. All the runs were done on 2.80GHz computers with 1GB memory and running the Linux operating system, with a 300s CPU limit.

The results show that the graph theory (GRA) Ramsey number problems are very difficult - this was expected. For the remaining domains the problems pose interesting challenges for the ATP systems. The systems have individual strengths and weaknesses across the domains, e.g., IsabelleP is strong on set theory (SE?) problems, LEO-II is strong on algebra (ALG), and Tps is strong on logical calculi (LCL) problems. The differences between the systems lead to some problems being solved uniquely by each of the systems, with Tps having the highest number of unique solutions. Satallax’s combined solution count (proofs and (counter-)models) is 1633. This might reasonably be compared with the combined IsabelleP+IsabelleN

Table II. Results for THF problems

	ALG	COM	GRA	GRP	LCL	MSC	NUM	PUZ	SE?	SWV	SYN	Total	Unique
Problems	80	1	93	1	107	2	212	51	1407	37	738	2729	
IsabelleP	36	0	0	1	63	1	167	26	857	11	422	1584	75
Satallax	19	0	0	0	53	0	133	21	804	11	452	1493	38
LEO-II	49	0	0	1	68	0	147	26	727	19	438	1475	37
Tps	32	1	0	1	83	1	161	20	723	17	410	1449	82
Any	53	1	0	1	86	1	176	28	1026	20	493	1885	232
All	15	0	0	0	48	0	112	14	514	9	346	1058	
IsabelleN	3	0	0	0	12	0	6	18	75	11	164	289	52
IsabelleM	3	0	0	0	11	0	6	17	75	11	87	210	0
Satallax	0	0	0	0	9	0	4	2	40	8	77	140	15
Any	3	0	0	0	16	0	6	18	75	11	175	304	77
All	0	0	0	0	5	0	4	2	40	8	39	98	
None	24	0	93	0	5	1	30	5	306	6	70	540	

count of 1873, although Satallax has the advantage of being a single monolithic system. The individual problem results show that IsabelleN solves a strict superset of the problems solved by IsabelleM, indicating that the **refute** command may now be redundant when using Isabelle in the normal interactive mode.

### 4.3 TPTP Tools

A range of tools are provided to support use of the TPTP [88]. Some are distributed with the TPTP problem library, some are available on request from the first author, and all are available as online services in the **SystemOnT\*TP** interfaces described in Section 4.3.4.

**4.3.1 Parsing and Formatting.** From a TPTP user perspective, the TPTP2X and TPTP4X utilities are often the most useful for manipulating THF problems. These two utilities have similar capabilities, of parsing (which provides a syntax check), analyzing, transforming, and formatting THF formulae. TPTP2X additionally provides format modules for outputting THF problems in the Tps [6], Twelf [77], OmDoc [62], Isabelle [73], and S-expression formats. TPTP2X is implemented in Prolog, thus taking advantage of the TPTP language's compatibility with Prolog, and being easily extended with new functionality. TPTP4X is implemented in C, and is thus faster than TPTP2X. TPTP2X is distributed with the TPTP problem library, while TPTP4X is available online and as part of the TPTPWorld package (available on request). In addition to the Prolog and C parsers, a Java parser built using **antlr** is available (on request). The Java parser is used in the IDV derivation viewer described in Section 4.3.3.

TPTP2X and TPTP4X provide imprecise syntax checking of THF formulae. More precise parsing and syntax checking is available by using the **BNFParser** family of parsers. These parsers are generated automatically from the BNF definition of the THF language, using the **lex/yacc** framework [97]. The **lex/yacc** files used to build these parsers are available (on request).

**4.3.2 Type Checking.** The THF syntax provides a lot of flexibility. As a result, many syntactically correct expressions are meaningless because they are not well typed. For example, it does not make sense to write  $(\& = \sim)$  because the equated expressions have different types. It is therefore necessary to type check THF for-

mulae using an inference system. This is achieved for THF0 in the TPTP by using TPTP2X to export THF problems in Twelf format, and submitting the result to the Twelf tool [77]. The type checking in Twelf is based on typing rules, e.g.,

$$\frac{\Gamma \vdash F :: A > B \quad \Gamma \vdash S :: A}{\Gamma \vdash F @ S :: B} \quad \frac{\Gamma \vdash F :: A > \$o}{\Gamma \vdash !! F :: \$o}$$

are the typing rules for application and  $\Pi$ , where  $S :: A$  denotes the judgement that  $S$  has type  $A$ . The normative definition is given by representing THF0 in the logical framework LF [51]. A base signature  $\Sigma_0$  is defined, and a list  $L$  of THF0 formulae is translated to a list of declarations  $\Sigma_L$  extending  $\Sigma_0$ .  $L$  is well-formed iff  $\Sigma_0, \Sigma_L$  is a well-formed Twelf signature. See [20] for full details.

