First Workshop on Challenges and Novel Applications for Automated Reasoning

In conjunction with CADE-19, Miami, USA

Organisers: Simon Colton, Jeremy Gow,
Volker Sorge & Toby Walsh

Date: 28th July 2003

Preface

The workshop on Challenges and Novel Applications for Automated Reasoning was held in Miami, USA on July 28, 2003 in conjunction with the 19th International Conference on Automated Deduction (CADE-19).

The aim of the workshop was to identify challenges for automated reasoning that will fire both the imaginations of new researchers and those long established in the field. The workshop encompassed two distinct kinds of challenge: *Grand Challenges* propose inspirational projects that could take the efforts of many researchers over a decade or more to achieve; *Novel Applications* describe, in detail, new and relatively unexplored areas where automated reasoning can be employed right now.

Grand Challenges have recently been the focus of an initiative by the UK Computing Research Committee, whose published criteria are given after this preface. Examples of Grand Challenges from computer science include: to prove whether P = NP (open), to develop a world class chess program (completed, 1990s), or to automatically translate Russian into English (failed, 1960s).

A huge range of challenges are represented in this workshop. Several suggest a change of emphasis in the tasks performed by automated reasoners. Sutcliffe, Gao and Colton suggest developing systems that discover and prove new and interesting theorems, rather than just the ones we supply. Walther argues for lemma speculation that invents new concepts. Schultz proposes bridging the gap between formal systems and informal users.

Both McCasland and Sorge's and Calmet's challenges suggest attacking computationally intensive problems in mathematics. Tinelli sets the challenge of integrating ground satisfiability into first-order reasoning without losing completeness. In contrast, other authors envisage reasoning within large knowledge systems: Andrews with a universal formalisation for science; Walsh with a mathematical assistant that has a broad and flexible expertise.

The Novel Applications papers describe works in progress that apply automated reasoning to new problem areas. Currently, the most well-known application is reasoning about computer systems — in particular, software and hardware verification. Although we applaud this success, we believe a diversity of applications helps researchers to more thoroughly test techniques, explore new issues and problems, and so prevent the field becoming too narrowly defined.

The applications here are very diverse: Monroy proposes to formally model, and then verify properties of, the human immune system; Baumgartner et al. describe their KRHyper system, and its application to a range of tasks, including electronic publishing; Katsiri and Mycroft discuss real-time reasoning about a rapidly changing environment.

The range of challenges and applications presented at this workshop suggests that the future of automated reasoning will be extremely interesting.

Simon Colton, Jeremy Gow, Volker Sorge and Toby Walsh
July 2003

UKCRC Grand Challenge Criteria

The UK Computing Research Committee has proposed a set of criteria for assessing grand challenges in computing[1], which can be applied to our own subfield of automated reasoning. There is no expectation that a grand challenge will meet all of these criteria, but it should meet some of them:

- It arises from scientific curiosity about the foundation, the nature or the limits of the discipline.
- It gives scope for engineering ambition to build something that has never been seen before.
- It will be obvious how far and when the challenge has been met (or not).
- It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it.
- It has international scope: participation would increase the research profile of a nation.
- It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines.
- It was formulated long ago, and still stands.
- It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project.
- It calls for planned co-operation among identified research teams and communities.
- It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won.
- It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails.
- It will lead to radical paradigm shift, breaking free from the dead hand of legacy.
- It is not likely to be met simply from commercially motivated evolutionary advance.

[1] `http://www.nesc.ac.uk/esi/events/Grand_Challenges`

Table of Contents

Part I:

Grand Challenges

A Grand Challenge of Theorem Discovery

Geoff Sutcliffe[1], Yi Gao[1], and Simon Colton[2]

[1] Department of Computer Science, University of Miami
geoff@cs.miami.edu, yigao@mail.cs.miami.edu
[2] Department of Computing, Imperial College, London
sgc@doc.ic.ac.uk

Abstract. A primary mode of operation of ATP systems is to supply the system with axioms and a conjecture, and to then ask the system to produce a proof (or at least an assurance that there is a proof) that the conjecture is a theorem of the axioms. This paper challenges ATP to a new mode of operation, by which interesting theorems are generated from a set of axioms. The challenge requires solutions to both theoretical and computational issues.

1 Introduction

Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the conjecture) is a logical consequence of a set of statements (the axioms). ATP systems are used in a wide variety of domains: many problems in mathematics have been tackled with ATP techniques, software and hardware have been designed and verified using ATP systems, and applications to the WWW seem possible. In all of these applications, a primary mode of operation is to supply an ATP system with the axioms and a conjecture, and to ask the system to produce a proof (or at least an assurance that there is a proof) that the conjecture is a theorem of the axioms. Full automation of this task has been highly successful when the problem is expressed in classical 1st order logic, so that a proof by refutation of the clause normal form of the problem can be obtained. There are some well known high performance ATP systems that search for a refutation of a set of clauses, e.g., Gandalf [Tam], SPASS [WBH+02], E [Sch02a], Vampire [RV02]. The Grand Challenge presented in this paper is suitable for ATP systems for any formal system, but will be presented in terms of ATP systems for classical 1st order logic, and henceforth all discussion is in that context.

ATP systems have made some noteworthy contributions to mathematics [SFS95,McC97], are used in real-world applications [Sti94,Sch02b,CHM02], and there is empirical evidence of progress in ATP [SFS01]. Successes of ATP systems have to be viewed in the context of the super-exponential growth of the search space of ATP systems, which is $O(NumberOfFormulae^{2^{searchdepth}})$. The successes are testiment to the heuristics that guide the search towards a proof (in refutation based systems, towards the derivation of a contradiction). Without such targeted search, one might suspect that ATP systems would be of little use to man or beast. Some might even claim that current state-of-the-art ATP systems are capable of proving only an exceptional few interesting theorems. While the huge search space of ATP systems is part of their challenge, it is also the source of the Grand Challenge presented in this paper.

The logical consequences of a set of (non-contradictory) axioms form the theory of those axioms. In such a theory there are many boring theorems, and scattered amoung them there are a few interesting ones. The few interesting ones include those that are *singled out* as theorems by human experts in the domain. Although humans identify many interesting theorems (of a given set of axioms), it seems inevitable that there are more out there. Our Grand Challenge is to *use ATP to generate and identify these unnoticed interesting theorems.*

If an ATP system is given a set of axioms (and infinite resources) it may generate (all) the theorems of the axioms. In order to output only interesting theorems, modifications will be necessary. The modifications may be in the form of internal guidance to prevent or reduce the generation of boring theorems, or in the form of post-processing filters that identify interesting theorems in the output stream. If such modifications can be designed and implemented, the resultant tool may be able to discover new interesting theorems of a set of axioms. Such a system will be proactive in the theorem discovery and proof process, rather than reactive, as ATP systems are now. They will, it might be said, become part of the problem rather than part of the solution.

The notion and a drive for automated discovery in science, including the use of ATP systems, is not new [Lan98,Col01]. Newell and Simon predicted [SN58] that a computer would discover and prove an important maths theorem, Alan Bundy placed discovery in his "DReaM" (the acronym for his research group, "**D**iscovery and **Rea**soning in **M**athematics"), and Bob Kowalski has expressed the opinion that it is "more important to discover the right theorems than to prove unimportant ones" [Kow]. ATP systems have been used, e.g., to discover proofs of open conjectures [McC97,Sla02], to discover new axiomatizations of theories [Hod98,Wos01,MVF^{+}02], to investigate plane geometry [BSZC93], and to test automatically generated conjectures in mathematics [Zha99,Col02]. In all these applications the ATP systems have been used as assistants to prove conjectures that are generated using other techniques. That contrasts with the approach proposed in this Grand Challenge, in which the ATP systems will themselves discover new theorems, and other techniques may be used to determine whether or not the theorems are interesting.

This Grand Challenge of discovering new interesting theorems is not one that can be easily met (see Section 3). However, any short term successes can make an immediate contribution to the advancement of ATP: The development and testing of ATP systems is somewhat dependent on the availability of test problems that are somehow representative of applications. At present the TPTP problem library [SS98] is a de facto standard for testing ATP systems (it includes, amoung others, many theorems that have been idenitifed as interesting by humans). In order for the TPTP to continue to be effective it is necessary to regularly add new problems to the TPTP, so that systems do not become over-tuned to solving problems in the TPTP. Since the first release of the TPTP in 1993 it has grown from 2295 problems in 23 domains, to the current 6672 problems in 30 domains. Despite this reasonable long-term growth, some years have been leaner than others, and consistent significant growth would make the TPTP a more valuable resource. Any new theorems produced in partial answer to this Grand Challenge can be added to the TPTP, which will be of immediate benefit to ATP system developers. It is necessary that the theorems be difficult (in the TPTP sense [SS01]) for state-of-the-art ATP systems.

2 Automatic Conjecture and Theorem Creation

There are at least four ways in which new conjectures and theorems can be created, including the approach proposed in Section 1.

The first approach, the *inductive* approach, is often used by humans. Upon observing many similar events, people often induce a general conjecture. For example, noting that the prime numbers 5, 7, 11, 13 are all odd, the general conjecture that all prime numbers are odd may be made. An attempt is then made to prove (or disprove) the conjecture, and if disproved, modifications may be made, e.g., that all prime numbers greater than 2 are odd. Such reparation of faulty conjectures is described in [Lak76], and a project to automate such tasks is described in [PCSL01]. The inductive approach has the advantage of being stimulated by observations in reality, but has the disadvantage that the rule used to produce the conjectures, induction, is unsound. As a result, many of the conjectures produced are non-theorems, and effort is expended finding disproofs. In some cases a disproof many be hard to find, and a conjecture remains open, possibly considered in folklore to be a theorem, for a considerable period. An example is the conjecture that $P \neq NP$.[1]

The second approach, the *generative* approach, is an extension of the inductive approach. There are various ways in which the generative approach can proceed. The simplest form of generation is syntactic, in which conjectures are created by mechanical manipulation of symbols, e.g., [Pla94]. The MCS system [Zha99] generates conjectures syntactically and filters them against models of the domain. A more intelligent, semantically based, approach is taken by the HR system [Col02]. HR starts with an initial set of concepts – supplied with both a set of examples and a definition – and some axioms that relate the concepts. It then iteratively invents new concepts based on previous ones, and uses the examples of concepts to check for empirical relationships between the concepts. Such relationships are stated as conjectures and are passed along with the axioms to an ATP system to test for theoremhood. Conjectures that pass the filtering are sent to an ATP system to test for theoremhood. Like induction, generation is unsound. However, if the rules by which the generation is performed are sufficiently conservative then this approach may generate a higher fraction of theorems than the inductive approach. The effectiveness of just the

[1] One could argue that unproved conjectures found inductively, such as Goldbach's conjecture and Fermat's last theorem, are very advantageous to mathematics.

generation functions used in HR is illustrated by an empirical study in which all 46186 group theory conjectures generated were proved to be theorems, and of those 184 were added to the TPTP [CS02]. Note that this was just an initial study of HR's ability to find conjectures of difficulty for theorem provers, and no measures were used to filter the conjectures. A more sophisticated application of HR to conjecture generation is described in [ZFCS02].

The third approach, the *manipulative* approach, generates conjectures from existing theorems. An existing theorem is modified by operations such as generalization, specialization, combination, etc. This approach is used in abstraction mapping, which converts a problem to a simpler problem, and uses a solution to the simpler problem to help find a solution of the original problem [Pla80]. Manipulation of ATP problems has also been used to produce new problems for testing the robustness of ATP systems' performances [Vor00]. An advantage of the manipulative approach is that if the manipulations are satisfiability preserving, then theorems, rather than conjectures, are produced from existing theorems. However, the conjectures produced by the manipulative approach are typically artificial in nature, and therefore uninteresting.

The fourth approach, the *deductive* approach, is that proposed in this paper. It generates only theorems consequences, and may or may not use internal mechanisms to guide the generation towards interesting theorems. The advantage of this approach is that every output is known to be a theorem. The disadvantage is that many boring theorems will be generated. A major part of the challenge is to overcome this tedium, so that boring theorems are either not generated or are discarded internally. The deductive approach, like many functions performed by computers, is one that cannot be realistically adopted by humans. The large number of logical consequences that need to be considered would swamp the ponderous human brain. It is only the capability of high-speed computing that makes this approach viable. To paraphrase Emma Lazarus' words [Laz83], computers happily accept "your boring, your trivial, your huddled theorems who yearn TPTP".

3 Attacking the Challenge

Using the deductive approach to discover new interesting theorems is a significant computational challenge. Goal directed theorem proving, e.g., the search for the empty clause in CNF refuation based ATP, provides some obvious opportunities for pruning and ordering the search space. In contrast, pruning and ordering so as to ignore boring theorems and to discover interesting theorems early in the search, seems to be a more difficult task - if ATP systems have any difficulty with deducing a target formula, removing the target is going to make things worse. Interesting theorems may be widely distributed in the space of logical consequences of a set of axioms, and a search through a larger fragment of that space seems likely to be necessary.

As a basis for a first attack on this challenge, a filtering approach is being implemented. It is already possible to have a CNF based ATP system generate a stream of logical consequences, filtered so that no generated formula is subsumed by an earlier one. These logical consequences can then be assessed for interestingness. Several filters are being considered:

Non-obviousness requires a theorem to be reasonably difficult to prove. This can be measured from the size of its proof tree, or the time taken for its proof. It may be the case that the derivation that created a theorem is significantly non-minimal. A directed search for a smaller derivation may lead to a theorem being given a higher measure of obviousness than it initially received.

Novelty requires that a theorem is non-tautologous and non-redundant with respect to other theorems and the axioms. This includes subsumption checking, and ensuring that a theorem is not easily proved from the axioms and any of its descendants.

Surprisingness measures new relationships between concepts. In the logic formalism, one way to measure is to examine the extent to which new combinations of predicate and functions symbols occur in a theorem.

Intensity measures the extent to which a theorem summarizes the information contained in the axioms from which it is deduced. A naive possibility is to consider the ratio of the symbol count of the theorem and the symbol count of the axioms (symbol counting is a common heuristic used for evaluating the quality of a clause in ATP systems [SM96]). More complex measures involve consideration of the structure of the proof tree leading to the logical consequence. One idea which seems to warrant further attention is to rate a formula by the average branching factor of the formula's proof tree. This measures "bushiness" of the proof tree. Initial empirical tests of this measure against the instincts of a mathematician seems to correlate bushiness with interestingness. A related notion is the *axiom dependency* of a theorem, which measures the number of axioms required to prove the theorem. Given a set of axioms for a domain, a particular theorem may be provable using only a subset of those axioms. For instance, in group theory, various theorems are provable using only the identity axiom. One could argue that such theorems are more general, hence more interesting. However, it seems more likely that if you were using a theorem generation program to discover theorems in group theory, then the theorems which are not true of a more general algebra would be more interesting. A good example of a theorem in group theory requiring all the axioms to prove it is: $\forall a \; (a * a = a \leftrightarrow a = id)$.

Usefulness measures how much a theorem may contribute to the proof of further theorems, i.e, its usefulness as a lemma. There have been several efforts to identify relevant lemmas produced by ATP systems, especially for model elmination based systems, e.g., [Fuc00,DS01]. The focus of those efforts is different from that here, their aim being to identify lemmas that will contribute to the completion of a search for a refutation. Despite this difference, it may be possible to adapt the underlying ideas to identifying interesting theorems. The identification of interesting lemmas has also been investigated for proof presentation purposes [DS96]. Another way to measure usefulness of implications may be to generate a suite of models of the axioms, and determine the fraction of models that make the antecedent true.

Comprehensibility estimates the effort required for a user to understand the theorem. Theorems with many or deeply nested structures may be considered incomprehensible.

Applicability measures the number of objects of interest to which a theorem applies. For instance, suppose we have this number theory theorem:

$$\forall X\, (iseven(X) \land isprime(X) \rightarrow isodd(sum\_of\_divisors\_of(X)))$$

The left hand side of this theorem states that X is an even prime, which is only true of the number 2. Hence this theorem scores low for applicability because it boils down to saying that the number 3 (the sum of divisors of 2) is odd.

Interestingness in theorem discovery is a highly subjective thing. The final arbitration as to the interestingness of a theorem must be left to human experts. It will be necessary to be *very* selective about which theorems are presented for consideration, as a stream of boring theorems will soon leave the arbitrators disillusioned, and will leave the generator without arbitrators. The difficulty will include *predicting* whether a theorem will turn out to be interesting. The worth of a theorem may not become obvious for some time, perhaps even years, and theorem generation programs have to predict this worth. Such prediction is notoriously difficult.[2]

Most of the above measures of interestingness apply as much to open conjectures as to proved theorems. Moreover, measures such as novelty, surprisingness, usefulness, comprehensibility, axiom dependency and applicability have been identified elsewhere in the literature (in particular [CBW00]). Indeed, these measures have been implemented in the HR system, as described in chapter 10 of [Col02].

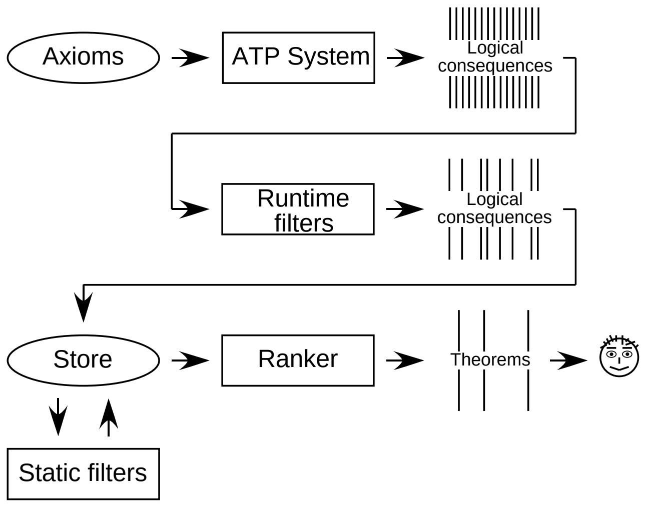
Given the above considerations, Figure 1 shows a reasonable architecture for a system that discovers interesting theorems. The logical consequences generated by the ATP system are filtered at runtime to reduce the number to a manageable level. Techniques used here have to be very quick so as to cope with the stream of production. They include requiring a minimal proof tree size (an aspect of non-obviousness), minimal intensity, and exclusion of easily identified tautologies (an aspect of novelty). The results are stored in secondary storage. When sufficient logical consequences have been stored the ATP system is stopped, and static filters are used to reduce the number of stored formulae. Techniques used here include further non-obviousness, applicability, and novelty checks. Once filtered the logical consequences are ranked in order of interestingness for presentation to the user. Measures used here include intensity, suprisingness, comprehensibility, and usefulness.

4 Conclusion

A Grand Challenge for ATP has been described: *Use ATP systems to discover interesting theorems of the axioms of a domain.* Additionally, for theorems to be

[2] Would Diophantine equations have been so studied if Fermat had left a proof of his so-called "Last Theorem" in the margin of his book?

Fig. 1. System Architecture



added to the TPTP, they should be difficult for ATP systems. This challenge can be approached incrementally:

- Generate theorems.
- Generate theorems that are difficult for ATP systems to prove.
- Generate theorems that are interesting to humans.
- Generate theorems that are interesting to humans and difficult for ATP systems to prove.
- Generate theorems that are interesting to humans and difficult for humans and ATP systems to prove.

Progress past the first of these is certainly possible, and progress to the last will be a wonderful success.

The UK Computing Research Committee has proposed a set of criteria for assessing grand challenges in computing [GC-], which can be applied to grand challenges in automated reasoning. Does this Grand Challenge meet these?

1. It arises from scientific curiosity about the foundation, the nature or the limits of the discipline.
   Although some theorems are proved out of commercial or other mundane need, the search for interesting theorems is largely a scientific endeavour. The theorems that may be generated by ATP in response to this challenge are not specifically of interest to ATP research. Rather, it is a challenge to see whether or not such theorems can be generated automatically.
2. It gives scope for engineering ambition to build something that has never been seen before.

The development of ATP systems is not new, and some of the techniques proposed for filtering out boring theorems have been tried and tested. The use of these in combination to find interesting theorems is new. Further, the development of techniques that will internally guide ATP systems towards interesting theorems will be a new feature of ATP systems.

3. It will be obvious how far and when the challenge has been met (or not).
   Human arbitration of the interestingness of the theorems produced will measure progress and success.
4. It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it.
   Not yet ... in fact, only a tiny proportion of automated reasoning researchers have addressed the question of discovering theorems, rather than proving known theorems.
5. It has international scope: participation would increase the research profile of a nation.
   Not yet ...
6. It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines.
   The idea is simple, and comprehensible to anyone who has ever had to prove a high school geometry theorem. The possibility of a computer inventing something new has long been a matter of debate between proponents and critics of artificial intelligence [Dre79]. Success here would weigh heavily in that debate, and would be of direct interest to the AI community and to those from the domain of the theorems.
7. It was formulated long ago, and still stands.
   Automated discovery in science is not a new idea (although I came up with this specific instance in the shower last week), and the challenge of generating theorems has been identified previously, most notably in [Col01].
8. It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project.
   At this stage only the first level of challenge (as itemized above) can be met. Current understanding of what makes a theorem interesting, rather than simply true, will be extended and refined. Effective implmentations of filtering and search techniques will have to be developed. Appropriate interfaces between ATP systems, filtering systems, and human arbitrators, will have to be designed.
9. It calls for planned co-operation among identified research teams and communities.
   There is scope and necessity for cooperation between the ATP community and experts from the domains of application.
10. It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won.
    The organizers of the annual CADE ATP System Competition [Nie02] have discussed the design of a competition for the submission of the "most interesting ATP problem". Some of the adjudication would be subjective, and done by human experts, while some would be based on empirical testing of the

problems on ATP systems. These ideas may be further developed to evaluate success in meeting this challenge.

11. It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails.
The application of existing criteria, and the development of new criteria, for recognizing a theorem as "interesting" will be interesting in their own right, even if they cannot be used successfully to filter the output of ATP systems. A deeper understanding of the structure of ATP systems' search spaces, resulting from attempts to guide ATP systems towards generating interesting theorems, would be valuable to ATP in general.
12. It will lead to radical paradigm shift, breaking free from the dead hand of legacy.
Using ATP systems to generate theorems, rather than find proofs for theorems, is a departure from their common usage.
13. It is not likely to be met simply from commercially motivated evolutionary advance.
Current commercial applications of ATP aim to find proofs and models. At this point in time there is no apparent commercial demand for the generation of interesting theorems.

Working on this challenge will be a pleasure and a virtue, because, as Bob Boyer remarked about working on open problems, it is impossible to cheat.

References

[BSZC93] R. Bagai, V. Shanbhogue, J.M. Zytkow, and S-C. Chou. Automatic Theorem Generation in Plane Geometry. In H.J. Komorowski and Z.W. Ras, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, number 689 in Lecture Notes in Artificial Intelligence, pages 415–424. Springer-Verlag, 1993.

[CBW00] S. Colton, A. Bundy, and T. Walsh. On the Notion of Interestingness in Automated Mathematical Discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.

[CHM02] K. Claessen, R. Hähnle, and J. Mårtensson. Verification of Hardware Systems with First-Order Logic. In G. Sutcliffe, J. Pelletier, and C. Suttner, editors, *Proceedings of the CADE-18 Workshop - Problem and Problem Sets for ATP*, number 02/10 in Department of Computer Science, University of Copenhagen, Technical Report, 2002.

[Col01] S. Colton. Automated Theorem Generation: A Future Direction for Theorem Provers. In M. Kerber, editor, *Proceedings of the IJCAR 2001 workshop: Future Directions in Automated Reasoning*, 2001.

[Col02] S. Colton. *Automated Theory Formation in Pure Mathematics.* Springer-Verlag, 2002.

[CS02] S. Colton and G. Sutcliffe. Automatic Generation of Benchmark Problems for Automated Theorem Proving Systems. In B. Faltings, editor, *Proceedings of the 7th International Symposium on Artificial Intelligence and Mathematics*, pages AI–M 4–2002., 2002.

[Dre79] H. Dreyfus. *What Computers Can't Do: The Limits of Artificial Intelligence*. Harper collins, 1979.

[DS96] J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21:523–541, 1996.

[DS01] J. Draeger and S. Schulz. Improving the Performance of Automated Theorem Provers by Redundancy-free Lemmatization. In I. Russell and J. Kolen, editors, *Proceedings of the 14th Florida Artificial Intelligence Research Symposium*, pages 345–349. AAAI Press, 2001.

