Initial experiments on deriving a complete HOL simplification set

Cezary Kaliszyk and Thomas Sternagel
{cezary.kaliszyk,thomas.sternagel}@uibk.ac.at

University of Innsbruck, Innsbruck, Austria

Abstract

Rewriting is a common functionality in proof assistants, that allows to simplify theorems and goals. The set of equations to use in a rewrite step has to be manually specified, and therefore often includes rules which may lead to non-termination. Even in the case of termination another desirable property of a simplification set would be confluence. A well-known technique from rewriting to transform a terminating system into a terminating and confluent one is completion. But the sets of equations we find in the context of proof assistants are typically huge and most state-of-the-art completion tools only work on relatively small problems. In this paper we describe our initial experiments with the aim to close the gap and use rewriting to compute a complete first-order simplification set for a HOL-based proof assistant fully automatically.

1 Introduction

Proof assistants are computer programs that aid the user in building a proof that can be mechanically checked. A typical proof assistant includes a number of algorithms that perform common proof operations in an automated way. One of such mechanisms is rewriting and conditional rewriting (often called *simplification* in the context of formal proof). There are two ways of performing rewriting in a proof assistant, depending on the working style: rewriting provided as a forward derivation rule that lets one rewrite a theorem using (possibly conditional) equalities and rewriting as a tactic, that uses equations to rewrite the current goal to an equivalent goal.

To perform a rewriting step, the user needs to choose the equations to rewrite with. Typically choosing this set is done completely manually. In certain cases, however, there exist defaults (for example Isabelle [17] simplification set) or lists of theorems to use (for example `ARITH` in HOL Light [8] or hint databases in Coq [1]). Such default sets are also defined manually, and developers try hard to avoid creating simplification sets that are non-terminating. Unfortunately this problem is quite hard, and for example the usual simplification set defined for many theories in Isabelle includes rules that lead to non-termination of the `simp` tactic.

In order to obtain an even stronger proof technique with the help of rewriting, one can consider normal forms of expressions with relation to some theory. Completion of rewriting is a technique that lets us derive rewrite systems that follow given sets of equations, but are terminating and confluent. A terminating and confluent rewrite system for a theory would give a complete decision procedure for establishing equalities in this theory, giving a strong proof technique.

Termination of term rewrite systems (TRSs) is an undecidable property. Nevertheless a vast number of methods have been developed to determine termination, many of which are suitable for implementation. For example the tools $\mathsf{T_TT_2}$ [12] or AProVE [6] implement techniques for automatically proving the termination of a first-order rewrite system. The termination methods implemented by these systems are sound, but the tools may produce unsound proofs (for example due to implementation bugs). Therefore a number of proof certification techniques have been developed, for example CeTA [22] which can certify termination (or non-termination) proofs provided by a termination tool.

Likewise confluence of TRSs is undecidable in general. In the presence of termination, however, confluence and local confluence coincide according to Newman's Lemma [16]. Based on this information most automatic completion tools follow a common strategy: They maintain termination of an oriented version of an initial set of equations and try to make it locally confluent by means of deducing and adding new consequences. If this succeeds, at some point all critical pairs will be joinable and the tool yields a terminating and confluent term rewrite system, which has the same equational theory as the initial set of equations. There is very little work that bridges the gap between proof assistants and termination, the most notable being [13].

The aim of this paper is to use rewriting techniques to automatically derive a terminating and confluent first-order rewrite system for a proof assistant. In this research we considered HOL Light [8] and its Multivariate [9] library, as together with the Flyspeck project (developing a formal proof of the Kepler conjecture [7]) they form a large library of mathematical knowledge. The completion tool we used is KBCV [21], which also features an interactive mode and the generated completion proves may be certified by CeTA.

In the setting of proof assistants we work on higher-order terms with types whereas in the rewrite community most tools work on first-order term rewriting systems. A commonly used input format to termination as well as completion tools is the TPDB format[1] and its XML-based successor.[2]

We have implemented a translation mechanism from HOL Light to the TPDB format in order to give the theorems present in HOL Light/Multivariate to the available rewriting tools. To give the computed results back to HOL Light we implemented the converse translation from the TPDB format to typed λ-terms. We proposed an interaction model between HOL Light and KBCV and did initial experiments on deriving new theorems from the critical pairs. The initial set of 3,267 orientable equations has been passed in one go to KBCV and 305 thousand critical pairs were found which gave rise to 167 thousand HOL equations.