Via the Curry-Howard correspondence [37, 56], the representation of THF in Twelf can be extended to represent proofs – it is necessary only to add declarations for the proof rules to  $\Sigma_0$ , i.e.,  $\beta$ - $\eta$ -conversion, and the rules for the connectives and quantifiers.

**4.3.3 Solution Analysis.** Tools for analyzing and presenting proofs output by higher-order ATP systems are available. The two most salient of these are the GDV derivation verifier, and the IDV derivation viewer.

GDV [87] is a tool that uses structural and then semantic techniques to verify a derivation in TPTP format. Structural verification checks that inferences have been done correctly in the context of the derivation, e.g., that the derivation is acyclic, that assumptions have been discharged, etc. Semantic verification encodes the semantic relationships between inferred formulae and their parent formulae as logical obligations (in the form of ATP problems), and the obligations are discharged by trusted ATP systems.

IDV [94] is a tool for graphical rendering of derivations in TPTP format. The rendering of the derivation DAG uses shape and color to visually provide information about the derivation. The node shape indicates the role of the formula, the color indicates THF, FOF or CNF, and tags indicate features of the inference step. For proofs that rely on translation between forms, e.g., LEO-II proofs (see Section 5.2), the phases are separated into layers. The user can interact with the rendering in various ways, e.g., to examine the formulae, zoom in, hide parts of the proof, and form a proof synopsis using information from the AGInT tool [80].

**4.3.4 Online Access via SystemOnTPTP.** The SystemOnTPTP service provides online access to ATP systems and all TPTP tools. The service can be used interactively in a browser, starting at <http://www.tptp.org/cgi-bin/SystemOnTPTP>, or can be accessed programmatically using `http POST` requests. The back-end of the SystemOnTPTP service is available (on request) for users who want to run ATP systems and tools locally, but under the control of the service's harness.

#### 4.4 The CADE ATP System Competition

The CADE ATP System Competition (CASC) [93] is an annual evaluation of the performance of fully automatic ATP systems – it is the world championship for such systems. CASC has been a catalyst for impressive improvements in ATP, stimulating both theoretical and implementation advances [71]. The first THF division of CASC was part of CASC-22 at CADE-22. Four THF ATP systems

entered this division, and their robust performance demonstrated their potential to prospective users. The full results and resources used in CASC-22 are available from the web site, <http://www.tptp.org/CASC/22/>.

While the primary purpose of CASC is a public evaluation of the relative capabilities of ATP systems, it also aims to promote ATP research. The THF division of CASC provides insight and stimulus for the development of higher-order ATP systems.

## 5. HIGHER-ORDER ATP SYSTEMS

Research and development of computer-supported reasoning for higher-order logic has been in progress for as long as that for first-order logic. It is clear that the computational issues in the higher-order setting are significantly harder than those in first-order. Problems such as the undecidability of higher-order unification, the handling of equality and extensionality reasoning, and the instantiation of set variables, have hampered the development of effective higher-order automated reasoning. Thus, while there are many *interactive* proof assistants based on some form of higher-order logic [99], there are few *automated* systems for higher-order logic. This section describes the (fully automatic) ATP systems for Church's simple type theory, that we know of.

### 5.1 TPS

TPS is a fully automated version of the higher-order theorem proving system TPS [5, 6]. TPS can be used to prove theorems of Church's type theory automatically, interactively, or semi-automatically. When searching for a proof, TPS first searches for an expansion proof [69] or an extensional expansion proof [28] of the theorem. Part of this process involves searching for acceptable matings [3]. Using higher-order unification, a pair of occurrences of subformulae (which are usually literals) is mated appropriately on each vertical path through an expanded form of the theorem to be proved. Strategies used by TPS in the search process include re-ordering conjunctions and disjunctions to alter the enumeration of paths through the formula, the use of primitive substitutions and gensubs [4], path-focused duplication [60], dual instantiation of definitions, generating substitutions for higher-order variables that contain abbreviations already present in the theorem to be proved [25], component search [24], generating and solving set constraints [32], and generating connections using extensional and equational reasoning [28].