[Fuc00] M. Fuchs. Controlled Use of Clausal Lemmas in Connection Tableau Calculi. *Journal of Symbolic Computation*, 29(2):299–341, 2000.

[GC-] Grand Challenges for Computing Research. http://umbriel.dcs.gla.ac.uk/NeSC/general/esi/events/Grand_Challenges/.

[Hod98] K. Hodgson. Shortest Single Axioms for the Equivalential Calculus with CS and RCD. *Journal of Automated Reasoning*, 20(3):283–316, 1998.

[Kow] R.A. Kowalski. A Short Story of My Life and Work. http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/history.html.

[Lak76] I. Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press, 1976.

[Lan98] P. Langley. The Computer-Aided Discovery of Scientific Knowledge. In S. Arikawa and A. Motoda, editors, *Proceedings of the 1st International Conference on Discovery Science*, number 1532 in Lecture Notes in Computer Science, pages 25–39. Springer-Verlag, 1998.

[Laz83] E. Lazarus. The New Colossus. 1883.

[McC97] W.W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[MVF$^+$02] W.W. McCune, R. Veroff, B. Fitelson, K. Harris, A. Feist, and L. Wos. Short Single Axioms for Boolean Algebra. *Journal of Automated Reasoning*, 29(1):1–16, 2002.

[Nie02] R. Nieuwenhuis. Special Issue: The CADE ATP System Competition. *AI Communications*, 15(2-3), 2002.

[PCSL01] A. Pease, S. Colton, A. Smaill, and J. Lee. A Multi-Agent Approach to Modelling Interaction in Human Mathematical Reasoning. In *Proceedings of the 2001 International Conference on Intelligent Agent Technology*, Intelligent Agent Technology: Research and Development. World Scientific, 2001.

[Pla80] D.A. Plaisted. Abstraction Mappings in Mechanical Theorem Proving. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th International Conference on Automated Deduction*, number 87 in Lecture Notes in Computer Science, pages 264–280. Springer-Verlag, 1980.

[Pla94] D.A. Plaisted. The Search Efficiency of Theorem Proving Strategies. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 57–71. Springer-Verlag, 1994.

[RV02] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[Sch02a] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[Sch02b] J. Schumann. *Automated Theorem Proving in Software Engineering*. Springer-Verlag, 2002.

[SFS95] J.K. Slaney, M. Fujita, and M.E. Stickel. Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. *Computers and Mathematics with Applications*, 29(2):115–132, 1995.

[SFS01] G. Sutcliffe, M. Fuchs, and C. Suttner. Progress in Automated Theorem Proving, 1997-1999. In H. Hoos and T. Stützle, editors, *Proceedings of the IJCAI'01 Workshop on Empirical Methods in Artificial Intelligence*, pages 53–60, 2001.

[Sla02] J.K. Slaney. More Proofs of an Axiom of Lukasiewicz. *Journal of Automated Reasoning*, 29(1):59–66, 2002.

[SM96] G. Sutcliffe and S. Melville. The Practice of Clausification in Automatic Theorem Proving. *South African Computer Journal*, 18:57–68, 1996.

[SN58] H. Simon and A. Newell. Heuristic Problem Solving: The Next Advance in Operations Research. *Operations Research*, 6(1):1–10, 1958.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS01] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[Sti94] M.E. Stickel. The Deductive Composition of Astronomical Software from Subroutine Libraries. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 341–355. Springer-Verlag, 1994.

[Tam] T. Tammet. Gandalf Family of Automated Theorem Provers. http://www.cs.chalmers.se/ tammet/gandalf/.

[Vor00] A. Voronkov. CASC-16 1/2. Technical Report CSPP-4, Department of Computer Science, University of Manchester, Manchester, England, 2000.

[WBH$^+$02] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS Version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 275–279. Springer-Verlag, 2002.

[Wos01] L. Wos. Conquering the Meredith Single Axiom. *Journal of Automated Reasoning*, 27(2):175–199, 2001.

[ZFCS02] J. Zimmer, A. Franke, S. Colton, and G. Sutcliffe. Integrating HR and tptp2X into MathWeb to Compare Automated Theorem Provers. In G. Sutcliffe, J. Pelletier, and C.B. Suttner, editors, *Proceedings of the CADE-18 Workshop - Problem and Problem Sets for ATP*, number 02/10 in Department of Computer Science, University of Copenhagen, Technical Report, 2002.

[Zha99] J. Zhang. System Description: MCS: Model-Based Conjecture Searching. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 393–397. Springer-Verlag, 1999.

A Universal Automated Information System for Science and Technology

Peter B. Andrews

Carnegie Mellon University, Pittsburgh, PA, U.S.A.
andrews@cmu.edu
http://gtps.math.cmu.edu/andrews.html

Abstract. It may be centuries before there is a well developed Universal Automated Information System for Science and Technology, but such a system is starting to grow even now. We consider various aspects of such a system.

Let us try to look ahead a few hundred years and imagine what we may find. Surely there will be computer-based information sources which can be used as aids in answering questions from all realms of mathematics, science, applied science, and technology. Just as those searching for information on the web today tend to regard "the web" as one monolithic information source, those who use these information sources of the future are likely to think of them as comprising a Universal Automated Information System for Science and Technology. What will this system be like? One possibility is that it will be a huge kluge of systems designed by specialists from various fields using whatever techniques and programming methodologies they found most immediately convenient. Another possibility is that it will be a well organized union of information systems in which formal logic and techniques of automated deduction play significant roles.

When one uses an information system, one may be seeking information which is not explicitly in the system, but which can be derived from information which is there. Thus, it is desirable that information systems be organized so that one can apply logical inferences to them. This is particularly important if it is a computer, rather than a person, that is looking for the information. People often make inferences almost subconsciously, but reasoning by computers must be explicit. The need to derive consequences of information suggests that the information should be represented in a form which can at least be readily translated into a formal language for which there are well known rules of logical inference, i.e., a formal logical language.

The Automated Information System of the future will undoubtedly have many components which have been contributed by many different people and projects at many different times. It seems desirable that the content of information in the system be separated from mechanisms for retrieving and deriving information, so that improvements in the retrieval and inference mechanisms

[0] This work was supported by NSF grant CCR-0097179.

can be made in a modular way which will benefit users of all components of the system.

Sometimes one asks a question which can be answered only by using knowledge from several disciplines. Thus, it is highly desirable that the Automated Information System be well integrated to facilitate dealing with such questions.

Let us consider as a very grand challenge the design and development of a Universal Automated Information System for Science and Technology having at least the following features:

- There are formal logical systems (formal languages with rules of reasoning) in which one can represent, in useful and natural ways, all the statements which might be made in the fields of mathematics, computer science, physics, chemistry, biology, engineering, medicine, and other physical sciences and applications of physical science. In particular, all information which is used in these fields is representable in these logical systems.
- There are good interfaces for these logical systems. There are algorithms for translating statements between different formal languages, and between formal languages and natural languages.
- Virtually all of the knowledge which constitutes the fields mentioned above is expressed in these formal systems. This includes not only factual data, but also definitions and fundamental principles which one may regard as axioms, hypotheses, or theorems. This knowledge, as so represented, is stored in computerized information libraries.
- The system grows continually as new knowledge is made available to it.
- There are Automated Reasoning Systems which can access this knowledge and apply logical inferences to derive answers to a great variety of questions. Calculations, algorithms, decision procedures, and special techniques for special problems are applied appropriately. These Automated Reasoning Systems can be applied to work automatically, semi-automatically, or interactively.

Of course, many people have been thinking about such a system, in at least a casual way, for many years. Indeed, the QED project [7] is concerned with building such a system for mathematics, and many of the same considerations apply to both. A system having all the features described above may take centuries to develop, but the Automated Information System of the future is growing right now, and it is important to encourage growth in the right directions. Good plans about how such a system should grow could have enormous future benefits. Let's think about the design of such a system.

It is not easy to make good plans about complex projects, and people will disagree about many details. One should try to reach as much consensus as possible on basic ideas and principles underlying the project, and explicitly plan for ways of handling disagreements. For example, different choices of formal languages may be accommodated by suitable interfaces and translation mechanisms. In the long run, the system will evolve in ways that no one can foresee or control.

Most information is written in a natural language (such as English or French), but to obtain the benefits of automated deduction one needs (some of) this information to be expressed in a formal logical language. Enormous progress could be made in developing automated information systems if suitable portions of natural language could be translated automatically into symbolic logic. For example, if one could translate all the theorems and definitions in Bourbaki's *Elements of mathematics* [4] into machine-readable formulas of symbolic logic, one would have a very impressive mathematical library for automated reasoning systems to use.

Formalization of the knowledge in any intellectual discipline is an enormous task. However, handling the details involved in such a task and testing various aspects of the formalization are greatly facilitated by using a general-purpose theorem-proving system (such as TPS [10]) which can be used in a mixture of automatic and interactive modes and has suitable library facilities. Automated deduction makes formalization much more tractable than it has ever been, and formalization will continue to become more tractable as theorem-proving systems improve.

Mathematics, which is often referred to as the language of science, plays a role in all technical disciplines, so a language which is used to formalize a technical discipline should at least be adequate for formalizing mathematics. Type Theory [1] and axiomatic set theory [9] have been studied extensively as general purpose languages for expressing mathematics. If one has an information system which is based on first-order logic, one can simply regard it as being based on type theory, which includes first-order logic. If one uses some extension of axiomatic set theory, it must be an extension of a formulation such as [9] which accommodates entities which are not sets.

As far as formalization is concerned, mathematics has received much more attention than other disciplines. Still richer languages than those needed for mathematics may be needed to formalize certain disciplines in suitably natural ways. For example, it may be desirable to incorporate into the formal language of science some representation of the diagrams of molecular structure which chemists use. Actually, diagrams are used in mathematical reasoning too[1], and the relevant theory for Venn diagrams is well developed [6][8]. However, at present, facilities for using Venn diagrams are not integrated into most general-purpose automated theorem proving systems.

Answering a question or solving a problem involves much more than deriving a statement which expresses the answer from relevant information. One must *find* the answer as well as prove that it is correct. This may involve search, logical deduction, and a variety of special techniques. We mention [5] and [11] as examples of research in this area. One way of testing problem-solving techniques is to formalize the information in relevant textbooks, and see if the problem-solving techniques are adequate for automatically doing the exercises in those textbooks.

[1] See [3] for persuasive arguments about using diagrams in mathematical proofs in ways that involve no compromise with complete rigor.

Finding relevant information becomes an increasingly serious problem as the amount of available information increases. There will be an ongoing need for research on this problem.

One well known method of finding relevant information is to classify items of information, and retrieve items with relevant classifications. Existing classification systems need to be extended so that (in mathematics, for example) one can classify not only books and articles, but also theorems, definitions, and examples. Current classification systems leave much to be desired, at least in mathematics. Mathematicians sometimes have trouble finding out whether a theorem they have proven has been proven before. The most practical method of answering this question is usually to rely on the the memories of experts in the field.

One would hope to at least have effective ways of determining whether a specified theorem is already in the Automated Information System. This can be a nontrivial problem, since there are many ways of expressing mathematical theorems. For example, the statements “Every function which has an inverse is bijective” and “A function which is not bijective cannot have an inverse” are clearly two ways of expressing the same theorem. One can think of a variety of ways to make trivial changes in the ways theorems are expressed without changing the essential mathematical content of the statements. Can we find some natural equivalence relation on the set of statements of a formal language for expressing mathematical theorems such that equivalent statements should be regarded as expressing the same theorem? Clearly logical equivalence is much too broad for this purpose, since all provable statements are logically equivalent to each other.

A classification scheme for mathematics should be robust. Different ways of expressing a theorem should not lead to different classifications for it.

One would like to have a classification scheme for mathematics which is based on some fundamental understanding of the structure of the field of mathematics, and which is objective. We don’t want to have to convene a committee of mathematicians to decide how a new theorem should be classified. Unfortunately, we don’t yet have any such fundamental understanding of the structure of mathematics.

Of course, classification is another area where there can be disagreements. Different classification schemes may be most appropriate for different purposes, or simply preferred by different people. One solution to such disagreements is to permit many classification schemes to be available simultaneously, and permit users to decide which classification to use at any particular time. Such a multiple classification system has been implemented for the TPS library by Chad Brown; it is described briefly in [2]. It remains to be seen whether complications arise when one tries to establish multiple classification schemes for enormous amounts of information.

Since the information in the Automated Information System will be contributed by many people and processes at many different times, some of it is likely to be unreliable or obsolete. Systems which provide some degree of authentication for certain components of the entire information system will be needed, and automated reasoning systems will need to keep track of the sources of information they use.

The development of an Automated Information System such as we have been discussing is indeed a very grand challenge. Developing such a system would not only provide an intellectual tool which would elevate our ability to reason reliably about complex technical questions to a new level, it would also lead to a deeper understanding of the nature and structure of our knowledge. Developing such a system involves working on a number of tasks which are themselves grand challenges. We close by summarizing some of these:

- Develop general standards which would permit the many components of the Automated Information System to be used together in a harmonious and efficient way.
- Promote widespread discussion and adoption of these standards.
- Study what extensions of existing formal languages may be needed for the formalization of various scientific and technical disciplines. Investigate whether various formal languages satisfying these requirements may be regarded as specializations of a single more general formal language.
- Formalize the knowledge in all scientific and technical disciplines.
- Develop automated systems for translating between various formal and natural languages.
- Develop improved methods of retrieving relevant knowledge.
- Develop good classification schemes for knowledge in all technical disciplines.
- Improve theorem-proving systems.
- Improve and extend question-answering and problem-solving systems which are based on automated deduction.
- Develop mechanisms for safeguarding, verifying, monitoring, and assessing the reliability of information obtained from the Automated Information System.
- Find ways to influence the education of students in scientific disciplines so that eventually workers in these fields will be familiar with formal logic and automated reasoning tools, and will be able to effectively use and contribute to those components of the Automated Information System which are relevant to their special interests.

References

1. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Kluwer Academic Publishers, second edition, 2002.
2. Peter B. Andrews, Matthew Bishop, Chad E. Brown, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS *User's Manual*, 2003. 103+iv pp. Available from http://gtps.math.cmu.edu/tps.html.
3. Jon Barwise and John Etchemendy. Visual Information and Valid Reasoning. In Walter Zimmermann and Steve Cunningham, editors, *Visualization in Teaching and Learning Mathematics*, pages 9–24. Mathematical Association of America, 1991.

4. Nicolas Bourbaki. *Elements of mathematics.* Hermann, 1966. 10 volumes. Translation of *Elements de mathematique.*
5. Cordell Green. Application of Theorem Proving to Problem Solving. In Donald E. Walker and Lewis M. Norton, editors, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 219–239, Washington, D.C., 1969.
6. Eric M. Hammer. *Logic and Visual Information.* CSLI Publications & FoLLI, Stanford, California, 1995.
7. The QED Manifesto. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 238–251, Nancy, France, 1994. Springer-Verlag.
8. Sun-Joo Shin. *The Logical Status of Diagrams.* Cambridge University Press, 1994.
9. Patrick Suppes. *Axiomatic Set Theory.* D. Van Nostrand Company, Inc., Princeton, N.J., 1960.
10. TPS and ETPS Homepage. http://gtps.math.cmu.edu/tps.html.
11. R. Waldinger, P. Jarvis, and J. Dungan. Pointing to Places in a Deductive Geospatial Theory. In A Kornai and B Sundheim, editors, *Workshop on Analysis of Geographical References; Human Language Technology Conference*, Edmonton, Canada, May 2003. NAACL.

Are There True Grand Mathematical Challenges in Mechanized Mathematics?

Jacques Calmet [*]

Institut for Algorithms and Cognitive Systems (IAKS)
University of Karlsruhe (TH)
calmet@ira.uka.de

Abstract. Grand mathematical challenges do exist in pure mathematics. Are some of the acute mathematical problems that we face when mechanizing mathematics true challenges? This short paper tries to assess through a few examples that they are indeed so.

1 Introduction

The quest for mathematically challenging problem is present in any field of science. A very recent example is from material science (Taylor [1]). However, the most famous challenges concern Mathematics itself. A puzzling one was the proof of Fermat's theorem. It is barely necessary to cite the landmark presentation of David Hilbert at the 1900 Paris Conference. His list of problems for the 20th century was then extended to the well-known set of the 21 Hilbert's problems. On the eve of the 21th century, the International Mathematical Union asked a selected number of top mathematicians to contribute a similar list for the coming century. The contribution of Steve Smale has been widely distributed. It appeared first in the Mathematical Intelligencer (Smale [2]). Two years later a version in French (Smale [3]) appeared in the January issue of the Bulletin of the French Mathematical Society. It looks like Smale's list got a wide agreement and no other list of challenge problems has, apparently, been published. The three greatest open problems of mathematics are: the Riemann Hypothesis (Hilberth's 16th), Poincaré conjecture and "Does P=NP?". The latter is already tightly linked to our domain. One of the remaining challenging mathematical problems is very relevant to this community. It amounts to answer the question "What are the limits of (artificial or natural) intelligence?". This simple, apparently philosophical question leads in fact to very difficult mathematical problems such as the decidability of the Mandelbrot set. Besides, it is not obvious whether or not there is a link to the implications of the Gödel incompleteness theorem.

To propose a definition of a mathematical challenge in theorem proving or in computer algebra that can be acknowledged by mathematicians is always touchy and sometimes impossible. However, if we introduce either of the words

[*] Work partially supported by the Calculemus Reserach Training Network, HPRN-CT-2000-00102

"mechanized" or "constructible" to qualify the part of Mathematics we deal with, we can then open a few tracks along the following directions.

(i) Mechanize new areas of Mathematics such as algebraic topology or better Grothendieck's theory when also including geometry. Indefinite symbolic integration is a well-known example where a problem in analysis was turned into an algebraic problem. The methodology is constructive since Risch's algorithm decides whether or not integrability exists. It is also mechanized since the solution, when it does exist is constructed,

(ii) Identify and master new representations of mathematical objects. This is well understood when designing algebraic algorithms for computer algebra systems. A certainly challenging task is to investigate how algebraic fields (Laumon [4]) could be introduced in mechanizing algebraic geometry problems. Also relevant are proof techniques in algebraic topology. This is a domain where the infinity plays a very special part compared to algebra, analysis, geometry of algebraic geometry. In these domains, the infinity is not really a challenge since proofs do not have to namely address this concept. In algebraic topology, the concept of infinity is very important when designing proofs,

(iii) Devise new proof techniques for domains where the amount of computation, not the theoretical difficulties is the challenge. An example is to prove some theorems on p-groups that would take a lifetime by hand calculation,

(iv) Space and time complexity issues when designing proofs and algorithms. Besides the "P=NP ?" problem already quoted, a prototypical example is the factorization of integer numbers. More practical examples arise when trying to improve doubly exponential algorithms such as the Gröbner bases algorithm, which play an important part in theorem proving in geometry, or when dealing with parameters as in constraint programming, which concerns any computing problem

(v) What means to prove? This looks like a silly question but a domain as demanding and "theoretical" as provable security sheds some light on this question.

The remaining part of this abstract is devoted to the presentation of some specific problems.

2 Provable security and proofs

The deduction community is much concerned by designing proof techniques for security protocols and there are many publications in this domain. It may be worthwhile to restrict the question to the simpler one "what is provable security?" and then to assess the part played by proofs in the answer. This is a domain of standard cryptography and a very nice state of the art is available in (Stern [7]). Provable cryptography is an attempt to mathematically establish security. This is indeed very difficult and as a result, what is available is a form of "practical" provable security. It is possible to decompose provable security into 5 steps:

1. define the goal of the adversary,

2. define a security model,
3. provide a proof by reduction,
4. check the proof,
5. interpret the proof.

A first remark is that we are in engineering, not in mathematics but concepts are expressed mathematically. A second one is that provable security does not necessary yield proofs that are sound. What really matters, is that public key encryption cannot be broken. As a consequence it is not that surprising to notice that the 4th point on checking proofs may assert that a proof for an encryption mechanism is false. In fact "this does not matter so much". Indeed, there is usually enough time left, before the encryption is broken, to come out with a right proof.

An open but difficult problem is to extend provable security to security protocols. One may guess that this task implies to introduce a concept of randomness in the date structures and of probability in the proof techniques.

3 Involutive bases

Systems of polynomial equations are solved using Gröbner bases and the related Buchberger's algorithm. A Gröbner basis is simply a basis with "good" properties in a given ideal. Involutive bases are a very special kind of Gröbner bases with additional combinatorial properties that make them very useful for many applications (Calmet [5]). They were first introduced by Janet a very long time ago and almost forgotten for many years.They exist in many polynomial algebras (also non-commutative ones) including ordinary polynomials and linear differential or difference operators. They are thus a possible approach to investigate symbolic solutions to system of (partial) differential equations. This is a domain where we need to find a suitable representation for differential objects. An overview is given in the final report of an INTAS project (Calmet [6]).

On the theoretical side, numerous results on the relationships between different kinds of involutive bases, Gröbner bases and characteristic sets have been obtained both for ordinary and for differential ideals. Several characterisation theorems for involutive bases have been proven and the computation of (differential) dimension polynomials has been studied. We have thoroughly investigated the homological approach to involution via Spencer cohomology. An algebraic algorithm for the geometric completion to involution was developed (including a constructive solution of the problem of the so-called delta-regularity).

Although we label these results "theoretical", they are in fact pretty technical. Any of them require to establish existence and validity proofs. The leading idea is that when we identify the right representation, then proofs and thus the algorithms that are images of such proofs are much easier to discover. A much more challenging facet of what involutive bases are leading to is subsumed by the concept of global integrability.

4 Global integrability

Given a system of non-linear partial differential equations, can we decide of its integrability? A first answer is that we have tools, such as the Cartan-Kuranishi theorem, to decide of the local integrability but none to assess the global integrability. Physicists and mathematicians are investigating this problem for around 50 and 100 years respectively and no satisfactory solution is yet found.

A possible approach is to investigate the impact of involutive techniques outlined in the previous section in field theory, a domain of Physics. Most, if not all, physical models are represented by systems of partial differential equations. Among such systems are the well-known Yang-Mills or Einstein equations for instance. Without aiming at doing better than what the very many expert physicists of string theory are doing, it is possible to study whether some systems are integrable or not. What is challenging is to solve symbolically over- or under-determined systems of polynomial or differential equations or in simpler terms to extend the concept of Gröbner bases to such systems. This is again an old, well-known problem that was much earlier investigated by Cartan and his co-workers before being put aside. The need to design constructive methods in mechanized mathematics was at the origin of a revival. But, we still need to find out the proper representations in which to better formulate involutive bases. Again in very simple words, we are in a situation where we can get some information on local solutions of non-linear systems and we aim at extending them to some kind of non-local neighborhood. At this stage it is worthwhile to assess whether algebraic topology can be the key tool leading to a breakthrough in this domain. Physics texbooks such as (Weinberg [8]) report that algebraic topology is already playing a role in obtaining approximate solutions to physical systems. This is a truly challenging problem where the challenge is to design constructive proofs and decision methods. This is a task better suited for computer scientists.

5 An example in group theory

This is a prototypical domain where problems are more tedious and repetitive than dificult. Some open proof problems could take a lifetime to be completed by hand calculation. This area reminds of the beginning of computer algebra. The first successes that established the field were obtained in high energy particle physics, celestial mechanics or general relativity where repetitive, very long and tedious computations were required. Besides exceeding the human capability, they were also error prone when done on paper.

A test problem is as follows, where the word suitable is used to avoid a too long presentation of the problem:

> *Given a "suitable" infinite collection of p-groups, give a formula for the least n such that the i-th group in the collection can be embedded in S_n, not in S_{n-1}.*

This is, according to the experts, a very long term project even when coupling DSs and CASs. However, when analyzing the problem, it is possible to identify sub-problems. Many of them are purely computational ones. For instance, one must compute determinants of matrices. Depending on the size of these matrices, a very thorough management of the computation is required. There are deduction problems as well. One of them is supposed to be simple and can be seen as a test of feasibility.

Can we prove, by machine, that every subgroup of Q_{n^2}, the quaternion group of order 2^n, is normal?

Solving such problems would bring fame in the group theory community.

6 Conclusion

This selection of a few computational domains where a need for new proof techniques looks pretty obvious shows, hopefully, that we are facing some very challenging mathematical problems. Some may be qualified to be grand. The list here, as said by Smale in his paper, is only taken from problems where the author has some experience. This is a further proof that it ought to be easely enlarged. It is not explicitely mentioned in the section on global integrability that the required tools belong to algebraic topology. This is a domain where contacts to computer science are rather limited. Two pieces of works are worth citing. A first one is by Jesus Aransay at the University of La Rioja in Spain. It is on progress and performed within Calculemus. It deals with proving theorems in algebraic topology. A second one is the KENZO computer algebra system of F. Sergeraert at the Fourier Institute in Grenoble. It is the only computer algebra system enabling to perform computations in algebraic topology.

