Proceedings of the
Automated Reasoning Workshop 2015
Bridging the Gap between Theory and Practice
ARW 2015

9th-10th April 2015
School of Computer Science
University of Birmingham
Birmingham, United Kingdom

Editor:
Volker Sorge
Organising Committee:

- Alexander Bolotov (University of Westminster)
- Simon Colton (Imperial College London)
- David Crocker (Escher Technologies)
- Louise Dennis (University of Liverpool)
- Clare Dixon (University of Liverpool)
- Jacques Fleuriot (University of Edinburgh)
- Ullrich Hustadt (University of Liverpool)
- Mateja Jamnik (University of Cambridge)
- Ekaterina Komendantskaya (University of Dundee)
- Alice Miller (University of Glasgow)
- Renate Schmidt (University of Manchester)
- Volker Sorge (University of Birmingham)

Local Organiser:
Volker Sorge (University of Birmingham)
Table of Contents

1. Invited Talk:
   Nonmonotonic reasoning and learning in biological networks
   *Oliver Ray*

2. Invited Talk:
   Verifying Quantum Protocols using Interactive Theorem Proving
   *Florian Kamm"uller*

3. Combining SEPIA and ML4PG
   *Thomas Gransden*

4. Event-B and Cloud Provers
   *Alexei Iliasov, Paulius Stankaitis and David Ebo Adjepon-Yamoah*

5. AVATAR: a new Architecture for First-Order Theorem Provers
   *Giles Reger, Martin Suda and Andrei Voronkov*

6. Reasoning about Dynamic Auctions
   *Marco B. Caminati, Manfred Kerber and Colin Rowat*

7. Characterising the workload of SMT solvers for program verification
   *Andrew Healy, Rosemary Monahan and James Power*

8. Saturation-Based Reasoning for SHI Knowledge Bases with Applications to Forgetting and Uniform Interpolation
   *Patrick Koopmann and Renate Schmidt*

9. Debugging of ALC-Ontologies via Minimal Model Generation
   *Fabio Papacchini and Renate Schmidt*

10. Concept Forgetting for ALCOI-Ontologies
    *Yizheng Zhao and Renate Schmidt*

11. A Tool for the Formal Verification of Quantum Optical Computing Systems
    *Sidi Mohamed Beillahi, Mohamed Yousri Mahmoud and Sofine Tahar*

12. CLProver++: An Ordered Resolution Prover for Coalition Logic
    *Ullrich Hustadt, Paul Gainer, Clare Dixon, Claudia Nalon and Lan Zhang*

    *Alexei Morozov and Alexander Polupanov*

14. Distributed Agent-Based Load Balancer for Cloud Computing
    *Leszek Sliwko, Vladimir Getov and Alexander Bolotov*

15. Discovering new ontologies and lemmas through concept blending and analogical reasoning
    *Meenakshi Kesavan*
16. Capturing Temporal Aspects in Ontologies
   Jared Leo, Uli Sattler and Bijan Parsia

17. Using Herbrand Bases for Building a Smart Semantic Tree Theorem Prover
   Nourah Shenaiber, Manfred Kerber and Volker Sorge

18. Semantic Understanding of Mathematical Formulae in Documents
   Randa Almomen and Volker Sorge
This talk describes a nonmonotonic reasoning system called XHAIL (eXtended Hybrid Abductive Inductive Learning) and shows how it can be used for analysing and revising biological networks. In particular, we introduce a new open-source implementation of XHAIL and investigate its scalability in a case study involving a complete logical model of yeast metabolism and real growth data collected by a Robot Scientist.
Invited Talk
Verifying Quantum Protocols using Interactive Theorem Proving

Florian Kammlüer
Middlesex Uni and TU Berlin

In this invited presentation, I first give an overview of interactive theorem proving and its application in Software and Security Engineering to set the scene and put my work into the right context within the field of automated reasoning. I focus on Higher Order Logic theorem provers, in particular Coq and Isabelle.

As a major case study, I present our current efforts on formalizing Quantum Protocols with Coq and Isabelle. I give a brief overview of the background of this application, the main ideas of the mathematical model and how it is formalized, showing some preliminary results but also analysing shortcomings and sketching future plans.