The behavior of TPS is controlled by hundreds of flags. A set of flags, with values for them, is called a mode. Forty-nine modes have been found that collectively suffice for automatically proving virtually all the theorems that TPS has proved automatically thus far. As the modes have quite different capabilities, and it is expected that any proofs found by any mode will be found quickly, strategy scheduling the modes is a simple way of obtaining greater coverage. A `perl` script has been used to do this, running each of the 49 modes for a specified amount of time.

### 5.2 LEO-II

LEO-II [19] is a higher-order ATP system based on extensional higher-order resolution. LEO-II is designed to cooperate with specialist systems for fragments of



higher-order logic. Currently, LEO-II is capable of cooperating with the first-order ATP systems E [85], SPASS [98], and Vampire [83]. LEO-II is often too weak to find a refutation amongst the steadily growing set of clauses on its own. However, some of the clauses in LEO-II's search space attain a special status: they are first-order clauses modulo the application of an appropriate transformation function. The default transformation is Hurd's fully typed translation [59]. Therefore, LEO-II launches a cooperating first-order ATP system every  $n$  iterations of its (standard) resolution proof search loop (currently  $n = 10$ ). If the first-order ATP system finds a refutation, it communicates its success to LEO-II in the standard SZS format. Communication between LEO-II and the cooperating first-order ATP system uses the TPTP language and standards. LEO-II uses a monolithic search strategy, i.e., it does not use strategy scheduling to try different search strategies or flag settings.

In addition to the fully automatic mode, LEO-II provides an interactive mode [19]. This mode supports debugging and inspection of the search space, and also the tutoring of resolution based higher-order theorem proving to students. The interactive mode and the automatic mode can be interleaved.

LEO-II is implemented in Objective Caml, and is available from <http://www.ags.uni-sb.de/~leo/> under a BSD-like licence.

### 5.3 IsabelleP, IsabelleM, IsabelleN

Isabelle [73] is a well known proof assistant for higher-order logic. It is normally used interactively through the Proof General interface. In this mode it is possible to apply various automated tactics that attempt to solve the current goal without further user interaction. Examples of these tactics are **blast**, **auto**, and **metis**. It is (a little known fact that it is) also possible to run Isabelle from the command line, passing in a theory file containing a **lemma** to prove. Finally, Isabelle theory files can include ML code to be executed when the file is processed. While it was probably never intended to use Isabelle as a fully automatic system, these three features have been combined to implement a fully automatic Isabelle. The TPTP2X Isabelle format module outputs a THF problem in Isabelle HOL syntax, augmented with ML code that runs tactics in sequence, each with a CPU time limit until one succeeds or all fail. The tactics used so far are **simp**, **blast**, **auto**, **metis**, **fast**, **fastsimp**, **best**, **force**, **meson**, and **smt**. The system is named **IsabelleP**.

The ability of Isabelle to find (counter-)models using the **refute** and **nitpick** commands has also been integrated into automatic systems, called **IsabelleM** and **IsabelleN** respectively. This provides the capability to find models for THF formulae, which confirm the satisfiability of axiom sets, or the countersatisfiability of non-theorems. This has been particularly useful for exposing errors in some THF problem encodings, and revealing bugs in the THF theorem provers (and conversely, the theorem provers have been useful in debugging **IsabelleM** and **IsabelleN**!).

### 5.4 Satallax

Satallax is based on a complete ground tableau calculus for higher-order logic with a choice operator. Instantiations at base types other than the type  $o$  are restricted to discriminating terms (terms involved in an inequality) as described in [30]. In case there are no discriminating terms at a base type  $\iota$ , some term of type  $\iota$  is used as an instantiation just to make progress. Since any term would do, either the first

Table III. Statistics for set theory problems

System	Solved	Avg. CPU	System	Solved	Avg. CPU
Satallax	78	0.01s	leanCoP	76	9.82s
Tps	78	2.68s	iProver	75	4.95s
LEO-II	76	0.06s	Vampire	73	5.19s
IsabelleP	76	7.12s	E	69	5.77s

term of type  $\iota$  encountered is used, or a fresh variable of type  $\iota$  is used, or the choice operator at  $\iota$  applied to the empty set is used, depending on flag settings and the particulars of the problem. When instantiations of type  $o$  are required, terms corresponding to both of the Boolean values ( $\perp$  and  $\top$ ) are used.