A last word is that a true grand mathematical challenge means to solve a problem leading to fame in either Mathematics or Physics rather than in Computer Science.

References

1. J. E. Taylor: Bulletin of the AMS, Vol. 40, No. 1, January 2003
2. S. Smale: Mathematical Problems for the Next Century, Mathematical Intelligencer Vol 20 No. 2, 7-15, 1998
3. S. Smale: Problèmes mathématiques pour le prochain siècle, Gazette des mathématiques, SMF, Janvier 2000.
4. G. Laumon and L. Moret-Bailly: "Champ algébriques",Vol. 39 in A Series of Modern Surveys in Mathematics, Springer, 1999
5. J. Calmet, M. Hausdorf and W. Seiler: A Constructive Introduction to Involution. Proceedings of ISACA2000 "Applications of Computer Algebra", R. Akerkar ed., Allied Publishers Limited, pp. 33-50, 2001
6. J. Calmet et al.: "INTAS - Final report", http://iaks-www.ira.uka.de/iaks-calmet/intas.html, 2002
7. J. Stern: Why Provable Security Matters?, In "Advances in Cryptology ", Proceedings of Eurocrypt 2003, Warsaw, May 2003, Biham E., Ed., LNCS 2656 , Springer, 2003

8. S. Weinberg: The Quantum Theory of Fields, Volume II, Chapter 23, Cambridge University Press, 1996

Concept Formation

Christoph Walther [⋆]

Fachgebiet Programmiermethodik
Technische Universität Darmstadt

1 Introduction

The frequent need of user interaction is commonly considered as one of the main obstacles for accepting theorem proving technology in applications, like e.g. program verification. User interaction is often needed to support a system in the construction of a proof. This requires a certain competence in logic and in proof engineering, a competence non-expert users neither possess nor are willing to acquire. Therefore many efforts in the community aim to improve the inference machinery of a system in order to relieve a user from the burden of proof construction as far as possible.

But user interaction is also needed to provide a system with useful lemmas and concepts which are necessary to prove a certain theorem. These kind of interactions often requires a deep insight into the domain of discourse, and working out the missing notions can be a tedious and frustrating effort, even for users well familiar with theorem proving.

In this note, we focus on the latter problem and illustrate the need of machine support for concept formation by four examples. By "concept formation" we understand the invention of mathematical notions which are not provided by a given formal presentation, i.e. an axiomatization of some theory, but are needed to formulate a required generalization of a statement or to speculate a required lemma.

2 Generalization and Lemma Speculation

When proving theorems by induction, often lemmas are required to complete the proofs of the induction formulas representing the base and the step cases of the induction. For a statement to be verified, a system creates the induction formulas and then calls a first-order theorem prover to compute the proofs for these formulas. However, quite often an induction formula is not *valid* because some lemma is missing, and then the theorem prover gets stuck. Now to continue, such a lemma has to be provided and verified, and then the failed proof attempt is resumed having the new lemma available now.

Sometimes the required lemma evolves by a *generalization* from the proof-goal under consideration, e.g. by replacing terms with (universally quantified)

[⋆] Chr.Walther@informatik.tu-darmstadt.de variables, dropping disjunctive parts of a formula, strengthening the proof-goal by adding a conjunctive part etc., cf. e.g. [10].

For instance, when proving the ordering property of *Insertion Sort*, a prover may get stuck with the proof obligation

$$\forall k{:}list.\ k \neq \varepsilon \wedge ord(isort(tl(k)) \curvearrowright ord(insert(hd(k), isort(tl(k)))) \tag{1}$$

from which the missing lemma

$$\forall k{:}list, n{:}nat.\ ord(k) \curvearrowright ord(insert(n,k)) \tag{2}$$

is easily spotted by a generalization.[1]

In other cases, properties of the functions involved with a statement have to be speculated and formulated as a lemma. E.g. when verifying the correctness of a *Byte Adder*, a theorem prover may get stuck with the proof obligation[2]

$$\begin{aligned}&\forall c{:}bit, x{:}byte.\ c \neq O \wedge lsb(x) \neq O \curvearrowright \\ &dbl(byte2nat(byte\text{-}inc(rshift(x)))) = succ(succ(dbl(byte2nat(rshift(x)))))\end{aligned} \tag{3}$$

a proof of which requires a lemma about *byte-inc*, viz.

$$\forall x{:}byte.\ byte2nat(byte\text{-}inc(x)) = succ(byte2nat(x))\ . \tag{4}$$

These problems are a bit harder than the generalization problems, because the missing lemmas do not evolve directly from the proof goal on which the theorem prover failed.

The generalization problem is a special case of the *lemma speculation* problem. Both problems are well recognized in the theorem proving community, and several proposals exist to spot the missing lemma by machine, cf. e.g. [2], [3], [4], [5], [6], [7], [8], [9], [11], [12].

3 Cases for Concept Formation

Matters become more complicated, if additional notions are required to formulate the missing lemma. Here a new concept has to be invented, i.e. some mathematical notion which is not directly available from the given presentation. We illustrate some cases for concept formation by four examples.

Example 1 In the first example, we consider the permutation property of *Mergesort*. When proving the step case, a theorem prover may get stuck with

[1] Throughout this note, linear lists `list` over natural numbers `nat` are defined by the data structures `structure nat <= 0, succ(pred:nat)` and `structure list <=` ε`, add(hd:nat,tl:list)`.

[2] Bits and bytes are defined here by the data structures `structure bit <= O, I` and `structure byte <= null, mkbyte(rshift:byte,lsb:bit)`.

the proof obligation

$$\begin{aligned} &\forall k{:}list, n{:}nat.\ k \neq \varepsilon \wedge tl(k) \neq \varepsilon \wedge n = hd(k) \wedge n = hd(tl(k)) \wedge \\ &\quad \forall n'{:}nat.\ occurs(n', dis.ev(k)) = occurs(n', msort(dis.ev(k))) \wedge \\ &\quad \forall n''{:}nat.\ occurs(n'', dis.odd(k)) = occurs(n'', msort(dis.odd(k))) \\ &\curvearrowright occurs(n, merge(msort(dis.ev(k)), msort(dis.odd(k)))) \\ &\quad = succ(succ(occurs(n, tl(tl(k))))) \ . \end{aligned} \tag{5}$$

Here the problem is, that the induction hypotheses cannot be applied by the presence of *merge* in the (simplified) induction conclusion.[3] Now, being familiar with the meaning of the procedures in (5), it is obvious that the lemma

$$\forall k, l{:}list, n{:}nat.\ occurs(n, merge(k, l)) = plus(occurs(n, k), occurs(n, l)) \tag{6}$$

will help, because it removes *merge* from the goal-equation in (5), thus making the induction hypotheses applicable. To complete the proof, further lemmas, and in particular the commutativity of *plus*, are required.

However, the definition of *plus* is not provided with the original problem, which means that the concept of addition has to be invented before starting to speculate lemma (6). To solve the problem by machine, a system may use an undefined procedure

```
function F(x:nat, y:nat):nat <= ?  ,
```

start an induction proof of

$$\forall k, l{:}list, n{:}nat.\ occurs(n, merge(k, l)) = F(occurs(n, k), occurs(n, l)) \tag{7}$$

and retrieve the definition of F from the unsolved proof obligations in this induction proof. But unfortunately this approach fails, because F had to be synthesized as a *commutative* definition of addition.

Example 2 Sometimes, there is not an unique concept to invent (as in the *Mergesort* case), but alternatives exist. This introduces the risk to do more work than required, simply because one failed to discover are more superiour concept. We illustrate this problem by an example from the verification of *Quicksort*.

When verifying the ordering property of this algorithm, a theorem prover may get stuck (in the step case) with the proof obligation[4]

$$\begin{aligned} &\forall k{:}list.\ k \neq \varepsilon \wedge \\ &\quad ord(qsort(sm(hd(k), tl(k)))) \wedge ord(qsort(lg(hd(k), tl(k)))) \\ &\curvearrowright ord(app(qsort(sm(hd(k), tl(k))), add(hd(k), qsort(lg(hd(k), tl(k)))))) \ . \end{aligned} \tag{8}$$

[3] The procedure `dis.ev` computes a sublist of its argument list by disregarding all list elements on odd argument positions, and `dis.odd` performs a similar computation by disregarding all list elements on even positions.

[4] A procedure call `sm(`n`,`k`)` computes a sublist of k by disregarding all list elements which are *larger* than n, and `lg(`n`,`k`)` performs a similar computation by disregarding all list elements which are *smaller* than or *equal* to n.

Being familiar with the "idea" of *Quicksort*, it is rather obvious what is missing here: If l and k are ordered lists, then $app(k,l)$ is also ordered, provided the last element of k is less than or equal to the first element of l. To formulate this observation, the concept of the *last list element* is needed, and one obtains

$$\forall k,l{:}list.\ ord(k) \wedge last(k) \leq hd(l) \wedge ord(l) \curvearrowright ord(app(k,l)) \tag{9}$$

as one of the key-lemmas for this verification problem. In course of the proof, another concept, viz. *list membership*, is also required to state, e.g., that $sm(n,k)$ and $qsort(k)$ both compute a sublist of k.

When proving (8) with the √eriFun system, cf. [1], [13], 9 auxiliary lemmas have to be formulated, all of which have an automated proof.[5] However, the proof of (8) requires 7 user interactions to tell the system which of the auxiliary lemmas has to be applied next.

As this is an unusual bad performance of the system, we looked for an alternative for proving (8). To this effect, the concepts of a *lower* and an *upper bound* of a list are invented, and using these notions, one of the key-lemmas for proving (8) is formulated as

$$\forall k,l{:}list.\ ord(k) \wedge upper.bound(k,hd(l)) \wedge ord(l) \curvearrowright ord(app(k,l))\ . \tag{10}$$

This time, 13 auxiliary lemmas about upper and lower bounds are needed, stating e.g. that an upper bound of k also is an upper bound of $sm(n,k)$ and of $qsort(k)$. All these lemmas have an automated proof too, but now the proof of (8) requires only *one* user interaction. Moreover, this interaction is not needed, if the system is given more resources for search, whereas with the former attempt, the need for all interactions persists.

As the auxiliary lemmas in both attempts do not express deep thoughts and are rather obvious to spot, the concepts of an upper and an lower bound are obviously more useful here than the concepts of the last list element and list membership. Hence this example rises the question, how to find a *good* concept *a priori*, i.e. without exploring several alternatives.

Example 3 Having an idea for a good concept sometimes is not enough. Also a good *representation* matters, in particular when the domain of discourse is not so simple as for the sorting algorithms. To illustrate this point, let us consider an example from the verification of *Binary Search.* Imagine a procedure

```
function find(key:nat, a:list, i:nat, j:nat):bool <= ...
```

which decides whether a *key* is stored between the indices i and j in an array a with dimension 0.. $|a|$-1 (represented by a linear list). The procedure $find$ operates according to the binary search method: If $i > j$, then `false` is returned, and the result is $a\,[i] = key$ if $i = j$. In the remaining case $i < j$ holds, and $find(key, a, i, i + \lfloor \frac{j-i}{2} \rfloor - 1)$ is recursively computed if $key < a\left[i + \lfloor \frac{j-i}{2} \rfloor\right]$,

[5] The statements about system performance refer to system version *2.6.1*.

$find(key, a, i + \lfloor\frac{j-i}{2}\rfloor + 1, j)$ is recursively computed if $a\left[i + \lfloor\frac{j-i}{2}\rfloor\right] < key$, and otherwise `true` is returned as result.

The completeness property of $find$ is stated as

$$\forall a{:}list, key{:}nat.\ ord(a) \land key \in a \curvearrowright find(key, a, 0, |a|\text{-}1) \tag{11}$$

where $key \in a$ is computed by searching key in a step by step. The difficulty in proving (11) is typical for a statement involving a tail-recursive procedure like $find$ stemming from the body of a loop: The variables which vary in a procedure (or in the loop-body respectively), viz. i and j in the present case, are replaced in the statement with non-variable terms representing the initialization of the loop-variables, viz. 0 and $|a|$-1 here, thus preventing an induction upon these variables. So to enable an induction, (11) has to be generalized, and to formulate the required generalization a new concept, viz. the *partition* $a\,[i..j]$ of an array has to be invented. This concept allows to formulate the generalization

$$\begin{array}{l}\forall a{:}list, key, i, j{:}nat.\\ \quad j < |a| \land ord(a) \land key \in a\,[i..j] \curvearrowright find(key, a, i, j)\end{array} \tag{12}$$

from which a proof of (11) immediately is obtained, by using a further lemma, viz.

$$\forall a{:}list, key{:}nat.\ key \in a \curvearrowright key \in a\,[0..\,|a|\,\text{-}1]\ \ . \tag{13}$$

Having spotted the missing concept, a representation has to be developed for it such that subsequent proofs are supported. A straightforward representation of $key \in a\,[i..j]$ is obtained by defining a procedure

```
function partition(a:list,i:nat,j:nat):list <= ...
```

representing $a\,[i..j]$ by computing the sublist of a between the indices i and j. Now the lemmas

$$\begin{array}{l}\forall a{:}list, key, i, j{:}nat.\\ \quad j < |a| \land ord(a) \land key \in partition(a, i, j) \curvearrowright find(key, a, i, j)\end{array} \tag{14}$$

and

$$\forall a{:}list, key{:}nat.\ key \in a \curvearrowright key \in partition(a, 0, |a|\text{-}1) \tag{15}$$

have to be proved. However, we failed in proving (14) when following this idea. New subgoals were created involving the creation of new lemmas, which in turn raised new subgoals etc., until we gave up. An analysis of this tedious and frustrating effort revealed that the problem is the use of *partition*. Roughly speaking, this procedure (using `tl`) strips of the list elements in a between positions 0 and $i-1$ from the beginning of a and cuts of (using a procedure *butlast*) the list elements between $|a|$-1 and $j+1$ working from the end of a towards the beginning. The problem caused by this approach is that by the presence of *butlast*, the use of the induction hypotheses and auxiliary lemmas is spoiled so that the proofs do not get through, cf. [15] for details.

But fortunately, this problem immediately disappears if $key \in a\,[i..j]$ is implemented *directly*, i.e. by using a procedure

```
function in.partition(key:nat, a:list, i:nat, j:nat):bool <= ...
```

instead of *partition* as the new concept. Now the required generalization (and the auxiliary lemma) are expressed as

$$\begin{aligned}&\forall a{:}list, key, i, j{:}nat.\\ &\quad j < |a| \land ord(a) \land in.partition(key, a, i, j) \curvearrowright find(key, a, i, j)\end{aligned} \tag{16}$$

and

$$\forall a{:}list, key{:}nat.\ key \in a \curvearrowright in.partition(key, a, 0, |a|\text{-}1)\ \ . \tag{17}$$

Both lemmas are easily proved, because the definition of *in.partition* is less complicated than the definition of *partition*. This example illustrates the general problem to distinguish a good *representation* of a new concept from a bad one *a priori*.

Example 4 In the final example, we consider the verification of a code generator, computing machine code from `WHILE`-programs. The language of `WHILE`-programs consists of conditional statements, while-loops, assignments, compound statements and a statement for doing nothing. `WHILE`-programs are defined by a data structure *wprog* and represent abstract syntax trees which e.g. are computed by a compiler from a program conforming to the concrete syntax of a programming language to be available for subsequent code generation.

An operational semantics for `WHILE`-programs is given by an interpreter

```
function eval(r:state, wp:wprog):state <= ...
```

which maps a program state and a `WHILE`-program to a program state. Instead of interpreting a `WHILE`-program *wp* by the interpreter *eval*, *wp* can be compiled into a machine program by some procedure

```
function code(wp:wprog):mprog <= ...
```

in order to execute the generated code by some target machine. Machine programs are defined by some data structure *mprog*, and the operation of the target machine is defined by another procedure, viz.

```
function exec(r:state, pc:nat, mp:mprog):state <= ... .
```

This procedure defines an operational semantics for machine programs, which are executed step-by-step by fetching and executing the instruction to which the program counter *pc* actually points in the machine program *mp*.

The correctness property for the code generator *code* is formally stated by

$$\forall wp{:}wprog, r{:}state.\ eval(r, wp) = exec(r, 0, code(wp))\ \ . \tag{18}$$

The induction proof of (18) develops into 3 base and 3 step cases. The base cases are easily proved, but the step cases require lemmas which are not that easy to spot. In the first step case, for instance, in which wp is a compound statement of the form $wp1; wp2$, the proof obligation

$$\begin{array}{l}\forall wp1, wp2{:}wprog, r{:}state. \\ \quad r \neq \perp \curvearrowright \begin{array}{l} exec(exec(r, 0, code(wp1)), 0, code(wp2)) \\ = exec(r, 0, app(code(wp1), code(wp2))) \end{array}\end{array} \tag{19}$$

is obtained after simplifying the induction conclusion. This proof obligation can be straightforwardly generalized to

$$\begin{array}{l}\forall mp_1, mp_2{:}mprog, r{:}state, pc{:}nat. \\ \quad exec(exec(r, pc, mp1), 0, mp2) = exec(r, pc, app(mp1, mp2))\end{array} \tag{20}$$

in order to prove (19) with this equation. However, the generalization (20) does not hold. The "debugged" version of (20) reads as

$$\begin{array}{l}\forall mp_1, mp_2{:}mprog, r{:}state, pc{:}nat. \\ \quad pc < |mp1| \wedge HALT.free(mp1) \wedge \\ \quad closed^+(mp1) \wedge closed^-(mp1) \wedge closed^-(mp2) \\ \quad \curvearrowright exec(exec(r, pc, mp1), 0, mp2) = exec(r, pc, app(mp1, mp2))\end{array} \tag{21}$$

where the new concepts of a $HALT.free$, a $closed^+$ and a $closed^-$ machine program had to be invented in order to formulate (21). Here a machine program mp is $HALT.free$ iff mp is free of `HALT` instructions, mp is $closed^+$ iff mp has no `JUMP`- or `BRANCH`-instructions moving the pc beyond the last instruction of mp, and mp is $closed^-$ iff mp has no `JUMP`- or `BRANCH`-instructions moving the pc before the first instruction of mp. Without demanding $HALT.free$, $closed^+$ and $closed^-$ for the machine programs in (21), counterexamples for (20) exist, see [14] for details.

The new concepts are also required to formulate auxiliary lemmas similar to (21), which are needed for the remaining step cases. This example illustrates, that sometimes a deep insight into the domain of discourse is required in order to develop the necessary concepts.

4 Summary

We have demonstrated the need of concept formation by four examples with increasing difficulty in the concept formation task. Computer support for problems of the kind given in Example 1 seem not that difficult to obtain, and examples of this kind provide a good starting point to analyze and tackle the problem. However, problems like in the remaining examples seem to be much harder to solve by machine, as they require much more insight into the domain of discourse than problems like in the first example.

It is our strong believe that the acceptance of theorem proving technology will grow significantly, if reasoning systems are able to solve concept formation problems of the kind illustrated by the examples of this note.

References

1. http://www.informatik.tu-darmstadt.de/pm/verifun/.
2. R. Aubin. Mechanizing structural induction Part II: Strategies. *Theoretical Computer Science*, 9(3):347–362, 1979.
3. R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, New York, 1979.
4. R. S. Boyer and J. S. Moore. Proving theorems about LISP functions. *J. ACM*, 22(1):129–144, 1979.
5. A. Bundy. The automation of proof by mathematical induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 13, pages 845–911. Elsevier Science, 2001.
6. A. Ireland and A. Bundy. Extensions to a Generalization Critic for Inductive Proofs. In M. McRobbie and J.Slaney, editors, *Proc. of the 13th Inter. Conf. on Automated Deduction (CADE-96)*, volume 1104 of *Lecture Notes in Artifical Intelligence*, pages 47–61, New Brunswick, 1996. Springer-Verlag.
7. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *J. Automat. Reason.*, 16(1–2), 1996.
8. D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In M. McRobbie and J.Slaney, editors, *Proc. 13th Intern. Conf. on Automated Deduction (CADE-96)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 538–552, New Brunswick, 1996. Springer-Verlag.
9. T. Walsh. A Divergence Critic for Inductive Proof. *J. Artifical Intelligence Research*, 4:209–235, 1996.
10. C. Walther. Mathematical Induction. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 127–227. Oxford University Press, Oxford, 1994.
11. C. Walther and T. Kolbe. On Terminating Lemma Speculations. *Information and Computation*, 162:96–116, 2000.
12. C. Walther and T. Kolbe. Proving theorems by reuse. *Artificial Intelligence*, 116:17–66, 2000.
13. C. Walther and S. Schweitzer. About $\surd$`eriFun`. In F. Baader, editor, *Proc. of the 19th Inter. Conf. on Automated Deduction (CADE-19)*, volume 2741 of *Lecture Notes in Artifical Intelligence*, pages 1–5, Miami, 2003. Springer-Verlag.
14. C. Walther and S. Schweitzer. A Machine-Verified Code Generator. In M. Y. Vardi and A. Voronkov, editors, *Proc. of the 10th Inter. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-10)*, volume 0000 of *Lecture Notes in Artifical Intelligence*, pages 1–15, Almaty, Kazakhstan, 2003. Springer-Verlag.
15. C. Walther and S. Schweitzer. A Verification of Binary Search. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning: Techniques, Tools and Applications*, volume 2605 of *LNAI*, pages 1–18. Springer-Verlag, 2003.

A Grand Challenge for Computing Research: A Mathematical Assistant

Toby Walsh

Cork Constraint Computation Centre, University College Cork, Ireland.
tw@4c.ucc.ie

The mathematical assistant

Scientists, engineers and students would all benefit greatly from the help of a mathematical assistant. Such an assistant should be rigorous and indefatigable, and have vast amounts of mathematical knowledge at her fingertips. Since these are precisely the qualities we appreciate most in computers, computers ought to make excellent mathematical assistants. Indeed, in specialized domains, computers already are useful mathematical assistants. For example:

– Computer algebra systems can compute complex indefinite integrals and solve difficult algebraic equations;
– Matrix packages can perform large and tedious matrix computations.

However, we lack systems that have the breadth as well as the depth of knowledge of a working mathematician. Systems typically do not reason at the meta-level about how they solve problems. They are unable therefore to explain their answers, to apply their expertise to new domains, or to reason about the quality of their answers. In addition, systems are neither pro-active nor adaptive. They do not leap in and offer the user help. They require the user to know when and how to call them.

The challenge then is to develop an automated mathematical assistant with both breadth and depth of mathematical expertise. The assistant should cooperatively help users solve their mathematical problems, adapting and learning over time. Such an assistant would be able to:

– Prove that a complicated series diverges;
– Identify parameters for which an indefinite integration is "dangerous";
– Construct a counter-example to the security of the user's cryptographic scheme, and suggest how to modify it;
– Explain an integral over the real line by identifying a suitable contour in the complex plane and locating all the poles;
– Find a large combinatorial object like a projective plane of order 10;
– Prove the uniqueness of a solution to Laplace's equation by appealing to a general purpose uniqueness proof method.

A mathematical assistant will have skills across a wide range of topics, from the very formal and axiomatic (e.g. constructing theories, identifying inconsistencies, proving meta-theoretic results) to the very applied (e.g. numerically solving a set of partial differential equations).

Is this research?

Such an assistant will require research in a wide number of areas. These include:

Knowledge representation: a mathematical assistant will need a large ontology of mathematical information at both the object and the meta level;
Automated reasoning: a mathematical assistant will need rich and complex inference mechanisms;
Learning: a mathematical assistant will need to learn new mathematics;
User modelling: a mathematical assistant will need to infer the user's goals and intentions from their actions;
Databases: a mathematical assistant will need to access vast mathematical databases in complex ways (e.g. search a database for a balanced incomplete block designs with some given properties)
Distributed computation: a mathematical assistant will need to know how to break large computations down to tap into the GRID;

Is it a grand challenge?

