

GRUNGE: A Grand Unified ATP Challenge^{*}

Chad E. Brown¹, Thibault Gauthier¹, Cezary Kaliszyk²,
Geoff Sutcliffe³, and Josef Urban¹

¹ Czech Technical University in Prague

² University of Innsbruck and University of Warsaw

³ University of Miami

Abstract. This paper describes a large set of related theorem proving problems obtained by translating theorems from the HOL4 standard library into multiple logical formalisms. The formalisms are in higher-order logic (with and without type variables) and first-order logic (possibly with types, and possibly with type variables). The resultant problem sets allow us to run automated theorem provers that support different logical formalisms on corresponding problems, and compare their performances. This also results in a new “grand unified” large theory benchmark that emulates the ITP/ATP hammer setting, where systems and metasytems can use multiple formalisms in complementary ways, and jointly learn from the accumulated knowledge.

Keywords: Theorem Proving · Higher-Order Logic · First-Order Logic · Many-Sorted Logic

1 Introduction

A hammer [7] for an interactive theorem prover (ITP) [22] typically translates an ITP goal into a formalism used by an automated theorem prover (ATP). Since the most successful ATPs have so far been for untyped first-order logic, the focus has been on first-order translations. There is also interest in ATPs working in richer formalisms, such as monomorphic and polymorphic, typed first-order, and higher-order logics. The TPTP formats for various formalisms have been adopted for this work, viz. FOF [46] TF0 [47], TF1 [8], TH0 [5], and TH1 [26]. An interesting related task is the creation of a (grand) unified large-theory benchmark that allows fair comparison of such ATP systems, their combination and integration with premise selectors and machine learners [1], across different formalisms. As a step towards creating such benchmarks we present two families of translations from the language of HOL4 [42] to the various TPTP formats. We have implemented these translations and plan to use them as the first “GRand UNified ATP challenGE” (GRUNGE) benchmarks, generalizing existing benchmarks such as

^{*} Supported by the ERC grant no. 649043 AI4REASON and no. 714034 SMART, by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466, the European Regional Development Fund, and the National Science Foundation Grant 1730419 - “CI-SUSTAIN: StarExec: Cross-Community Infrastructure for Logic Solving”.

the CakeML export [31] that was used in the large-theory benchmark (LTB) division of the CASC-J9 ATP competition [48].

The rest of the paper is structured as follows. Section 2 introduces notation and the HOL syntax. Section 3 introduces the problems – the HOL4 standard library. Section 4 introduces the first family of translations, and Section 5 introduces the second family of translations. Section 6 discusses and compares the translations on an example, and Section 7 evaluates the translations using existing ATPs. Section 8 describes the CASC-27 LTB division, which is based on these translations. Related work is discussed in Section 9.

2 Preliminaries

Since this work is based on the HOL4 standard library, it is necessary to start with brief comments about the syntax and notion of proof in HOL4. More detailed information is in [42,19]. HOL4, like several other ITPs (e.g., Isabelle/HOL [36], HOL Light [20] or ProofPower [28]), is based on an extension of Church’s simple type theory [12] that includes prefix polymorphism and type definitions [19]. HOL4 includes a type o of propositions, a type ι of individuals and a type $(\sigma \rightarrow \tau)$ of functions from a type σ to a type τ . Parentheses are omitted, with $\rightarrow$ associating to the right. In addition, there are type variables α and defined types. At each point in the development of the HOL4 library there is a finite set of (previously defined) constants c , and a finite set of (previously defined) type constructors κ giving a type $\kappa(\sigma_1, \dots, \sigma_n)$ for types $\sigma_1, \dots, \sigma_n$. For simplicity we consider the signature to be fixed, to avoid the need to specify the types and terms relative to an evolving signature.

Terms are generated from constants c and variables x using application ($s\ t$) and λ -abstractions ($\lambda x : \sigma. t$) in the expected way, for terms s and t . Parentheses are omitted, with application associating the left. Binders have scope as far to the right as possible, consistent with parentheses. Multiple binders over the same type can be written in a combined form, e.g., $\lambda xy : \sigma. t$ means $\lambda x : \sigma. \lambda y : \sigma. t$.

Constants may be polymorphic. There are two primitive polymorphic logical constants: $=^\alpha$ is polymorphic with type $\alpha \rightarrow \alpha \rightarrow o$ and ε^α is polymorphic with type $(\alpha \rightarrow o) \rightarrow \alpha$, where α is a type variable. When terms are defined, such constants are used with a fixed type for α written as a superscript. New polymorphic constants can be defined within a HOL4 theory.

Aside from $=^\alpha$ and ε^α , implication $\Rightarrow$ of type $o \rightarrow o \rightarrow o$ is primitive. From these primitive logical constants it is possible to define $\wedge$, $\vee$ and $\neg$, as well as polymorphic operators $\forall^\alpha$ and $\exists^\alpha$. The usual notation is used for these logical connectives, so that the binder notation $\forall^\sigma(\lambda x : \sigma. t)$ is written as $\forall x : \sigma. t$, using the same binding conventions as for λ -abstractions. Similarly, $\exists^\sigma(\lambda x : \sigma. t)$ is written as $\exists x : \sigma. t$.

Terms of type o are called propositions, and we use φ and ψ to range over propositions. A sequent is a pair $\Gamma \vdash \varphi$ where Γ is a finite set of propositions and φ is a proposition. There is a notion of HOL4 provability for sequents. While our translations map HOL4 sequents to TPTP formulae, it is not our intention to mirror HOL4 provability in the target format. The intention, roughly speaking,

is to gain information about when a HOL4 theorem is a consequence of previous HOL4 theorems, in some logic weaker than HOL4.

From the simplest perspective, each translation translates HOL4 types and HOL4 terms (including propositions) to terms in the target format. A type, term or sequent with no type variables is called *monomorphic*. As an optimization, some of the translations translate some monomorphic HOL4 types to types in the target language. As a common notation throughout this paper, a HOL4 type σ translated as a term is written as $\hat{\sigma}$, and σ translated as a type is written as $\tilde{\sigma}$. Another optimization is to translate HOL4 propositions (and sequents) to the level of formulae in the target language.

3 Problem Set: The HOL4 Standard Library

Version Kananaskis-12 of the HOL4 standard library contains 15733 formulae: 8 axioms, 2294 definitions, and 13431 theorems. If most of the formulae were monomorphic and fell into a natural first-order fragment of HOL4, then there would be a natural translation into the FOF format. However, many formulae are either polymorphic or higher-order (or both), as Table 1 shows (note that the numbers are not cumulative, e.g., the 2232 monomorphic first-order formulae do not include the 1632 uni-sorted first-order formulae, which could also be processed by an ATP that can handle the monomorphic types). The problem set consists of 12140 theorems proven in the HOL4 standard library⁴, in the context of a finite set of dependencies used in the HOL4 proof [16].