Instantiations at function types are generated using primitive substitutions. Instead of searching in the tableau calculus directly, the LISP code (1) progressively generates all tableau steps that might be used in a refutation, (2) creates corresponding propositional clauses, and (3) calls MiniSat [40] to check for propositional satisfiability. If MiniSat finds the current set of clauses to be unsatisfiable, then there is a corresponding tableau refutation. If a problem only requires instantiations at base types (and never at function types), the generation of clauses for that problem might terminate. In such a case, if a final call to MiniSat indicates the set of clauses is satisfiable, then the original higher-order problem was satisfiable.

Satallax is implemented in Steel Bank Common LISP, and is available from <http://www.satallax.com>.

## 6. APPLICATIONS

### 6.1 Set Theory

As noted in Section 3.2, set theory problems can be elegantly encoded in THF0 by exploiting the fact that higher-order logic provides a naturally built-in set theory, based on the idea of identifying sets with their characteristic functions. As a result the higher-order encoding of a set theory problem can often be solved more efficiently than the corresponding first-order encoding [21]. The data in Table III illustrates this effect. The data is for 78 set theory problems presented in [95], encoded in FOF for the (now defunct) ILF project [38], and encoded in THF by export from the TPS test suite.<sup>5</sup> Data is given for the four FOF ATP systems that have the most solutions (for the FOF versions of these problems) in the TSTP solution library, and the four THF theorem provers described in Section 5. The FOF ATP systems are leanCoP 2.1 [74], iProver 0.7 [64], Vampire 11.0 [83], and E 1.1 [85]. The columns provide the number of problems solved, and the average CPU time taken over the problems solved. All the runs were done on 2.80GHz computers with 1GB memory and running the Linux operating system, with a 300s CPU limit.

These results confirm that for these problems, the higher-order encoding in conjunction with the higher-order ATP systems is more effective. The number of problems solved by the THF systems is slightly higher than the number solved by the FOF systems, and the average times taken by the THF systems are significantly lower than the times taken by the FOF systems. These results are amplified by the

<sup>5</sup>This is the largest group of consistently comparable FOF and THF set theory problems in the TPTP.

fact that THF ATP technology is in its infancy relative to the mature FOF ATP systems.

A second application of THF0 to set theory comes from the DetSeT project [27].<sup>6</sup> The Scunak system was used to axiomatize and develop axiomatic (untyped) set theory within a dependently typed framework [33]. Using this system, a version of ZFC (Zermelo-Fraenkel set theory with choice) was axiomatized and a number of theorems were proven. Even though the development took place within a dependent type theory, it is possible to translate this into simply typed higher-order logic by erasing the dependencies and appropriately handling logical constants. This translation has been done and resulted in a development of set theory within higher-order logic along the lines described in [48]. The translation provided 326 theorems of ZFC. For each of these theorems, two THF problems were created: a *global* version and a *local* version. The global version of the  $n^{th}$  theorem  $Z_n$  states that all axioms and all previous theorems  $Z_i$  with  $i < n$  imply  $Z_n$ . The local version states that some subset of the axioms and previous theorems imply  $Z_n$ . The subset for the local version was chosen using the proof term produced by Scunak. The proof term explicitly contains a reference to every axiom and previous theorem used in the proof. One could imagine using higher-order ATP systems to help obtain proof terms in Scunak (or similar systems). The collection of ZFC problems in the TPTP allows the feasibility of this idea to be tested.

## 6.2 Embeddings and Combinations of Logics

Prominent non-classical logics, including quantified multimodal logics and intuitionistic logic, can be elegantly embedded in THF0 [18]. THF0 can hence serve as a platform for flexibly combining classical and non-classical logics [14]. These embeddings can in turn be exploited to model further embeddings of logics, for example, logics of access control in computer security. The embedding of access control logics in THF0 [13] exemplifies how different logic embeddings can be elegantly combined in THF0. In this specific case a translation of prominent access control logics in modal logic S4 [43] is combined with the above embedding of modal logics into THF0.