It is certainly a challenge since we could fail. AI has had success in narrow domains (witness expert systems) but broad expertise, like that proposed here, is a much more challenging and uncertain goal. What about the other criteria identified in the call for submissions to the workshop? This challenge arises from curiosity about the limits of how much mathematics we can automate. It aims to build something never seen before. It ought to be obvious when the challenge is met since we will stop asking our mathematical colleagues for help. It will be useful to the whole scientific community so should gain their support. It is of a scale that will require international participation. It will be comprehensible to the general public. It was formulated long ago (at least as far back as Leibnitz's desire to reduce all mathematics to calculation). It will take us way beyond the domain specific mathematical tools available today. It will require planned cooperation between many different research projects. Even partial success will improve the mathematical tools available. Finally, given the scale and ambition of the challenge, it is unlikely to happen through the evolution of existing commercial products.

Position Statement

Stephan Schulz

schulz@informatik.tu-muenchen.de

While there are many challenges and open questions in automated reasoning, there are two I consider to be particularly interesting (for me) and pressing (for the community).

1 Understanding the Search Space

Nearly 40 years have passed since Robinson's ground-breaking paper on resolution and unification, and Otter, the earliest and probably still most prominent of modern theorem provers, is about 15 years old. In this time, hardware performance has increased by an incredible factor. However, we are still performing proof search in essentially the same way we did 15 or 20 years ago – by locally evaluating simple search alternatives. While refined calculi and better implementations allow us to penetrate deeper, the basic problem of good search strategies in an exponentially growing search space has not been solved.

It is my belief that search strategies that offer more than a slow, gradual improvement will need to be based on introspection, either via the analysis of successful previous searches, or on the analysis of certain global properties of the search state.

2 Breaking out of the Ghetto

Despite the many open research questions, automated reasoning systems today are powerful enough for many interesting application. However, there are two significant barriers for new users. First, users don't know how to formalize their problems. Secondly, they don't know if the effort of learning how to do it will pay off. Improving the power of reasoning systems will increase the lure, but will still leave the high initial outlay on the part of the user.

I believe another way of winning more application is the lowering of the initial cost for potential users. There are large systems that already are specified in a more or less formal way: Hare ware designs in languages like Verilog and VHDL, and, of course, computer programs in various programming languages. The AR community should work on tools that allow us to automatically translate such specifications into forms that make the application of automated reasoning tools to prove relevant properties of designs and programs possible.

Automating Algebra's Tedious Tasks: Computerised Classification

Roy L. McCasland and Volker Sorge

School of Computer Science
University of Birmingham, UK,
$\{$V.Sorge$|$R.L.McCasland$\}$@cs.bham.ac.uk
http://www.cs.bham.ac.uk/{\homedir}$\{$vxs$|$rlm$\}$

1 Problem

We present classification of (finite) algebraic structures as a possible challenge for automated reasoning systems.

Classification of algebraic entities is one of the more difficult problems in algebra. Nevertheless, it is rarely attempted by working mathematicians. Not only is it a relatively tedious task, but also it is often nearly impossible for a human since the number of structures and therefore the number of possible cases that have to be considered is generally quite large. Still, classification problems can provide interesting results or information; the most prominent example is of course the complete classification of finite simple groups by Feit and Thompson [1].

Some examples of algebraic constructs for which relatively little classification has been done so far are:

– Simple non-associative structures such as quasigroups or loops (for some classification results in this area see [7])
– Direct-sum decompositions (apart from a few cases such as finite Abelian groups and free modules, for example)
– Prime submodules for a given module
– Lattices of radical submodules

While the most obvious classification is of course to distinguish different among isomorphism classes, structures can also be classified with respect to other criteria. For instance, quasigroups and loops can be grouped into isotopy classes, and modules can be distinguished with respect to their uniform dimensions. In fact, any equivalence relation on a class of structures can be exploited for classification purposes, and indeed the more aspects that are investigated, the more information can be gained about the structures under consideration. The results can not only prove interesting for mathematicians working in the particular field, but also for possible applications — for instance, non-associative algebras have applications in circuit algebra. Moreover, classification is not only interesting in algebra but could also be applied in other mathematical domains such as analysis or differential geometry.

For some of the structures mentioned above there already exist techniques to count the number of elements for separate classes. For example, the number of isotopy and isomorphism classes for quasigroups and loops can be counted up to at least order 9, and in some cases representatives for each class can be generated [5]. However, a particularly interesting question to ask is whether, for a given type of structure, one can identify non-trivial properties which discriminate among the different classes. This is usually a much more difficult problem to solve than simply determining the numbers of classes, or even finding a representative for each class.

Although there have been some promising results, particularly the discovery of some previously unknown examples [8, 9], exhaustive classification by automated reasoning systems has never been seriously attempted. In the following we shall present some ideas on how to handle classification problems, and what would be required from reasoning systems.

2 Bootstrapping

Classification problems can often be simplified by decomposing the structures, and thereby reducing the problem to entities of smaller size. For example, if a given group can be decomposed into a direct sum of proper subgroups, then a great deal can be learnt about the given group, simply by examining the summands. In particular, if one has already classified all groups (relative to some equivalence relation) of order less than the order of the given group, then one can hopefully use this knowledge in classifying the given group. The classification process itself should therefore be designed in such a way that it is possible to reuse knowledge gained from previous classifications. Therefore, we argue for flexible classification process that allows for bootstrapping.

Additionally, knowledge from experts should be taken into account. Ideally, results of classification runs would be examined by mathematicians, who then might be able to generalise these results, and thereby identify (or perhaps discover) theorems that could significantly facilitate and improve the overall classification process. Naturally, we want to be able to incorporate this knowledge into our system, and to exploit it for classification purposes, without having to redesign or restart the overall process.

3 Grid Computing

Even if we consider algebraic structures of relatively small size, there will often be a large number of these objects. There is generally too much data for a single reasoning system to deal with effectively. Moreover, the necessary computation time could be astronomical. On the other hand, it has been shown by other projects which involve vast amounts of data, that breaking the task down into manageable pieces and using distribution and grid technology can not only significantly speed up the process, but can make the endeavour feasible in the first place.

The SETI project (`http://setiathome.ssl.berkeley.edu/`) is successfully examining hundred of millions of radio signals from space to search for extraterrestrial life, using grid-like technology. Therefore, we believe that grid computing is the predestined tool for examining billions of algebraic structures.

4 The Tools

There already exist tools that aid with the detection and examination of different classes of algebraic structures. For instance, there exist programs that can be used for counting isomorphism classes [5] or for isomorph-free exhaustive generation [4]. Moreover, model generators, constraint solvers, theorem provers, and computer algebra systems have already helped to answer certain open problems, and to determine the structure of specific algebraic entities [3, 2]. There have also been experiments with machine learning to construct discriminating properties for non-isomorphic algebraic structures [6].

However, none of these tools alone can be expected to have the necessary strength for successful classification in various algebraic domains. Instead we need a range of techniques and state-of-the-art systems that implement these techniques, combined in a dynamic and flexible framework, which allows for the described bootstrapping. A proof planning approach could be a first attempt at such a framework. Additionally, it should contain elements of machine learning, both to discover discriminating properties, and to enhance the desired bootstrapping effect.

Moreover, we will need tools that can organise distributed reasoning in a Grid environment. This goes far beyond the current means for distributing reasoning in agent-based scenarios, for instance, since we will need far more sophisticated techniques to separate and collate reasoning processes, disseminate and collect data, and to organise and reuse results.

References

1. W. Feit and J. G. Thompson. A solvability criterion for finite groups and some consequences. *Proc. Nat. Acad. Sci. USA*, 48:968–970, 1962.
2. Carla P. Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In B. Kuipers and B. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–226, USA, 1997. AAAI Press, Menlo Park, CA, USA.
3. Kenneth Kunen. G-loops and permutation groups. *J. Algebra*, 220:694–708, 1999.
4. Brendon D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26:306–324, 1998.
5. Brendon D. McKay, Alison Meinert, and Wendy Myrvold. Counting small latin squares. Available at `http://www.csr.uvic.ca/\~{}wendym/ls.ps`, 2003.
6. Andreas Meier, Volker Sorge, and Simon Colton. Employing theory formation to guide proof planning. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Proceedings of Joint International Conference AISC 2002 and Calculemus 2002*, *LNAI* 2385, pages 275–289, 2002. Springer Verlag, Berlin, Germany.

7. J. Schwenk. A classification of abelian quasigroups. *Rendiconti di Matematica, Serie VII*, 15:161–172, 1995.
8. John Slaney, Masayuki Fujita, and Mark E. Stickel. Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. *Computers and Mathematics with Applications*, 29:115–132, 1995.
9. Mark E. Stickel and Hantao Zhang. Studying Quasigroup Identities by Rewriting Techniques: Problems and First Results. In J. Hsiang, editor, *Proceedings of the* 6^{th} *International Consference on Rewriting Techniques and Applications*, *LNCS* 914, pages 450–456, 1995. Springer Verlag, Berlin, Germany.

Grand Challenges for Automated Reasoning: Integrating Decision Procedures into First-Order Reasoning

Cesare Tinelli[1]

Department of Computer Science
The University of Iowa
USA
tinelli@cs.uiowa.edu

It has been argued in many places and by several researchers (see [6] for a recent account) that the key to scaling Automated Reasoning to real world applications lies into the effective integration of special-purpose reasoners into general-purpose reasoning framework. In this respect, I think that one of the main challenges for the field in the next years will be:

efficiently integrating, without loss of completeness, ground satisfiability procedures into first-order reasoning systems.

By ground satisfiability procedure I mean the following. Given a logical theory T, a ground satisfiability procedure for T is a procedure that decides the satisfiability in T of conjunctions of literals. More precisely, it is a procedure that, for all conjunctions φ of literals in the signature of T, is always able to determine whether there is a model of T that satisfies (the existential closure of) φ. Ground satisfiability procedures have been discovered for many theories of interests in computer science. Moreover, several of them are also quite efficient in practice, if not in theory.

By integrating a ground satisfiability procedure for a theory T into first-order reasoning system, I mean modularly incorporating the procedure into a more general mechanism, such as for instance a first-order calculus, so as to obtain a system for the satisfiability in T of arbitrary first-order formulas. Because of basic undecidability results, we know that such a system will not be terminating in general. The challenge however requires that it be complete, that is, able to recognize all first-order formulas unsatisfiable in T.

In addition to *completeness*, the key requirements in this challenge are the time/space *efficiency* of the integrated system and the limitation to *ground satisfiability* procedures, as opposed to non-ground satisfiability procedures. The challenge truly consists in satisfying all of these desiderata at the same time, because without one or more of them the challenge can be considered as already met. In the following I argue why this is the case, and why it is important to achieve all of the challenge's requirements.

Efficiency. I realize that efficiency is not an objective requirement unless it is defined in quantitative terms. Therefore some reasonable and measurable notion of efficiency must be discussed and established before this challenge can be even considered. Assuming, however, that this has been done and appealing an intuitive appreciation of what an efficient system should be, let me comment on why efficiency is part of the challenge.

If one does not care about efficiency, the challenge is easily achievable by ground instantiation, at least for "Herbrand theories" T.[1] Given a formula to be checked for satisfiability, one can Skolemize it, enumerate its ground instances, and check the satisfiability of each of them in T until an unsatisfiable one is found.[2]

Since improving the efficiency of automated reasoning is the very reason for using special-purpose reasoners such as ground satisfiability procedures in the first place, the efficiency requirement is one that we certainly cannot lift.

Completeness. At least at the theoretical level, forgoing completeness almost trivializes the challenge. In fact, using past results on *theory reasoning* calculi (see [1, 2] for an overview), a ground satisfiability procedure for a theory T can be easily integrated into a general first-order calculus to obtain a sound and complete system for the satisfiability in T of *quantifier-free formulas.* The integrated system, however, cannot deal with (universally) quantified formulas, and as a consequence it is incomplete with respect to the class of all first-order formulas.

The difficulty lies exactly in dealing with quantifiers in a complete way. Theory reasoning calculi are indeed complete, but at the cost of requiring more than a mere ground satisfiability procedure (see later). Some existing theorem provers embedding ground satisfiability procedures[3] do provide a limited treatment quantifiers. They use simple but incomplete heuristics to eliminate quantifiers by guessing "right" instantiations. The challenge is in going beyond these heuristics and devise complete ways to deal with quantifiers while at the same time using decision procedure that accept only quantifier-free formulas.

Ground satisfiability. In a sense, having decision procedures for the satisfiability of sets of literals is not so much a desideratum as a necessity. Designing and implementing efficient decision procedures for *non-ground* satisfiability in a theory T is extremely hard, when possible at all. There is quite a large and ever increasing number of efficient ground satisfiability procedures in the literature, and for more and more theories of practical interest in AR applications. On the other hand, one is hard pressed to find efficient decision

[1] By Herbrand theory I mean a theory T such that a formula is satisfiable in T iff it is satisfiable in a term-generated model of T. For instance all, but not only, universal theories are Herbrand in this sense.

[2] Note that this requires the ground satisfiability procedure for T to accept formulas containing arbitrary Skolem symbols in addition to the symbols of T. Recent results, however, show that any ground satisfiability procedure can be modularly extended to do that [7].

[3] Step [3], Simplify [4], and PVS [5] come to mind.

(or even semi-decision) procedures for non-ground satisfiability. This is possibly the leading reason the theory reasoning paradigm has not fulfilled so far its promise of improving the efficiency of automated reasoning. The basic research in theory reasoning has succeeded in integrating in a complete way theory-specific *background reasoners* into general-purpose calculi. But it has done so by requirinig a background reasoners to answer more general satisfiability problems than groung satisfiability. In basically all theory reasoning calculi, the background reasoner for a theory T is given a conjunction φ of literals and is asked for a *refuter*, a substitution σ that makes the existential closure of $\varphi\sigma$ unsatisfiable in T. Implementing efficiently background reasoners of this sort has proven very challenging except for a few very simple background theories. The implementation task is a lot simpler, as proven by the many ground satisfiability procedures in circulation, if the background reasoner is just required to find out whether the existential closure of φ itself is satisfiable in the theory.

At an abstract level then, we could perhaps say that the challenge consists in moving the computation of refuters from the background reasoner into the main calculus, while still achieving completeness overall.

References

1. Peter Baumgartner, Ulrich Furbach, and Uwe Petermann. A unified approach to theory reasoning. Research Report 15–92, Universität Koblenz-Landau, Koblenz, Germany, 1992. Fachberichte Informatik.
2. Peter Baumgartner and Uwe Petermann. Chapter II.6: Theory Reasoning. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume I: Foundations. Calculi and Refinements, pages 191–224. Kluwer Academic Publishers, 1998.
3. N. Bjørner, A. Browne, E. Chang, M. Colon, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. STeP: deductive-algorithmic verification of reactive and real-time systems. In R. Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.
4. Greg Nelson and Dave Detlefs. *The Simplify user's manual.* HP Systems Research Center. (`http://research.compaq.com/SRC/esc/Simplify.html`).
5. Natarajan Shankar. Using decision procedures with a higher-order logic. In *Proceedings of the 14th International Conference on Theorem Proving in Higher-Order Logics (Edimburgh, Scotland)*, volume 2152 of *Lecture Notes in Computer Science*, pages 5–26. Springer, 2001.
6. Natarajan Shankar. Little engines of proof. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *FME 2002: Formal Methods - Getting IT Right, Proceedings of the International Symposium of Formal Methods Europe (Copenhagen, Denmark)*, volume 2391 of *Lecture Notes in Computer Science*, pages 1–20. Springer, July 2002.
7. Cesare Tinelli and Calogero Zarba. Combining non-stably infinite theories. In I. Dahn and L. Vigneron, editors, *Proceedings of the 4th International Workshop on First Order Theorem Proving, FTP'03 (Valencia, Spain)*, volume 86.1 of *Electronic Notes in Theoretical Computer Science*, 2003.

Part II:

Novel Applications

Computer Modelling the Human Immune System

Raúl Monroy

Departamento de Ciencias Computacionales
Tecnológico de Monterrey, Campus Estado de México
Carretera al Lago de Guadalupe, Km. 3.5, Atizapán, México
raulm@itesm.mx

Abstract. Problem solving strategies, borrowing ideas from the immune system analogy, have been successfully applied to challenging problems of modern computing. However, no formal model of an immune system has so far been proposed. This paper discusses what one should consider if embarked in such an enterprise. We envisage a model where each individual component is specified in terms of its observable behaviour. We then discuss how to use the model in order to formalise important properties of immune systems and then outline the sort of mechanisms that are required to formally verify them.

1 Introduction

The human immune system has recently captured increasing interest in the computing community. It portrays properties that are highly desirable in computer systems, for example effectiveness, robustness and reliability. Interestingly, immune systems are made out of a number of simple components, each of which carries out a small, specific task. This simplicity of the individual, together with the striking effect of the collection, has driven scientist to use the immune system as a model for building computer systems. Their hypothesis is that artificial immune systems will resemble the natural ones. Yet, research seems to proceed without a formal model of an immune system.

This paper discusses what aspects need to be considered in building a computer model of the immune system. The model we have in mind is intended to be employed both by biologists, as a controlled laboratory, and by computer scientists, as a reference for hopefully discovering new problem solving strategies. The model is not concerned with the nature of the interactions amongst components.[1] It comprehends only aspects of behaviour and so is based on a suitable process algebra.

Process algebras are suitable for modelling, building and analysing complex systems out of simpler ones. They provide a mathematical model to reason about the structure and behaviour of a communicating system in terms of a few primitive ideas. Example basic process algebras are CCS [15], CSP [11] and

[1] If interested in this topic, the reader is referred to [19].

ACP [2]. Process algebras are well-established in both industry and academy and have quickly evolved into an international standard, LOTOS [13]. Current research has yielded a great variety of successful extensions, including message passing [9], mobility [16, 17, 20], performance evaluation [10] and probabilistic quantification for non-determinism [10, 25].

We shall argue that for a model to be faithful to immune system behaviour, it must portray issues about mobility, action duration, probabilistic quantification for non-determinism and multiple communication. We shall motivate the sort of process algebra required to do this task and show how to formalise interesting properties of the immune system within such a framework. Finally, we will outline directions as to how to build the sort of toolbox required to formally analyse the intended model.

2 Modelling an Immune System

This section aims to motivate the sort of formalism required to model a system as complex as the immune one. We first highlight the features of the immune system that are relevant to our enterprise and then we discuss the formalism we suggest to embrace it.

2.1 Immune Systems: General Features

The immune system consists of a number of components, including cells, antigens, the blood stream, etc. Each component plays a simple, specific role. The behaviour of each element is simple and so in principle amenable to modelling and verification. Components are all independent but they coordinate one another, overcoming individual limitations. Hence, the immune system is highly distributed. Its overall behaviour is assumed to be the product of the number and kind of interactions that happen amongst its individual components.

The immune system is diverse: cells of the same kind vary from one to other. This involves a cell's ability to non-self detection. What is distinguishable to one cell may go unnoticed to others [24]. Despite cell diversity, the immune system is highly robust. This is because, in order to maximise the chance for successful detection, components, such as lymphocytes, complement their recognition abilities by travelling the body through the blood stream. Mobility is hence crucial to immune system performance.

An immune system is highly dynamic. The population size of each type of component, called *the configuration of the immune system*, changes with time and according to circumstances. Some cells last a few days but some others last years. Cells therefore obey to a sort of universal clock, they are programmed to die—*apoptosis* [14]. Non-living components, such as antigens, last as long as the body does.

To compensate for short-lived cells, the body is continuously producing cells and so are some cells of the immune system itself. Yet the population self-stabilises, it does not grow boundlessly. Population increase amounts to detection of invading microorganisms. It is followed by the corresponding decrease when the disease is eliminated [21].

Learning and memory are key to body protection. If the immune system detects an intruder it has not seen before, not only will it battle the intruder, but it will also learn the intruder structure [21]. As a result of this learning process, the immune system will evolve a collection of lymphocytes, specially designed to and designated for detecting and protecting the body against this invader.

While travelling through the blood stream, a group of immune system's components may coincide and form a chain, *clustering*. Together, the chain is able to provide a sophisticated defense, stronger than that given by each component in isolation, *emergent behaviour*.

Other features, such as the selection of a target interlocutor or the secondary effects of an interaction, should also be considered. Indeed, there are too many issues regarding immune system behaviour, that we are bound to leave necessarily some of them out. In what follows, we discuss how process algebras is an adequate means to approach modelling immune system behaviour.

2.2 Process Calculi: General Features

Basic Process calculi, such as CCS [15] CSP [11] or ACP [2], provide a means suitable for modelling and analysing complex communicating systems. They are well-established both in industry and in academia and have evolved into an international standard, LOTOS [13]. Here, we aim to show that process algebras is a suitable means to express immune system behaviour. We shall argue that for that purpose, the selected process algebra should be equipped with mobility, message-passing, action duration, multiple communication and probabilistic quantification for non-determinism.

In a process algebra, terms represent processes. Processes have their own identity, characterised by their entire capabilities of interaction. Interactions occur either between two agents, or between an agent and its environment. They are communicating activities and referred to as *actions*. An action is said to be *observable*, if it denotes an interaction between an agent and its environment, otherwise it is said to be *unobservable*. This interpretation of observation underlies a precise and amenable theory of behaviour: whatever is observable is regarded as the behaviour of a system. Two agents are considered equivalent if their behaviour is indistinguishable to an external observer.

Message-passing process algebras [9] extend basic ones by giving actions a structure so as to represent the sending or reception of values of any kind on communication channels. They are suitable to explain some aspects of immune system behaviour. Cell diversity, for example, can be properly modelled by associating the model of each cell with a parameter, so-called an *affinity set*. With it, an agent cell will be capable of recognising others if, through interaction, it gets a datum belonging to this set. Not only are message-passing process algebras suitable to express cell diversity but also memory, involved in learning.

Immune system choices cannot be assumed to be non-deterministic. While interacting with others, an immune system component may show specific preferences to do one thing over another. For example, a lymphocyte is more reactive to the presence of cytosine in the blood stream than to the presence of an antigen presenting cell. *Probabilistic process calculi* extend classical ones by adding a probabilistic quantification for non-determinism. Probabilistic processes are realised as Markov, stochastic process and, hence, existing theories within such frameworks can be used to prove properties about them. WSCCS [25] is an example probabilistic process algebra, where choices are quantified with *weights*. Thus, probabilistic specification amounts to *relative frequency*: the heavier an action's weight the more chances for it to happen.

Similarly, immune system actions cannot be assumed to be instantaneous. While specifying action duration may be burdensome, it is useful for performance analysis. *Markovian process algebras*, such as PEPA [10, 5], extend classical ones to deal with performance modelling and evaluation by quantifying time and uncertainty. Unlike Petri nets, Markovian process algebras do not provide true concurrency models. Instead they capture the uncertainty of both how long an action will take and what action will happen next. In PEPA, this is achieved by associating a random variable with each action, representing its duration, and so yielding a Markov process. The delay inherent in each action yields a race condition determining a probabilistic branching. Under certain conditions, a PEPA model can be transformed into a Markov process, which may then be used to compute performance measures.

WSCCS and PEPA are different in many ways. WSCCS is a synchronous calculus, while PEPA is an asynchronous one. *Asynchrony* is the assumption that concurrent agents run at different speeds, while *synchrony* is the assumption that they run in lockstep [15]. So far nothing suggests the preservation of synchrony in modelling immune system behaviour.

Upon reaction to a given stimulous, a cell may aim to interact with a specific target interlocutor. The messages exchanged can be simultaneously received by other components, causing secondary, side reactions. The message content is the same, but the intensity of it, the cytosine dose, is lower. Thus modelling immune system behaviour involves *multiple process communication*. Most asynchronous process algebras support this feature or can be easily extended to do so. Including it within synchronous process algebras is awkward though, since agent inaction may cause the entire system to deadlock.

Mobility and distributivity can be captured using a *higher-order process calculus*, such as the π-calculus [16, 17] or fusion [20]. For example, in the π calculus, action names, as well as processes themselves, are communicated allowing the specification of dynamic reconfiguration of the process linkage. Thus an agent's capabilities of interaction evolve through time, implicitly modelled via transition. These changes in process structure express mobility: process behaviour depends upon ubiquity. A mobile process calculus is indeed required for producing a precise account for an immune system. However, the model is far more complex and so here we constrain ourselves to a more modest, limited but intuitive model, which involves only a single site.

Given that we constrain ourselves to a singled-site system, our model does not include clustering either. Clustering requires specifying coordination and competence protocols, which increases the complexity of the model.

We have taken a few steps towards verifying if we can build a process algebra from the union of the ones above mentioned [22]. A preliminary model of the immune system is available upon electronic request to the author.

3 Immune System Analysis

The immune system protects the body against invading organisms. It has a number of properties computer scientist aim to reify in contemporary computer systems. Example of these properties are effectiveness, reliability, robustness, adaptability and self-stabilisation—to mention a few—. In this section, we aim to show how some of these properties can be expressed and then analysed. Our formalism relies on the notion of immune system configuration.