In the final part, I wrap up some general experiences made in this and other projects on the verification of security protocols but also programming languages, for example, type systems and security proofs for active objects, giving a summary of the state of the art, current and future challenges.
Combining SEPIA and ML4PG

Thomas Gransden

Department of Computer Science, University of Leicester, UK tg75@student.le.ac.uk

Abstract: SEPIA is an approach that infers state based models from existing proof corpora. These models can then be used for the development of new proofs. As with similar approaches, selecting the best facts to use in new proofs is challenging. We investigate the potential use of ML4PG as a relevance filter for our approach - achieving a model only inferred from the lemma suggestions by ML4PG. These reduced models still contain the necessary tactics to achieve proofs.

1 Introduction

In the past two decades, Interactive Theorem Provers (ITP) have been used in a number of significant proof developments. These range from verifying proofs of mathematical properties such as the Four-Color and Odd-Order theorems through to proving compilers and OS kernels are correct. These developments contribute large amounts of knowledge to the many proof corpora available.

The proof libraries have now become extremely large, and keeping track of the best facts to use in new proofs can be challenging. For instance, Sledgehammer [4] outsources proof obligations to automated theorem provers. Along with a conjecture, existing facts must also be provided – however in the presence of too many facts even the most powerful automated tools will struggle to deliver a proof.

Therefore, relevance filtering must occur to select only the most relevant facts to provide to the automated systems. Initially, the relevance filter implemented was reasonably naive (selecting facts based on symbols and functions from the conjecture) but still effective. Kühlwein et al. have recently improved this with the help of machine learning techniques [3].

We propose the use of ML4PG [2] to act as a relevance filter for our model inference approach (briefly outlined in the next section). ML4PG has been shown to identify commonalities between lemmas in large proof libraries, and harnessing this knowledge may be beneficial to our approach. The overall search space will be reduced whilst still leading to models that contain the necessary tactics needed to derive a proof.

2 Modelling proofs with state machines

In our previous work [1], we have applied model inference techniques to corpora of Coq proofs (we have since named this approach SEPIA - Search for Proofs using Inferred Automata). The models inferred are Extended Finite State Machines (EFSM). The main conceptual difference between EFSMs and traditional finite state machines is that transitions may have a guard that places constraints on the parameters that may be used. Figure 1 presents a small example of an EFSM inferred from Coq proofs.

The inferred models have been shown to accurately capture the reasoning patterns within a set of proofs. We showed how the models can be used manually to derive proofs new properties that weren’t in the original corpora. As the size of the libraries increases, the models become too large to consider processing manually. However, ongoing work is investigating the automation of this process.

A transition in an EFSM model of Coq proofs may look like the following:

\[
\text{rewrite (p=\text{"plus}_n\_O" || p=\text{"O_minus"})}
\]

The intended semantics of this transition is that it should only be followed if the Coq tactic \textit{rewrite} is applied with either \textit{plus}_n\_O or \textit{O_minus} given as the parameters. The full discussion of the inference process and it’s application to interactive proofs can be found in previous work [1, 5].

3 Employing ML4PG as a relevance filter

Our current approach when inferred the models is to use \textit{whole} theories of Coq proofs. However, at any one point only a handful of the lemmas within these theories may be useful in a proof attempt. We propose the use of ML4PG [2] as a relevance filter for our model inference technique.

ML4PG is a tool that can take large corpora of Coq/SSReflect proofs and identify commonalities between lemmas and definitions - for instance the whole SSReflect library has been clustered and analysed. We use ML4PG to produce clusters that identify the most useful lemmas to infer a model from. This leads to a reduction in complexity, without affecting the accuracy of the inferred models.

We demonstrate this idea with a small example. Consider a scenario where we are trying to prove the lemma \textit{take\_size} from the \textit{seq} theory in the SSReflect development.\footnote{http://ssr.msr-inria.inria.fr} The theory contains 393 other proofs, and inferring a model from all of these proofs leads to a state machine containing 88 states and over 250 transitions (this is too large to show in this paper). The challenge we are faced with is this: can we reduce the complexity of the model (using ML4PG) and still prove the lemma?

By using ML4PG, the lemma \textit{take\_size} appears in a cluster with 33 other lemmas. Instead of inferring the model from the whole theory, we instead infer only from these proofs. This provides us a much smaller state machine containing 13 states with much fewer transitions than before. The model in Figure 1 shows the reduced EFSM.
This reduced state machine yields the necessary proof (found by following the transition from state 0 to state 15):
by elim: s => /= x s -->. Although a fairly trivial example, this demonstrates the potential benefits of using ML4PG to filter large proof corpora. Of course, using a reduced set of lemmas not only improves the time taken to infer the model but also reduces the possible search time.