	First-order	Higher-order	Combined
Uni-sorted	1632 (FOF)	0	1632
Monomorphic	2232 (TF0)	3683 (TH0)	5915
Polymorphic	1536 (TF1)	6650 (TH1)	8186
Combined	5400	10333	15733

Table 1. Number of HOL4 formulae in each category.

4 Syntactic Translations via First-Order Encodings

There already exist many families of translations for HOL to the TPTP format, usually developed for hammers [27,35]. We have adopted and adapted them: (i) We have made the translations more local, by making them independent for each theorem, i.e., unaffected by other theorems. In particular, this means that when the same lambda function appears in two theorems, lambda-lifting will produce two new functions. (ii) We have made more problems provable (in principle) by introducing additional axioms and relying on an embedding of

⁴ 1291 theorems were not included due to dependencies being erased during the build of the HOL4 library.

polymorphic types instead of relying on heuristic monomorphization. (iii) For the TF0 and TH0 formats, we have made use of their polysortedness by expressing the type of monomorphic constants directly using TF0 and TH0 types.

The translations are described in the order $\text{TH1} \rightarrow \text{TF1} \rightarrow \text{FOF} \rightarrow \text{TF0} \rightarrow \text{TH0}$, as translations to the later formats take advantage of translation techniques used for earlier formats.

4.1 Translating to TH1

TH1 is a language that is strictly more expressive than HOL4. Therefore HOL4 formulae can be represented in TH1 with minimal effort. This produces the TH1-I collection of ATP problems.

Alignment of Logical Constructions. The TPTP format contains a set of defined constructs that have implicit definitions, and HOL4 objects are mapped to their TPTP counterparts in a natural way. The boolean type *o* of HOL4 is mapped to the defined TPTP type *\$o*. The arrow type operator is mapped to the TPTP arrow *>*. All other type operators are declared to take *n* types and give a type, using the TPTP “type of types” *\$tType*. For example, the type operator *list* has type *\$tType > \$tType*.

The TPTP logical connectives $\wedge, \vee, \Rightarrow, \neg, =, \forall, \exists$ are used at the top-level of the translated formula whenever possible, but the corresponding HOL4 constants are used when necessary. Equivalences relating HOL4 logical constants to TPTP connectives are included.

Explicit type arguments. A HOL4 constant *c* carries a type ν' . This type is an instance of type ν that was given to *c* when *c* was created. By matching ν with ν' , a type substitution *s* can be inferred. Ordering the type variables in the domain of *s*, the implicit type arguments of *c* are deduced. Making the quantification of type variables and the type arguments explicit is required in the TH1 format. The effect that this requirement has on a constant declaration and a formula is shown in Example 1.

Example 1. Explicit type arguments and type quantifications

	HOL4	TH1
Type of <i>I</i>	$\alpha \rightarrow \alpha$	$\forall \alpha : \$tType. \alpha \rightarrow \alpha$
Formula	$\forall x : \alpha. (I\ x) = x$	$\forall \alpha : \$tType. \forall x : \alpha. ((I\ \alpha)\ x) = x$

4.2 Translating to TF1

To produce the TF1-I collection of ATP problems, all the higher-order features of the HOL4 problems have to be eliminated. This is done in a sequence of steps.

Lambda-lifting and Boolean-lifting. One of the higher-order features is the presence of lambda-abstraction. The translation first uses the extensionality property to add extra arguments to lambdas appearing on either side of an equality, and then beta-reduce the formula. Next lambda-lifting [14,35] is used. Lambda-lifting creates a constant **f** for the leftmost outermost lambda-abstractions appearing at the term-level. This constant **f** replaces the lambda-abstraction in the formula. A definition is given for **f**, which may involve some variable capture - see Example 2. This procedure is repeated until all the atoms are free of lambda-abstractions. The definitions are part of the background theory, and are considered to be axioms in the **TF1** problem even if they were created from the conjecture.

Example 2. Lambda-lifting with variable capture

Original formula: $\forall k. \text{linear } (\lambda x. k \times x)$
 Additional definition: $\forall k x. \mathbf{f} k x = k \times x$
 New formula: $\forall k. \text{linear } (\mathbf{f} k)$

A similar method can be applied to move some logical constants from the term-level to the formula-level - see Example 3 (this optimization is not applied in our second family of translations).

Example 3. Boolean-lifting

Original formula: $\forall x. x = \text{COND } (x = 0) 0 x$
 Additional definition: $\forall x. \mathbf{f} x \Leftrightarrow (x = 0)$
 New formula: $\forall x. x = \text{COND } (\mathbf{f} x) 0 x$

To allow the ATPs to create their own encoded lambda-abstractions, axioms for the combinators **S**, **K** and **I** are added to every problem. These axioms are omitted in the second family of translations. In the **TH0-II** versions of the problem, combinators are not needed since all simply typed λ -calculus terms are already representable. In the **TF0-II** and **FOF-II** versions only an axiom for **I** and a partially applied axiom for **K** are included. Combinator axioms enlarge the search space, which hinders the ATPs unnecessarily because they are not needed for proving most of the theorem.

Apply operator and arity equations. As functions cannot be passed as arguments in first-order logic, an explicit apply operator **ap** is used to apply a function to an argument. This way all objects (constants and variables) have arity zero except for the apply operator which has arity two. The **HOL4** functional extensionality axiom is added to all problems, as it expresses the main property of the apply operator:

$$\forall f g (\forall x. \mathbf{ap} f x = \mathbf{ap} g x) \Rightarrow f = g$$

This axiom also demonstrates how the higher-order variables *f* and *g* become first-order variables after the introduction of **ap**.

To limit the number of apply operators in a formula, versions of each constant are defined for all its possible arities, in terms of the zero-arity version. These constants are used in the translated formula - see Example 4.

Example 4. Using constants with their arity

Original formula: $\text{SUC } 0 = 1 \wedge \exists y. \text{MAP SUC } y \neq y$

Arity equations: $\text{SUC}_1 x = \text{ap SUC}_0 x, \dots$

New formula: $\text{SUC}_1 0_0 = 1_0 \wedge \exists y. \text{MAP}_2 \text{SUC}_0 y \neq y$

If the return type of a constant is a type variable then some of its instances can expect an arbitrarily large numbers of arguments. In the case where the number of arguments n' of an instance exceeds the number of arguments n of the primitive constant, variants of this constant are not created for this arity. Instead, the apply operator is used to reduce the number of arguments to n . For example, the term $\text{I I } x$ is not translated to $\text{I}_2 \text{I}_0 x$ but instead to $\text{ap } (\text{I}_1 \text{I}_0) x$.