The configuration of a system, as defined in §2.1, is the size of each component's population, including the link structure. Upon action execution, the system configuration changes. Thus, we can take action execution to be a language generation process: the output language being the set that contains all possible configurations. This approach is often used in the analysis of infinite-state systems [18].

Effectiveness means that the immune system often gets rid of both all antigens and all infected cells. Thus, it can be modelled as follows: Given an initial configuration with a non-empty population of invaders, the system will eventually become idle, getting rid of the non-self. Effectiveness can be formally verified using methods borrowed from probabilistic process calculi: A system is effective if the expected value of non-self population tends to zero.

Using the notion of configuration, we can model other properties.

Self-stabilisation is achieved if, after activity, the immune system goes back to a configuration similar to the initial one. Put differently, an idle system does not evolve. Thus, the expected size of the configuration of an infected immune system should tend to the size of that configuration after eliminating the non-self.

Robustness is achieved if, no matter the population of a particular component, the system does not loose effectiveness. Thus, the expected value of non-self population should tend to zero, even though we omit part of the immune system population from the initial configuration.

There are of course properties, such as adaptability, that cannot be even specified within our model. That would take giving a full account of learning.

To formally analyse our system, we only need existing proof methods. However, these methods ought to be re-implemented so that they can be used to analyse partial system behaviour. This is because, prior to verification, existing methods first compute the entire system behaviour but this is not always possible, especially when the system space is infinite. The space associated with our immune system is essentially infinite and so is the language it generates. This is because, the configuration evolves dynamically. Accordingly, we need to do our observations along a predefined number of transitions.

4 Related Work

Hofmeyr and Forrest have invented ARTIS, an architecture for immune computer systems [12]. While ARTIS involves the basic ingredients of an immune system, the authors do not develop the underlying model. Rather they include only operational ideas associated with immunology — attack detection, learning and rejection.

Artificial Immune System, AIS [1], portrays a collection of mechanisms that combine characteristics found in some elements of a natural immune system (cells and proteins). The mechanisms are used to address intrusion detection, so the underlying model does not consider the entire variety of system elements. A similar argument applies to both [4] and [3, pages 242-261]. Other applications of immune system phenomena can be found in [23].

More related to ours is the work of Hatcher, Tofts and Dunn [8, 7, 6]. They have applied WSCCS to computer modelling issues of biological systems, such as sex ratio, sex determination and parasite transmission rate. Focusing on a very specific problem, they have been able to formally analyse or simulate the behaviour of biological systems. Their papers have been a source of inspiration for the work presented here.

5 Conclusions

We have argued that process algebras is an appropriate means to model the immune system. The intended model would involve only aspects of behaviour, ignoring other important subtleties, such as learning and detection. The model could be used to analyse immune system's expected population, with which we can address properties like effectiveness, robustness and self-stabilisation. To model the immune system, we need a process algebra combining existing ones. To conduct formal analysis, we need to extend existing techniques so as to incorporate the analysis of partially developed infinite-state systems. While this development is time consuming, the associated payoffs are worthwhile. We reckon the resulting tool could be used as a controlled laboratory by biologists.

Acknowledgments

This research was partially supported by three grants: CONACYT 33337-A, CONACYT-DLR J200.1442/2002 and ITESM CCEM-0302-05. This paper is an extended abstract of another that appeared in the proceedings of MICAI'02 [22], one of the winners of best ten paper award.

References

1. R. Belew and S. Forrest. Learning classifier systems, from foundations to applications. In P. L. Lanzi, W. Stolzmann, and W. Wilson S, editors, *IWLCS'99*, volume 1813 of *Lecture Notes in Computer Science*. Springer, 2000.
2. J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.
3. D. Dasgupta. *Artificial Immune System and Their Applications*. Springer, U.S.A., 1998.
4. P. DeHaeseleer, S. Forrest, and P. Helman. A immunological approach to change detection: Algorithms analisis and implications. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 110–119. IEEE Computer Society Press, 1996.
5. S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, May 2001.
6. M.J. Hatcher, A.M. Dunn, and C. Tofts. The effect of the embryonic bottleneck on vertically transmitted parasites. In *Proceedings of 1st International Conference on Information Processing in Cells and Tissues*, page In Press. University of Liverpool, 1996.
7. M.J. Hatcher and C. Tofts. The effect of point of expression on ess sex ratios. *Journal of Theoretical Biology*, 175:263–266, 1995.
8. M.J. Hatcher and C. Tofts. The evolution of polygenic sex determination with potential for environmental manipulation. Technical Report Series, Deptarment of Computer Science UMCS-95-4-2, University of Manchester, 1995.
9. M. Hennessy and H. Lin. Proof systems for message passing process algebras. *Formal Aspects of Computing*, 8(4):379–407, 1996. Also available from Sussex as Computing Science Technical Report 3/93.
10. Jane Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
11. C.A.R. Hoare. Communicating sequential processes. *Communications of the Association for Computing Machinery*, 21(8):666–77, 1978.
12. S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473, 2000.
13. ISO. *Information processing systems - Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*. ISO 8807, 1989.
14. R. López, F. Díaz, and S. Arias. *Biología Celular*. Editorial Iberoamérica, México, 1991.
15. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
16. R. Milner, J. Parrow, and D. Walker. A calculus for mobile processes, pt. 1. *Information and Computation*, 100(1):1–40, 1992. Also available from Edinburgh, as Research Report ECS-LFCS-89-95.
17. R. Milner, J. Parrow, and D. Walker. A calculus for mobile processes, pt. 2. *Information and Computation*, 100(1):41–77, 1992. Also available from Edinburgh, as Research Report ECS-LFCS-89-96.
18. F. Moller. Infinite results. In U. Montanari, editor, *CONCUR'96: Concurrency Theory*, pages 195–216. Springer-Verlag, 1996. Lecture Notes in Computer Science, v. 1119.

19. F. Murad. Discovery of the biological effects of nitric oxide and its role in cell signalling. *Bioscience Reports*, 19:133–154, 1999.
20. J. Parrow and V. Björn. The Fusion Clculus: Expressiveness and Symmetry in Mobile Processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 176–185, 1998.
21. S. Robbins and R. Cotran. *Pathologic Basis of Disease.* W B Saunders Co, U.S.A., 1984.
22. R. Saab, R. Monroy, and F. Godínez. Towards a model for an immune system. In C.A. Coello-Coello, A. De Albornoz, L.E. Sucar, and O. Cairó, editors, *2nd Mexican International Conference on Artificial Intelligence, MICAI'02*, pages 401–410. Springer-Verlag, 2002.
23. L. A. Segel and I. R. Cohen. *Design Principles for the Immune System and Other Distributed Autonomous Systems.* Oxford University Press, New York, U.S.A., 2000.
24. D. Stites and T. Abba. *Medical Immunology.* McGraw-Hill, 1997.
25. C. Tofts. Processes with probablities, priority and time. *Formal Aspects of Computing*, 6(5):536–564, 1994.

KRHyper Inside — Model Based Deduction in Applications

Peter Baumgartner,
Ulrich Furbach,
Margret Gross-Hardt,
Thomas Kleemann, and
Christoph Wernhard

Institut für Informatik, Universität Koblenz-Landau, D-56070 Koblenz, Germany,
{peter,uli,margret,tomkl,wernhard}@uni-koblenz.de

Abstract. Three real world applications are depicted which all have a full first order theorem prover based on the hyper tableau calculus as their core component. applications concern information retrieval in electronic publishing, the integration of description logics with other knowledge representation techniques and XML query processing.

1 Introduction

Automated theorem proving is offering numerous tools and methods to be used in other areas of computer science. An extensive overview about the state of the art and its potential for applications is given in [9]. Very often there are special purpose reasoning procedures which are used to reason for different purposes, like e.g. knowledge representation [22] or logic programming [15].

The most popular methods used for practical applications are resolution-based procedures or model checking algorithms. In this paper we want to demonstrate, that there is an important potential for model based procedures. Model based theorem proving can be based very naturally on tableau calculi [19], and in particular there is a line of development, which started with the *SATCHMO* approach [28] and was later refined and extended towards the hyper tableau calculus [8].

In this paper three real world applications are depicted which all have a full first order theorem prover based on the hyper tableau calculus as their core component. I.e. deduction is not used to produce or verify the software, but a deduction system is a part of the running application system. The three applications are

- information retrieval in electronic publishing
- reasoning in description logic and knowledge representation
- query answering and optimization in XML databases

These applications all stem from research and development projects, which do not deal with automated reasoning primarily. The model generation theorem prover was an obvious tool for the purposes of these projects, because in each of the tasks it was not just a yes/no answer required, but the model to be returned by the prover was the answer to the query in the respective application.

In the following section we shortly depict the hyper tableau prover and on this basis we can describe the applications in the successive sections.

2 Theorem Proving with Hyper Tableau

2.1 Features

In the hyper tableau approach application tasks are specified using first order logic — plus possibly non-monotonic constructs — in clausal form. While a hyper tableau prover can be used straightforwardly to prove theorems, it also allows the following features, which are on one hand essential for knowledge based applications, but on the other hand usually not provided by first order theorem provers:

1. Queries which have the listing of predicate extensions as answer are supported.
2. Queries may also have the different extensions of predicates in alternative models as answer.
3. Large sets of uniformly structured input facts are handled efficiently.
4. Arithmetic evaluation is supported.
5. Non-monotonic negation is supported.
6. The reasoning system can output proofs of derived facts.
7. The system can be used as reasoner for a description logic, enhanced with rules and ABox reasoning capability.

Hyper tableau is a "bottom-up" method, which means that it generates instances of rule[1] heads from facts that have been input or previously derived. Derived facts are stored as lemmas. This has the heuristic effect of avoiding redundant re-computations and supports the use of the calculus for model generation, since it makes it possible to detect when a fixed point of rule application is reached.

If a hyper tableau derivation terminates without having found a proof, the derived facts form a representation of a model of the input clauses.[2]

A rule head may be a disjunction. In hyper tableau, disjunctions are handled by exploring the alternative branches in a systematic way. Backtracking can be used to generate one model after the other.

[1] We use Prolog notation for clauses throughout this paper: A clause is viewed as rule "*Head* `:-` *Body*.", where *Head* consists of its positive literals, combined by "`;`" (*or*), and *Body* consists of its negative literals, combined by "," (*and*). If a clause contains only positive literals, i.e. is a fact or a disjunction, it is notated as "*Head*.", if it contains only negative literals, as "`false :-` *Body*.".

[2] The Herbrand model output consists of all ground instances of the derived facts. Since the derived facts must not necessarily be ground, in some cases they can characterize an infinite model.

Of the features listed above, items 1 and and 2, the generation of answer sets, are made possible through model generation.

Large sets of uniformly structured input facts play a role comparable to base relations of databases in conventional applications, however nested and incomplete data structures can be represented by terms. So item 3 benefits from implementation techniques used in database systems, which can be smoothly integrated with the hyper tableau method.

The handling of special language constructs, items 4 and 5, is facilitated by two aspects of the controlled way in which the hyper tableau calculus builds up data structures: First, it does not generate new clauses with negative literals, which means that only input clauses have negative literals, and so information about the context in which special predicates will be evaluated is statically available at preprocessing. Second, the implementation of nonmonotonic operations such as negation as failure is facilitated by the possibility to detect when a fixed point of inferencing is reached. Intuitively speaking, we then know, that it is not possible to infer certain information, and can use this knowledge positively.

Regarding item 6, many first order theorem provers can output the proofs of refutations, albeit often in an idiosyncratic syntax that makes it difficult to process them further. For model generation, it is additionally desirable that derivations of the facts belonging to a model are available.

Item 7, the practical suitability as a processor for description logic extended by rules and ABox reasoning, is a consequence of the other features. It is described in more detail in section 4.

2.2 A Small Example

The following example illustrates how our hyper tableau calculus based system, *KRHyper*, proceeds to generate models. Figure 1 shows four subsequent stages of a derivation for the following input clauses:

```
p(a).                                   (1)
q(X, Y) ; r(f(Z)) ; r(X) :- p(X).       (2)
false :- q(X, X).                       (3)
s(X) :- p(X), not r(X).                 (4)
```

KRHyper provides stratified negation as failure. The set of input clauses is partitioned into *strata*, according to the predicates in their heads: if a clause c_1 has a head predicate appearing in the scope of the negation operator `not` in the body of c_2, then c_1 is in a lower stratum than c_2. In the example, we have two strata: the lower one containing clauses (1), (2) and (3), the higher one clause (4).

Stage (I) shows the data structure maintained by the method, also called *hyper tableau*, after the input fact (1) has been processed. One can view the calculus as attempting to construct the representation of a model, the *active branch*, shown with bold lines in the figure. At step (I), this model fragment contradicts for example with clause (2): a model containing `p(a)` must also

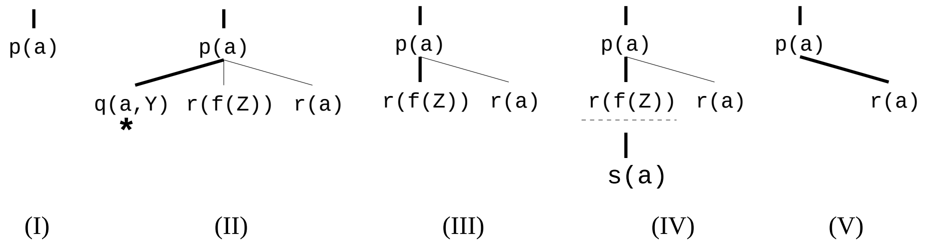


Fig. 1. Stages of a KRHyper derivation

contain all instances of `q(a,Y)` or of `r(f(Z))` or `r(a)`. The model fragment is "repaired" by derivating consequences and attaching them to the hyper tableau: The corresponding instance of clause (2) is attached to the hyper tableau. Since it has a disjunctive head, the tableau splits into three branches. The first branch is inspected and proved contradictory with clause (3). This state is shown in (II).

Computation now tracks back and works on the second branch. With the clauses of the lower stratum, no further facts can be derived at this branch, which means that a model for the stratum has been found, as shown in step (III). Computation then proceeds with the next higher stratum: `s(a)` can be derived by clause (4). Since no further facts can be derived, a model for the whole clause set has been found, represented by the facts on the active branch: {`p(a)`, `r(f(X))`, `s(a)`}, as shown in (IV).

If desired, the procedure can backtrack again and continue to find another model, as shown in state (V). Another backtracking step then finally leads to the result, that there is no further model.

2.3 The KRHyper System

Our system, *KRHyper*, implements the hyper tableau calculus by a combination of semi-naive rule evaluation [29] with backtracking over alternative disjuncts and iterative deepening over a term weight bound. It extends the language of first order logic by stratified negation as failure and built-ins for arithmetic.

For the applications described here, this system is used "embedded" in different ways: As a knowledge maintenance and processing unit in the server component of a client-server system and as target system for the transformation of a higher level language and a knowledge representation language based on description logic.

3 Living Book — Electronic Publishing

Living Book [7] is an electronic book system based on the *Slicing Information Technology (SIT)* [13] for the management of personalized documents: Documents or textbooks are fragmented into small semantic units, so called *slices* or *units*, such as e.g. the definition of a concept, an example, an exercise or a paragraph. Slicing Information Technology evolved from an electronic library system for mathematics, the *ILF Mathematical Library*, which was developed within the *Deduction* research program of Deutsche Forschungsgemeinschaft in the nineties.

Meta data play an important role to describe dependencies among slices, which may originate from a single document or from different ones. Keywords can be assigned to slices to indicate their contents. The process of "slicing", i.e. fragmenting and annotating given documents such as manuals or mathematical textbooks, is partially automated, but usually needs some further manual work.

The Living Book system has a client-server architecture, which is shown in figure 2.

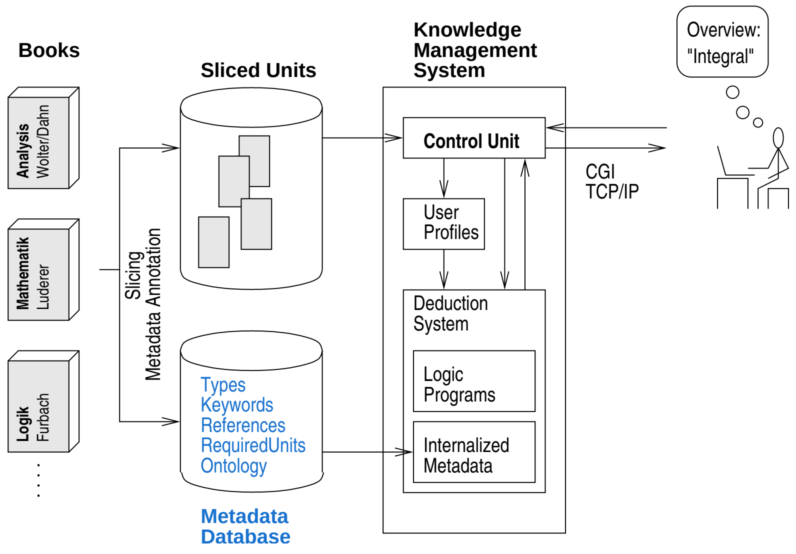


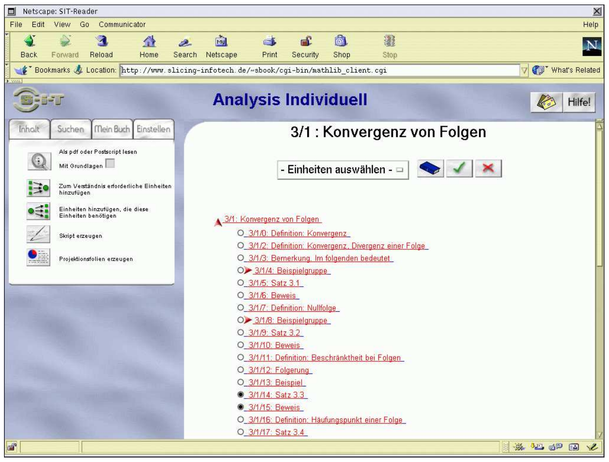
Fig. 2. Living Book — system architecture

3.1 The User View

A client side program, the *SIT-Reader*, offers all functionality of the system to the user through a common Web browser; figure 3 shows a screenshot.

To use the system, the user can mark units, like e.g. `analysis/3/1/15`[3] and `analysis/3/1/16` representing e.g. theorem 3.1.15 in the analysis book together with its proof. Then she can tell the system that she wants to read the marked units and gets a generated PDF document containing just those units. If the user thinks that this information is not sufficient for her understanding, she can tell the system to include all units which are prerequisites of the units selected.

But also more information about the user's profile can be incorporated in generating tailored documents: she may select a certain chapter, say e.g. chapter 3 containing everything about integrals in the analysis book. But instead of requesting all units from this chapter the user wants the system to take into account that she knows e.g. unit 3.1 already, and she possibly wants just the material that is important to prepare for an exam. Based on the units with their meta data, the deduction system can exploit this knowledge and combine the units to a new document (hopefully) fitting the needs of the user.


Netscape: SIT-Reader
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.slicing-infotech.de/~sbook/cgi-bin/mathlib_client.cgi
What's Related
Analysis Individuell
Hilfe!
Inhalt Suchen Mein Buch Einstellen
Als pdf oder Postscript lesen
Mit Grundlagen
Zum Verständnis erforderliche Einheiten hinzufügen
Einheiten hinzufügen, die diese Einheiten benötigen
Skript erzeugen
Projektionsfolien erzeugen
- Einheiten auswählen -

3/1 : Konvergenz von Folgen

3/1: Konvergenz von Folgen
- ○ 3/1/0: Definition: Konvergenz
- ○ 3/1/2: Definition: Konvergenz, Divergenz einer Folge
- ○ 3/1/3: Bemerkung. Im folgenden bedeutet
- ○ 3/1/4: Beispielgruppe
- ○ 3/1/5: Satz 3.1
- ○ 3/1/6: Beweis
- ○ 3/1/7: Definition: Nullfolge
- ○ 3/1/8: Beispielgruppe
- ○ 3/1/9: Satz 3.2
- ○ 3/1/10: Beweis
- ○ 3/1/11: Definition: Beschränktheit bei Folgen
- ○ 3/1/12: Folgerung
- ○ 3/1/13: Beispiel
- ● 3/1/14: Satz 3.3
- ● 3/1/15: Beweis
- ○ 3/1/16: Definition: Häufungspunkt einer Folge
- ○ 3/1/17: Satz 3.4

Fig. 3. Living Book — screenshot

In conclusion, we not only have the text of the books, we have an entire knowledge base about the material, which can be used by the reader in order to generate personalized documents from the given books. If we are running the system with three books in combination, we have altogether more the 12.000 facts and between 50 and 100 rules in the knowledge base.

3.2 The Knowledge Management System

From the viewpoint of deduction, the most interesting component of Living Book is the knowledge management system on the server side. As shown in figure 2, the knowledge management system handles meta data of various types: **Types** of units (*Definition, Theorem* etc), **Keywords** describing what the units are about (*Integral* etc), **References** between units (e.g. a *Theorem* unit about *Integral* refers to a *Definition* unit), and what units are **Required** by other units in order to make sense.

[3] "/" is a binary function symbol, written as right-associative infix operator.

Further, a **User Profile** stores what is *known* and what is *unknown* to the user. It may heavily influence the computation of the assembly of the final document. The user profile is built from *explicit* declarations given by the user about units and/or topics that are known/unknown to him. This information is completed by deduction to figure out what other units must also be known/unknown according to the initial profile.

3.3 The Logic Behind

On a higher, research methodological level the deduction technique used in the knowledge management system is intended as a bridging-the-gap attempt. On one side, our approach builds on results from the area of *logic-based knowledge representation and logic programming* concerning the semantics of (disjunctive) logic programs (see [10] for an overview). On the other side, our KRHyper system used in Living Book is built on calculi and techniques developed for *classical first order classical reasoning.* To formalize our application domain we found features of both mentioned areas mandatory: the logic should be a first order logic, it should support a default negation principle, and it should be "disjunctive". To our knowledge, such a "cross-over" is novel, and therefore we will motivate the logic used by some examples now.

First order Specifications. In the field of knowledge representation, and in particular when non-monotonic reasoning is of interest, it is common practice to identify a clause with the set of its ground instances. Reasoning mechanisms often suppose that these sets are finite, so that essentially propositional logic results. Such a restriction should not be made in our case. Consider the following clauses, which are actual program code in the knowledge management system about user modeling:

```
unknown_unit(analysis/1/2/1).                (1)
known_unit(analysis/1/2/_ALL_).              (2)
refers(analysis/1/2/3, analysis/1/0/4).      (3)
known_unit(Book_B/Unit_B) :-                 (4)
        known_unit(Book_A/Unit_A),
        refers(Book_A/Unit_A, Book_B/Unit_B).
```

The fact (1) states that the unit named `analysis/1/2/1` is "unknown"; the fact (2), the `_ALL_` symbol stands for an anonymous, universally quantified variable. Due to the `/`-function symbol (and probably others) the Herbrand-Base is infinite. Certainly it is sufficient to take the set of ground instances of these facts up to a certain depth imposed by the books. However, having thus exponentially many facts, this option seems not really a viable one. The rule (4) expresses how to derive the known-status of unit from a known-status derived so far and using a refers-relation among units.

Default Negation. Consider the following program code, which is also about user modeling:[4]

```
%% Actual user knowledge:
known_unit(analysis/1/2/_ALL_).                (1)
unknown_unit(analysis/1/2/1).                  (2)
refers(analysis/1/2/3, analysis/1/0/4).        (3)

%% Program rules:
known_unit_inferred(Book/Unit) :-              (5)
       known_unit(Book/Unit),
       not unknown_unit(Book/Unit).
unknown_unit_inferred(Book/Unit) :-            (6)
       unit(Book/Unit)
       not known_unit_inferred(Book/Unit).
```

The facts (1), (2) and (3) have been described above. It is the purpose of rule (5) to compute the known-status of a unit on a higher level, based on the `known_units` and `unknown_units`. The relation called `unknown_unit_inferred`, which is computed by rule (6) is the one exported by the user-model computation to the rest of the program.

Now, facts (1) and (2) together seem to indicate inconsistent information, as the unit `analysis/1/2/1` is both a `known_unit` and a `unknown_unit`. The rule (5), however, resolves this apparent "inconsistency". The pragmatically justified intuition behind is to be cautious in such cases: when in doubt, a unit shall belong to the `unknown_unit_inferred` relation. Also, if nothing has been said explicitly if a unit is a `known_unit` or an `unknown_unit`, it shall belong to the `unknown_unit_inferred` relation as well. Exactly this is achieved by using a default negation operation `not`, when used as written, and when equipping it with a suitable semantics[5].