The rest of this paper is organized as follows. In Section 2 we describe how the theorems of HOL Light may be translated to first-order equations in the TPDB format. Following this we briefly recall completion in Section 3. Our main idea — the interaction between HOL Light and KBCV — is presented in Section 4. Next we present our experiments in Section 5. Finally we conclude in Section 6.

2 Translation from logic to rewriting

The first issue in implementing a translation mechanism from theorem statements to rewriting is to choose which theorem statements can be used in a rewriting setting. Modern completion tools only support unconditional equations, so we choose to work only with unconditional orientable equations that can be encoded in a first-order format.

We start with all the theorems available in HOL Light/Multivariate. To obtain a list of all these, we use the `update_database` functionality of HOL Light, which can produce a list of name–theorem pairs accessible from the top level by analyzing OCaml's internal data structures. We proceed by eliminating repetitions (theorems that have the same statement, but have been

[1]`https://www.lri.fr/~marche/tpdb/format.html`

[2]`http://www.termination-portal.org/wiki/XTC_Format_Specification`

assigned different names), and selecting only the theorems that have the form of unconditional equations (possibly universally quantified).

The two most common approaches for eliminating λ-expressions from higher-order logic formulas are to use λ-lifting or to encode them as combinators. Such approaches are often used to translate HOL to first-order logic [15, 10]. Lambda lifting does not seem to be possible, as in rewriting we are not able to quantify inside a term. As the theory of SKI combinators is equational, it might be possible to obtain a translation of λ-expressions to combinators. In our first experiments we chose to translate the λ-expressions which can be expressed as rewrite rules using simple transformations (β-reduce all redexes in the theorem statements and η-expand equalities that have a λ-expression on one of the sides using extensionality) and to ignore the rest of the equations.

Next we define the weight function, as a partial function on terms that returns polynomials over variables (we will ignore the equations for which the function is undefined). The function that we present ignores the types of the terms completely. In case of type-aware encodings the function would need to include the type variables in the resulting polynomial.

Definition 2.1 (weight of a higher-order logic term).

$$w(t) := \begin{cases} 1 & \text{if } t \text{ is a constant} \\ x & \text{if } t \text{ is a variable } x \\ w(l) + w(r) & \text{if } t = l\,r \\ \textit{undefined} & \text{if } t = \lambda y.s \end{cases}$$

Given such a weight function, we can filter the orientable equations. We consider a HOL theorem as an orientable equation, if after specializing all the top level universally quantified variables the weights of the left- and right-hand sides of the equation are defined and one of them is strictly greater than the other in the usual polynomial order sense. Not all equations are orientable, for example the usual associativity or commutativity theorems have the same polynomials returned by w for both sides, so they are not orientable.

In order to eliminate the higher-order applications, we use the `apply` functor. We employ the algorithm introduced by Meng and Paulson [15]. For each higher-order constant c we compute the minimum arity n_c with which it appears in a problem, and the first n_c arguments are passed to c directly. If the constant is also used with more arguments in the problem, `apply` is used. Blanchette [4, p. 105–106] gives simple examples when this encoding introduces incompleteness in the encoding to ATP formats. Due to the lack of general quantifiers, however, this works quite well for rewriting.

We can now proceed to encode the equations in the TPDB format. A file in this format starts with a number of variable declarations, followed by a number of equations. To synchronize the symbols appearing in the theorems, the TPDB export declares the signature of the constants, functions and variables (for polymorphic constants or functions only one symbol for all occurrences; this could be strengthened with monomorphisation). The variables present in all the equations are written to the file, followed by the equations oriented in the direction implied by the weights. In order to verify our implementation of the polynomial ordering we proved the following theorem.