TF1 types. As a final step, type arguments and type quantifications are added as in Section 4.1. Moreover, the boolean type of HOL4 is replaced by $\$o$ at the formula-level, and by o at the term-level (because $\$o$ is not allowed at the term-level in a first-order formula). This causes a mismatch between the type of the atom o and the type of the logical connective $\$o$. Therefore an additional operator $\mathbf{p} : o \rightarrow \o is applied on top of every atom. The following properties of $\mathbf{p}$ and o are added to every translated problem (written here in the first-order style for function application):

$$\begin{aligned} \forall xy : o. (\mathbf{p}(x) \Leftrightarrow \mathbf{p}(y)) \Rightarrow (x = y) \\ \mathbf{p}(\text{true}), \neg \mathbf{p}(\text{false}), \forall x : o. x = \text{true} \vee x = \text{false} \end{aligned}$$

In a similar manner, the TPTP arrow type cannot be used whenever a function appears as an argument. Instead the type constructor **fun** is used, as illustrated by the following constant declaration:

$$\mathbf{ap} : \forall \alpha : \$tType \beta : \$tType. ((\mathbf{fun}(\alpha, \beta) \times \alpha) \rightarrow \beta).$$

4.3 Translating to FOF

The translation to FOF, which produces the FOF-I collection of ATP problems, follows exactly the same path as the translation to TF1 except that the types are encoded as first-order terms. To represent the fact that a first-order term t has type ν , the tagging function **s**, introduced by Hurd [23], is used: every term t of type ν is replaced by $\mathbf{s}(\hat{\nu}, t)$. Going from the type ν to the term $\hat{\nu}$ effectively transforms type variables into term variables, and type operators into first-order functions and constants. Type arguments are unnecessary as the tags contain enough information. In practice, the **s** tagging function prevents terms of different types from unifying, and allows instantiation of type variables - see Example 5.

Example 5. Type instantiation

TF1	FOF
$\forall x : \alpha. \text{I}(\alpha, x) = x$	$\forall \hat{\alpha} \hat{x}. \mathbf{s}(\hat{\alpha}, \hat{\text{I}}(\mathbf{s}(\hat{\alpha}, \hat{x}))) = \mathbf{s}(\hat{\alpha}, \hat{x})$
$\forall x : \text{num}. \text{I}(\text{num}, x) = x$	$\forall \hat{x}. \mathbf{s}(\widehat{\text{num}}, \hat{\text{I}}(\mathbf{s}(\widehat{\text{num}}, \hat{x}))) = \mathbf{s}(\widehat{\text{num}}, \hat{x})$

4.4 Translating to TF0

An easy way to translate HOL4 formulae to TF0, which produces the TF0-I collection of ATP problems, is to take the translation to FOF and inject it into TF0.

Trivial Injection from FOF to TF0. The first step is to give types to all the constants and variables appearing in the FOF formula. A naive implementation would be to give the type $\iota^n \rightarrow \iota$ to symbols with arity n . However, since it is known that the first argument comes from the universe of non-empty types, and the second argument comes from the universe of untyped terms, an explicit distinction can be made. The type of **s** is defined to be $\delta \times \mu \rightarrow \iota$, with δ being the universe of non-empty types, μ being the universe of untyped terms, and ι being the universe of typed terms. After this translation a type operator (or type variable) with m arguments has type $\delta^m \rightarrow \delta$, and a function (or term variable) with n arguments has type $\iota^n \rightarrow \mu$. The type of **p** is $\mathbf{p} : \iota \rightarrow \text{\$o}$. Declaring the type of all these objects achieves a trivial translation from FOF with tags to TF0.

Using Special Types. To take full advantage of the polysortedness of TF0, a constant $\tilde{c}_n^\nu$ is declared for every constant c_n , with arity n and monomorphic type ν . The type of $\tilde{c}_n^\nu$ is declared to be $(\tilde{\nu}_1 \times \dots \times \tilde{\nu}_n) \rightarrow \tilde{\nu}_0$, where $\tilde{\nu}_1, \dots, \tilde{\nu}_n$, and $\tilde{\nu}_0$ are basic types. A basic type constructs a single type from a monomorphic type, e.g., *list_real* for *list[real]*, *fun_o_o* for *fun(o, o)*. The basic types are special types, and are declared using **\\$tType**. Thanks to these new constants monomorphic formulae can be expressed in a natural way, without type encodings in the formula. Nevertheless, an ATP should still be able to perform a type instantiation if necessary. That is why we relate the monomorphic representation with its tagged counterpart.

If a term has a basic type then it lives in the monomorphic world where as a term of type ι it belongs to the tagged world. All monomorphic terms (constructed from monomorphic variables and constants) can be expressed in the monomorphic world. To relate the two representations of the same HOL4 term an “injection” i_ν and a “surjection” j_ν are defined for each basic type $\tilde{\nu}$. The constants $i_\nu : \tilde{\nu} \rightarrow \mu$ and $j_\nu : \iota \rightarrow \tilde{\nu}$ must respect the following properties, which are included as axioms in the translated problems:

$$\begin{aligned} \forall x : \mu. \mathbf{s}(\hat{\nu}, i_\nu(j_\nu(\mathbf{s}(\hat{\nu}, x)))) &= \mathbf{s}(\hat{\nu}, x) \\ \forall x : \tilde{\nu}. j_\nu(\mathbf{s}(\hat{\nu}, i_\nu(x))) &= x \end{aligned}$$

Whenever $\tilde{c}_n^\nu$ is an instance of a polymorphic function $\hat{c}_n$, the following equation is included in the TF0 problem, which relates the two representatives:

$$\forall x_1 : \tilde{\nu}_1 \dots x_n : \tilde{\nu}_n. \mathbf{s}(\nu_0, i_{\nu_0}(\tilde{c}_n^\nu(x_1, \dots, x_n))) = \mathbf{s}(\nu_0, \hat{c}_n(\mathbf{s}(\nu_1, i_{\nu_1}(x_1)), \dots, \mathbf{s}(\nu_n, i_{\nu_n}(x_n))))$$

Example 6 shows how “injections”, “surjections”, and special types can be used to translate a theorem mixing polymorphic and monomorphic elements.

Example 6. Special types

The polymorphic function I is applied to the monomorphic variable x . Using special types, type tags can be dropped for x .

$$\begin{aligned} \text{FOF: } & \forall \hat{x}. s(\widehat{\text{num}}, \hat{l}(s(\widehat{\text{num}}, \hat{x}))) = s(\widehat{\text{num}}, \hat{x}) \\ \text{TF0: } & \forall \tilde{x} : \widehat{\text{num}}. j_{\text{num}}(\hat{l}(s(\widehat{\text{num}}, i_{\text{num}}(\tilde{x})))) = \tilde{x} \end{aligned}$$

Effect on defined operators. The **ap** operator is treated in the same way as every other constant. In particular, a different version of **ap** is created for each monomorphic type. The type of **p** becomes $\tilde{o} \rightarrow \$o$, and the projection j_o is used to transfer atoms from the tagged world to the monomorphic world.

