Research Proposal:

Automating Coherent Logic

Marc Bezem* Thierry Coquand† Arild Waaler‡

June 1, 2006

Abstract

We propose to build an automated reasoning system for first-order logic (FOL) by translating reasoning problems to a fragment of FOL called coherent logic (CL) and then solving them by a fast problem solver for CL. Both the translation and the fast CL solver are to be developed in the project. Automated reasoning in FOL is an established research field with many applications in mathematics, logic and computer science (e.g., the verification of hard- and software, deductive databases, semantic web applications etc.) The advantages of the use of CL instead of weaker logics (for example, resolution logic) are: strong support for interactive problem solving and for explanation facilities, which is of crucial importance for many applications. Moreover, some inefficiencies inherent to the translation of FOL to weaker logics than CL are avoided.

1 Introduction to coherent logic

Coherent Logic (CL) is the fragment of first-order logic (FOL) with formulas

$$\forall \vec{x}\ (A_1 \wedge \cdots \wedge A_n \rightarrow \exists \vec{y}_1\ C_1 \vee \cdots \vee \exists \vec{y}_m\ C_m)$$

where the A_i are atoms and the C_j are conjunctions of atoms. The strong point of CL is that it has both quantifiers $\forall, \exists$ together with a simple proof procedure, forward ground reasoning, which is nevertheless complete. Due to this expressive power, many reasoning problems can be formulated directly in CL, whereas in less expressive logics (for example, resolution logic) they would have to be encoded. This has the following important advantages.

- The *interaction* between the automated reasoning system and the user greatly benefits from the fact that both work on the same problem.
- The *explanation* of solutions found by the system is facilitated. Furthermore, proof objects can be generated and these can be verified by and imported in other systems.
- Inefficiencies inherent to certain encodings are avoided by a direct formulation in CL.

*Department of Computer Science, Bergen University, Postboks 7800, N-5020 Bergen, Norway, bezem@ii.uib.no

†Department of Computer Science, Chalmers University of Technology and Gothenburg University, SE-412 96 Göteborg, Sweden, coquand@cs.chalmers.se

‡Department of Informatics, University of Oslo, Postboks 1080 Blindern, N-0316 Oslo, Norway, arild@ifi.uio.no

CL goes back to the Norwegian Thoralf Skolem, who developed CL in 1920 to obtain meta-mathematical results in lattice theory and projective geometry [30]. The bottom-up proof procedure used today in deductive databases and the system SATCHMO [25] (for logics weaker than CL), can already be found in [30]. Only recently CL has been rediscovered: see [14] for its relevance to computer science and [27] for one example of its interesting proof-theoretic properties. Reasoning in CL is constructive and can be used for, e.g., the constructivization of classical abstract algebra, see [21, 20].

CL extends Horn clause logic (on which the programming language Prolog [23] is based) and resolution logic [29] in that we may have a whole disjunction of existentially quantified conjunctions of atoms in the conclusion. The point of CL is that coherent formulas can be used as generating rules, which forms a complete proof procedure for this fragment. Though possible, the reduction to resolution clauses is unattractive for two reasons: (1) the Skolemization of the existential variables is not logically equivalent and spoils their dependence on the premiss; (2) Skolem functions complicate the proof objects in a serious way; (3) the disjunction of conjunctions leads in most clausification procedures to a multitude of clauses.

CL is still somewhat less expressive than full FOL. However, every first-order theory has a conservative (definitional) extension which is equivalent to a coherent theory. (The proof theory of full FOL is generally considered to be too difficult to be automated.) CLs simple proof theory (forward ground reasoning) can easily be automated and proof objects are easily obtained. Proof objects are important for explanation facilities, independent verification and use in other systems, as well as for algorithm extraction.

2 Background

The automation of reasoning (AR) is one of the great dreams of computer science. Together with natural language processing, AR has proved to be a real challenge. For comparison, chess computers now play at the grandmaster level and occasionally beat the world champion, whereas AR in the mathematical domain generally stays at the undergraduate level. Yet, in some specialized mathematical subjects, AR occasionally obtains new results, such as a proof of the Robbins' Conjecture, see [4]. Another application of automated reasoning is the construction and verification of mathematical proofs that, because of their length and the use of ad-hoc computer solutions, have not immediately been accepted by the mathematical community. See for example recent issues of Science [2] and The Economist [3] where the Four Color Theorem and a proof of the Kepler Conjecture are discussed. Important application areas are that of the verification of computer hardware and software (see [12]), and so-called deductive databases. An upcoming application area is the semantic web [1], with its vision that future web content will be marked up with semantical information. The processing of this semantical information requires software agents that can reason automatically.