4 Conclusion and Future Work

We have demonstrated that combining ML4PG with SEPIA can be beneficial when deriving proofs in Coq. By reducing the size of the inferred models, the overall process of finding a proof becomes simpler. The models are still useful, even when reducing their complexity. Current work is focussing on automating this process, so that proof attempts can be completed fully-automatically.

This work naturally brings up many potential ways of combining the two approaches. Initially, we have studied reducing the search space by disregarding lemmas from the model inference process. We plan to empirically evaluate the combined methods further to gain deeper insight into the potential benefits of the combined tools.

References


1 Overview

Event-B [1] is one of the more popular notations for model-based, proof driven specification. It offers a fairly high-level mathematical language based on FOL and ZF set theory and an economical yet expressive modelling notation centred around the notion of an atomic event - a form of before-after predicate. Leaving aside the methodological qualities of Event-B, one can regard an Event-B model as a high-level notation from which a number of proof obligations may be automatically derived. Proof obligations seek to establish properties like the preservation of invariance and satisfaction of refinement obligations. A model is deemed correct when all proof obligations are successfully discharged.

Event-B employs syntactic schemas for constructing its safety and refinement proof obligations. A number of provers, some tailor-made and some external, attempt to automatically discharge an open proof obligation. Failing this, a user may provide proof hints (but not a complete proof script) to direct a combination of built-in rewrite procedures and automatic provers.

There was a concerted effort, funded by a succession of EU research projects, to make Event-B and its toolkit appealing and competitive in an industrial setting. One of the lessons of this mainly positive exercise is the general aversion of industrial users to interactive proofs. It is possible in principle (although hardly unrealistic in the context) to learn, through experience and determination, the ways of underlying verification tools and master refinement and decomposition to minimize proof effort. The methodological implications of avoiding manual proofs are far more serious: building a good model is always a trial and error process; any non-trivial design necessitates restarting from scratch or doing considerable model refactoring (i.e., re-arranging refinement layers). This means redoing manual proofs which makes time spent proving dead-end efforts seem wasted. Hence, proof-shy engineers too often do not make good use of the formal specification stage as they tend to hold on to an initial, often incoherent design.

Proof economy is thus one of the worthy goals to explore. We propose to attack the problem from several directions at once.

First, we want to change the way proofs are done, at least in an industrial setting. Instead of doing an interactive proof - something that is an inherently one-off effort - we shall invite users to write and proof a schematic lemma that is strong enough to discharge an open proof obligation. Such a lemma may not refer to any model variables or types and is, in essence, a property supporting axiomatization of Event-B mathematical language. If such a lemma cannot be found or seems too difficult to prove, the model must be changed. Experience suggests that a modelling project is likely to have a fairly distinctive usage of data types and mathematical constructs; we conjecture that this leads to a distinctive set of supporting lemmas. We also hypothesise that such distinctiveness is pronounced and dictated by modeler’s experience and background as well as model subject domain. We have also observed that the style of informal requirements - structured text, hierarchical diagram, structural diagram - has an impact on a modelling style.

A schematic lemma is a tangible and persistent outcome of any modelling effort, even an abortive one. The ‘distinctiveness’ hypothesis suggests that such a lemma is likely to be useful in a nested iteration and, by an extension, there is a point when all relevant lemmas are collected; then modelling, in a given context, becomes completely free of interactive proofs. We intend to demonstrate that such a point is reachable at least for a number of non-trivial existing models.

One problem we foresee is the accumulation of lemmas of which majority could be irrelevant outside of some narrow context. Adding an extra lemma in a proof context generally makes proof more difficult. It could conceivably reach a point where nothing may be proven due to the sheer number of supporting properties. Some form of filtering is thus paramount. From the outset, we make a set of support lemmas bound to a specific application and private to a group of developers. To filter out irrelevant lemmas within such context, we are looking into state-of-the-art techniques in hypothesis filtering. At the very basic level, we start with filtering by user-defined template expressions where a lemma is included only if the goal or some hypothesis predicate matches a given template.