Theorem 2.1. *The theorems of HOL Light/Multivariate orientable by $w(t)$ and translated to first-order rewriting form a terminating TRS.*

Proof. We have used $\mathsf{T_TT_2}$ to find a proof that the system is terminating. The automatic strategy is quite slow. Limiting $\mathsf{T_TT_2}$ to polynomial interpretations (matrix interpretations of dimension

$$\textsc{Deduce}\ \frac{(\mathcal{E},\mathcal{R})}{(\mathcal{E}\cup\{s\approx t\},\mathcal{R})}\ \text{if } s\ {}_{\mathcal{R}}\!\leftarrow u \to_{\mathcal{R}} t \qquad \textsc{Orient}\ \frac{(\mathcal{E}\cup\{s\mathrel{\dot{\approx}}t\},\mathcal{R})}{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}\ \text{if } s>t$$

$$\textsc{Compose}\ \frac{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}{(\mathcal{E},\mathcal{R}\cup\{s\to u\})}\ \text{if } t\to_{\mathcal{R}} u \qquad \textsc{Delete}\ \frac{(\mathcal{E}\cup\{s\approx s\},\mathcal{R})}{(\mathcal{E},\mathcal{R})}$$

$$\textsc{Collapse}\ \frac{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}{(\mathcal{E}\cup\{u\approx t\},\mathcal{R})}\ \text{if } s\ \overset{\sqsupset}{\to}_{\mathcal{R}}\ u \qquad \textsc{Simplify}\ \frac{(\mathcal{E}\cup\{s\mathrel{\dot{\approx}}t\},\mathcal{R})}{(\mathcal{E}\cup\{u\mathrel{\dot{\approx}}t\},\mathcal{R})}\ \text{if } s\to_{\mathcal{R}} u$$

Figure 1: The inference rules of *completion.*

1) over the naturals, however, is able to find a proof in a reasonable time (about 25 minutes). In particular the SMT prover invoked by TTT2 does not find a satisfiable assignment for a bit-width of 5, but finds one for a bit-width of 6. Parsing the system (consisting of a few thousand rewrite rules) takes the biggest part of the running time of TTT2. We used CeTA to certify that the proof found by TTT2 is indeed a valid termination proof of the system. CeTA requires about 10 minutes to check the proof. □

We have also implemented a combinator parser, which is able to read files generated by termination and confluence tools and output HOL Light preterms. When reading the TPDB file, applications give rise to `Combp` preterms and constants or variables give rise to `Varp` preterms. When exporting the HOL theorems as a TPDB file, we have declared a signature which is used to map the TPDB concepts (names of functions, constants, variables) back to their HOL counterparts. Such preterms can later be type-checked by the standard HOL Light term parser. Due to an encoding that does not preserve types, some of the preterms may fail to type-check (and as we will see in Section 5, some will fail). The rewrites performed on the KBCV side are not type-valid in the HOL setting, therefore such equations do not give rise to valid HOL critical pairs and can be forgotten.

Given a terminating rewrite system, we can proceed to completing the system.

3 Completion

We briefly recall the basics of completion. See for example [2] for a comprehensive introduction to completion and term rewriting.

Completion is a procedure which takes as input a (finite) set of equations $\mathcal{E}$ and optionally a reduction order $>$ (older tools need the reduction order in advance whereas modern tools try to construct the reduction order dynamically with the help of external termination tools, see [23]) and attempts to construct a terminating and confluent TRS $\mathcal{R}$ with the same equational theory as $\mathcal{E}$. Provided the completion procedure succeeds, two terms are equivalent with respect to $\mathcal{E}$ if and only if they reduce to the same normal form with respect to $\mathcal{R}$, that is, $\mathcal{R}$ represents a decision procedure for the word problem of $\mathcal{E}$.

The procedure generates a finite sequence of intermediate TRSs which constitute approximations of the equational theory of $\mathcal{E}$. Following Bachmair and Dershowitz [3] the completion procedure may be modeled as a system of inference rules (see Figure 1). These inference rules work on pairs $(\mathcal{E},\mathcal{R})$ where $\mathcal{E}$ constitutes a finite set of equations and $\mathcal{R}$ is a finite set of rewrite rules. The goal of the procedure is to transform an initial pair $(\mathcal{E},\varnothing)$ into a pair $(\varnothing,\mathcal{R})$ such that $\mathcal{R}$ is terminating, confluent and equivalent to $\mathcal{E}$. A completion procedure based on these rules may either succeed (find $\mathcal{R}$ after finitely many steps), loop indefinitely, or fail. In Figure 1