Because of the intrinsic difficulty of automated reasoning, the field has roughly split in two along the following lines:

1. focusing on useful reasoning problems which are difficult to solve for humans (involving large quantities of data in combination with weak intuitions about these), but not too difficult for a machine. This kind of problems is often found in connection with the formal verification of computer hardware and software, deductive databases and in semantic web applications.

2. taking the interactive approach, where the reasoning is not fully automatic. Here the spectrum ranges from proofs largely encoded by hand (where the automation boils down to *checking* the proofs) to proofs which to a large extent are *generated* by the machine.

In principle CL will contribute to both approaches, but is most promising for the second. In the first approach the best results are obtained with more specialized systems, e.g. for propositional logic, or even domain-specific logics. In the second approach one finds systems like Automath [26] (only proof checking), Coq [8], Isabelle [9] and Agda [5]. The latter three systems have some reasoning power, but far less than desirable and far less than CL.

3 Objectives

The primary aim of the project is to build a system for automated reasoning based on CL to support *logical frameworks*, also called *proof assistants*, such as Coq [8], Isabelle [9] and Agda [5]. This requires an optimal translation of FOL into CL as well as a fast CL theorem prover, which will both be developed in the project.

As a secondary aim we would like to compare CL to other approaches to automated reasoning (resolution, tableaux, matrix methods). This includes the tactics offered by the logical frameworks (Isabelle seems to have the most powerful tactics). But there are some indications [17, Table 1, p.258] that CL can compete with resolution or tableaux style theorem provers, at least on problems that fit naturally into the CL format. Our system should be good enough to participate in CASC, the CADE ATP System Competition [7].

Success can be defined as: our system outperforms the existing tactics in logical frameworks and has a good performance on problems that other general-purpose theorem provers also can solve. Although the CASC competition focuses on full automation (no interaction, no standardized proof objects), we want our system to end among the 5 best. (Winning is very difficult with limited resources, one competes against systems in which at least 10 man-years have been invested.)

The overall objective is to present the results obtained at good conferences (say, 3 per PhD project) and/or to publish in good journals (say, 2 publications per postdoc year). Of course the PhD student will collect his/her results in a PhD thesis.

4 State of the art in automating CL

As yet the potential of CL for AR has only been explored to a limited extent. Based on a depth-first (incomplete) variant of the proof procedure from [16, 17], a prototype prover has been developed in Prolog. This prover has been inspired by the system SATCHMO [25] for resolution logic, but extends it in two essential ways: by the existential quantifications allowed in the CL format and by generating proof objects in the format of Coq [8].

This prototype has been tested on several examples in fields that can naturally be formalized in CL: projective geometry, rewriting theory and lattice theory. A small selection of the problems solved can be found on the website [6]. A typical example is the induction step in the proof of Newman's Lemma, as described in [16]. There has been no other system that was able to find such a short proof in a few milliseconds. This problem and some others have been submitted as challenges to the TPTP data base [11].

De Nivelle [28] has developed a model finder for a rather restricted variant of CL where functions are encoded as relations. Although this is good for finding finite models, the correspondence between the original problem and the translated one is lost. Also, proof generation in this system is problematic.

For a newly found natural translation of FOL to CL, see [17]. The undecidability of CL without function symbols has been proved in [15].

5 Research plan

The automated theorem prover consists of two main components.

The first component translates an arbitrary first-order problem, that is, a set of formulas forming the theory and a goal, to CL, the logic of the theorem prover. Starting point is the translation given in [17]. It is of critical importance that no information gets lost in translation and that it results in a manageable problem for the prover. The translation itself is seldom time-critical. The input format should be TPTP format [11].

The second component is the actual proof search. This is the most critical component, both with respect to time complexity and to space complexity. Depth-first search is fast but incomplete, breadth-first search is slow but complete. The challenge is to find a good compromise. The first approach will be to extend the existing prototype based on a depth-first strategy to breadth-first search.