Combination of logics, e.g., combining logics of knowledge and belief with logics of time and space, are important for many real world applications, and the application contexts can be difficult to model directly. The THF0 language and the THF0 ATP systems can be employed to both (i) model and solve problems that use combinations of logics (cf. the modeling of the famous Wise Men Puzzle in [14]), and (ii) meta-reason about such combinations of logics, e.g., prove their consistency.

The experiments conducted in [14] are basic, and further work is needed to investigate the scalability of the logic embeddings approach. However, besides its promising meta-reasoning aspects, further advantages of this approach are flexibility and reuse: it is not necessary to create new theorem proving calculi and systems for each new combination of logics. The THF0 embedding and the THF0 ATP systems can be applied uniformly. The direct approach of creating new calculi and systems for each new combination of logics is, in contrast, usually very tedious and also error prone.

<sup>6</sup><http://mathgate.info/detsetitem.php>

## 7. CONCLUSION

This paper has described the TPTP THF infrastructure for automated reasoning in higher-order logic, with performance data and example applications that show that the infrastructure is useful and effective. The goals of developing THF were to provide pragmatic standards for writing higher-order logic ATP problems and solutions, provide libraries of test problems and solutions, and consequently support and stimulate the development and deployment of ATP systems for higher-order logic in applications. These goals are already being realized, through the TPTP and TSTP libraries of test problems and solutions, the development of several new higher-order ATP systems, the use of higher-order ATP in a range of domains, and renewed interest in higher-order ATP in the community, e.g., at the CADE conference during CASC.

The development of the THF part of the TPTP and its infrastructure has progressed far more rapidly than was the case for the FOF part, for which it took some years to convert existing systems and their developers to the TPTP standards. The THF development has benefited from being built on top of the established FOF part of the TPTP. Future systems and applications of higher-order ATP are expected to use THF from the start, thus making it immediately possible to use all the TPTP THF infrastructure. Further development of THF will also be guided by the more mature FOF part of the TPTP. For example, ATP systems that output proofs are particularly important, allowing proof verification. In the long term we hope to see burgeoning research and development of ATP for higher-order logic, with a richness similar to first-order ATP, with many ATP systems, common usage in applications, meta-systems, etc.

*Acknowledgments:* Thanks to Chad Brown for contributions of THF problems to the TPTP, developing the automated version of TPS, and implementing Satallax. Thanks to Florian Rabe for help with the Twelf and OmDoc formats, and to Lucas Dixon and Stefan Berghofer for help with the Isabelle format. Thanks to Florian Rabe for implementing the Twelf-based type checking. Thanks to Makarius Wenzel and Stefan Berghofer for writing the IsabelleP code. Thanks to Jasmin Blanchette for explaining how to implement IsabelleM, and subsequently implementing IsabelleN. Thanks to Andrei Tchaltev and Alexandre Riazanov for developing the first-order parts of the Java parser for the TPTP language.

## References

- [1] S. Allen, M. Bickford, R.L. Constable, R. Eaton, C. Kreitz, L. Lorigo, and E. Moran. Innovations in Computational Type Theory using Nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.
- [2] P. B. Andrews. Resolution in Type Theory. *Journal of Symbolic Logic*, 36(3):414–432, 1971.
- [3] P. B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28(2):193–214, 1981.
- [4] P. B. Andrews. On Connections and Higher-Order Logic. *Journal of Automated Reasoning*, 5(3):257–291, 1989.