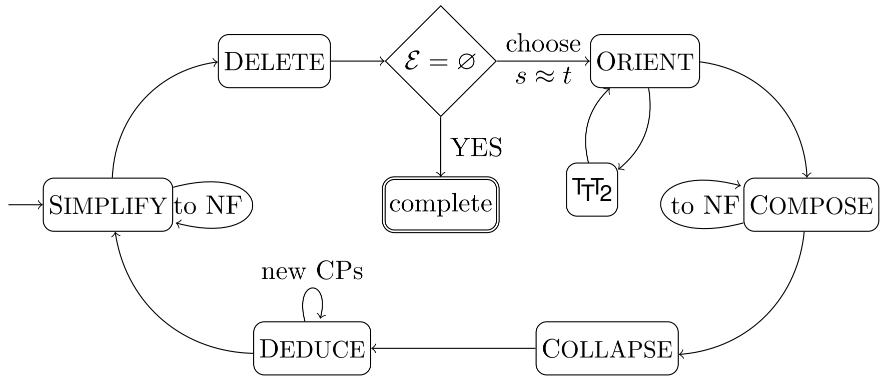

Figure 2: KBCV's automatic completion procedure.

a reduction order $>$ is provided as part of the input (whereas most modern completion tools construct this ordering on the fly as described above). We write $s \overset{\sqsupset}{\rightarrow}_{\mathcal{R}} u$ to express that s is reduced by a rule $\ell \to r \in \mathcal{R}$ such that ℓ cannot be reduced by $s \to t$. The notation $s \dot{\approx} t$ denotes either of $s \approx t$ and $t \approx s$.

In order to make the simplification set more complete we want to use an automatic completion tool. Most modern completion tools use some variant of the above inference system and then commit to a specific strategy to implement a completion procedure. There are several such tools available today, e.g., Slothrop [23], MKBTT [18], Maxcomp [11], and KBCV [21]. One of the main issues was that most of these tools have not been designed to handle problems with a magnitude counting thousands of equations. In the end we decided to use KBCV for several reasons:

- It has both an automatic and an interactive mode, where the completion inference rules may be applied freely.
- It records a history [19] of how rules were applied, which is useful to reprove the corresponding equations in HOL Light.
- Its parser is fast enough to load the thousands of equations of the problem at hand in reasonable time.
- One of the authors is the main developer of KBCV so it was relatively easy to adapt the tool and optimize it [20].

Tools like KBCV typically work on small problems, e.g., those which can be found in the TPDB problem database.[3] When given a problem KBCV tries to complete it by issuing the inference rules of completion in the order depicted in Figure 2.

First SIMPLIFY is used on all equations as long as a normal form with respect to the current TRS $\mathcal{R}$ is reached. Next all trivial equations, i.e., equations where the left- and right-hand sides are the same, are deleted. Now the tool checks whether $\mathcal{E}$ is empty, if this is the case $\mathcal{R}$ is complete and the procedure finishes. Otherwise KBCV chooses an equation which it will try to orient. The heuristic here is to select an equation with minimal left- and right-hand sides. The depicted procedure actually runs in two threads in parallel. The first of those always tries to orient equations from left to right and only if this does not succeed the other way round. The