If the presence of the **p** predicate and the inclusion of additional equations are ignored, our translation of a HOL4 first-order monomorphic formula using special types to TF0 is simply the identity transformation.

4.5 Translating to TH0

Translating from HOL4 to TH0, which produces the TH0-I collection of ATP problems, is achieved in a way similar to the translation to TF0. The HOL4 formulae are first translated to FOF and then trivially injected into TH0. Special types are used for basic types extracted from monomorphic types. The set of higher-order basic types is slightly different from the first-order one, where we recursively remove arrow types until a non-arrow constructor is found. In the higher-order setting a single monomorphic constant $\tilde{c}^\nu$ is used to replace all arity versions of c : $\forall f x. \tilde{a}p^\nu f x = f x$. Another benefit of the expressivity of TH0 is that the basic type $\tilde{o}$ can be replaced by $\$o$, and the predicate **p** can be omitted. The effect of the previous steps is illustrated in Example 7.

Example 7. Translations of $\exists f. f\ 0 = 0$

In this example $\tilde{a}p^\nu$ has type $(\text{fun_num_num} \times \widehat{\text{num}}) \rightarrow \widehat{\text{num}}$ where fun_num_num is the special type corresponding to $\text{num} \rightarrow \text{num}$.

$$\begin{aligned} \text{TF0: } & \exists \tilde{f} : \text{fun_num_num}. \tilde{a}p^\nu(\tilde{f}, \tilde{0}^{\text{num}}) = \tilde{0}^{\text{num}} \\ \text{TH0: } & \exists \tilde{f} : \widehat{\text{num}} \rightarrow \widehat{\text{num}}. \tilde{f}\ \tilde{0}^{\text{num}} = \tilde{0}^{\text{num}} \end{aligned}$$

In order to have the same shallowness result for TH0 as for TF0, it would be necessary to replace monomorphic constants created by the lifting procedure by their lambda-abstractions. We chose to keep the definitions for the lifted constants, as they allow some term-level logical operators to be pushed to the formula level.

5 Semantic Translations via Set Theory Encodings

The second family of translations into TH0, TF0, and FOF is semantically motivated [38]: we make use of constructors known to be definable in set theory. Types and terms are translated to sets, where types must translate to non-empty sets. The translation may optionally use other special types for monomorphic

types in the HOL4 source. In the TH0 case the builtin type $\$o$ can be used for the HOL4 type o . In the first-order cases HOL4 terms of type o are sometimes translated to terms, and sometimes to formulae, depending on how the HOL4 term is used. In the TF0 case a separate type $\tilde{o}$ of booleans is declared, which is used as the type of terms translated from HOL4 terms of type o . In the FOF case this approach is not possible, as all terms have the same type (intuitively representing sets). The other main difference between the translation to TH0 and the translations to the first-order languages is that the first-order translations make use of lambda lifting [14,35]. As a result of the translations we obtain three new collections of ATP problems are produced: TH0-II, TF0-II and FOF-II.

5.1 Translating to TH0

The base type o for propositions is written as $\$o$ in TH0, and ι for individuals is written as $\$i$. In addition a base type δ is declared. The translation treats elements of type ι as sets, and elements of type δ as non-empty sets. The basic constants used in the ATP problems are as follows:

- $\text{bool} : \delta$ is used for a fixed two element set.
- $\text{ind} : \delta$ is used for a fixed non-empty set corresponding to HOL4's type of individuals.
- $\text{arr} : \delta \rightarrow \delta \rightarrow \delta$ is used to construct the function space of two sets.
- $\text{mem} : \iota \rightarrow \delta \rightarrow o$ corresponds to the membership relation on sets, where the second set is known to be non-empty. The term $\text{mem } s \ t$ is written as $s \in t$, and the term $\forall x. x \in s \rightarrow t$ is written as $\forall x \in s. t$.
- $\text{ap} : \iota \rightarrow \iota \rightarrow \iota$ corresponds to set theory level application (represented as a set).
- $\text{lam} : \delta \rightarrow (\iota \rightarrow \iota) \rightarrow \iota$ is used to build set bounded λ -abstractions as sets.
- $\text{p} : \iota \rightarrow o$ is a predicate that indicates whether or not an element of bool is true or not.
- $\text{i}_o : o \rightarrow \iota$ is an injection of o into ι , essentially translating false to a set and true to a different set.

The basic axioms included in each ATP problem are:

Inj_o: $\forall X : o. \text{i}_o X \in \text{bool}.$
Iso_o¹: $\forall X : o. \text{p}(\text{i}_o X) = X.$
Iso_o²: $\forall X \in \text{bool}. \text{i}_o(\text{p} X) = X.$
ap_{tp}: $\forall AB : \delta. \forall f \in (\text{arr } A \ B). \forall x \in A. (\text{ap } f \ x) \in B.$
lam_{tp}: $\forall AB : \delta. \forall F : \iota \rightarrow \iota. (\forall x \in A. F \ x \in B) \rightarrow (\text{lam } A \ F) \in (\text{arr } A \ B).$
FunExt: $\forall AB : \delta. \forall f \in (\text{arr } A \ B). \forall g \in (\text{arr } A \ B).$
 $(\forall x \in A. \text{ap } f \ x = \text{ap } g \ x) \rightarrow f = g.$
beta: $\forall A : \delta. \forall F : \iota \rightarrow \iota. \forall x \in A. (\text{ap } (\text{lam } A \ F) \ x) = F \ x.$

If ι is interpreted using a model of ZFC and δ using a copy of the non-empty sets in this model, then the constants above can be interpreted in an obvious way so as to make the basic axioms true.

Given this theory, a basic translation from HOL4 to TH0 is as follows. Each HOL4 type α (including type variables) is mapped to a term $\hat{\alpha}$ of type δ . HOL4 type variables (constants) are mapped to TH0 variables (constants) of type δ . For the remaining cases `bool`, `ind`, and `arr` are used. Each HOL4 term $s : \alpha$ is mapped to a TH0 term $\hat{s}$ of type ι , for which the context $\hat{s} \in \hat{\alpha}$ is always known. The invariant can be maintained by including the hypothesis $\hat{x} \in \hat{\alpha}$ whenever x is a variable or a constant. The `ap` and `lam` constants are used to handle HOL4 applications and λ -abstractions. The axioms `aptp` and `lamtp` ensure the invariant is maintained. Finally HOL4 propositions (which may quantify over type variables) are translated to TH0 propositions in an obvious way, using `p` to go from ι to o , and `io` to go from o to ι , when necessary. As an added heuristic, the translation makes use of TH0 connectives and quantifiers as deeply as possible, delaying the use of `p` whenever possible.