- [5] P. B. Andrews, M. Bishop, S. Issar, Nesmith. D., F. Pfenning, and H. Xi. TPS: A Theorem-Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.
- [6] P. B. Andrews and C. E. Brown. TPS: A Hybrid Automatic-Interactive System for Developing Proofs. *Journal of Applied Logic*, 4(4):367–395, 2006.
- [7] A. Asperti, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. User Interaction with the Matita Proof Assistant. *Journal of Automated Reasoning*, 39(2):109–139, 2007.
- [8] A. Asperti and E. Tassi. Higher order Proof Reconstruction from Paramodulation-Based Refutations: The Unit Equality Case. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Proceedings of the 6th International Conference on Mathematical Knowledge Management*, volume 4573 of *Lecture Notes in Computer Science*, pages 146–160, 2007.
- [9] J. Backes and C. Brown. Analytic Tableaux for Higher-Order Logic with Choice. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page To appear, 2010.
- [10] M. Beeson. Otter-lambda, a Theorem-prover with Untyped Lambda-unification. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.
- [11] C. Benzmüller. Extensional Higher-order Paramodulation and RUE-Resolution. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 399–413. Springer-Verlag, 1999.
- [12] C. Benzmüller, editor. *Special Issue on Assistance Systems for Mathematics*, volume 4. Elsevier, 2007.
- [13] C. Benzmüller. Automating Access Control Logic in Simple Type Theory with LEO-II. In D. Gritzalis and J. López, editors, *Proceedings of the 24th IFIP TC 11 International Information Security Conference*, number 297 in IFIP Advances in Information and Communication Technology, pages 387–398. Springer-Verlag, 2009.
- [14] C. Benzmüller. Simple Type Theory as a Framework for Combining Logics. In *Proceedings of the 3rd World Congress and School on Universal Logic*, 2010.
- [15] C. Benzmüller and C. E. Brown. A Structured Set of Higher-Order Problems. In J. Hurd and T. Melham, editors, *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, number 3606 in Lecture Notes in Artificial Intelligence, pages 66–81. Springer-Verlag, 2005.
- [16] C. Benzmüller, C. E. Brown, and M. Kohlhase. Higher-order Semantics and Extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.
- [17] C. Benzmüller and M. Kohlhase. LEO - A Higher-Order Theorem Prover. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 139–143. Springer-Verlag, 1998.

- [18] C. Benzmüller and L. Paulson. Multimodal and Intuitionistic Logics in Simple Type Theory. *Logic Journal of the IGPL*, 2010.
- [19] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 162–170. Springer-Verlag, 2008.
- [20] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 - The Core TPTP Language for Classical Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 491–506. Springer-Verlag, 2008.
- [21] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Combined Reasoning by Automated Cooperation. *Journal of Applied Logic*, 6(3):318–342, 2008.
- [22] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.
- [23] D. Bezem, D. Hendriks, and H. de Nivelle. Automated Proof Construction in Type Theory using Resolution. *Journal of Automated Reasoning*, 29(3-4):253–275, 2002.
- [24] M. Bishop. A Breadth-First Strategy for Mating Search. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 359–373. Springer-Verlag, 1999.
- [25] M. Bishop and P.B. Andrews. Selectively Instantiating Definitions. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 365–380. Springer-Verlag, 1998.
- [26] B. Brock, M. Kaufmann, and J. Moore. ACL2 Theorems about Commercial Microprocessors. In M. Srivas and A. Camilleri, editors, *Proceedings of the 1st International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, number 1166 in Lecture Notes in Computer Science, pages 275–293. Springer-Verlag, 1996.
- [27] C. E. Brown. Dependently Typed Set Theory. Technical Report SWP-2006-03, Saarland University, 2006.
- [28] C. E. Brown. *Automated Reasoning in Higher-Order Logic: Set Comprehension and Extensionality in Church's Type Theory*. Number 10 in Studies in Logic: Logic and Cognitive Systems. College Publications, 2007.
- [29] C. E. Brown. M-Set Models. In C. Benzmüller, C. E. Brown, J. Siekmann, and R. Statman, editors, *Reasoning in Simple Type Theory: Festschrift in Honour of Peter B. Andrews on his 70th Birthday*, number 17 in Studies in Logic, Mathematical Logic and Foundations, pages 175–186. College Publications, 2008.
- [30] C. E. Brown and G. Smolka. Extended First-Order Logic. In T. Nipkow and C. Urban, editors, *Proceedings of the 22nd International Conference on*