[3]`http://termination-portal.org/wiki/TPDB`

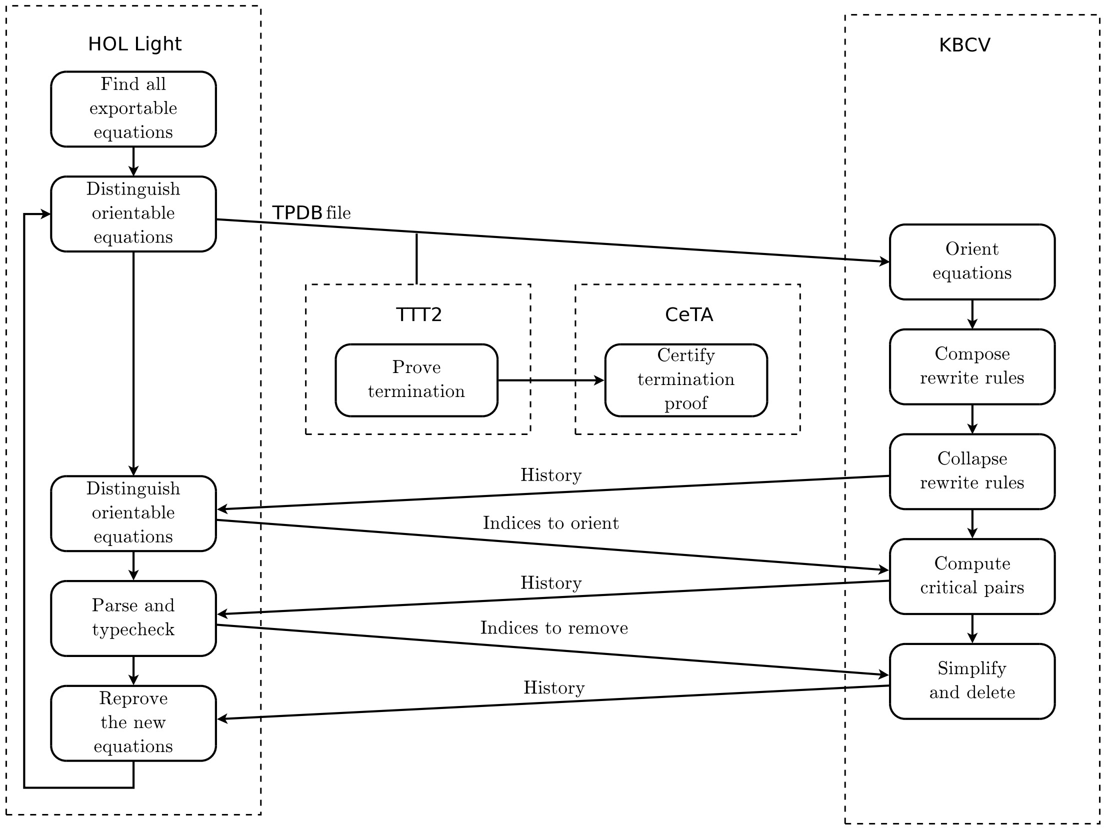

Figure 3: Control and data flow between HOL Light, KBCV, and external termination tools.

second thread behaves dually. KBCV implements a fast version of a lexicographic path ordering to orient equations. Only if this does not succeed it gives the problem to $\mathsf{T_TT_2}$ to establish a terminating system which also comprises the newly oriented equation. To keep the resulting system as small as possible KBCV now applies COMPOSE until all right-hand sides of rules are in normal form with respect to the current TRS $\mathcal{R}$. Next it also simplifies the left-hand sides of rules using COLLAPSE. Finally DEDUCE computes critical pairs between left-hand sides of rules and adds those to the set of equations.

4 Interaction between HOL Light and KBCV

In this section we describe how the steps performed in the completion procedure (described in the previous section) correspond to operations in HOL. For this we run KBCV in its interactive mode. The interaction is depicted schematically in Fig. 3.

Exporting the HOL Light simplification set. In Section 2 we have already described the export of HOL Light equations to the TPDB format. We export all the equations that can be written in the first-order rewriting format. We separate the orientable ones from the non-orientable ones but write the latter as well (as they may become orientable after simplification).

Importing the rewrite system to KBCV. KBCV can directly read the TPDB file. We issue the `orient` command to orient the part of the system which could be oriented in HOL. Because we already know that our chosen orientation will be terminating we set KBCV to use an external termination tool that will always output "YES".

Composing the rewrite rules. Given that the right-hand side of a rewrite rule can be simplified with another rule the same operation can be performed in HOL assuming that the types are correct. This means that we will later have to check that the derived rules correspond to provable equations in HOL Light.

Collapse of rewrite rules. This operation works in a similar way as compose, only that it produces equations which we will have to give back to HOL Light to orient and prove terminating.

Deducing critical pairs. A critical pair is a derivable equation which arises from the overlap of left-hand sides of two rules. Some of those pairs found by KBCV will not be well-typed. To improve the performance of SIMPLIFY we can remove such ill-typed pairs by parsing and type-checking them in HOL Light. In our experiments we decided to use type-checking rather than encoding of types in terms for two reasons: First, an ill-typed pair may give rise to a well-typed equation using unification. For example an equation where we have $list(\alpha)$ on one side and $list(num)$ on the other side will give a well-typed equation by instantiating α to num. Second, by throwing away not well-typed pairs we may remove equations that are needed to preserve confluence. We have already discussed at the end of Section 2, why such pairs can be forgotten. completion procedure.