Using Special Types. As with the first family of translations, the second family optimizes by using special types for HOL4 types with no type variables, e.g., `num` and `list num`. Unlike the first family, special types are not used for monomorphic function types. As a result it is not necessary to consider alternative `ap` operators. A *basic monomorphic type* is a monomorphic type that is not of the form $\alpha \rightarrow \beta$. If special types are used, then for each basic monomorphic type occurring in a proposition a corresponding TH0 type γ is declared, mappings and axioms relating γ to the type ι of sets are declared, and the type γ is used to translate terms of the type and quantifiers over the type when possible. For example, if a basic monomorphic type ν (e.g., `num`) occurs in a HOL4 proposition, then in addition to translating ν as a term $\hat{\nu} : \iota$ we also declare a TH0 type $\tilde{\nu}$, $i_\nu : \tilde{\nu} \rightarrow \iota$ and $j_\nu : \iota \rightarrow \tilde{\nu}$ along with axioms $\forall x : \tilde{\nu}. j_\nu(i_\nu x) = x$ and $\forall x : \iota. x \in \tilde{\nu} \rightarrow i_\nu(j_\nu x) = x$.

One obvious basic monomorphic type is o . In the case of o a new type is not declared, but instead the TH0 type `$o` is used. That is, $\tilde{o}$ denotes `$o`. Note that $i_o : \tilde{o} \rightarrow \iota$ is already declared. Additionally, `jo` is used as shorthand for `p`, which has the desired type $\iota \rightarrow \tilde{o}$.

Suppose a HOL4 constant c has type $\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$, where $\alpha_1, \dots, \alpha_n, \beta$ are basic monomorphic types with corresponding TH0 types $\tilde{\alpha}_1, \dots, \tilde{\alpha}_n, \tilde{\beta}$. Instead of translating a term $c t_1 \dots t_n$ as a term of type ι , each t_i is translated to a term $\hat{t}_i$ of type $\tilde{\alpha}_i$, and a first order constant $\tilde{c} : \tilde{\alpha}_1 \rightarrow \dots \rightarrow \tilde{\alpha}_n \rightarrow \tilde{\beta}$ is used to translate to the term $\tilde{c} \hat{t}_1 \dots \hat{t}_n$ of type $\tilde{\beta}$. In such a case an equation relating $\hat{c}$ to $\tilde{c}$ is also included. Since the translation may return a term of type ι or $\tilde{\alpha}$, where α is a basic monomorphic type, i_α and j_α are used to obtain a term of type $\tilde{\alpha}$ or ι when one is required. If a quantifier ranges over a monomorphic type α , a quantifier over type $\tilde{\alpha}$ is used instead of using a quantifier over type ι and using $\in$ to guard the quantifier.

5.2 Translating to TF0

There are two main modifications to the translation to TH0 when targeting TF0. Firstly, propositions cannot be treated as special kinds of terms in TF0. In order

to deal with this o is treated like other special types by declaring a new type $\tilde{o}$ and functions $i_o : \tilde{o} \rightarrow \iota$ and $j_o : \iota \rightarrow \tilde{o}$ along with corresponding axioms as above. Note that unlike the TH0 case, j_o differs from p . In TF0 p is a unary predicate on ι , and j_o is a function from ι to $\tilde{o}$. In the TF0 versions of the axioms $\mathbf{Iso}_o^1$ and $\mathbf{Iso}_o^2$, p is replaced with j_o . Secondly, the background theory cannot include the higher-order **lam** operator. Therefore the **lam** operator is omitted, and lambda lifting is used to translate (most) HOL4 λ -abstractions. The two higher-order axioms **lam_{tp}** and **beta** are also omitted.

In the TH0 case, the background axioms are enough to infer the following (internal) propositional extensionality principle

$$\forall Q \in \text{bool}. \forall R \in \text{bool}. (p\ Q \leftrightarrow p\ R) \rightarrow Q = R$$

from the corresponding extensionality principle $\forall QR : o. (Q \leftrightarrow R) \rightarrow Q = R$ valid in TH0. This is no longer the case in TF0, so propositional extensionality is added as an axiom.

There are two special cases where lambda lifting can be avoided: identity and constant functions. For this purpose a new unary function **I** on sets and a new binary function **K** on sets are added. Two new basic axioms are added to the ATP problem for these functions:

Id: $\forall A : \delta. \forall X \in A. (\text{ap } (\mathbf{I}\ A)\ X) = X.$

Const: $\forall A : \delta. \forall Y : \iota. \forall X \in A. (\text{ap } (\mathbf{K}\ A\ Y)\ X) = Y.$

A HOL4 term $\lambda x : \alpha. x$ is translated as $\mathbf{I}\ \hat{\alpha}$. For a HOL4 term $\lambda x : \alpha. t$, where x is not free in t , t is translated to a first-order term $\hat{t}$ of type ι , and the λ -term is translated to $\mathbf{K}\ \hat{\alpha}\ \hat{t}$. If there is already a function defined for $\lambda x : \alpha. t$ (with the same variable names), then that function is reused. Otherwise, lambda lifting of $\lambda x : \alpha. t$ proceeds as follows. Let $\alpha_1, \dots, \alpha_m$ be type variables occurring in $\lambda x : \alpha. t$ and $y_1 : \beta_1, \dots, y_n : \beta_n$ be the free variables occurring in $\lambda x : \alpha. t$. Assume $\hat{t}$ is a first-order term translation of t , with $\hat{x}$ of type ι corresponding to the variable x . (Note that this may have involved some lambda lifting.) Let **f** be a new $m + n$ -ary function returning sets. If special types are not being used, then each argument of **f** is a set. If special types are used, then each argument is a set unless it corresponds to $y_i : \beta_i$, where β_i is a monomorphic type in which case the argument has type $\tilde{\beta}_i$. The following axioms about **f** are added to the ATP problem:

f_{tp}: $\forall A_1 \dots A_n : \delta. \forall Y_1 \dots Y_m : \iota. \dots (\mathbf{f}\ A_1 \dots A_n\ Y_1 \dots Y_m) \in \hat{\alpha}.$

f_{beta}: $\forall A_1 \dots A_n : \delta. \forall Y_1 \dots Y_m : \iota. \dots \forall X \in \hat{\alpha}. \text{ap } (\mathbf{f}\ A_1 \dots A_n\ Y_1 \dots Y_m)\ X = \hat{t}.$

In these axioms the preconditions that each Y_i must be in $\tilde{\beta}_i$ if Y_i has type ι have been elided (otherwise special types are being used, β_i is monomorphic, Y_i has type $\tilde{\beta}_i$, and no guard is required).

5.3 Translating to FOF

In order to translate to FOF, all terms must be translated to the same type, effectively the type ι . This requires omission of any special treatment of monomorphic

types, and instead all HOL4 terms are translated to terms of type ι . The type δ of non-empty sets is also omitted. Instead, ι is used wherever δ was used in the TF0 setting, and quantifiers that were over δ are guarded by a new non-emptiness predicate $\text{ne} : \iota \rightarrow o$. Aside from these changes, the translation proceeds using lambda lifting as in the TF0 case.

