Conditional Simplification

Mihails Milehins

September 9, 2026

Abstract

The document presents a reference manual for the framework *Conditional Simplification* (CS): a collection of experimental general-purpose methods for the object logic *Isabelle/HOL* (e.g., see [1]) of the formal proof assistant *Isabelle* [4]. The methods that are provided in the collection offer the functionality that is similar to certain aspects of the functionality provided by the standard *proof methods* [8] of Isabelle that combine classical reasoning and rewriting/simplification, but use a different approach for rewriting/simplification. More specifically, the methods provided in the collection allow for the side conditions of the rewrite rules to be solved via intro-resolution.

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries.

Furthermore, the author would like to acknowledge the positive impact of [6] and [7] on his ability to code in Isabelle/ML [2, 7]. Moreover, the author would like to acknowledge the positive role that numerous Q&A posted on the Stack Exchange network (especially Stack Overflow and TeX Stack Exchange) played in the development of this work.

The author would also like to express gratitude to all members of his family and friends for their continuous support.

Contents

1 Introduction **5**
1.1 Background 5
1.2 Purpose and scope 5
1.3 Related and previous work 5

2 Syntax **6**

3 Known issues and limitations **8**

References **9**

1 Introduction

1.1 Background

This document presents a reference manual for the framework CS. The framework CS is a collection of experimental tactics and associated proof methods aimed at the automation of conditional simplification in the object logic Isabelle/HOL of the formal proof assistant Isabelle. The methods that are provided in the collection offer the functionality that is similar to certain aspects of the functionality provided by the standard proof methods of Isabelle that combine classical reasoning and simplification (e.g., the method *auto* [3, 8]), but there are notable differences. More specifically, the methods provided in the collection allow for the side conditions of the rewrite rules to be solved via intro-resolution.

1.2 Purpose and scope

The primary functionality of the framework is available via the proof methods *cs-concl-step*, *cs-prems-atom-step* and *cs-intro-step*. The methods *cs-concl-step* and *cs-prems-atom-step* accept a collection of (conditional) rewrite rules and execute one rewrite step on the conclusion or a premise of some goal, respectively. The application of the rewrite step produces new goals that are associated with the premises of the rewrite rules. These goals are meant to be discharged via a recursive application of either *cs-intro-step* or *cs-concl-step*. The procedure outlined above was automated and made available as part of the proof methods *cs-concl* and *cs-prems*.

1.3 Related and previous work

No claim with regard to the originality of the algorithms used in the methods implemented as part of the CS is made and, due to the experimental and evolving nature of this work, a comprehensive literature review is considered to be outside its scope. Therefore, the only contributions claimed by the author are the implementation of the algorithms associated with the methods provided as part of the CS in *Isabelle/ML* [2, 7] and their integration with the Isabelle/Isar infrastructure.

The implementation of the methods associated with the framework builds upon the existing infrastructure of Isabelle and provides only a very thin layer of additional or alternative functionality. As such, it may be possible to achieve integration of the functionality offered by the CS with the standard infrastructure for classical reasoning and simplification in Isabelle.

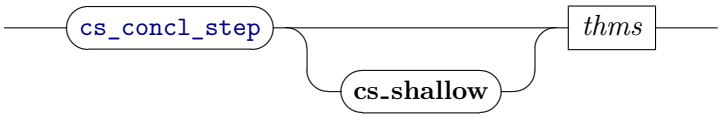
It should also be mentioned that the Isabelle/ML code from the main distribution of Isabelle2020 and from *The Isabelle/ML Cookbook* [6] was frequently reused (with amendments) during the development of the library. Some particular examples of such reuse include

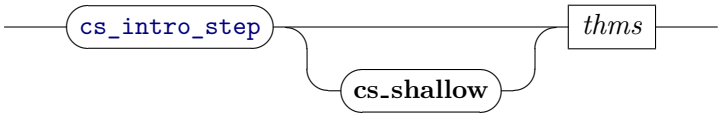
- The adoption of the code for the tactic *remdups-tac* from the file *~/Tools/Intuitionistic.ML*.
- The adoption of the code presented in subsection 3.3 of [6] for higher-order matching and unification.

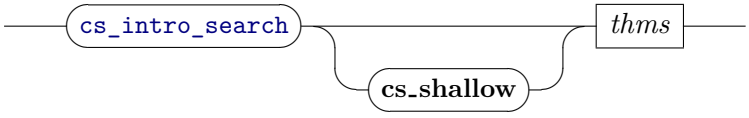
2 Syntax

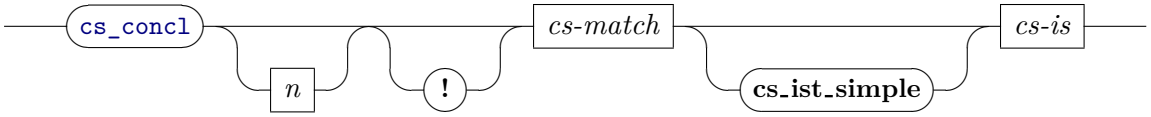
This section presents the syntactic categories that are associated with the methods *cs-concl-step*, *cs-intro-step*, *cs-intro-search*, *cs-concl*, *cs-prems-atom-step* and *cs-prems*. It is important to note that the presentation is only approximate.