Our second direction of attack is an extensive use of cheap computational resources made available by data centres and cloud computing [3]. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., servers, storage, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction. It provides different deployment
models: Infrastructure-as-a-service(IaaS), Platform-as-a-service(PaaS) and Software-as-a-service(SaaS). Generally, many functionalities are currently being provided as a service under a broad term of ‘Everything-as-a-service(XaaS)’ on the cloud. Provers-as-a-service is a natural direction given that provers are CPU and memory intensive; at the same time, running a collection of (distinct) provers on the same conjecture is a trivial and fairly effective way to speed up proof given plentiful resources. Usability perception of interactive modelling methods such as Event-B is sensitive to peak performance when a burst of activity (new invariant) is followed by a relatively long period of idling (modeller thinking and entering model). The cloud paradigm, where only the actual CPU time is rented, seems well suited to such scenario. Also, the cloud’s feature of scalability plays a critical role in this situation. Virtualization, which is a key concept of cloud computing, enables the installation of multiple virtual machines on the same physical machine. It supports scaling through load balancing. Here, a service running on a provisioned virtual machine scales either up or down to handle varying amounts of requests using a live migration process.

From the technical perspective one low hanging fruit was the absence of interface between Event-B and TPTP provers. Thus hosting a collection of TPTP provers on the cloud and providing an interface to them seems a promising direction. To simplify translation, we decided to use Why3 [2] umbrella prover that offers a far more palatable input notation and also supports SMT-LIB provers. A plug-in to the Rodin Platform was realised to map between the Event-B mathematical language and Why3 theory input notation (we don’t make use of its other part - a modelling language notation). The syntactic part of the translation is trivial: just one Tom/Java class. The bulk of the effort is in the axioms and lemmas defining the properties of the numerous Event-B set-theoretic constructs. We are already at a stage where there is a working prototype able to discharge (via provers like SPASS and Alt-Ergo) a number of properties that previously required interactive proof. At the same time, we realize that axiomatization of a complex language like Event-B is likely to be an ever open problem. It is apparent that different provers prefer differing styles of operator definitions: some perform better with an inductive style (the size of an empty set is zero, adding one element to a set increases its size by one) others prefer regress to already known concepts (here exists a bijection such that ...). Since we don’t know how to define an optimal axiomatization, even for any given prover, we offer an open translator with which a user may define, with as many cross-checks as practically reasonable, a custom embedding of Event-B into Why3.

Despite providing users with axiomatization option, some general axioms are supplied by us. In early development stages it was noticed that with some provers (e.g. Z3), the machine quickly runs out of memory, thus axiomatization process of WHY3 functions was highly constrained by state space. Secondly, lack of axiomatization experience resulted in proving anything, where missing finiteness statement or use of bi-implication instead of implication resulted in contradicting axiom, which then allowed to prove anything. Simple, though not a completely reassuring solution was to have and check contradicting dummy-lemma at the end of your library. Whole operators axiomatization process was done in similar manner to [4], where all Event-B set operators, were defined in a form of membership. Analysing undischarged proof obligations of different Event-B models was a next step in axiomatization process. It was noticed that a lot of undischarged goals required fairly simple, but very model-specific lemmas. Therefore, discovered missing lemmas had a high potential to be redundant in another context. In order not to expand state space dramatically some generalization of lemmas is necessary or otherwise as previously mentioned some state-of-the-art hypothesis filtering.

Doing proofs on the cloud opens possibilities that we believe were previously not explored, outside, perhaps, proof contests results. The cloud service keeps a detailed record of each proof attempt along with (possibly obfuscated) proof obligations, supporting lemmas and translation rules. There is a fairly extensive library of Event-B models constructed over the past 15 years and these are already a source of proof obligations. Some of these come from academia and some from industry. We are now starting to put models through our prover plug-in in order to collect some tens of thousands of proof obligations. One immediate point of interest is whether one can train a classification algorithm to make useful prediction of relative prover performance. If such a prediction can yield statistically significant results, prover call order may be optimized to minimize resource utilization while retaining or improving average proof time.

References


AVATAR: a new Architecture for First-Order Theorem Provers

Giles Reger  Martin Suda  Andrei Voronkov
University of Manchester, Manchester, UK
{regerg,sudam,voronkov}@cs.man.ac.uk

Abstract: AVATAR is a new architecture for first-order resolution and superposition theorem provers which tightly integrates the saturation loop with a SAT solver (or an SMT solver) to efficiently implement the clause splitting rule. AVATAR employs the SAT solver to pick splitting branches, thus delegating the propositional essence of the given problem to the dedicated solver. This leads to a combination which is shown to be highly successful in practice. Moreover, replacing the SAT solver with an SMT solver opens up the possibility for efficient reasoning with both quantifiers and theories.