6 Case Study

A very simple HOL4 theorem is $\forall f : \alpha \rightarrow \beta. \forall x : \alpha. \text{LET}^{\alpha, \beta} f x = f x$, where $\text{LET}^{\alpha, \beta}$ is defined to be $\lambda f : \alpha \rightarrow \beta. \lambda x : \alpha. f x$. Informally the proof is clear: expand the definition of **LET** and perform two β -reductions. However, proving various translated versions of the problem range from trivial to challenging.

The first family of translations make use of a preprocessing step (Section 4.2) that changes the definition of **LET** from $\text{LET}^{\alpha, \beta} = \lambda f : \alpha \rightarrow \beta. \lambda x : \alpha. f x$ to

$$\forall x : \alpha \rightarrow \beta. \forall x' : \alpha. \text{LET}^{\alpha, \beta} x x' = x x'.$$

This step makes the definition of **LET** the same (up to α -conversion) as the theorem. Even if further encodings are applied to obtain a first-order problem, the axiom will still be the same as the conjecture. Consequently all versions resulting from the first family of translations are trivially provable.

The TH0-II version has conjecture

$$\forall AB : \delta. \forall f \in (\text{arr } A B). \forall x \in A. \text{ap } (\text{ap } (\text{LET } A B) f) x = \text{ap } f x$$

and the axiom (corresponding to the definition of **LET**)

$$\forall AB : \delta. \text{LET } A B = \text{lam } (\text{arr } A B) (\lambda f : \iota. \text{lam } A (\lambda x : \iota. \text{ap } f x)).$$

The axiom defining **LET** combined with the basic axiom **beta** is enough to prove the theorem. However, the TH0-II version also includes all the other basic axioms along with internal versions of the logical constants for universal quantification and equality. The extra axioms make the problem hard for ATP systems, but if only the necessary axioms are provided the problem is easy. In TF0-II and FOF-II the conjecture is the same as in the TH0-II version, but the definition of **LET** is split into two functions declared when lambda lifting:

$$\forall A \dots \forall B \dots \text{LET } A B = \mathbf{f}_{14} A B,$$

$$\forall A \dots \forall B \dots \forall f \in (\text{arr } A B). \text{ap } (\mathbf{f}_{14} A B) f = \mathbf{f}_{13} A B f$$

and

$$\forall A \dots \forall B \dots \forall f \in (\text{arr } A B). \forall x \in A. \text{ap } (\mathbf{f}_{13} A B f) x = \text{ap } f x.$$

All the first-order versions of this problem are easy for current ATP systems.

7 Results

Since the HOL4 library has a natural order of the problems, each translation can generate two versions of each problem. The *bushy* (small) version contains only the (translated) library facts that were needed for the HOL4 proof of the theorem. The *chainy* (large) version contains all the facts that precede the theorem in the library order, i.e., the real task faced by hammer systems. Chainy problems typically include thousands of axioms, requiring the use of *premise selection* algorithms [1] as a front-end in the ATP systems. Thus, in order to maintain the focus on ATP system performance, the results of running the ATP systems on the bushy problems are presented here.

Nineteen ATPs were run on the 12140 problems in each of the bushy problem sets, according to the ATPs’ support for the various TPTP formats. In each case we ran the ATP with a CPU time limit of 60s per problem. Table 2 summarizes the results. In union, more proofs were found in the first family of translations than in the second family, in all formats. However, some provers like Vampire 4.3 and SPASS 3.9 do better on FOF-II than on FOF-I. This indicates that these provers are probably better at reasoning with type guards than with type tags. Of the 12140 problems 7412 (61.1%) were solved by some ATP in one of the representations.

System	TH1-I	TH0-I	TH0-II	TF1-I	TF0-I	TF0-II	FOF-I	FOF-II	Union
agsyHOL 1.0 [32]	1374	1187							1605
Beagle 0.9.47 [3]				2008	2047	2449	2498		3183
cocATP 0.2.0	899	599							1000
CSE.E 1.0 [51]							4251	3102	4480
CVC4 1.6 [2]				4851	3991	5030	3746		5709
E 2.2 [41]				4277	3622	4618	3844		5118
HOLyHammer 0.21 [27]	5059								5059
iProver 2.8 [29]							2778	2894	3355
iProverModulo 2.5-0.1 [11]				2435	1639	1433	1263		2852
LEO-II 1.7.0 [4]	2579	1923					2119	1968	3702
Leo-III 1.3 [44,43]	6668	5018	3485	3458	4032	3421	3986	3185	7090
Metis 2.4 [24]							2353	474	2356
Princess 170717 [39,40]				3646	2138	3162	2086		4096
Prover9 1109a [33]							2894	1742	3128
Satallax 3.3 [10]	2207	1292							2494
SPASS 3.9 [50]							2850	3349	3821
Vampire 4.3 [30]				4837	4693	4008	4928		5929
ZenonModulo [15]			1071	1038	1041	1026	1198		1751
Zipperposition 1.4 [13]	2252	2161	3771	3099	2576	2531	1795		4251
Union	6824	5209	3771	4663	5732	5074	5909	5249	7412

Table 2. Number of theorems proved, out of 12140. Each ATP is evaluated on all its supported TPTP formats.

The *TacticToe* [17,18] prover built into HOL4 has been tested as a baseline comparison, and it (re)proves 5327 of 8855 chainy versions of the problems

(60.2%). **TacticToe** is a machine-learning guided prover that searches for a tactical proof by selecting suitable tactics and theorems learned from human-written tactical proofs. By design, this system works in the chainy setting. In total 8840 (72.8%) of the 12140 problems can be proved by either **TacticToe** or one of the ATPs using one of the translations.

8 GRUNGE as CASC LTB Division

The CADE ATP System Competition (CASC) [45] is the annual evaluation of fully automatic, classical logic Automated Theorem Proving (ATP) systems – the world championship for such systems. CASC is divided into divisions according to problem and system characteristics. Each competition division uses problems that have certain logical, language, and syntactic characteristics, so that the systems that compete in the division are, in principle, able to attempt all the problems in the division. For example, the First-Order Form (FOF) division uses problems in full first-order logic, with each problem having axioms and a conjecture to be proved.

While most of the CASC divisions present the problems to the ATP systems one at a time, with an individual CPU or wall clock time limit per problem, the Large Theory Batch (LTB) division presents the problems in batches, with an overall wall clock time limit on the batch. As the name also suggests, the problems in each batch come from a “large theory”, which typically has many functors and predicates, and many axioms of which only a few are required for the proof of a theorem. The problems in a batch typically have a common core set of axioms used by all problems, and each problem typically has additional axioms that are specific to the problem. The batch presentation allows the ATP systems to load and preprocess the common core set of axioms just once, and to share logical and control results between proof searches. Each batch is accompanied by a set of training problems and their solutions, taken from the same source as the competition problems. The training data can be used for ATP system tuning and learning during (typically at the start of) the competition.