Disjunctions and Integrity Constraints. Consider the following clause:

```
computed_unit(Book1/Unit1) ;
computed_unit(Book2/Unit2) :-
       definition(Book1/Unit1,Keyword),
       definition(Book2/Unit2,Keyword),
       not equal(Book1/Unit1, Book2/Unit2).
```

It states that if there is more than one definition unit of some `Keyword`, then (at least) one of them must be a "computed unit", one that will be included in the generated document (the symbol `;` means "or"). Beyond having proper disjunctions in the head, it is also possible to have rules without a head, which act as integrity constraints.

[4] The `not` operator has been illustrated by the example in section 2.2.

[5] Observe that with a classical interpretation of `not`, counterintuitive models exist. We use a variant of the perfect model semantics for stratified disjunctive logic programs.

4 Knowledge Representation Beyond Description Logic

In this section we will argue that automated deduction techniques, in particular those following the model computation paradigm, are very well suited for knowledge representation purposes. This is an argument leaving the mainstream of knowledge representation research, which currently has its focus on the development of description logic (DL) systems. We want to point out that we consider the DL direction of research extremely successful: it led to a deep insight into computational properties of decidable subclasses of first order reasoning; it made clear some interesting links to non-classical logics, and, moreover, DL systems are nowadays outperforming most modal logic theorem provers. Despite these successful developments we find two reasons which motivate our approach to use a first order theorem prover for knowledge representation purposes instead of dedicated description logic systems.

First, even the key researchers in the field of description logics are stating some severe deficiencies of their systems (e.g. [18]): research into description logics focused on algorithms for investigating properties of the terminologies, and it is clear that for realistic applications the query language of description logic systems is not powerful enough. Only recently has the community investigated seriously the extension of description logic systems towards ABox and query answering, which is not trivial [24, 23].

Second, the most advanced systems are essentially confined to classical semantics and do not offer language constructs for non-monotonic features (which are a core topic in another branch of the knowledge representation community). Although there are some results on extending DL languages with nonmonotonic features [4], it seems that this direction of research is vastly unexplored.

Additionally, it has been widely recognized that adding to the terminological language of DL a complementary knowledge representation scheme based on rules (as used in logic programs) would greatly improve expressivity. This issue is currently addressed in particular within the *Semantic Web* context, but no really convincing solution has been found so far [6, 16, 17].

As mentioned above, our focus is on the development of a language and system that combines a terminological language with a rule-like language and nonmonotonic features. The specifications may be "mixed", in the sense that concepts and roles defined in the terminological part may be used or further extended/constrained in the logic program part. Regarding computation with such specifications, we follow a model-computation paradigm. That is, a bottom-up procedure is employed that computes a (minimal) model of the whole specification. (The usefulness of the model-generation paradigm in general and in particular in conjunction with DL will be argued for below and in other parts of this paper.) The computation uses a naive transformation of the description logic syntax into first order predicate logic. Clearly, we are loosing decidability in the general case; however, being careful with the definition of knowledge bases in the application, we can retain decidability. With respect to performance we give some figures in the conclusion. In the following we will roughly sketch the system we are targeting at.

4.1 Kernel + KRHyper

Our approach is oriented at the paradigm of logic programming and model-based theorem proving. Instead of starting with a small and efficient kernel language like ALC,[6] which is stepwisely extended towards applicability, we start with a rather general DL language and the rather general language of first order logic programs, and then we identify sub-languages that are decidable and practically feasible.

Our approach is a transformational one, which embeds DL into logic programming by translating DL constructs into logic programming constructs. This way, the semantics of the original specification is given the semantics of the resulting logic program. The largest subclass that we can handle is that of the Bernays-Schönfinkel fragment extended by a default-negation principle. The user of our system can decide to stay within this class or whether she wants to use some language constructs which leave this class.

Our approach can be summarized as

$$\mathsf{Kernel} + \mathsf{KRHyper}$$

where

– *Kernel* is an OIL-like language which is augmented by some additional constructs, like non-monotonic negation and second-order features (reification).
– *KRHyper* means the extended first order predicate logic which can be processed by our system. As a logic programming system, it provides rules, axioms, constraints and concrete domains.

4.2 The Kernel Language

OIL class definitions, e.g.

```
class-def defined carnivore
       subclass-of animal
       slot-constraint eats
           value-type animal
```

have a similar concrete syntax in our kernel language. Most parts of OIL are covered, in particular all kinds of class definitions, inverse roles, transitive roles etc. The constructs from the Kernel language are translated to our logic programming language following standard schemes.

Beyond this, we are able to handle the following points which are mentioned explicitly as missing in [18]:

[6] I.e. concept descriptions are formed using the constructors negation, conjunction, disjunction, value restriction and existential qualification.

Rules and Axioms. In addition to constructs in the syntax of the knowledge representation language we can use arbitrary formulae as constraints, rules or axioms. For instance, we can state in the rule part

```
dangerous(X) :- carnivore(X), larger_than(30,X).
```

to express sufficient conditions for being `dangerous`. The `larger_than` relation would be defined by the user as a binary predicate.

Using Instances in Class Definitions. Although it is well known (cf. [3]) that reasoning with domain instances certainly leads to EXPTIME-algorithms, it is very clear that exactly this is mandatory in practical applications. For instance, the previous example could also be supplied as[7]

```
dangerous <= carnivore & larger_than(30).
```

in the terminological part.

Default Reasoning. In our system we included a closed world assumption, such that we can use a default negation principle "not " following the perfect model semantics. Default negation may be used both in the rule part and in the terminological part. For the latter case, the previous example might more appropriately be written as

```
dangerous <= carnivore & not smaller_than(30).
```

Switching Back and Forth. One may switch back and forth between the terminological part and the rule part, by keeping in mind that concepts translate into unary predicates, and that roles translate into binary predicates.

ABoxes. Concrete instances of concepts (roles) are handled via unary (binary) predicates. This is a very natural and well-understood method for model generation procedures. For instance, from

```
dangerous <= carnivore & not smaller_than(30).
```

and the ABox consisting solely of

```
carnivore(leo).
```

the model generation prover will derive `dangerous(leo)`. Unlike other systems, no grounding in a preprocessing phase takes place, and the system is capable of computing with ABoxes consisting of tens of thousands of objects.

[7] Notice that description logic languages such as OIL usually permit concept definitions via equivalences (`<=>` in our syntax) or via necessary conditions (`=>` in our syntax). However, we start off with a concrete ABox that is assumed to implicitly represent a model of some TBox, and that can be extended to an explicitly represented model by using sufficient conditions, as shown.

Limited Second-Order Expressivity. Very often it is necessary to treat statements of the language as objects and to apply procedures for some kind of evaluation to them. This can be done in our context by meta-language constructs à la Prolog. For instance, via `concept_instance(Concept,Instance)` one has access to the `Concept` names where `Instance` is an instance of. For example,

```
all_dangerous(X) :-
        call(findall(Z,
            (dangerous(Y), concept_instance(Z,Y)),X)).
```

describes as a Prolog-list all the concepts that have an instance of the `dangerous` concept.

4.3 Sample Application

This method of using Kernel + KRHyper for reasoning in description logic is the core of an application we built for a major German bank. It is a knowledge management system, which is used as decision support for the communication department of the bank. An important characteristic of this system, is that it contains besides the TBox a large ABox, containing press articles and excerpts thereof. The reasoning mechanism is Kernel + KRHyper which in particular has to deal with the ABox. The system is extended with a graphical user interface, which allows easy modifications of the ABox and the TBox and which supports the usual queries one wants to get answered from knowledge base. The system is realized as a client-server architecture and can be used via an ordinary web browser. This knowledge management system is already in use in the bank and, currently it is extended with automatic learning-based mechanisms, in order to extend the knowledge base.

Working on this application brought two rather trivial, but nevertheless important aspects to our attention. One is, that the user of such an application does not care at all, which technique is inside the system. From a deduction point of view, we are happy seeing a theorem prover running as part of an application software, the user, however, is only interested in things like reliability, efficiency and costs. We learned, that the use of an existing theorem prover helps to meet exactly these requirements. The second lesson we learned is, that it is extremely difficult for people who are not computer scientists to use a language of description logic for defining concepts or to express complex queries. Although we defined a Windows-like graphical user interface, we found that users encounter difficulties to use it. Currently we are redesigning the interface of the system, such that the knowledge representation format is completely hidden behind functionalities which are only from the application domain.

5 Flexible Database Queries for XML Data

In the last few years, semistructured data and in particular XML have played an increasing role within the database community. Databases for XML data have evolved in the context of database integration tasks (*Enterprise Application Integration, EAI*); these databases have a strong focus on data that is structured irregularly, is implicit, incomplete and therefore often result in large schemas [1].

Various query languages for semistructured data have been proposed in order to deal with the complex and nested structure of these data. Many languages use path queries [26, 14, 12, 2, 25] which have emerged as an important class of browsing-style queries on the Web. Navigational database access by these path queries require the user to know lot of structural details. It has been pointed out that declarative query constructs are needed in order to reduce or even avoid explicit navigation through the data [20, 11].

Here, we present an approach that relies on the assumption that techniques from logics and in particular knowledge representation, as e.g. subsumption of XML types [27] may be useful for querying XML data.

5.1 Example

```
<university>
  <researcher>
    <name>Smith</name>
    <publications>
      <monograph>
        <title>Basics of databases</title>
        <author><name>Smith</name>
        </author>
        <isbn>7899</isbn>
        <subject>DB</subject>
      </monograph>
      <article>
        <title>Flexible queries</title>
        <author><name>Smith</name>
        </author>
        <author><name>Miller</name>
        </author>
        <proceeding>VLDB 2000
        </proceeding>
      </article>
      <publication>
        <title>XML Schema</title>
        <author>
          <name>Smith</name>
        </author>
      </publication>
    </publications>
  </researcher>
  <-- more researcher elements -->
  <library>
    <books>
      <book>
        <title>Databases</title>
        <author>
          <name>Smith</name>
        </author>
        <isbn>1234</isbn>
      </book>
      <book>
      <!-- more book elements -->
      </book>
    </books>
  <library>
</university>
```

Fig. 4. Example XML data

Let us consider an example XML document representing a university with a library and researchers working in the university. A library consists of books where each book has a title, an author and an ISBN. Researchers have a name and an associated set of publications e.g. articles, monographs or some general kind of publication without further specialization. Figure 4 shows an excerpt from the represented data.

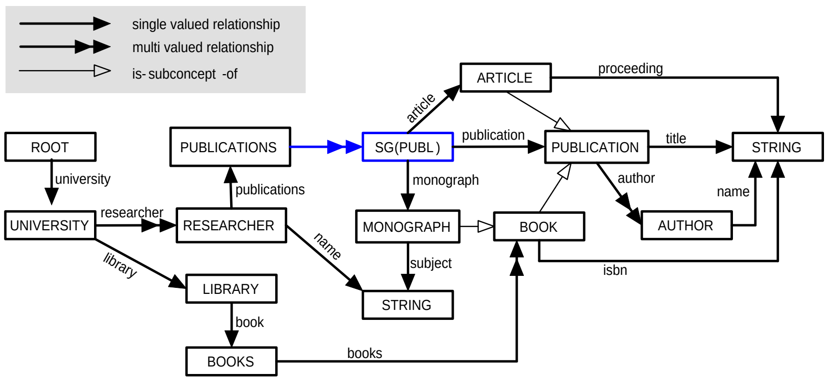


Fig. 5. Schema for database from figure 4

The content of a database is described by means of a schema; often for XML data *Document Type Definitions (DTDs)* are used. Recently, *XML Schema* [31] has evolved with the advantage, that user defined types can be represented; furthermore, besides referential relationships between elements of the XML data, XML Schema provides the possibility to model generalization and specialization relationships. Figure 5 uses a graphical notion to represent an example schema for the database in figure 4.

5.2 Querying the Data

Existing query languages use path queries, that navigate along the structures of the XML data. For instance, in order to access the name of all researchers of a university in *XPath* [30] we use `/university/researchers/researcher/name`. Path queries usually allow some form of "abbreviation". For instance, with `//researcher/name` we address all descendants of the "root" that are *researcher*-elements and navigate to their names. However, because path queries work directly on the XML data and not on the schema, it is not possible to query those elements from a data source, that belong to the same type or concept. In particular, in order to ask e.g. for all kinds of publications, we would have to construct the union of path queries navigating to publication, book, article and monograph, explicitly.

This problem has been addressed in [21], where the notions and the issues have been described. One possibility is to add a concept based query facility by means of graph based technology. Another possibility, presented here, is to rely on existing technologies and in particular systems from the area of description logics [5]. In DL, retrieving instances from a certain concept is a well known requirement and standard services are available out-of-the-box in existing DL systems such as *RACER* [22].

In order to use the DL-system, we represent the types in an XML schema by concept expressions in DL as follows. For every type c in the schema with attributes (outgoing edges) $a_1, a_2, \ldots, a_n$ leading to types $t_1, t_2, \ldots, t_n$ a concept expression

$$c \equiv \exists a_1.t_1 \sqcap \exists a_2.t_2 \sqcap \ldots \sqcap \exists a_n.t_n.$$

is introduced. Specialization edges from c to c' are represented by inclusion dependencies

$$c \sqsubseteq c'.$$

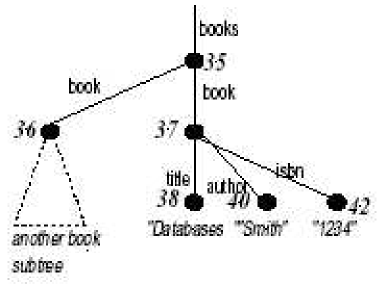


Fig. 6. Tree representation of XML documents

Furthermore, the data in an XML database is represented by ABox-facts. To this end, we represent each element in the database by a unique object identifier (see figure 6 for illustration). The appropriate ABox representation is sketched as follows. For every element o being an instance of concept c we introduce the fact

$$o : c$$

(e.g. o_{36}*:Book*) and furthermore, if the element o has a child o' with tag name a we write:

$$(o', o) : a.$$

For instance, (o_{35}, o_{37}) : *book*. This representation is used when using standard services from DL-systems. For instance, we now can use the service *retrieve-instances* applied to the concept PUBLICATION, in order to retrieve PUBLICATION, MONOGRAPH, BOOK and ARTICLE elements. Flexible query processing therefore makes use of TBox-reasoning (finding the relevant concepts and paths from the database description) and combines this with ABox reasoning and the retrieval of instances.

5.3 Path Completion

By *path completion* we mean, roughly, the problem of deriving from a concept name in a schema (the *start concept*) and an *end concept* a path through this schema connecting the start and the end concept. Such a path then can immediately be turned into a fully specified (e.g.) XPath query. Notice that this way the problems with XPath queries as explained in Section 5.2 can be avoided.

The question remains how to compute path completions. To this end, we propose to build on the DL formalization as described above. Then, it seems natural to appeal to some standard service offered by contemporary DL-systems and let it compute the path completion. Unfortunately we did not find a way of doing so. However, we succeeded in finding a solution based on *model computation*. It works as follows:

1. The start concept, say, c_s, is populated by filling the ABox with $a : c_s$, where a is a fresh name.
2. The DL specification of the schema graph is transformed into clause logic, using the well-known standard mapping. Also, the (singleton) ABox $a : c_s$ is transformed into a fact $c_s(a)$.
3. To the clause logic part the following two clauses are added, where `false :-` $\texttt{c}_e\texttt{(X)}$ is the clause logic transformation of the concept $\neg c_e$:

    ```
    end :- ce(X).
    false :- not end.
    ```

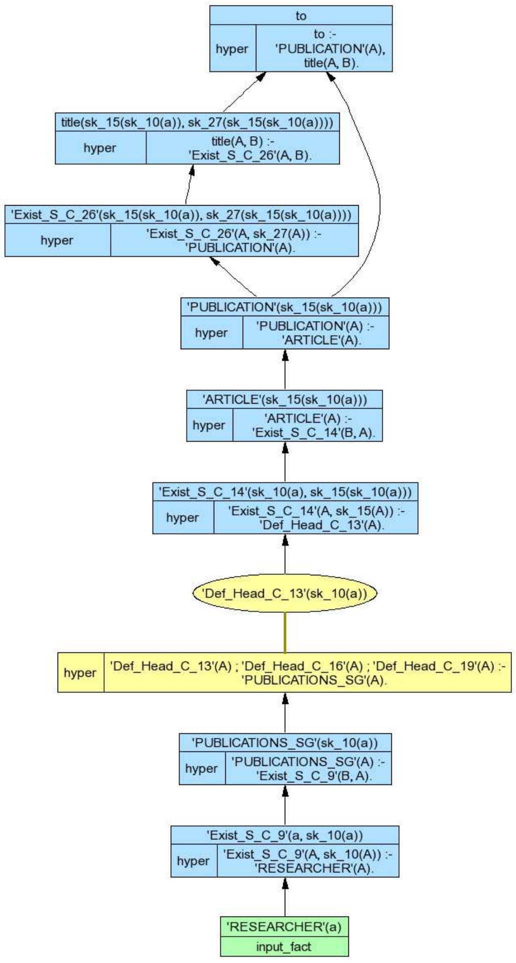

to
hyper
to :- 'PUBLICATION'(A), title(A, B).
title(sk_15(sk_10(a)), sk_27(sk_15(sk_10(a))))
hyper
title(A, B) :- 'Exist_S_C_26'(A, B).
'Exist_S_C_26'(sk_15(sk_10(a)), sk_27(sk_15(sk_10(a))))
hyper
'Exist_S_C_26'(A, sk_27(A)) :- 'PUBLICATION'(A).
'PUBLICATION'(sk_15(sk_10(a)))
hyper
'PUBLICATION'(A) :- 'ARTICLE'(A).
'ARTICLE'(sk_15(sk_10(a)))
hyper
'ARTICLE'(A) :- 'Exist_S_C_14'(B, A).
'Exist_S_C_14'(sk_10(a), sk_15(sk_10(a)))
hyper
'Exist_S_C_14'(A, sk_15(A)) :- 'Def_Head_C_13'(A).
'Def_Head_C_13'(sk_10(a))
hyper
'Def_Head_C_13'(A) ; 'Def_Head_C_16'(A) ; 'Def_Head_C_19'(A) :- 'PUBLICATIONS_SG'(A).
'PUBLICATIONS_SG'(sk_10(a))
hyper
'PUBLICATIONS_SG'(A) :- 'Exist_S_C_9'(B, A).
'Exist_S_C_9'(a, sk_10(a))
hyper
'Exist_S_C_9'(A, sk_10(A)) :- 'RESEARCHER'(A).
'RESEARCHER'(a)
input_fact

Fig. 7. Result of a KRHyper run. See text for explanation.

Now, speaking figuratively, computing a *minimal* model of the thus obtained clause set corresponds to labeling in the schema graph the start concept and propagate the label according to the concept hierarchy and via the roles encountered. Appealing to *minimal* model computation is an issue in order to avoid unwanted population of concepts. Notice that "substitution groups" [31] in the schema graph may introduce disjunctions in the head of the clauses, which may lead to different models or models that do not populate the end concept. It is exactly the purpose of the clauses in 3. above to avoid the latter problem. Observe that as small as the use of the default negation operator "`not`" may seem, it is a very useful feature to filter out unwanted models. The figure on the right shows the result of running our KRHyper prover, where the start concept is $\mathsf{RESEARCHER}$ and the end concept $\mathsf{PUBLICATION} \sqcap \exists \mathsf{title}.\top$.

6 Conclusion

The KRHyper theorem prover is an integral part of all three applications, that derive from quite different fields. Contrasting schemes that limit the use of a prover to the setup or configuration of an application, in our case the prover is a continuously used core component of the depicted systems. Any concerns about performance degradation imposed by the use of a full featured KRHyper system did not come true. Comparisons with a widely respected DL reasoner rendered KRHyper superior with respect to execution time as well as memory consumption. Based on these experiences we are continuing to investigate the applicability of deduction systems in upcoming projects.

References

1. S. Abiteboul. Querying semistructured data. In *ICDT*, pages 1–18, 1997.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
3. C. Areces, P. Blackburn, and M. Marx. A roadmap of the complexity of hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in LNCS, pages 307–321. Springer, 1999. Proc. of the 8th Annual Conf. of the EACSL, Madrid, September 1999.
4. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proc. of the Third Int. Conf.*, pages 306–317, San Mateo, California, 1992. Morgan Kaufmann.
5. F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *Description Logic Handbook*, pages 47–100. Cambridge University Press, 2002.
6. M. Balaban and A. Eyal. Dfl - a hybrid integration of descriptions and rules, using f-logic as an underlying semantics. In *DL*, 1996.
7. P. Baumgartner, U. Furbach, M. Gross-Hardt, and A. Sinner. `'Living Book' :- 'Deduction', 'Slicing', 'Interaction'.` – system description. In F. Baader, editor, *CADE-19 – The 19th Int. Conf. on Automated Deduction*, LNAI. Springer, 2003. To appear.
8. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in LNAI. European Workshop on Logic in AI, Springer, 1996.
9. W. Bibel and P. H. Schmitt, editors. *Automated Deduction. A basis for applications.* Kluwer Academic Publishers, 1998.
10. G. Brewka, J. Dix, and K. Konolige. *Nonmonotonic Reasoning*, volume 73 of *Lecture Notes*. CSLI Publications, 1997.
11. F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Proceedings of the Int. Conf. on Logic Programming (ICLP)*. Springer-Verlag LNCS, 2002.
12. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM Symposium on Principles of Database Systems*, 1994.

13. I. Dahn. Slicing book technology - providing online support for textbooks. In H. Hoyer, editor, *Proc. of the 20th World Conference on Open and Distance Learning*, Düsseldorf/Germany, 2001.
14. J. V. den Bussche and G. Vossen. An extension of path expressions to simplify navigation in oject-oriented queries. In *Proc. of Int. Conf. on Deductive and Object-Oriented Databases (DOOD)*, 1993.
15. J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In A. Voronkov and A. Robinson, editors, *Handbook of Automated Reasoning*, pages 1121–1234. Elsevier-Science-Press, 2001.
16. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Al-log: integrating datalog and description logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.
17. F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.
18. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proc. of the European Knowledge Acquisition Conf. (EKAW-2000)*, Lecture Notes In Artificial Intelligence. Springer-Verlag, 2000.
19. U. Furbach. Automated deduction. In W. Bibel and P. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume I: Foundations. Calculi and Refinements. Kluwer Academic Publishers, 1998.
20. L. L. G. Grahne. On the difference between navigating semistructured data and querying it. In *Workshop on Database Programming Languages*, 1999.
21. M. Gross-Hardt. Querying concepts — an approach to retrieve xml data by means of their data types. In *17. WLP - Workshop Logische Programmierung*, Technical Report. Technische Universität Dresden, 2002.
22. V. Haarslev and R. Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701, 2001.
23. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic SHIQ. In D. MacAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, Germany, 2000. Springer Verlag.
24. I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI'2000, Proc. 17th (U.S.) National Conf. on Artificial Intelligence*, pages 399–404. AAAI Press/The MIT Press, 2000.
25. Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *PODS*, 2001.
26. M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *SIGMOD*, 1992.
27. G. M. Kuper and J. Simen. Subsumption for XML types. In J. V. den Bussche and V. Vianu, editors, *Int. Conf. on Database Theory*. Springer LNCS 1973, 2001.
28. R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In E. L. Lusk and R. A. Overbeek, editors, *Proc. of the 9th Conf. on Automated Deduction*, LNCS, pages 415–434. Springer, 1988.
29. J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, Rockville, Maryland, 1989.
30. W3C. XPath specification. `http://www.w3.org/TR/xpath`, 1999.
31. W3C. XML Schema - part 0 to part 2. `http://www.w3.org/TR/xmlschema-0,-1,-2`, 2001.