- Theorem Proving in Higher Order Logics*, number 5674 in Lecture Notes in Computer Science, pages 164–179. Springer-Verlag, 2009.
- [31] C. E. Brown and G. Smolka. Terminating Tableaux for the Basic Fragment of Simple Type Theory. In M. Giese and A. Waaler, editors, *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 5697 in Lecture Notes in Artificial Intelligence, pages 138–151. Springer-Verlag, 2009.
  - [32] C.E. Brown. Solving for Set Variables in Higher-Order Theorem Proving. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 408–422. Springer-Verlag, 2002.
  - [33] C.E. Brown. Combining Type Theory and Untyped Set Theory. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 205–219. Springer-Verlag, 2006.
  - [34] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:5668, 1940.
  - [35] R. Constable and M. Bickford. Formal Foundations of Computer Security. In *Information and Communication Security*, number 14 in NATO Science for Peace and Security Series, pages 29–52. 2008.
  - [36] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2-3):95–120, 1988.
  - [37] H.B. Curry and R. Feys. *Combinatory Logic I*. North Holland, Amsterdam, 1958.
  - [38] I.B. Dahn, J. Gehne, T. Honigmann, L. Walther, and A. Wolf. Integrating Logical Functions with ILF. Technical report, Institut für Mathematik, Humboldt Universität zu Berlin, Berlin, Germany, 1995.
  - [39] N.G. de Bruijn. The Mathematical Language Automath, its Usage, and Some of its Extensions. In R. Nederpelt, J. Geuvers, and R. de Vrijer, editors, *Selected Papers on Automath*, number 133 in Studies in Logic and the Foundations of Mathematics, pages 73–100. North-Holland, 1994.
  - [40] N. Eén and N. Sörensson. MiniSat - A SAT Solver with Conflict-Clause Minimization. In F. Bacchus and T. Walsh, editors, *Posters of the 8th International Conference on Theory and Applications of Satisfiability Testing*, 2005.
  - [41] F. Frege. *Grundgesetze der Arithmetik*. Jena, 1893,1903.
  - [42] D. Garg. Principal-Centric Reasoning in Constructive Authorization Logic. Technical Report CMU-CS-09-120, School of Computer Science, Carnegie Mellon University, 2009.
  - [43] D. Garg and M. Abadi. A Modal Deconstruction of Access Control Logics. In R. Amadio, editor, *Proceedings of the 11th International Conference on the Foundations of Software Science and Computational Structures*, number 4962 in Lecture Notes in Computer Science, pages 216–230, 2008.
  - [44] J-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

- [45] P. Godefroid. Software Model Checking: the VeriSoft Approach. Technical Report Technical Memorandum ITD-03-44189G, Bell Labs, Lisle, USA, 2003.
- [46] R. Goldblatt. *Logics of Time and Computation*. Center for the Study of Language and Information - Lecture Notes. The University of Chicago Press, 1992.
- [47] G. Gonthier. Formal Proof - The Four-Color Theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008.
- [48] M. Gordon. Set Theory, Higher Order Logic or Both? In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics*, number 1125 in Lecture Notes in Computer Science, pages 191–201. Springer-Verlag, 1996.
- [49] M. Gordon and T. Melham. *Introduction to HOL, a Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
- [50] T. Hales. A Proof of the Kepler Conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.
- [51] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, 1993.
- [52] J. Harrison. HOL Light: A Tutorial Introduction. In M. Srivas and A. Camilleri, editors, *Proceedings of the 1st International Conference on Formal Methods in Computer-Aided Design*, number 1166 in Lecture Notes in Computer Science, pages 265–269. Springer-Verlag, 1996.
- [53] J. Harrison. Floating-Point Verification using Theorem Proving. In M. Bernardo and A. Cimatti, editors, *Proceedings of the 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, number 3965 in Lecture Notes in Computer Science, pages 211–242. Springer-Verlag, 2006.
- [54] J. Harrison. HOL Light: An Overview. In T. Nipkow and C. Urban, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, number 5674 in Lecture Notes in Computer Science, pages 60–66. Springer-Verlag, 2009.
- [55] L. Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15:81–91, 1950.
- [56] W. Howard. The Formulas-as-types Notion of Construction. In J. Seldin and J. Hindley, editors, *To H B Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.
- [57] G. Huet. A Complete Mechanization of Type Theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 139–146, 1973.
- [58] G. Huet. A Unification Algorithm for Typed Lambda-Calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.
- [59] J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Muñoz, editors, *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.