Both components should be equipped with a back-end that generates proof objects. In case of the first component we need a proof object for the equivalence of the FOL and CL versions of the problem. In case of the second component we need a proof object for the solution found (based on the search log). These proofs should be verified, explained to the user and/or converted to other formats in which they can be used in various proof assistants. Proof generation is seldom time-critical, but the size of the proof and its structure are of crucial importance for the success. (Of course, no proofs will be generated when participating in the CASC competition.)

For each of these two components one can distinguish two phases in the development: the prototyping phase and the phase in which the final product is produced. For the prototyping phase we will choose a high-level declarative programming language which is particularly suited to create executable environments for different logics and theorem provers. There are a number of good possibilities such as Prolog, Haskell or Maude [10]. There is ample competence with these languages in the project. Declarative programming languages are excellent for prototyping, but typically yield slow code which consumes a lot of memory. In particular for the second component, the proof search, which is highly critical for the overall performance of the system, a language like C (or C++) will be used. Indexing techniques are crucial.

The next challenge is to extend CL with a native equational logic. Reasoning with equality is of great importance in mathematics, computer science and almost all applications. It is seldom good enough to axiomatize equality as a congruence and to treat it thereafter as any other binary predicate. The proof procedures have to be reconsidered for CL with equality. A good starting point is a rule in resolution logic called *paramodulation*.

Finally, to make the system efficient enough, the following refinements have to be taken into account:

1. *Relevancy analysis of CL.* Traditionally this means the following: if one infers $p \vee q$, splits and finds in the branch with p that the goal can be proved *without using* p, then one need not

consider the branch with q. Both the proof search and the proof object will greatly benefit from such an optimization. In the case of CL one can do more: if one infers $\exists x.p(x)$, continues with $p(c)$ for a new constant c and finds that the goal can be proved *without using* $p(c)$, then $\exists x.p(x)$ is irrelevant and the proof can be simplified. The systems SATCHMORE [24] and R-SATCHMO [22] can serve as starting points.

2. *Stålmarck's method* [31] has been a very important improvement to automated reasoning in propositional logic. It has been extended to FOL in [13] with some promising results. As the reasoning in CL is ground, the method can also be applied here with the same expected success as in propositional logic.

3. *Heuristics for proof search* will be as important here as in any other system of AR: iterative deepening, weights and a 'hot list' are obvious candidates.

6 Workplan

We ask for support for a PhD student in Bergen and a postdoctoral researcher in Oslo. There exist good candidates, both in Norway and abroad. The PhD student will develop the prototypes, guided by the supervisors. The postdoc will undertake the (more difficult) implementation of the final programs. Therefore the postdoc starts one semester later than the PhD. The postdoc also develops the 'missing parts' of the theory later on, such as, for example, the completeness proof for forward reasoning in CL with equality. An important part of the workplan are the two workshops, the first after one year and the second after two years. Here *all* participants will meet and discuss future directions in the project. Apart from this, regular visits between the sites are planned, including two budgetted halfyear stays of the PhD and the postdoc (at Chalmers). The exact scheduling of the tasks in the workpackages below can be found in the e-application.

6.1 Work packages

W1. Supervision and project management

(T1) **Task:** Project coordination
Deliverables: Internal reports, work notes, drafts of reports and articles, presentations

(T2) **Task:** Project reporting
Deliverables: Annual reports to NFR, status reports, web-updates, final report

(T3) **Task:** Organisation of the two workshops
Deliverables: Proceedings of the workshops

(T4) **Task:** Supervision of PhD student, postdoc and MSc students
Deliverables: PhD and MSc graduations in the field
The PhD student will join the *PhD Research School in Information and Communication Technology* at the Department of Informatics, University of Bergen.