Simplify and Delete. Simplify rewrites the left- and right-hand sides of equations to normal form in order to eliminate joinable critical pairs. At this point the information about newly obtained equations to orient can be send back to HOL together with their recorded history which allows to reprove them.

Now we have a somehow more complete approximation of the initial theorems in HOL Light for which we want to repeat the loop until we arrive at a complete system.

5 Experiments

We did our experiments with HOL Light revision 153 from December 2012 and Flyspeck revision 3130 from March 2013. The experiments were performed on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU. HOL Light uses only one process, whereas KBCV uses threads to exploit all the available cores.

The number of all theorems as obtained using the `update_database` mechanism is 17,807. After removing repetitions and considering only the universally quantified equations there are 6,273 theorems. Our weight function returns a defined value for 4,186 theorems. Our reduction order divides these in two parts: the 3,267 equations that are orientable and 919 that are not.

To verify our heuristic order, we used $\mathsf{T_TT_2}$. We have exported the orientable equations in the TPDB format and having used $\mathsf{T_TT_2}$ with polynomial interpretations, we have found a proof. The proof is 2.6MB in size and we have used CeTA to certify it.

We next proceeded by loading the TPDB file in KBCV. Since we already have established termination of these rules beforehand orienting them only takes a view seconds. Deducing all critical pairs between the left-hand sides of the oriented equations proved to be the first hurdle for KBCV. The previous version of KBCV did not finish the computation of critical pairs in a few days. To speed it up we introduced parallelization. Now KBCV is able to compute all 305,708 critical pairs in about two hours.

The critical pairs can be exported together with their history. As described in Section 2, we have implemented a combinator parser to read back the KBCV history output as HOL Light terms. Only the well-typed ones can be properly parsed. Trying to parse the terms takes about 4 hours and 137,888 of the newly generated equations are ill-typed. The indices of the ill-typed equations are passed back to KBCV, which can in turn remove them.

The next bottleneck was SIMPLIFY. Using the previous version of KBCV, simplification of this big number of equations was infeasible. With the help of caching techniques and parallelization we are able to delete the joinable critical pairs; this, however, takes a few days. The remaining step to close the cycle is to reprove the equations obtained using rewriting techniques in HOL Light. With the history of performed simplifications, this is straightforward but has not been done yet. So far we have performed only one iteration, so the TRS is not a confluent one.

Two example critical pairs found by KBCV are (the numbers are the internal indices used to reference equations in KBCV, e.g., 309551 is a consequence of theorems 49 and 882 from the initial set):

```
309551 : i(i(realu_lt(), i(i(realu_add(), x), y)), i(realu_ofu_num(), u_0())) =
  i(i(realu_lt(), x), i(realu_neg(), y)) : 309551 : 49, 882,
```

which corresponds to the equation $x + y < 0 \Longleftrightarrow x < -y$ and

```
309569 : i(i(realu_lt(), i(i(realu_add(), x), y)), i(Re(), ii())) =
  i(i(realu_lt(), x), i(realu_neg(), y)) : 309569 : 139, 882,
```

which corresponds to $x + y < \mathsf{Re}(i) \Longleftrightarrow x < -y$. The latter of the two theorems will be simplified using the rule that rewrites $\mathsf{Re}(i)$ to 0 and will be discarded.

6 Conclusion

This paper presents initial experiments in automatically deriving a terminating and confluent simplification set for HOL Light using tools coming from termination and completion research. We started with all the (unconditional) equations present in HOL Light/Multivariate and using a manually defined order we proved termination for a large subset of the rules. We have presented a possible loop for deriving confluence and we have done some experiments with the first loop of the confluence derivation.

The simplification set that we derive, includes a number of equations translated to first-order logic. Because higher-order matching is used for rewriting in most proof assistants, the properties of the simplification set (like termination or confluence) derived for first-order translations do not immediately give rise to the same properties for the original rules. There are at least two ways to proceed in order to preserve the termination and confluence for the obtained HOL simplification set:

- Use first-order rewriting in the proof assistant;
- Further restrict the initial simplification set to the first-order theorems.