In CASC-J9 [48] – the most recent edition of the competition – the LTB division used FOF problems exported from **CakeML** [31]. At the time there was growing interest in an LTB division for typed higher-order problems, and it became evident that a multi-format LTB division would add a valuable dimension to CASC. For the CASC-27 LTB division each problem was presented in multiple formats: TH1, TH0, TF1, TF0, and FOF. The work described in this paper provides the problems. Systems were able to attempt whichever versions they support, and a solution to any version constitutes a solution to the problem. For example, **Leo-III** is able to handle all the formats, while **E** can attempt only the **tffzero** and **FOF** formats.

The batch presentation of problems in the LTB division provides interesting opportunities for ATP systems, including making multiple attempts on problems and learning search heuristics from proofs found. The multi-format LTB division extends these possibilities by allowing multiple attempts on problems by virtue of the multiple formats available, and learning from proofs found in one format

to improve performance on problems in another format. The latter is especially interesting, with little known research in this direction.

9 Related Work

The HOL4 library already has translations for SMT solvers such as Yices [49], Z3 [9] and Beagle. A link to first-order ATPs is also available thanks to exports [16] of HOL4 theories to the HOL(y)Hammer framework [27]. Another notable project that facilitates the export of HOL4 theories is Open Theory [25]. The general approach for higher-order to first-order translations is laid out in Hurd [23]. An evaluation of the effect of different translations on ATP-provability was performed in [35]. A further study shows the potential improvements provided by the use of supercombinators [14]. In our work, the use of lambda-lifting (or combinators) is not necessary in TH0-II thanks to the use of the higher-order operator `lam`. This is similar to using higher-order abstract syntax to model syntax with binders [37].

A method for encoding of polymorphic types as terms through type tags (as in our first translation) or type guards (as in our second translation) is described in [6]. Translations [21,44] from a polymorphic logic to a monomorphic poly-sorted logic without encoding typically rely on heuristic instantiations of type variables. However, heuristics may miss useful instantiations, and make the translation less modular (i.e., context dependent). Our translations to TH0 and TF0 try to get the best of both worlds by using a type encoding for polymorphic types and special types for basic monomorphic types.

10 Conclusion

This work has defined, compared, and evaluated ATP performance on two families of translations of the HOL4 logic to a number of ATP formalisms, and described a new unified large-theory ATP benchmark (GRUNGE) based on them. The first family is designed to play to the strengths of the calculi of most ATP systems, while the second family is based on more straightforward semantics rooted in set theory. The case study shows how different the translated problems may be, even in a simple example. A number of methods and optimizations have been used, however it is clear that the translations can be further optimized and that different encodings favour different provers. Out of 12140 HOL4 theorems, the ATP systems can solve 7412 problems in one or more of the formats. The TacticToe system that works directly in the HOL4 formalism and uses HOL4 tactics could solve 5327 problems. Together the total number of problems solved is 8840. Leo-III was the strongest system in the higher-order representations. In the first-order representations the strongest systems were Zipperposition, CVC4, E and Vampire. A pre-release of the bushy versions of the problems was provided before CASC-27⁵, to allow system developers to adapt and tune their systems before the competition.

⁵ <http://www.tptp.org/CASC/27/TrainingData.HL4.tgz>

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsvitsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning* **52**(2), 191–213 (2014). <https://doi.org/10.1007/s10817-013-9286-5>
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) *Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011), https://doi.org/10.1007/978-3-642-22110-1_14
3. Baumgartner, P., Waldmann, U.: Hierarchic superposition with weak abstraction. In: Bonacina, M.P. (ed.) *Conference on Automated Deduction (CADE)*. Lecture Notes in Computer Science, vol. 7898, pp. 39–57. Springer (2013), http://dx.doi.org/10.1007/978-3-642-38574-2_3
4. Benzmüller, C., Paulson, L., Theiss, F., Fietzke, A.: LEO-II – A Cooperative Automatic Theorem Prover for Higher-Order Logic. In: Baumgartner, P., Armando, A., Dowek, G. (eds.) *Proceedings of the 4th International Joint Conference on Automated Reasoning*, pp. 162–170. No. 5195 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag (2008)
5. Benzmüller, C., Rabe, F., Sutcliffe, G.: THF0 – the core of the TPTP language for classical higher-order logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12–15, 2008, Proceedings*. Lecture Notes in Computer Science, vol. 5195, pp. 491–506. Springer (2008), <http://christoph-benzmueller.de/papers/C25.pdf>
6. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: Piterman, N., Smolka, S.A. (eds.) *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Lecture Notes in Computer Science, vol. 7795, pp. 493–507. Springer (2013), https://doi.org/10.1007/978-3-642-36742-7_34
7. Blanchette, J.C., Kaliszky, C., Paulson, L.C., Urban, J.: Hammering towards QED. *J. Formalized Reasoning* **9**(1), 101–148 (2016). <https://doi.org/10.6092/issn.1972-5787/4593>
8. Blanchette, J.C., Paskevich, A.: TFF1: The TPTP typed first-order form with rank-1 polymorphism. In: Bonacina, M.P. (ed.) *CADE*. Lecture Notes in Computer Science, vol. 7898, pp. 414–420. Springer (2013)
9. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) *Conference on Interactive Theorem Proving (ITP)*. Lecture Notes in Computer Science, vol. 6172, pp. 179–194. Springer (2010), http://dx.doi.org/10.1007/978-3-642-14052-5_14
10. Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) *IJCAR*. Lecture Notes in Computer Science, vol. 7364, pp. 111–117. Springer (2012)
11. Burel, G.: Experimenting with deduction modulo. In: Sofronie-Stokkermans, V., Bjørner, N. (eds.) *CADE 2011*. Lecture Notes in Artificial Intelligence, vol. 6803, pp. 162–176. Springer (2011)
12. Church, A.: A formulation of the simple theory of types. *J. Symbolic Logic* **5**, 56–68 (1940)
13. Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. (*Extensions de la Superposition pour l’Arithmétique*)