Knowledge-Representation and Scalable Abstract Reasoning for Sentient Computing using First-Order Logic

Eleftheria Katsiri[1] and Alan Mycroft[2]

[1] Laboratory for Communication Engineering, University of Cambridge,
[2] Computer Laboratory, University of Cambridge
William Gates Building, 15 JJ Thompson Avenue, Cambridge CB3 0FD, UK
ek236@eng.cam.ac.uk am@cl.cam.ac.uk

Abstract. We present a dynamic knowledge-base maintenance system for representing and reasoning with knowledge about the Sentient Computing environment. Sentient Computing has the property that it constantly monitors a rapidly-changing environment, thus introducing the need for abstract modelling of the physical world which is at the same time computationally efficient. Our approach uses *deductive systems* in a relatively unusual way, namely, in order to allow applications to *register inference rules* that generate *abstract* knowledge from low-level, sensor-derived knowledge. *Scalability* is achieved by maintaining a *dual-layer* knowledge representation mechanism for reasoning about the Sentient Environment that functions in a similar way to a two-level cache. The lower layer maintains knowledge about the current state of the Sentient Environment at sensor level by continually processing a high rate of events produced by environmental sensors, e.g. it knows of the position of a user in space, in terms of his coordinates *x,y,z*. The higher layer maintains easily-retrievable, user-defined abstract knowledge about current and historical states of the Sentient Environment along with temporal properties such as the time of occurrence and their duration e.g. it knows of the room a user is in and for how long he has been there. Such abstract knowledge has the property that it is updated much less frequently than knowledge in the lower layer, namely only when certain threshold-events happen. Knowledge is retrieved mainly by accessing the higher layer, which entails a significantly lower computational cost than accessing the lower layer, thus ensuring that the lower-level can be replicated for distribution reasons maintaining the overall system scalability. This is demonstrated through a prototype implementation.

1 Introduction

Our research is focused on *Sentient Computing* [11] applications. Sentient computing is the proposition that *applications can be made more responsive and useful by observing and reacting to the physical world.* Awareness comes through *sensing* and a *sensor infrastructure* (Fig. 1(a)), distributed in the environment, provides information about the spatial properties of users and objects, i.e. their position in space, their containment within a region such as a room or their proximity to a known physical location. *Sentient Applications* make it possible for the user to perform easily complex computations involving spatial and temporal notions of a dynamic changing environment. E.g. when a user walks into his study at home, it is possible for his PC to automatically and seamlessly display the desktop from his office environment. Or, whenever two people are co-located in a single space, the system can make this information graphically available to an Active Map, or deliver a relevant reminder. e.g. *"You asked me to remind you, to give Tom his book back, when you meet him."* Sentient Computing applications can be viewed as a logical layer, namely, the *Application Layer* in the *Sentient Applications layered architecture* (Fig. 1(a)).

However, there is a significant *gap* between the level of abstraction in the *knowledge* about the Sentient World that Sentient Computing applications require for their functionality and the actual *low-level data* that get produced by the sensors and which constitute a *low-level, precise, knowledge layer.* To illustrate this using the previous example, the sensor that dispatches information about a user's position, only knows who the user is and his coordinates in space. An application that displays the user's screen in response to his proximity to his PC, needs to know when a more abstract situation has occurred, that is, when the user is close to his PC. The information about the user's proximity to his PC is a logical abstraction of his position in space and it is expressed in relation to the position of another physical object, namely his PC. To make matters worse, the above system will need to monitor a large number of users distributed among a number of distinct locations at the same time. Even so, it needs to react to the perceived changes with no perceptible delay.

We propose that the gap between the application-layer abstraction and the sensor-derived precision be bridged by using a *deductive* component that reasons with low-level, sensor-derived knowledge in order to *deduce* high-level, abstract knowledge, which can in turn be used easily by the application layer. Furthermore, we believe that the proposed *deductive reasoning* does not compromise computational efficiency and performance even for very large distributed environments.

This work tackles the above issue of *scalable, system-level, computationally-efficient abstract modelling* of the physical world. Its contributions are a formal definition of a knowledge representation as well as a mechanism for reasoning with such knowledge using *logical deduction* that combines expressiveness, *scalability* and *performance.*

1.1 Layered Interfaces

For the abstract model of the Sentient world we chose to design a dual-layer knowledge representation architecture. Our design approach is inspired by the *OSI paradigm* [14] for layered network architecture where each layer incorporates a set of similar functions and hides lower-level problems from the layer above it thus achieving simplicity, abstraction and ease of implementation.

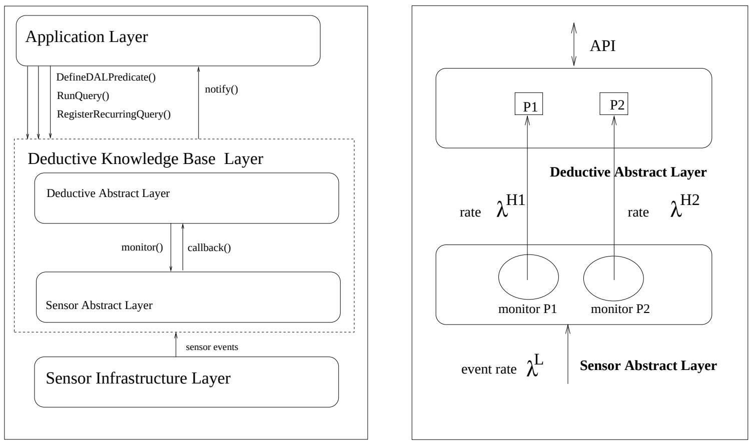


Fig. 1. The Sentient Applications layered architecture and its API (a) andthe dual-level architecture (b) .

1.2 API

The application Layer communicates with the dual Deductive Knowledge base layer via an API consisting of a ***DefineDALPredicate()***, ***RunQuery()*** and ***RegisterRecurringQuery()*** interfaces (see Fig. 1(a)). The *DefineDALPredicate()* interface, takes as arguments the predicate name along with its parameters, and creates the necessary representation in the Distributed Abstract Layer (DAL) for this piece of knowledge. The *RunQuery()* command is similar to an SQL SELECT statement in that it returns the current stored state of the Sentient world by first trying to obtain the information of interest from the Deductive Abstract Layer. If that piece of information is not contained there, a communication process between the two knowledge layers is triggered so that this piece of knowledge is generated from the low-level Sensor Abstract Layer and maintained dynamically in DAL before it is returned to the application layer. In this way, it will be available for future queries.

The *RegisterRecurringQuery()* command is used by the Application Layer in order to register interest in a particular, recurring situation in a way that the application layer is *notified* whenever the situation occurs, starting with its next occurrence. The *RegisterRecurringQuery()* command together with the *notify()* command, behave similarly to the *publish-subscribe* protocol.

The interface between the two layers, used by both API commands, is based on a *monitor-callback* mechanism similar to an *asynchronous invocation* between a *consumer* and a *publisher*. Nomenclature here is taken from the theory of Distributed Systems [2]. The *monitor* mechanism in the Sensor Abstract Layer watches all changes in the environment reported by the Sensor Infrastructure for certain threshold events, as specified in the *RunQuery()* and *RegisterRecurringQuery()* statements. The *callback* mechanism ensures that such threshold events, when they occur, trigger an update in the Deductive Abstract Layer accordingly, creating instances of the predicates that hold and destroying these that are no-longer true, as the case may be.

In this way, the two knowledge layers behave as a *two-level cache* for the Application Layer, enabling scalability.

1.3 Paper Outline

The rest of the paper is organised as follows: First, significant work in related areas is discussed and then, the architecture of the proposed system is described. Section 4 presents a formal definition of the knowledge representation architecture using first-order logic. Sentient Applications reason about the available knowledge through an interface which is described in Section 5. Next, a prototype implementation, discussed in Section 6, confirms the achieved scalability. Last, some conclusions and future work are discussed.

1.4 What is Not a Goal of this Paper.

We have chosen to describe the *deductive component* of the proposed architecture by using *first-order logic*. We believe first-order logic to be a very powerful and general tool for representing and reasoning with knowledge and we have found it to be appropriate and sufficient for the needs of Sentient Computing as is demonstrated in later sections. It is not our goal to provide a comparative study between first-order logic and other *reasoning schemes* such as *planning* and *description logics*, however a concise survey of literature related to planning and description logics can be found in section 2.

2 Related work

Having an efficient and scalable data model is of great importance for Sentient Computing, not only because of the *dynamic* and *rapidly changing* nature of the processed data in this paradigm, but also because of the *heterogeneity* of it. In this section we chose to survey literature in systems that are faced with similar concerns.

SPIRIT [10] is the most influential system in this area and is also the only system which tries to address scalability as an issue which is inherently linked with processing sensor-derived data. SPIRIT is a sophisticated system that provides a platform for maintaining spatial context, based on raw location information, derived from the Active Bat location system. SPIRIT has a similar goal to our proposed architecture in that it offers applications the ability to express relative spatial statements in terms of geometric containment statements. However, its approach towards both data-modelling and scalability is quite different from ours. SPIRIT models the physical world in a bottom-up manner, translating absolute location events for objects into relative location events, associating a set of spaces with a given object and calculating containment and overlap relationships among such spaces, by means of a scalable, spatial indexing algorithm. The indexing algorithm uses a quadtree for indexing spaces of arbitrary sizes and geometries and provides fast and predictable insertion, updating and query of spaces.

Since we believe *scalability* to be an *enabler for distribution*, it is useful to discuss the potential of efficiently distributing an architecture in order to cater for larger sizes of the physical world. In this aspect, although SPIRIT supports parallelism in the level of the storage of the world model objects [25], the distribution capabilities of the above mentioned *spatial indexing algorithm* are unclear. In fact, calculating relationships between spaces which are not stored on the same computer suggests a high communication overhead between the distributed elements which may affect significantly the response time of the algorithm.

Hence, SPIRIT uses a bottom-up, *engineering* approach in order to address scalability in calculating spatial updates in the world model.

FLAME [13, 3] is a development platform [13, 3] providing middleware support for applications by modelling location information. FLAME uses a Spatial Relation Model to assign regions to location events and generate *events* that denote region containment and overlapping.

Apart from the above mentioned development platforms, context-aware [24] applications include Teleporting [23], ComMotion [19], CyberMinder [1] and Proem [17]. The Teleporting System developed at Olivetti Research Laboratory is a tool for experiencing mobile X sessions. It provides a familiar, personalised way of making temporary use of X displays as the user moves from place to place. ComMotion is a GPS-based [4] system [19] which uses location and time information in order to deliver location-related information such as a grocery list, when the user is close to a super-market. CyberMinder is a system that delivers context-aware reminders, based on a specific set of contextual information such as location, time and agenda items. Proem is a peer-to-peer system, which matches pairs of mobile users according to their given profile of preferences. Last, Narayanan [20] presents the notion of grouping together locations that share spatial or logical features.

Our approach differs in that it tackles scalability in a top-down manner. It looks at the *stored data* through a *knowledge perspective*, and uses *logical deduction* in order to produce a mapping from the *heterogenous* data to different levels of abstract knowledge and exploiting the latter property in order to ensure efficient allocation of resources. By separating concerns between three key areas, namely, the sensor infrastructure that instruments the physical world, precise and abstract knowledge about the physical world and the application layer, we achieve a clearer view of abstract knowledge and provide good grounds for scalability in terms of knowledge queries as well as the creation of efficient interfaces. Furthermore, although our work is focused on the Sentient Computing environment, it is modelled in a formal way, which can be reapplied to all the above-described development platforms, independent of the underlying sensor technology, accommodating at the same time application systems such as [19], [1], [17] as users of the proposed architecture. For this reason, we adopted the formalism of *first-order logic*, as we believe this to be a powerful mechanism for reasoning. First-order logic, provides the desired expressiveness that applications need in order to define their requirements in knowledge. Several other alternatives, that may be applicable to *knowledge-representation and reasoning* in this area, are discussed briefly below:

Planning Systems [22] are systems that given a *formula* that represents a *goal* γ they attempt to find a sequence of *actions* that produces a *world state* described by some state description S such that $S \models \gamma$. We say then that the state description satisfies the goal. Although the *state-space* approach of planning systems *looks promising* for Sentient Computing, the goals pursued by planning systems are different and a lot of work needs to be done in this direction to determine any potential contributions in this area.

Description Logics [21] are logic-based approaches for knowledge-representation systems. Description logics represent entities using a "UML"-like language called DL Language. The logic used for reasoning with such entities is derived from first-order logic but is much less powerful and expressive than first-order logic. Description Logics have the advantage that they *allow* for the specification of *logical constraints*. Similarly to planning, any potential contribution of Description Logics to Sentient Computing is yet to be determined.

3 Knowledge Representation

This section describes the architecture (Fig. 1(b)) of the two logically distinct layers of knowledge representation and discusses their functionality as a two-layered cache for the Sentient application layer.

We use a definition introduced by Samani and Sloman in [18], according to which an *event* is a happening of interest that occurs instantaneously at a specific time. Furthermore, we define the *Sentient environment* to be the *physical* environment and the *current logical state* of the Sentient environment to be the set of all known facts about the Sentient environment between an *initial event* and a *terminal event. Initial* and *terminal* events can be any events that are of interest to the Sentient application layer. Based on the above, we can say that the *Sensor Abstract Layer* maintains a *low-level* but *precise* view of the *current logical state* of the Sentient environment, as produced by sensors that are distributed throughout the environment and continually updated through events. Equally, we can say that the Deductive Abstract Layer maintains an *abstract* view of the *current logical state* of the Sentient environment.

A particularly interesting source of events are the ones that characterise the *location* of an object. These are generated by a *location system* such as the Active BAT [10] where the position of users in 3-D space is tracked typically once per second, by means of an ultrasonic transmitter called BAT. The *Sensor Abstract Layer* processes all the generated events and thus knows of the *last position* of all users in the system in terms of their co-ordinates.

A more abstract view about the state of the Sentient environment can easily be *inferred* from the knowledge stored in the Sensor Abstract Layer. E.g. from a user's position (x, y, z) and from a set of known polyhedra that represent regions, the room the user is in follows logically. Furthermore, from a known set of nested polyhedra, additional locations that the user is present in, can also be inferred. [1]

The *Deductive Abstract Layer* (DAL) through its interaction with the Sensor Abstract Layer (SAL) maintains such abstract knowledge about the *current* and *past* states of the Sentient Environment together with temporal information about the *initial events* that triggered them, the duration of each state, as well as when they stopped holding. Such data can be used by statistical models in order to generate a likelihood estimation of situations that may occur in the future, based on their past occurrences [16].

The two layers interact through a monitor-callback communication scheme. A *monitor call* initiated by the Application Layer, causes the Sensor Abstract layer to *filter through* to DAL only those low-level changes that affect the abstract knowledge stored in the Deductive Abstract Layer, thus alleviating the Deductive Abstract Layer from the cost of continually monitoring all the data that are produced by the sensors. Consequently, knowledge in DAL is updated in a significantly lower rate $(\lambda_1^H, \lambda_2^H)$ than it does in SAL (λ^L) ensuring in this way that any large amount of physical data can be processed by replicated SALs, maintaining at the same time the overall system scalability.

Example. In order to illustrate the functionality of the dual-layer architecture in more detail, let us consider the case where the Application layer is interested in receiving notification whenever two or more users are co-located. Through a *RunQuery()* statement initiated by the Sentient Application Layer, unless it already knows about co-located users, DAL will register a *Monitor()* call to SAL in order for the latter to start monitoring the sensor data that signify co-location occurrences. As a result, the Sensor Abstract Layer monitors the incoming events in order to determine from the users' positions whether two or more users are contained in the same room. When this occurs, the respective knowledge about the user's co-location will be generated in the Deductive Abstract Layer through a *callback()* call. All further changes in the position of these users in the Sensor Abstract Layer are monitored in order to determine whether the two users are still co-located. If any of the co-located users exits the containing region, the change in the users' location in combination with the co-location predicate in-

[1] The query: "Is user X in Cambridge?" needs to answer positively even if User X is in FC15, which is in the William Gates Building in Cambridge.'

stance in the current, abstract state (DAL), signals an inconsistency. As a result, another *callback()* call is triggered from SAL to DAL, invalidating the current state, logging it as a historical state and generating a new current state. In practice, not the whole state of the Sentient Environment is changed, as most of abstract knowledge remain unaltered. Our approach in generating a new current state is similar to updating a table in a traditional database.

3.1 Scalability concerns

The main benefit of the proposed architecture is that it maintains a consistent, abstract state of the Sentient environment in the Deductive Abstract Layer which can be made available to the application layer at a significantly low cost than if it would be generated directly from the Sensor Abstract Layer. There are three key reasons that enable DAL to act similarly to a fast cache for the application layer: The availability of the abstract knowledge in DAL, the fact that this knowledge changes at a lower rate than it does in SAL make DAL more computationally efficient at keeping its stored knowledge consistent. Figure 1(b) depicts the different rates with which knowledge is updated in each layer. Section 5.2 discusses in more detail, computational concerns associated with the functionality of the two layers.

4 Formal definition

This section presents a formal definition of the proposed scalable knowledge representation architecture for Sentient Computing. The concepts of the dual-layer architecture that were discussed in the previous section are now formally defined using first-order logic.

4.1 First order logic

First order logic [6] or predicate calculus, was chosen as being appropriate and sufficient for the description of the two knowledge layers as they both maintain either current knowledge only (Sensor Abstract Layer) or a combination of current and historical knowledge (Deductive Abstract Layer) about the Sentient Environment. Time is implicit in SAL and explicit in DAL. However, when describing the monitoring mechanism that establishes the links between the two layers, temporal aspects of the described predicates are addressed by realising that as the Sensor Abstract Layer is updated first, until the changes are updated to the Deductive Abstract Layer, this will contain the last known abstract state of the world.

Concepts and Definitions We assume that the physical world contains N individual values that represent autonomously mobile objects such as people that work in a building. We also assume that the physical environment contains M individual values which represent known physical *locations* of interest. Locations can be classified into *atomic* locations and *nested* locations. *Atomic* locations will typically be polyhedral named regions, such as *"the coffee-area"*, *"mike's desk"* and rooms. Via a process of *nesting* we produce a set of aggregated polyhedral regions such as floors and buildings (each floor may contain a specific set of rooms and each building a particular set of floors) as well as logically aggregated spaces such as departments (each department may contain a number of floors, or buildings).

We call a *knowledge base* K a system that stores knowledge about the Sentient environment. A knowledge base represents predicates that are *true* by storing an instance of each of these predicates. We refer to this instance as a *fact*. The *assertion* of a *fact* in the knowledge-base is equivalent to it being *stored* in the knowledge base as a *true* statement. A fact being *retracted* from the knowledge base has as a result the *removal* of the fact from the knowledge base. In fact, the *assert* command is similar to a database ADD, whereas the *retract* command is equivalent to a database DELETE. When a fact is *asserted* in the knowledge base, this signifies that the predicate that the fact corresponds to has the value TRUE. When the fact is *retracted* from the knowledge base, this signifies that the corresponding predicate has the value FALSE. Nomenclature is taken from logic programming.

4.2 Naming convention for predicates

For reasons of clarity and simplicity, we adopt the following naming convention for logical predicates throughout this document:

$$\begin{aligned}\text{L\_}\langle \textit{SAL predicate name}\rangle \; &((\text{argument name } ?\textit{argument value})\cdots \\ &\cdots \; (\text{argument name } ?\textit{argument value})) \\ \text{H\_}\langle \textit{DAL predicate name}\rangle \; &((\text{argument name } ?\textit{argument value})\cdots \\ &\cdots \; (\text{argument name } ?\textit{argument value}))\end{aligned}$$

The main difference is that DAL predicates have additional time parameters which represent the beginning and wherever appropriate, the end of the situation they refer to. For the description of the predicates we have used a *named parameter notation* based on the CLIPS [26, 9] syntax. Table 1 portrays some significant predicates. Each *predicate argument* has an associated *value* which is denoted with *?argument-value*. The predicates and their arguments are discussed in detail in sections 4.3 and 4.4.

4.3 Sensor Abstract Layer (SAL)

The knowledge base for this layer contains up to N facts of the type $L\_UserAtLocation(uid, x, y, z)$[2] that represent an object's last known position

[2] The positional parameters notation *L_UserAtLocation(uid,x,y,z)* is used interchangeably throughout this paper with the named parameter notation *L_UserAtLocation*(uid *?uid*)(x *?x*)(y *?y*) (z *?z*)) for simplicity reasons.

	Current Predicates	Historical Predicates
DAL	**(H_UserAtLocation** (uid ?*uid*)(rid ?*rid*) (start-time ?*start-time*))	**(H_ UserAtLocationHistoric** (uid ?*uid*)(x ?*x*)(y ?*y*)(z ?*z*) (start-time ?*start-time*) (end-time ?*end-time*))
	(H_UserColocation (uid-list $?uid_1 \cdots ?uid_n$) (rid ?*region-id*) (start-time ?*time-value*))	**(H_UserColocationHistoric** (uid-list $?uid_1 \cdots ?uid_n$) (rid ?*region-id*) (start-time ?*time-value*) (end-time ?*time-value*))
	(H_UserIsPresent (uid ?*uid*) (start-time ?*time-value*))	**(H_UserIsPresentHistoric** (uid ?*uid*) (start-time ?*time-value*) (end-time ?*end-time*))
SAL	**(L_UserAtLocation**(uid ?*uid*) (x ?*x*)(y ?*y*)(z ?*z*))	

Table 1. Naming convention for logical predicates

in 3-D space in terms of its Cartesian coordinates x, y, z. *L_UserAtLocation* is the most precise location known to the system for each user. The variable *?uid* represents the unique user identification for that particular user. In examples we use users' full names as identifiers. The variables *x,y,z* represent the user's last known co-ordinates. In this way, each user is associated with a position in space.

Apart from these positions, the knowledge base also contains M_1 facts of type *L_AtomicLocation* each corresponding to the M_1 known atomic regions of physical space (e.g. rooms and polyhedral areas of space) and M_2 nested regions (floors, larger areas, buildings, neighbourhoods). There are therefore four distinct type of predicates represented in this layer.

(*L_UserAtLocation* (uid *?uid*)(x *?x*)(y *?y*) (z *?z*))
(*L_AtomicLocation* (rid *?region-id*) (polyhedra $?n_1?n_2\cdots?n_j$))
(*L_NestedLocation* (rid *?region-id)* (site-list $?site_1\cdots?site_k$))
(*L_InRegion* (x *?x*)(y *?y*)(z *?z*)(rid *?region-id*))

As the people move in space, a location system generates in average λ^L *L_UserAtLocation* facts/sec per mobile user and asserts them in the knowledge base. For each new fact of type *L_UserAtLocation* the fact that represented the previous known position for that user is *retracted*, so that the knowledge base only contains the most recent known location for that.

The predicate *L_AtomicLocation* associates a named location such as "Room 5" characterised by a unique identifier, the *region-id*, with a set of j points, $n_1 \cdots n_j$, which form the nodes of a polyhedral region that defines that area.[3]

[3] We assume a co-ordinate system that assigns a set of co-ordinate values *x,y,z* to each position in space.

The predicate *L_NestedLocation* associates a *nested location* such as "The Computer Laboratory" with a list of nested and atomic locations that are directly contained in it. The predicate *L_InRegion* is created as a result of a spatial indexing algorithm, which determines the *smallest* region that contains the given co-ordinates, as expressed in the *L_UserAtLocation* predicate.

4.4 Deductive Abstract Layer (DAL)

The higher level is logically distinct from the lower level in that it maintains a complete view of the Sentient world. Although it lacks the knowledge of the accuracy of the exact user position (as this is only known to the Sensor Abstract Layer), it knows of high-level situations seen from a user-perspective as well as their temporal properties i.e. whether they hold at the current instant, or whether they happened in the past, when they first occurred and what was their duration. Such *dynamic knowledge* is modelled in the form of *current* and *historic predicates. Current* predicates represent a *dynamic situation* that still holds. *Historic* predicates represent a *situation* that occurred for a certain interval, beginning at a certain point in time and ending at a later point in time. As a consequence of the above modelling technique, the Deductive Abstract Layer has the important property that it accumulates gradually information about what has happened in the Sentient world. Now, the format of the DAL predicates is discussed in detail.

The DAL Current predicates. DAL current predicates describe a situation which occurred at an instant t_0 and which still holds at the current instant which is represented with the value *now*. Such predicates have the general format:

$$(\textit{predicate name}\ (arg_1\ ?arg_1)\cdots\ (arg_n\ ?arg_n)\ (\text{start-time}\ \textit{?time-value}))$$

Arguments arg_1 to arg_n represent the parameters of the situation that is described by the predicate and the variables $?arg_1$ to $?arg_n$ their respective values. The argument named "start-time" represents the time when the situation described by the above predicate became first known to the system.

An important *current* predicate is the one used to describe a high-level location, e.g. Mary being in the proximity of the coffee-machine, or James being on floor 4.