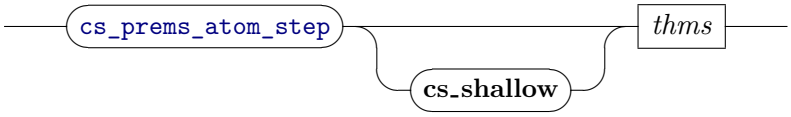
cs-concl-step : *method*
cs-intro-step : *method*
cs-intro-search : *method*
cs-concl : *method*
cs-prems-atom-step : *method*
cs-prems : *method*

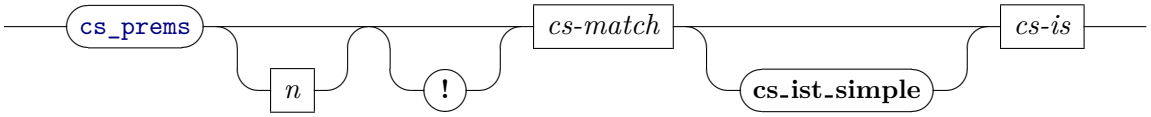




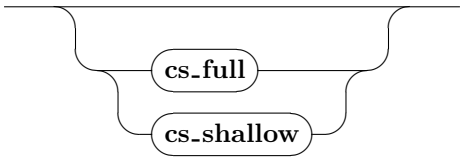




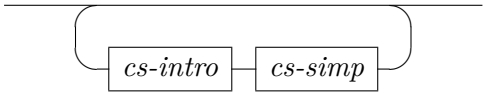




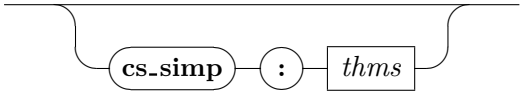
cs-match



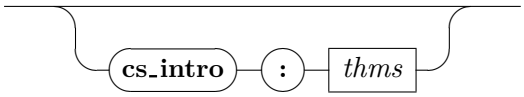
cs-is



cs-simp



cs-intro



cs-concl-step (**cs-shallow**) *thms* performs a single rewrite step of the conclusion of some goal using the collection of the rewrite rules *thms*. The rewriting is performed via the intro-resolution with the rewrite rule stated in an altered form: the application of *cs-concl-step* may produce new subgoals that are associated with the premises of the applied rewrite rule. If the optional argument **cs-shallow** is provided during the invocation of the proof method, then backtracking and all of the related infrastructure is disabled during the invocation of the method (disabling the infrastructure associated with backtracking can result in improved performance).

cs-intro-step (**cs-shallow**) *thms* performs a single refinement step via intro-resolution. The optional argument **cs-shallow** serves a purpose that is similar to its purpose in *cs-concl-step*.

cs-intro-search (**cs-shallow**) *thms* attempts to solve a single goal using a search procedure based on the algorithm outlined in the description of the method *cs-intro-step*. The optional argument **cs-shallow** serves a purpose that is similar to its purpose in *cs-concl-step*.

cs-concl (*n*) (!) (**cs-full** or **cs-shallow**) (**cs-ist-simple**) **cs-simp** : *simp-thms* **cs-intro** : *intro-thms* attempts to solve a single goal using a search procedure that employs the method applications *cs-concl-step simp-thms* and *cs-intro-step intro-thms* as individual steps. If the optional argument **cs-full** is provided during the invocation of the proof method, all possible rule-term matches are considered. Otherwise, only a single sensible default match is used for every applicable rule-term pair. As before, if the optional argument **cs-shallow** is provided during the invocation of the proof method, then backtracking and all of the related infrastructure is disabled. If the optional argument **cs-ist-simple** is provided, then the search space of the method is expanded by allowing backtracking after every atomic step (the default behavior uses a tailor-made empirically established routine that can be inferred from the implementation of the method). The optional positive integer argument *n* can be used for the invocation of a built-in profiling tool: *n* represents the number of trial runs of the method during profiling. The optional argument ! switches on the verbose mode. In this mode, the individual steps that are invoked during the search procedure associated with the method are printed.

cs-prems-atom-step (**cs-shallow**) *thms* performs a single rewrite step of the first premise of some goal using the collection of the rewrite rules *thms*. The optional argument **cs-shallow** serves a purpose that is similar to its purpose in *cs-concl-step*.

cs-prems (*n*) (!) (**cs-full** or **cs-shallow**) (**cs-ist-simple**) **cs-simp** : *simp-thms* **cs-intro** : *intro-thms* repeatedly performs a single rewrite step of the first premise of some goal using the collection of the rewrite rules *simp-thms*, followed by an attempt to solve all but the final subgoal using the method application (*cs-concl* **cs-simp** : *simp-thms* **cs-intro** : *intro-thms*). The optional arguments *n*, !, **cs-full**, **cs-shallow** and **cs-ist-simple** serve a purpose that is similar to their purpose in *cs-concl*.

3 Known issues and limitations

The collection of the proof methods that are associated with the framework CS is a result of experimentation during practical formalization work. The CS should be viewed as an idea or a proposal for further development, rather than a finished product. The limitations and the performance of the methods associated with the CS have not been investigated and there is little guarantee that they will be suitable for any specific target application. It is also important to note that the methods have only been tested extensively on the subgoals that do not contain any explicit occurrences of the *Isabelle/Pure* [5] universal quantifier. Only very limited and highly experimental support for the first-/higher-order reasoning is provided by the CS.

References

[1] O. Kunčar and A. Popescu. Comprehending Isabelle/HOL's Consistency. In H. Yang, editor, *Programming Languages and Systems*, volume 10201, pages 724–749. Springer, Heidelberg, 2017. ISBN 978-3-662-54433-4.

[2] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (revised)*. MIT Press, Cambridge, Massachusetts, 1997. ISBN 978-0-262-63181-5.

[3] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer Science & Business Media, Heidelberg, 2002. ISBN 978-3-540-43376-7.

[4] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.

[5] L. C. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[6] C. Urban. *The Isabelle Cookbook: A Gentle Tutorial for Programming Isabelle/ML*. 2019.

[7] M. Wenzel. The Isabelle/Isar Implementation. 2019.

[8] M. Wenzel. The Isabelle/Isar Reference Manual. 2019.