- [60] S. Issar. Path-Focused Duplication: A Search Procedure for General Matings. In Swartout W. Dietterich T., editor, *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 221–226. American Association for Artificial Intelligence / MIT Press, 1990.
- [61] M. Kaufmann, P. Manolios, and J. Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
- [62] M. Kohlhase. *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2006.
- [63] K. Konrad. *Model Generation for Natural Language Interpretation and Analysis*. Number 2953 in Lecture Notes in Computer Science. Springer-Verlag, 2004.
- [64] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.
- [65] E. Landau. *Grundlagen der Analysis*. Akademische Verlagsgesellschaft M.B.H, 1930.
- [66] X. Leroy. Formal Verification of a Realistic Compiler. *Communications of the ACM*, 52(7):107–115, 2009.
- [67] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
- [68] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An Introduction to the Syntax and Content of Cyc. In Baral C., editor, *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.
- [69] D. Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.
- [70] R. Nederpelt, J. Geuvers, and R. de Vrijer. *Selected Papers on Automath*. Number 133 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1994.
- [71] R. Nieuwenhuis. The Impact of CASC in the Development of Automated Deduction Systems. *AI Communications*, 15(2-3):77–78, 2002.
- [72] I. Niles and A. Pease. Towards A Standard Upper Ontology. In C. Welty and B. Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems*, pages 2–9, 2001.
- [73] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Number 2283 in Lecture Notes in Computer Science. Springer-Verlag, 2002.
- [74] J. Otten. leanCoP 2.0 and ileancop 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 283–291, 2008.

- [75] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 411–414. Springer-Verlag, 1996.
- [76] F. Pfenning. Logic Programming in the LF Logical Framework. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.
- [77] F. Pfenning and C. Schürmann. System Description: Twelf - A Meta-Logical Framework for Deductive Systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 202–206. Springer-Verlag, 1999.
- [78] T. Pietrzykowski and D. Jensen. A Complete Mechanization of Omega-order Type Theory. In J. Donovan and R. Shields, editors, *Proceedings of the ACM Annual Conference*, pages 82–92. ACM Press, 1972.
- [79] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, Edinburgh, United Kingdom, 1994.
- [80] Y. Puzis, Y. Gao, and G. Sutcliffe. Automated Generation of Interesting Theorems. In G. Sutcliffe and R. Goebel, editors, *Proceedings of the 19th International FLAIRS Conference*, pages 49–54. AAAI Press, 2006.
- [81] T. Rath, J. Otten, and C. Kreitz. The ILTP Problem Library for Intuitionistic Logic - Release v1.1. *Journal of Automated Reasoning*, 38(1-2):261–271, 2007.
- [82] J.C. Reynolds. Types, Abstraction, and Parametric Polymorphism. Number 83 in Information Processing, pages 513–523. Elsevier Science Publishers, 1983.
- [83] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
- [84] P. Rudnicki. An Overview of the Mizar Project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pages 311–332, 1992.
- [85] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.
- [86] J. Siekmann, C. Benzmüller, and S. Autexier. Computer Supported Mathematics with OMEGA. *Journal of Applied Logic*, 4(4):533–559, 2006.
- [87] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.
- [88] G. Sutcliffe. TPTP, TSTP, CASC, etc. In V. Diekert, M. Volkov, and A. Voronkov, editors, *Proceedings of the 2nd International Computer Science Symposium in Russia*, number 4649 in Lecture Notes in Computer Science, pages 7–23. Springer-Verlag, 2007.
- [89] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

- [90] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
- [91] G. Sutcliffe, C. Benz Müller, C. E. Brown, and F. Theiss. Progress in the Development of Automated Theorem Proving for Higher-order Logic. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 116–130. Springer-Verlag, 2009.
- [92] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81, 2006.
- [93] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.
- [94] S. Trac, Y. Puzis, and G. Sutcliffe. An Interactive Derivation Viewer. In S. Autexier and C. Benz Müller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.
- [95] A. Trybulec and H. Swieczkowska. Boolean Properties of Sets. *Journal of Formalized Mathematics*, 1(1):17–23, 1989.
- [96] L.S. van Benthem Jutting. *Checking Landau’s “Grundlagen” in the AUTOMATH System*. PhD thesis, Eindhoven University, Eindhoven, The Netherlands, 1979.
- [97] A. Van Gelder and G. Sutcliffe. Extending the TPTP Language to Higher-Order Logic with Automated Parser Generation. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 156–161. Springer-Verlag, 2006.
- [98] C. Weidenbach, A. Fietzke, R. Kumar, M. Suda, P. Wischniewski, and D. Dimova. SPASS Version 3.5. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 140–145. Springer-Verlag, 2009.
- [99] F. Wiedijk. *The Seventeen Provers of the World*. Number 3600 in Lecture Notes in Computer Science. Springer-Verlag, 2006.