(*H_UserAtLocation*(uid "Mary") (rid "coffee-machine-area") (start-time 11:02))

(*H_UserAtLocation*(uid "James") (rid "floor-4") (start-time 13:05))

where *?uid* represents the user's unique identification and *?region-id* is the value of the named parameter rid which represents the name of the smallest region that contains the user.

Similarly, applications can request through the API for the SAL to register their interest in situations where two or more users are co-located in the same high-level region by using the predicate *H_UserCoLocation*.

$(H\_UserCoLocation(\text{uid-list } ?uid_1 \cdots ?uid_n)(\text{rid } ?region\text{-}id)(\text{start-time } ?time\text{-}value))$

This process is explained in more detail in section 5.2.In the above formula, uid-list is the list of users that are co-located in a region with name *rid*. The variables $?uid_1$ to uid_n represent the unique identification of these users.

The DAL historical predicates. The DAL historical predicates describe a situation which occurred at a time instance t_0, remained holding for a duration d and ceased holding at a time instance t_1. Such predicates are expressed in the following general format:

$$(predicate\ name\ (arg_1\ ?arg_1) \cdots\ (arg_n\ ?arg_n)\ (\text{start-time } time\text{-}value)\\ (\text{end-time } ?time\text{-}value))$$

The argument "start-time" represents the time when the situation described by the above predicate became first known to the system. The argument "end-time" represents the time when the situation stopped being true e.g. when the user left the room he was in. E.g. the DAL historical predicate that describes the situation where Jane and Mike move into the meeting room in their office building at 12.46 pm, remain in the same room for 9 minutes and Jane leaves the meeting room at 12.55 pm, is expressed below: It is worth noting that there can be multiple instances of historic predicates for the same user.

(*H_UserLocationHistoric* (uid-list "Jane Hunter")(rid "Meeting Room")
(start-time 12.46) (end-time 12.55))

(*H_UserCoLocationHistoric* (uid-list "Mike Smith" "Jane Hunter")
(rid "Meeting Room")(start-time 12.46) (end-time 12.55))

4.5 User-Defined DAL Predicates

It is worth noting that whereas all SAL predicates are predefined, all predicates in DAL are *user-defined*. This means that in the initial state, DAL contains no predicates. DAL predicates get created through the *DefineDALPredicate()* API call (see section 1.2) and *instances* of these predicates (facts) get generated from the Sensor Abstract Layer by the *monitor()* and *callback()* calls (see Section 1.2).

5 Queries

Queries are used by the application layer in order to capture and return the current instance of the stored knowledge about the Sentient World. *Queries* are similar to SQL [5] SELECT statements in the theory of relational databases. We can view a ***query*** as a first-order logical expression $f(c_1, c_2 \cdots c_n)$ which has the ***property*** that upon the satisfaction of a set of *atomic formulae* $c_1, c_2 \cdots c_n$, an *answer* is triggered.

$$f(c_1, c_2 \cdots c_n) \Rightarrow \mathit{Answer}$$

where f is any first-order formula involving the formulae $(c_1, c_2 \cdots c_n)$.

Answer can have a value of "yes", "no", "I don't know" or a value extracted from a stored fact such as the user id. The interface through which the *answer is returned* to the user is subject to the application layer and can be implemented in various ways, e.g.by using a *print* function, by publishing a *structured event* or through an API. We have chosen to adopt the *structured event approach* where the answer is encoded as a *structured event* and is returned to the application layer via a *notify()* call (see Section 1.2).

Examples of logical queries are *"Who is present in the building now ?"* and *"Which users are co-located now?"* The first query may be useful in the case of an application that delivers reminders to anybody who is present in the building late in the evening in order to remind them to lock their door on the way out. The second query may be useful for the same application, delivering a reminder to one party which is semantically associated with the second e.g. the reminder: "Remember to ask Jane to return your book" will be delivered when the user is in the same room with Jane [1]. Equally interesting as an example is the case where a user enters a conference site and is interested to know if there is somebody present from the University of Cambridge.

The number of conditions in the queries depends on the underlying knowledge base model. E.g. if the above mentioned query *"Who is present in the building?"* was to be executed at a knowledge-base with a single layer of knowledge (i.e. the Sensor Abstract Layer), it could then be written as a query of the following form:

Query 1 Return All Present Users (Sensor Abstract Layer).

$$\begin{aligned}&\mathit{uid}|(\mathit{L\_UserAtLocation}(\text{uid } ?\mathit{uid})\ (\text{x } ?x)\ (\text{y } ?y)\ (\text{z } ?z))\wedge\\&(\mathit{L\_AtomicLocation}(\text{rid ?region-id})\ (\text{polyhedron } ?n_1?n_2\cdots?n_j)\ )\wedge\\&(\mathit{L\_InRegion}(\text{x ?x})\ (\text{y ?y})(\text{z ?z})\ (\text{rid } \mathit{?region\text{-}id}))\end{aligned}$$

The "|" operator is similar to an SQL *SELECT* operator in that it returns the values for the associated variables. In this case, *uid* represent the information that will be returned in the answer. Query 1 expresses the logical statement that in order for a user to be present in the building, three *conditions* need to hold simultaneously:

– He or she needs to be *seen* by the location system at a position which can be characterised by the co-ordinates x,y,z.
– The system must know of at least one region with id *rid* which can be characterised by a known polyhedral shape, and
– The system is able to determine that the coordinates of the user's position belong to a known region, such as the one described above.

If all of the above hold simultaneously, than the user is deduced to be *present.*

The same query, should it be applied on DAL it would assume a simpler form:

Query 2 Return All Present Users (Deductive Abstract Layer).

$$uid|(\textit{H\_UserIsPresent}(\text{uid } \textit{?user-id}) \; (\text{start-time } \textit{?t}))$$

Query 1 and Query2 are defined to be equivalent directly by the application layer through the RegisterRecurringQuery interface and its arguments. E.g. for the case of the (*H_UserIsPresent*(uid *?uid*) (start-time *?t*)) predicate, an application would have to issue the following statement:

RegisterRecurringQuery(*application identity*, (*H_UserIsPresent*(uid *?uid*) (start-time *?start-time*)), Query 1, Query 2)

In the above statement, the argument *application identity* describes an identification for the application that has issued the statement and to which the *answer* will be returned to. The *H_UserIsPresent* predicate is similar in content to the *H_UserAtLocation* one and it is useful as an abstraction of the user's location, where the actual location is of no interest to the application.

Similarly, the query *"Which users are co-located now?"* can be viewed as:

Query 3 Return All Co-Located Users (Sensor Abstract Layer).

$$\begin{aligned}
&(uid_1, uid_2)|\; (\textit{L\_UserAtLocation}(\text{uid } ?uid_1)(\text{x } ?x_1)(\text{y } ?y_1)\; (\text{z } ?z_1)\wedge\\
&(\textit{L\_UserAtLocation}(\text{uid } ?uid_2)(\text{x } ?x_2)(\text{y } ?y_2)\; (\text{z } ?z_2))\wedge\\
&(\textit{L\_AtomicLocation}(\text{rid ?region-id})\; (\text{polyhedron } ?n_1?n_2\cdots?n_j))\wedge\\
&(\textit{L\_InRegion}(\text{x } ?x_1)(\text{y } ?y_1)(\text{z } ?z_1)\; (\text{rid ?region-id}))\wedge\\
&(\textit{L\_InRegion}(\text{x } ?x_2)(\text{y } ?y_2)(\text{z } ?z_2)(\text{rid ?region-id}))\wedge\\
&(\textit{L\_InRegion}(x_2, y_2, z_2, rid))\wedge\\
&(uid_1 \neq uid_2)
\end{aligned}$$

The same query, should it be applied on the Deductive Abstract Layer instead of the Sensor Abstract Layer, assumes a simpler form.

Query 4 Return All Co-Located Users (Deductive Abstract Layer).

$$(uid_1, uid_2)|(\textit{H\_UserCoLocation}(\text{uid-list } ?uid_1?uid_2)(\text{rid } \textit{?rid})(\text{start-time } \textit{?t}))$$

Similarly, Queries 3 and 4 are declared to be equivalent through the Register-RecurringQuery interface:

RegisterRecurringQuery(*application identity*,
(*H_UserCoLocation* (uid-list $?uid_1?uid_2$)(rid *?rid*)(start-time *?t*)),
Query 3, Query 4)

Based on the RegisterRecurringQuery definitions, Queries 1 and 3 are equivalent to Queries 2 and 4 respectively. However, Queries 2 and 4 have in average fewer conditions than their equivalent Queries 1 and 3. This is due to the fact that the information of the users' *presence* and *co-location* is available in the Deductive Abstract layer in the form of the logical predicates, *H_UserIsPresent* and *H_UserCoLocation* respectively. Section 6 discusses in detail the effect of the above observation to the computational complexity involved in the execution of queries in the proposed reasoning system, demonstrating that queries executed in the Deductive Abstract Layer such as Queries 2 and 4, entail the use of significantly fewer computational resources than queries executed in the Sensor Abstract Layer (Queries 1 and 3).

5.1 Recurring queries

A second approach for the application layer to derive information from the knowledge base is by *registering interest* to a recurring situation that gets triggered by periodic timing events. Whenever such a situation occurs, a *notify()* call, returns a structured event that represents the predicate of interest to the application layer. Contrary to queries, recurring queries *do not examine the current state* of the Sentient World in order to establish whether the situation of interest holds at the current instance. Rather, they act similarly to a *subscribe* call in the *publish-notify* protocol for distributed systems, in registering interest in receiving information about the *future* occurrences of the situation in question.

E.g. an application may be interested in a regularly recurring event such as "Whenever any two people are co-located update the GUI so that co-located people are portrayed as being enclosed in a rectangular area." We can view a ***recurring query*** as a first-order logical expression $f(c_1, c_2 \cdots c_n)$ which has the ***property*** that upon the satisfaction of a set of *atomic formulae* $c_1, c_2 \cdots c_n$, a *set of actions* are triggered.

$$f(c_1, c_2 \cdots c_n) \Rightarrow notify(event)$$

The *notify(event)* call passes on to the application layer a *structured event* that contains the queried information. Such an event can be a Supervisor Alert event which is defined elsewhere in the system. When it is received by the application, the latter sends an appropriate e-mail message to the user. In fact, a particular case of recurring queries, is that, where upon satisfaction of the query, a notification action is being performed. E.g. "Whenever my supervisor enters the lab, notify me." Recurring queries can be expressed as *logical implications*, in which the left-hand-side is a simple query and the right-hand-side is a *notify(event)* predicate.

Query 5 Whenever my supervisor enters the lab, notify me by email.(Deductive Abstract Layer)

$$uid|(\mathit{H\_UserIsPresent}(\text{uid “Andy Hopper”}) \Rightarrow \text{notify(Supervisor Alert)})$$

The application layer, on receipt of the *Supervisor Alert* event, is responsible for issuing an appropriate e-mail notification. ***Note*** that this is equivalent to a high-level query, as it assumes that the predicate *H_UserIsPresent* is already available in the knowledge-base.

5.2 Analysis

Having discussed queries and recurring queries, we can now look into how the two-layer knowledge scheme ensures scalability .

In order to illustrate this, we consider a prototype implementation, where queries and recurring queries are implemented in each layer by means of a CLIPS [26] inference engine. Each query is mapped to a CLIPS rule. CLIPS implements a forward chaining rule interpreter that given a set of *rules* applied on a set of stored facts, cycles through a process of matching rules to available facts thus determining which queries are satisfied by the stored state of the Sentient environment. The process by which CLIPS determines which facts satisfy the conditions of each query or recurring query, is called *pattern matching* and the *Rete algorithm* [8] is used for this purpose.

The advantage of the proposed architecture is due to three important factors:

- First, as can be seen from Sections 5, 5.1, *queries* that are executed in the Deductive Abstract Layer such as Query 3, assume a *much simpler form* than those executed in the Sensor Abstract Layer (Query 1), as the latter have *more conditions in average* and therefore require *more computational resources for pattern matching.*
- Secondly, pattern matching is triggered *repeatedly* every time the stored knowledge changes by an *assert* or *retract* command. Therefore, the lower the rate of knowledge updates, the lower the computational load required (see figure 1(b)). Since the knowledge update rate in DAL is significantly lower than the one in SAL (produced by the regular updates of the sensor infrastructure), *DAL is computationally more efficient.*
- Finally, the machine that hosts DAL has fewer real-time constraints, introduced by the *interruptions* caused by the *assert* and *retract* statements that control knowledge updates.

The next session discusses the computational complexity associated with queries in more detail by analysing the Rete algorithm.

6 Prototype implementation

This section aims to give a quantitative evaluation of the proposed scheme and its algorithm by discussing an implementation of the proposed system and by comparing Query 3 (see Section 5) which is executed at the Sensor Abstract Layer to the same query (Query 4) which is executed at the Deductive Abstract Layer and demonstrate that the latter entails a significantly lower number of computational steps.

We have implemented the proposed architecture using the Jess[12] production system. Jess is a java-based implementation of CLIPS. For the acquisition of real-time location information, we have built a middleware component [15] that interfaces the Active BAT system using CORBA structured events, and translating them into Jess facts.

6.1 Sensor Abstract Layer.

A model was created in Jess for the LCE [7] based on location data produced by the Active BAT. The experiment involved 15 members of the lab moving around 21 known locations in the LCE. An instance of the lower-layer was captured and the following query "Return All Co-Located Users" was executed in the Sensor Abstract Layer.

Query 6 Return All Co-Located Users (Sensor Abstract Layer).

$$\begin{aligned}&(uid_1, uid_2)|\ (L\_UserAtLocation(\text{uid } ?uid_1)(\text{rid } ?\text{rid}))\wedge\\&(L\_UserAtLocation(\text{uid } ?uid_2)(\text{rid } ?\text{rid}))\wedge\\&(L\_AtomicLocation(\text{rid } ?\text{region-id})\ (\text{polyhedron } ?n_1?n_2\cdots?n_j))\wedge\\&(uid_1 \neq uid_2)\end{aligned}$$

The Rete Algorithm Our implementation uses the Rete Algorithm [8] for pattern matching. In the Rete algorithm, the pattern compiler creates a network by linking together nodes that test query elements. This network functions similarly to a finite state machine whenever a query is added in the knowledge base, or whenever a new fact s asserted or retracted. More specifically, for each predicate included in the query, the network creates a one-input node, portrayed in *red* in Fig 2. Node n_1 corresponds to the predicate *L_UserAtLocation* and node n_2 to the predicate *L_AtomicLocation*. Node n_3 corresponds to the condition $(uid_1 \neq uid_2)$. Also portrayed in red is the root node of the network, n_0. A two-input *(green)* node is created for each conjunction of predicates. Node n_5 corresponds to the conjunction:

$$\begin{aligned}&(L\_UserAtLocation(\text{uid } ?uid_1)(\text{rid } ?rid))\wedge\\&(L\_UserAtLocation(\text{uid } ?uid_2)(\text{rid } ?rid))\end{aligned}$$

Node n_6 corresponds to the conjunction:

$$\begin{aligned}&(L\_UserAtLocation(\text{uid } ?uid_1)(\text{rid } ?rid))\wedge\\&(L\_UserAtLocation(\text{uid } ?uid_2)(\text{rid } ?rid))\wedge\\&(L\_AtomicLocation(\text{rid } ?region\text{-}id)(\text{polyhedron } ?n_1\cdots?n_j))\end{aligned}$$

Node n_7 is also a two-input node, that represents the conjunction of the above predicates with the condition $(uid_1 \neq uid_2)$. Finally, node n_8 is a terminal node that determines whether the query is satisfied or not.

The Rete algorithm proceeds as follows: When the query is added to the Sensor Abstract Layer, for each stored fact, a *token* is created. Each token is an ordered pair of a tag which in this case has the value "UPDATE" and a description of the stored fact. All these tokens are passed to node n_0 which

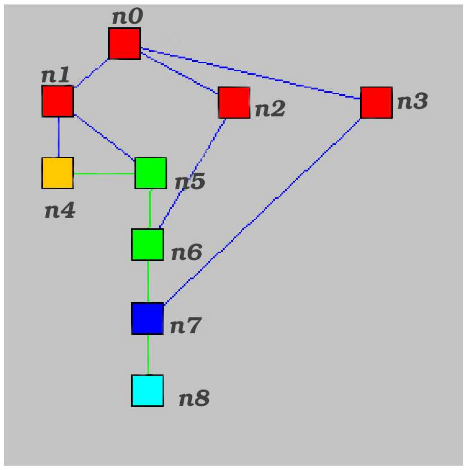


Fig. 2. The Rete Network for the Return All Co-Located Users query (Sensor Abstract Layer)

is the root node in the network. Node n_0 passes all the generated tokens to each of its successor nodes. Node n_1 checks whether any of the received tokens correspond to facts of type *L_UserAtLocation*[4] and passes all such tokens to node n_5. Node n_5 checks all UserAtPosition tokens against each other, in order to determine which pairs satisfy the conjunction:

$$(\textit{L\_UserAtLocation}(\text{uid } ?uid_1)(\text{rid } ?rid))\wedge \\ (\textit{L\_UserAtLocation}(\text{uid } ?uid_2)(\text{rid } ?rid))$$

For each of the pairs that satisfy the conjunction, it creates a new token and forwards this on to node n_6. Node n_2 tests for tokens that are of type *L_AtomicLocation* and passes these on to node n_6 too. Node n_6 joins the pairs that represent the conjunction :

$$(\textit{L\_UserAtLocation}(\text{uid } ?uid_1)(\text{rid } ?rid))\wedge \\ (\textit{L\_UserAtLocation}(\text{uid } ?uid_2)(\text{rid } ?rid))\wedge \\ (\textit{L\_AtomicLocation}(\text{rid } ?\textit{region-id})(\text{polyhedron } ?n_1 \cdots ?n_j))$$

into bigger tokens and forwards them on to n_7. Node n_7 tests that $uid_1 \neq uid_2$ thus excluding trivial co-locations of the same person. It forwards the eligible tokens to n_8, the success node. These tokens satisfy the whole query. For each token, n_8 creates an instantiation of the query.

In order to get an measure of the computational complexity that the implemented scheme entails, we chose to look at the number of node activations and

[4] In this prototype implementation the SPIRIT system was used to provide *L_UserAtLocation* predicates from the Active BAT positions.

the number of tests performed in total by the nodes on the network. The results are shown in Table 2 (SAL).

	Sensor Abstract	Deductive Abstract
	Layer (SAL)	Layer (DAL)
total of node activations	317	200
total of tests on nodes	1611	0

Table 2. Pattern Matching Costs.

6.2 Deductive Abstract Layer

Repeating the previous experiment, with the same initial state, we now consider Query 4 (see Section 5) which is executed in the Deductive Abstract Layer. The network for this query is portrayed in Fig. 3.

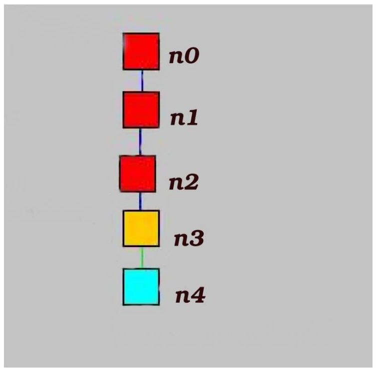


Fig. 3. The Rete Network for the Return All Co-Located Users query (Deductive Abstract Layer)

Node n_0 is the root node. Node n_1 tests whether the received token is of type *H_UserCoLocation*. Node n_2 passes on the tokens with the correct number of arguments and n_4 creates an instantiation of the query and adds it to the conflict set.

Performing the same analysis as before, the results are presented in Table 2. It is worth noting that the number of computational steps executed by the Rete algorithm for pattern matching each query are significantly lower for the DAL query. Taking into consideration that both networks (see Fig. 2, Fig.3) behave similarly to acyclic finite automata, which are triggered repeatedly each time a fact is asserted or retracted in each knowledge layer respectively, we can easily infer that the overall number of computational steps required for DAL is smaller than that required for SAL, as knowledge in DAL changes much less frequently. Last, SAL is continually *interrupted* by a very high event rate which has an immediate effect on the machine that hosts that layer.

7 Conclusions and future work

A scalable knowledge representation and abstract reasoning system for Sentient Computing was presented where knowledge was modelled formally using first-order logic. First-order logic proved suitable for Sentient Computing, especially in the context of the proposed architecture which is based on a cache-like, dual-layer scheme which maintains abstract knowledge in the higher Deductive Abstract layer as opposed to rapidly-changing low-level knowledge in the lower, Sensor Abstract layer. Abstract knowledge remains consistent with the rapidly changing state of the Sentient world by closely monitoring associated, low-level predicates as requested by the application layer through an API interface. Such predicates are contained in the Sensor Abstract Layer and by having only threshold changes reflected at the Deductive Abstract Layer. Maintaining abstract knowledge is a requirement of the Sentient Application layer and it is made available to Sentient Applications through a mechanism of queries which are mainly executed at the Deductive Abstract layer. Experiments with a prototype implementation confirm that the two-layered architecture is more efficient than a single-layered one. Future work will involve designing and implementing a large-scale, fully distributed architecture based on the proposed system.

8 Acknowledgements

We are grateful to Andy Hopper and Mike Gordon for discussion and encouragement.

References

1. Dey K Anind and Gregory D.Abowd. CyberMinder: A Context-aware System for Supporting Reminders. In *Proceeding of CHI 2000*. CHI2000, 2000.
2. George Coulouris, Tim Kindberg, and Jean Dollimore. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2001.
3. George Coulouris, Hani Naguib, and Scott Mitchell. Middleware Support for Context Aware Multimedia Applications.
4. P. Dana. Global Posioning System Overview", Department of Geography University of Texas at Austin. http://www.colorado.Edu/geography/gcraft/notes/gps/gps.html, 1998.
5. C. J. Date and Hugh Darwen. *A Guide to the SQL Standard, Third Edition*. Addison-Wesley Publishing Company, Inc., 1993.

6. Elliott Mendelson. *Introduction to Mathematical Logic*. Wadsworth and Brooks Cole Advanced Books Software, 1990.
7. Laboratory for Communications Engineering (LCE). http:://www-lce.eng.cam.ac.uk.
8. Charles L. Forgy. Rete: A fast Algorithm for the Many Pattern/many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.
9. Joseph C. Giarratano. *CLIPS User's Guide, Version 6.10*, chapter ch. 8. unknown, 1998.
10. Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The Anatomy of a Context-Aware Application. *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking August 15-19,1999, Seattle*, 1999.
11. Andy Hopper. The Royal Society Clifford Paterson Lecture: Sentient Computing, 1999.
12. http://herzberg.ca.sandia.gov/jess. Jess, the Java Expert System Shell.
13. http://www lce.eng.cam.ac.uk/qosdream/. QoSDREAM: Quality of Service for Distributed REconfigurable Adaptive Multimedia.
14. http://www.acm.org/sigcomm/standards/iso_stds/OSI_MODEL. OSI 7498, open System Interconnection Model.
15. D. Ipina and E. Katsiri. A Rule-Matching Service for Simpler Develpment of Reactive Applications. In *Middleware 2001*. http://computer.org/dsonline/0107/features/lop0107.htm, November 2001.
16. E Katsiri. Principles of Context Inferences. In *Ubicomp 2002 Additional Proceedings*, 2002.
17. G. Korteum, Z. Segall, and T.G. Thomson. Close Encounters: Supporting Mobile Collaboration through Interchange of User Profiles. In *HUC'99*, pages 171–185, 1999.
18. Masoud Mansouri-Samani and Morris Sloman. Gem: A Generalised Event Monitoring Language for Distributed Systems. *Distributed Systems Engineering Journal*, Vol. 4(No. 2), June 1997.
19. N. Marmasse. comMotion. In *CHI'99*, pages 320,321, 1999.
20. Ajith K. Narayanan. Realms and States: A Framework for Location Aware Mobile Computing. In *Workshop on Mobile Commerce (MOBICOM)*, 2001.
21. Danielle Nardi and Ronald J. Brachman. An Introduction to Description Logics. Published on the Internet.
22. Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.
23. Tristan Richardson. Teleporting - Mobile X Sessions. In *Proceedings Ninth Annual X Technical Conference, Boston MA, Technical Report 95.7*, 1995.
24. Bill Schilt, Norman Adams, and Roy Want. Context-Aware Computing Applications. *tbd*, 1994.
25. Pete Steggles, Paul Webster, and Andy Harter. The Implementation of a Distributed Framework to Support Distributed Applications. Technical report, The Olivetti and Oracle Research Laboratory, 1998.
26. www.ghg.net/clips/CLIPS.html. CLIPS: A Tool for Designing Expert Systems.