W2. PhD workpackage

(T1) **Task:** Prototype translation FOL→CL
Deliverables: Code and documentation of working prototype

(T2) **Task:** Prototype breadth-first CL prover
Deliverables: Code and documentation of working prototype

(T3) **Task:** Experiments with the combined T1,2 (FOL prover)
Deliverables: Reports(s) on experiments

(T4) **Task:** Prototype of proof object generator for T1,2
Deliverables: Code and documentation of working prototype

(T5) **Task:** Integration of T1,2,4 in proof assistants, experiments
Deliverables: Documented code, report(s) on experiments

(T6) **Task:** Writing of the PhD thesis
Deliverables: Phd thesis

W3. Postdoc workpackage

(T1) **Task:** Final translation FOL→CL
Deliverables: Code and documentation of final translation

(T2) **Task:** Fast implementation CL prover
Deliverables: Documented code

(T3) **Task:** Extending CL with equality, fast implementation
Deliverables: Publications, documented code

(T4) **Task:** Experiments with combined system, various search strategies
Deliverables: Report(s) on experiments

(T5) **Task:** Refinements and experiments
Deliverables: Report(s) on experiments

(T6) **Task:** CASC participation and accompanying publications
Deliverables: Place among the 5 best!

7 Publication plan: see e-application

8 Competences and collaboration

The main participants below are in italics, Bezem will lead the project.

The Bergen-group in Software Engineering (*M. Bezem*, M. Walicki, M. Haveraaen, U. Wolter) has a long-standing collaboration with the Oslo-group Precise Modeling and Analysis. Bezem has written a depth-first CL prover in Prolog, with a back-end for Coq proof objects. He will supervise the PhD student based in Bergen.

The Oslo-group has wide experience with Maude (E.B. Johnsen, O. Owe and P. Ølveczky) and Isabelle (Johnsen). *A. Waaler* is an expert in tableau and matrix methods [32, 33], and is currently running a project on matrix methods with 3 PhD students. He will supervise the postdoc based in Oslo.

T. Coquand is internationally recognized as expert in type theory, proof theory and constructive mathematics. For example, the widely used proof assistant Coq, based on [19], is named after him. He is involved in the development of the proof assistant Agda and in particular in preliminary work

on an interface between FOL and type theory. Coquand will contribute to the prototypes and to the extension of CL with equality, in particular to the completeness proof. He will supervise the PhD student and the postdoc during their stay at Chalmers (see the attached letter).

Bezem and Coquand have jointly published on CL [16, 17]. Both have participated in the ESPRIT working groups on type theory for over more than a decade (currently Coordination Action TYPES 510996).

The Gothenburg-group (`www.cs.chalmers.se/Cs/`) has further strong expertise in formal methods (R. Hähnle, *W. Ahrendt*, K. Claessen) and functional languages (J. Hughes).

H. de Nivelle is a well-known researcher in the field of AR. Some years ago he has particpated in the CASC competition with a system called Bliksem (2nd place in 1998). This is an invaluable experience and he will contribute to the final program that will compete in CASC. The Max Planck Institute (his current affiliation) is willing to host the PhD student for 3 months (see the attached letter).

References

[1] T. Berners-Lee, J. Hendler and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[2] D. Mackenzie. What in the name of Euclid is going on here? *Science*, Vol. **307**(5714):1402–1403, March 4th 2005. `www.sciencemag.org/cgi/content/full/307/5714/1402a`

[3] NN. Proof and Beauty. *The Economist*, March 31st 2005. `www.economist.com/science/displayStory.cfm?story_id=3809661`

[4] G. Kolata. Computer math proof shows reasoning power. *The New York Times*, December 10th 1996. `www.nytimes.com/library/cyber/week/1210math.html`

[5] Agda `www.cs.chalmers.se/~catarina/agda/`

[6] M. Bezem. Website for geometric/coherent logic. `www.ii.uib.no/~bezem/GL`

[7] The CADE ATP System Competition, `www.cs.miami.edu/~tptp/CASC/`

[8] The Coq Development Team, *The Coq Proof Assistant Reference Manual, Version 8.0*. Available at: `coq.inria.fr/`

[9] L.C. Paulson. *The Isabelle Reference Manual*. University of Cambridge, Computer Laboratory, 1998. Available at: `www.cl.cam.ac.uk/Research/HVG/Isabelle/`

[10] Maude: `http://maude.cs.uiuc.edu/`

[11] Thousands of Problems for Theorem Provers, *The TPTP Problem Library for Automated Theorem Proving*, `www.cs.miami.edu/~tptp`.