In the future, the first thing we intend to do is to automatically prove the equations derived by KBCV in HOL. Thanks to recording completion we know the equations used to derive the new ones, so we have all the necessary components to use the existing HOL Light decision procedures.

This will complete the cycle presented in Section 4 and will allow executing more iterations of the completion procedure. Further, the remaining efficiency bottlenecks (on both HOL Light and KBCV sides) need to be taken care of to make the procedure practical.

The approach that we presented is applicable not only to the large HOL theorem set, but also to an arbitrary subset of it. For example, one might consider rewrite rules concerning one specific domain of mathematics. In case completion for the whole set turns out to be unachievable (for such a big set it might not be possible to iterate the loop until the result is stable), the approach can be applied to the particular area, automatically deriving a decision procedure.

We intend to try out different encodings to rewriting, that would take types into account. Certain approaches presented for example in [5] could be directly applicable. Next, extensions to rewriting, like higher-order rewriting [14], conditional rewriting, or AC rewriting can be considered. Finally, we intend to investigate, how useful the automatically derived simplification set is for proving real HOL Light problems.

References

[1] B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J.C. Filliâtre, E. Giménez, H. Herbelin, et al. *The Coq Proof Assistant Reference Manual Version 8.4*. LogiCal project, 2012. `http://coq.inria.fr/refman/`.

[2] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[3] Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of the ACM*, 41:236–276, 1994.

[4] Jasmin Christian Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2012. `http://www21.in.tum.de/~blanchet/phdthesis.pdf`.

[5] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding Monomorphic and Polymorphic Types. In *TACAS*, 2013. To appear, `http://www21.in.tum.de/~blanchet/enc_types_paper.pdf`.

[6] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated termination proofs with AProVE. In Vincent van Oostrom, editor, *RTA*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220. Springer, 2004.

[7] Thomas C. Hales. Introduction to the Flyspeck project. In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[8] John Harrison. HOL Light: A tutorial introduction. In Mandayam K. Srivas and Albert John Camilleri, editors, *FMCAD*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996.

[9] John Harrison. The HOL Light theory of euclidean space. *J. Autom. Reasoning*, 50(2):173–190, 2013.

[10] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *CoRR*, abs/1211.7012, 2012.

[11] Dominik Klein and Nao Hirokawa. Maximal completion. In Manfred Schmidt-Schauß, editor, *Proc. of the 22nd International Conference on Rewriting Techniques and Applications (RTA 2011)*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 71–80. Schloss Dagstuhl, Dagstuhl, 2011.

[12] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In Ralf Treinen, editor, *RTA*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer, 2009.

[13] Alexander Krauss, Christian Sternagel, René Thiemann, Carsten Fuhs, and Jürgen Giesl. Termination of Isabelle functions via termination of rewriting. In Marko C. J. D. van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *ITP*, volume 6898 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2011.

[14] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theor. Comput. Sci.*, 192(1):3–29, 1998.

[15] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.

[16] Maxwell H. A. Newman. On theories with a combinatorial definition on "equivalence". In *The Annals of Mathematics*, pages 223–243. Annals of Mathematics, 1942.

[17] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[18] Haruhiko Sato, Sarah Winkler, Masahito Kurihara, and Aart Middeldorp. Multi-completion with termination tools (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proc. of the 4th International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 306–312. Springer, 2008.

[19] Thomas Sternagel. Automatic proofs in equational logic. Master's thesis, University of Innsbruck, 2012. `http://cl-informatik.uibk.ac.at/teaching/master/theses/TS.pdf`.

[20] Thomas Sternagel. KBCV 2.0 – Automatic Completion Experiments, 2013. Accepted for publication at IWC 2013, `http://cl-informatik.uibk.ac.at/users/csag9384/publications/iwc_2013_paper.pdf`.

[21] Thomas Sternagel and Harald Zankl. KBCV - Knuth-Bendix completion visualizer. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 530–536. Springer, 2012. `http://cl-informatik.uibk.ac.at/users/csag9384/publications/ST12_02.pdf`.

[22] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.

[23] Ian Wehrman, Aaron Stump, and Edwin Westbrook. Slothrop: Knuth-Bendix completion with a modern termination checker. In *Proc. of the 17th International Conference on Rewriting Techniques and Applications (RTA 2006)*, pages 287–296. Springer, 2006.