- Linéaire Entière, l'Induction Structurale, et bien plus encore). Ph.D. thesis, École Polytechnique, Palaiseau, France (2015), <https://tel.archives-ouvertes.fr/tel-01223502>
14. Czakajka, L.: Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions. In: Avigad, J., Chlipala, A. (eds.) *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, Saint Petersburg, FL, USA, January 20-22, 2016. pp. 49–57. ACM (2016). <https://doi.org/10.1145/2854065.2854069>
 15. Delahaye, D., Doligez, D., Gilbert, F., Halmagrand, P., Hermant, O.: Zenon modulo: When achilles outruns the tortoise using deduction modulo. In: McMillan et al. [34], pp. 274–290. https://doi.org/10.1007/978-3-642-45221-5_20, https://doi.org/10.1007/978-3-642-45221-5_20
 16. Gauthier, T., Kaliszyk, C.: Premise selection and external provers for HOL4. In: *Certified Programs and Proofs (CPP'15)*. Lecture Notes in Computer Science, Springer (2015). <https://doi.org/10.1145/2676724.2693173>
 17. Gauthier, T., Kaliszyk, C., Urban, J.: TacticToe: Learning to reason with HOL4 tactics. In: Eiter, T., Sands, D. (eds.) *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Maun, Botswana, May 7-12, 2017. EPiC Series in Computing, vol. 46, pp. 125–143. EasyChair (2017), <http://www.easychair.org/publications/paper/340355>
 18. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: Learning to prove with tactics. *CoRR* (2018), <http://arxiv.org/abs/1804.00596>
 19. Gordon, M.J.C., Melham, T.F. (eds.): *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press (1993), <http://www.cs.ox.ac.uk/tom.melham/pub/Gordon-1993-ITH.html>
 20. Harrison, J.: HOL Light: A tutorial introduction. In: Srivas, M.K., Camilleri, A.J. (eds.) *FMCAD*. Lecture Notes in Computer Science, vol. 1166, pp. 265–269. Springer (1996)
 21. Harrison, J.: Optimizing proof search in model elimination. In: McRobbie, M., Slaney, J. (eds.) *Conference on Automated Deduction (CADE)*. pp. 313–327. No. 1104 in LNAI, Springer (1996), https://doi.org/10.1007/3-540-61511-3_97
 22. Harrison, J., Urban, J., Wiedijk, F.: History of interactive theorem proving. In: Siekmann, J.H. (ed.) *Computational Logic, Handbook of the History of Logic*, vol. 9, pp. 135–214. Elsevier (2014). <https://doi.org/10.1016/B978-0-444-51624-4.50004-6>
 23. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports pp. 56–68 (2003)
 24. Hurd, J.: System description: The Metis proof tactic. In: Christoph Benzmueller, John Harrison, C.S. (ed.) *Workshop on Empirically Successful Automated Reasoning in Higher-Order Logic (ESHOL)*. pp. 103–104 (2005), <https://arxiv.org/pdf/cs/0601042>
 25. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M.G., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) *NASA Formal Methods*. Lecture Notes in Computer Science, vol. 6617, pp. 177–191. Springer (2011), http://dx.doi.org/10.1007/978-3-642-20398-5_14
 26. Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*. pp. 41–55. No. 1635 in *CEUR Workshop Proceedings* (2016)

27. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning* **53**(2), 173–213 (2014). <https://doi.org/10.1007/s10817-014-9303-3>
28. King, D., Arthan, R., Winnersh, I.: Development of practical verification tools. *ICL Systems Journal* **11**, 106–122 (1996)
29. Korovin, K.: iProver – an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *Automated Reasoning*, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12–15, 2008, Proceedings. *Lecture Notes in Computer Science*, vol. 5195, pp. 292–298. Springer (2008)
30. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) *CAV*. *Lecture Notes in Computer Science*, vol. 8044, pp. 1–35. Springer (2013)
31. Kumar, R., Myreen, M.O., Norrish, M., Owens, S.: CakeML: a verified implementation of ML. In: Jagannathan, S., Sewell, P. (eds.) *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL’14*, San Diego, CA, USA, January 20–21, 2014. pp. 179–192. ACM (2014). <https://doi.org/10.1145/2535838.2535841>
32. Lindblad, F.: A focused sequent calculus for higher-order logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) *Automated Reasoning – 7th International Joint Conference, IJCAR 2014*, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19–22, 2014. Proceedings. *Lecture Notes in Computer Science*, vol. 8562, pp. 61–75. Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_5, https://doi.org/10.1007/978-3-319-08587-6_5
33. McCune, W.: Prover9 and Mace4 (2005–2010), <http://www.cs.unm.edu/~mccune/prover9/>
34. McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.): *Logic for Programming, Artificial Intelligence, and Reasoning – 19th International Conference, LPAR-19*, Stellenbosch, South Africa, December 14–19, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8312. Springer (2013)
35. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning* **40**(1), 35–60 (2008)
36. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. No. 2283 in *Lecture Notes in Computer Science*, Springer-Verlag (2002)
37. Pfenning, F., Elliot, C.: Higher-order abstract syntax. In: *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*. pp. 199–208. PLDI ’88, ACM, New York, NY, USA (1988). <https://doi.org/10.1145/53990.54010>
38. Pitts, A.: The HOL logic. In: Gordon and Melham [19], <http://www.cs.ox.ac.uk/tom.melham/pub/Gordon-1993-ITH.html>
39. Rümmer, P.: A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*. pp. 274–289. No. 5330 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag (2008)
40. Rümmer, P.: E-Matching with Free Variables. In: Bjørner, N., Voronkov, A. (eds.) *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*. pp. 359–374. No. 7180 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag (2012)

41. Schulz, S.: System description: E 1.8. In: McMillan et al. [34], pp. 735–743. https://doi.org/10.1007/978-3-642-45221-5_49
42. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.) Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18–21, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5170, pp. 28–32. Springer (2008)
43. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning. IJCAR 2018. Lecture Notes in Computer Science, vol. 10900, pp. 108–116. Springer, Cham (2018), <http://christoph-benzmueller.de/papers/C70.pdf>
44. Steen, A., Wisniewski, M., Benzmüller, C.: Going polymorphic – TH1 reasoning for Leo-III. In: Eiter, T., Sands, D., Sutcliffe, G., Voronkov, A. (eds.) IWIL@LPAR 2017 Workshop and LPAR-21 Short Presentations, Maun, Botswana, May 7–12, 2017. Kalpa Publications in Computing, vol. 1. EasyChair (2017), <http://www.easychair.org/publications/paper/346851>
45. Sutcliffe, G.: The CADE ATP System Competition – CASC. AI Magazine **37**(2), 99–101 (2016)
46. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)
47. Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP Typed First-order Form with Arithmetic. In: Bjørner, N., Voronkov, A. (eds.) Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. pp. 406–419. No. 7180 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2012)
48. Sutcliffe, G.: The 9th IJCAR automated theorem proving system competition – CASC-J9. AI Commun. **31**(6), 495–507 (2018). <https://doi.org/10.3233/AIC-180773>
49. Weber, T.: SMT solvers: new oracles for the HOL theorem prover. International Journal on Software Tools for Technology Transfer **13**(5), 419–429 (2011), <http://dx.doi.org/10.1007/s10009-011-0188-8>
50. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischniewski, P.: SPASS Version 3.5. In: Schmidt, R.A. (ed.) CADE. Lecture Notes in Computer Science, vol. 5663, pp. 140–145. Springer (2009)
51. Xu, Y., Liu, J., Chen, S., Zhong, X., He, X.: Contradiction Separation Based Dynamic Multi-clause Synergized Automated Deduction. Information Sciences **462**, 93–113 (2018)