Creating a Dataset of Problems for Learning in Dependent Higher-Order Logic

Daniel Ranalter and Cezary Kaliszyk

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
`{d.ranalter,cezary.kaliszyk}@gmail.com`

Introduction Although Automated Theorem Proving using Higher-Order Logic has produced several powerful systems, including well-known names like Vampire [6], E [11], and Zipperposition [1], incorporating dependent types into it poses a challenge. While there exist some interactive systems utilizing dependent types [5, 4, 2], in order to keep their type-checking decidable, they differentiate between judgmental and provable equality. PVS [10] is an exception to this but supports predicate subtyping and not dependent types in general.

Rothgang et al. introduced Dependently-Typed Higher-Order Logic [9] (DHOL) and developed a translation to HOL as a first step into making viable automated theorem provers for dependently typed logics. Niederhauser et al. then extended the theorem prover Lash [7] as the first instance of a native ATP system able to deal with problems of this kind. We plan to further this work and hope to make DHOL an attractive base for systems in the future.

To develop modern AI-guided provers for DHOL, we are creating a problem set sizable enough to support meaningful benchmarks and learning data. This will allow us to check implementations and compare them against others, providing fertile ground on which DHOL can grow.

DHOL DHOL is an extension of the Simple Type Theory (STT) as introduced by Church [3]. More precisely, we use a classical formulation of STT that includes a base type for booleans, primitive logical connectives for implication $\Rightarrow$, and typed (term-)equality $=_A$ as well as the constant function False $\bot$. Rothgang et al. created DHOL [9] by changing the simple function constructor $A \to B$ to the dependent version $\Pi x : A.B$ and allowing for the declaration and instantiation of dependent base types. The extension results in undecidable type-checking, as now the type equality judgment $\Gamma \vdash as_1 ... s_n \equiv at_1 ... t_n$ requires proofs for the equivalence (and well-typedness) of all s_i and t_i. The classical nature and coinciding equalities lead to a logic much closer to those of established ATPs. Type-checking needs only happen once, as all inference rules preserve the well-typedness of the problem. Indeed, Rothgang et al. [9] show for their translation that, if the original DHOL problem is well-typed, a valid proof of the translated problem can be tied to a valid proof in DHOL.

Extension of TPTP We use the TPTP syntax for our problems. In particular, TPTP's support of quantifying over higher-order typed variables by using the `!>[X:T]:` in the TH1 form will be reused as the binder for quantifying over dependent types by allowing which forms `T` might take. Part of this work is the specification of a proper TPTP form for DHOL, which is syntactically straightforward but requires consideration for things like type-checking and semantics. For example, Niederhauser et al. define two new SZS ontology statuses [12] related to type-checking in [7]. A DHOL problem is therefore in fact a (dependent) conjunction of two problems, as the proof of the conjecture itself is not meaningful without proof of the problem's well-typedness.

```
          Lemma L_R_neq n m (p : t n) (q : t m) :  L m p <> R n q.
                                    ≙
thf(l_r_neq, conjecture, ![M:nat, N:nat]:
   (![P:(fin @ N), Q:(fin @ M)]: ((l @ N @ M @ P) = (r @ M @ N @ Q)))).
```

Figure 1: The original Rocq problem next to the TPTP conjecture.

Benchmark Creation The creation of the problem set is ongoing. Currently, the set includes problems related to fixed-length lists/vectors, fixed-size matrices, and fixed-size sets. Especially problems of the last category were translated manually from dependently typed theories in the standard library of the Rocq prover [4]. In addition to the already mentioned categories of problems, there are also several examples for sanity-checking type-inference, type-constraint generation, and/or direct type-checking of a problem. These are artificial problems that do not state any mathematical concepts. For example, a problem might establish instantiations of generic dependent types as well as some corresponding terms to finally conjecture the equality of those terms. While those equalities are meaningless and indeed usually false this constitutes a minimal example of creating typing constraints between two sides of an equality.

The translation of problems between different proof systems is an interesting AI task and is of major value for the systems themselves as dependently-typed reasoning can then be shared and optimized independently of the systems. Our initial manual experiments can serve as a training ground for future automated experiments. Instances of such problems were important enough to find their way into standard libraries or were previously under active investigation. This elevates a benchmark set from a collection of toy examples to a more meaningful representation of what a system has to be able to achieve. Another advantage is the variety of problems that come with it. The same concept can be expressed in different ways by different systems, of which some differences might translate over into our representation. It should also yield more variety in difficulty and possibly even in domains compared to handcrafting them. As such, a way to automatically translate conjectures and axioms is desirable, increasingly so when the number of potential problems of one source is large. However, modern proof assistants usually provide several convenience features, such as implicit arguments and types that are inferred from the "real" arguments. TPTP takes a more verbose approach and lists all terms and types for a function explicitly. Figure 1 illustrates that translation from one to the other is not trivial because some of the types are missing. Discrepancies such as these make a straight-forward translation impossible and a sophisticated program will have to be developed to rise to the task. While core functionality, such as the mentioned inference of implicit arguments, can be shared, it will be necessary to have a separate parser for all considered systems. Currently, there are plans to look further into the Rocq theorem prover [4] and expand to the Mizar Mathematical Library[1], Agda [2], and Lean [5].

Future Work The creation of this benchmark will facilitate future developments in DHOL, its automation, and proof guidance for Lash$_{\text{DHOL}}$ [7]. Lash already includes a strategy language, and using strategy invention for DHOL will only be possible with a sufficiently large and diverse set of problems. Similarly, several of the techniques developed in the system have alternatives (for example, there are two ways how choice can be specified and implemented [8]), and choosing the right approaches will require experimenting with them. With this, it would also be possible to develop strategies that implement internal guidance based on machine learning. One of the

[1]http://mizar.org/library/

major weaknesses of the tableaux architectures is enumerative approach to universal quantifies and again a dataset would allow optimizing such term enumeration schemes.

References

[1] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirovic. Superposition for full higher-order logic. In André Platzer and Geoff Sutcliffe, editors, *Proc. 28th International Conference on Automated Deduction*, volume 12699 of *LNCS*, pages 396–412. Springer, 2021.

[2] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda – a functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proc. 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 73–78, 2009.

[3] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[4] Coq Development Team. *The Coq Reference Manual, Release 8.18.0*, 2023.

[5] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Proc. 28th International Conference on Automated Deduction*, volume 12699 of *LNAI*, pages 625–635, 2021.

[6] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[7] Johannes Niederhauser, Chad E. Brown, and Cezary Kaliszyk. Tableaux for automated reasoning in dependently-typed higher-order logic. Submitted, http://cl-informatik.uibk.ac.at/cek/submitted/jncbck.pdf.

[8] Daniel Ranalter, Chad E. Brown, and Cezary Kaliszyk. Experiments with choice in dependently-typed higher-order logic. Submitted, http://cl-informatik.uibk.ac.at/cek/submitted/drcbck.pdf.

[9] Colin Rothgang, Florian Rabe, and Christoph Benzmüller. Theorem proving in dependently-typed higher-order logic. In Brigitte Pientka and Cesare Tinelli, editors, *Proc. 29th International Conference on Automated Deduction*, volume 14132 of *LNAI*, pages 438–455, 2023.

[10] John Rushby, Sam Owre, and Natarajan Shankar. Subtypes for specifications: Predicate subtyping in PVS. *IEEE Transactions on Software Engineering*, 24(9):709–720, 1998.

[11] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2019.

[12] Geoff Sutcliffe. The SZS ontologies for automated reasoning software. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 5330 of *LNCS*, 2008.