[12] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development*, Springer, 2004.

[13] M. Björk. *A First Order Extension of Staalmarck's Method* In G. Sutcliffe and A. Voronkov, editors, *Proceedings LPAR-12*, LNCS 3835, pages 276–291, Springer-Verlag, Berlin, 2005.

[14] A. Blass. Topoi and computation, *Bulletin of the European Association for Theoretical Computer Science (EATCS)* 36:57–65, October 1998.

[15] M.A. Bezem. *On the Undecidability of Coherent Logic*. In A. Middeldorp e.a., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, LNCS 3838, pages 6–13, Springer-Verlag, Berlin, 2005.

[16] M. Bezem and T. Coquand. Newman's Lemma – a Case Study in Proof Automation and Geometric Logic. In Y. Gurevich, editor, The Logic in Computer Science Column, *Bulletin of the EATCS* 79:86–100, February 2003. Also in G. Paun, G. Rozenberg and A. Salomaa, editors, *Current trends in Theoretical Computer Science*, Volume **2**:267–282, World Scientific, Singapore, 2004.

[17] M.A. Bezem and T. Coquand. Automating Coherent Logic. In G. Sutcliffe and A. Voronkov, editors, *Proceedings LPAR-12*, LNCS 3835, pages 246–260, Springer-Verlag, Berlin, 2005.

[18] F. Bry and S. Torge. A Deduction Method Complete for Refutation and Finite Satisfiability, *Proceedings 6th European Workshop on Logics in AI (JELIA)*, LNAI 1489, pages 122–138, Springer, 1998.

[19] T. Coquand and G.P. Huet. The Calculus of Constructions, *Information and Computation* 76(2/3):95–120, 1988.

[20] T. Coquand, A Logical Approach to Abstract Algebra. In: *CiE 2005: New Computational Paradigms*, LNCS 3526, pp. 23–34, 2005.

[21] M. Coste, H. Lombardi, and M.F. Roy. Dynamical methods in algebra: effective Nullstellensätze. *Annals of Pure and Applied Logic* **111**(3):203–256, 2001.

[22] L. He, Y. Chao and H. Itoh. R-SATCHMO: Refinements on I-SATCHMO. *Journal of Logic and Computation* **14**(2):117–143, 2004.

[23] R.A. Kowalski. Predicate Logic as Programming Language, *Proccedings IFIP Congress*, 569–574, 1974.

[24] D. Loveland, D. Reed and D. Wilson. SATCHMORE: SATCHMO with RElevancy, *Journal of Automated Reasoning*, 14:325-351, 1995

[25] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9-th Conference on Automated Deduction*, Lecture Notes in Computer Science 310, pages 415–434, Springer, 1988.

[26] R.P. Nederpelt, J.H. Geuvers and R.C. de Vrijer (eds.). *Selected Papers on Automath*. North-Holland, Amsterdam, 1994.

[27] S. Negri. Contraction-free sequent calculi for geometric theories, with an application to Barr's Theorem, *Archive for Mathematical Logic* **42**:389–401, 2003.

[28] H. de Nivelle and J. Meng. Geometric resolution: A proof procedure based on finite model search. In John Harrison, Ulrich Furbach, and Natarajan Shankar, editors, *International Joint Conference on Automated Reasoning 2006*, to appear.

[29] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM* 12(1): 23–41, 1965.

[30] Th. Skolem, Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen, *Skrifter* I **4**:1–36, Det Norske Videnskaps-Akademi, 1920. Also in Jens Erik Fenstad, editor, *Selected Works in Logic by Th. Skolem*, pp. 103–136, Universitetsforlaget, Oslo, 1970.

[31] G. Stålmarck. *A system for determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are generated from a formula*, European Patent No.0403 454 (1995), US Patent No.5 276 897, Swedish Patent No.467 076 (1989), 1989.

[32] A. Waaler and L. Wallen. Tableaux for Intuitionistic Logics. In M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, editors, *Handbook of Tableaux*, pages 255–296, Kluwer Academic Press, 1999.

[33] A. Waaler. Connections in nonclassical logics. In Robinson, A., Voronkov, A., editors, *Handbook of Automated Reasoning*, Volume II, pages 1487–1578, Elsevier Science, 2001.