1 Introduction

Modern first-order resolution and superposition theorem provers use saturation algorithms to search for a refutation in clauses derivable from the input. On hard problems, this search space often grows rapidly and performance degrades especially fast when long clauses are generated. One approach that has proved successful in taming the search space is splitting where clauses are split into components with disjoint variables and the components are asserted in turn. This reduces the length of clauses in the search space at the cost of keeping track of splitting decisions.

Two main approaches to implementing splitting have been described in the literature. In splitting with backtracking (introduced in SPASS [5]) a conceptual splitting tree is traversed in parallel to the saturation process and one explicitly keeps track of how clauses depend on splitting decisions. Deriving a conditional contradiction means that the currently explored branch of the tree is inconsistent and a new candidate branch must be selected. Splitting without backtracking [3] (as seen in Vampire [2]), on the other hand, introduces new propositional variables to represent the splitting components and replaces a split clause by several shorter clauses while preserving satisfiability.

AVATAR (Advanced Vampire Architecture for Theories and Resolution) [4] combines ingredients from both previous approaches to splitting in a clever way with the key addition of employing an efficient SAT solver to make splitting decisions. We provide an overview of this new architecture (Sect. 2), report on experimental results which demonstrate its practical potential (Sect. 3), and outline areas of our current research on this topic (Sect. 4).

2 AVATAR overview

The AVATAR architecture (see Fig. 1) consist of three parts: the first order solver FO, the SAT solver, and the intermediate Splitting interface. The FO solver implements the standard saturation loop except that it deals with first order clauses with assertions (see below). Splitting interface splits clauses into components, maintains an injective mapping [i] abstracting clause components by propositional variables and communicates with the SAT solver. The SAT solver receives information about split clauses and derived (conditional) contradictions and either produces a model, which defines a new branch of the conceptual split tree to explore, or reports Unsatisfiable, which means the original problem is unsatisfiable and the computation terminates.

When AVATAR encounters a clause C splittable into components: C = C₁ ∨ ... ∨ Cₙ, the clause is abstracted into the propositional split clause [C₁] ∨ ... ∨ [Cₙ] and passed to the SAT solver. This ensures that at least one of the variables [Cᵢ] will be true in the subsequent model.

When [Cᵢ] is true in the model, a clause with assertion Cᵢ ← [Cᵢ] is inserted into the FO solver. In general, a clause with assertions D ← A consists of a first-order clause D and a conjunction of propositional variables A, the assertions. The FO solver performs inferences on the first-order parts as usual, but collects assertions such that the assertion of inference’s conclusion equals the conjunction of assertions of the inference’s premises.

Unless the given problem is satisfiable, the FO solver eventually derives a contradiction clause ⊥ ← A, which in general depends on some assertions A = [C₁] ∨ ... ∨ [Cₙ]. The Splitting interface turns this information into a propositional contradiction clause ¬[C₁] ∨ ... ∨ ¬[Cₙ] and passes it to the SAT solver. Because the assertion A was true in the current model, this addition forces the SAT solver to compute a different model. This corresponds to advancing the search to a different branch in the conceptual split tree.
The above description of AVATAR omits some important details such as the maintenance of the component index, special treatment of ground components, and, most importantly, the handling of clause reductions and deletions by the FO solver which, similarly to splitting with backtracking, are in general conditional and may need to be undone when the current model changes. We refer the reader to the original paper [4] for a full account of these aspects.

3 Experimental evaluation

Our experiments [4] have shown that AVATAR shows outstanding results both in terms of its average performance and in the number of problems that it can solve and that were previously unsolvable by any existing prover.

Out of 6892 unsatisfiable problems from TPTP problem library the best splitting strategy from previous work [1] was able to solve 4381 problems while a strategy based on AVATAR solved 4716 problems. Moreover, Vampire with AVATAR was able to solve 421 problems unsolvable by Vampire without AVATAR and by any other prover. In comparison, Vampire using splitting with and without backtracking was able to solve only 17 problems unsolvable by any strategy using AVATAR. Solving over 400 previously unsolvable problems is a remarkable result since all these problems are very hard. In the past, the implementation of various novelties in Vampire would normally result in solving from a few to about 30 previously unsolvable problems.

The experimental results were so successful that all previously implemented code for handing splitting was completely removed from the latest versions of Vampire, resulting in considerable simplifications in its code and better maintainability.

4 Current and future work

Although AVATAR has already proved highly successful, its full power, and the best way to use it, is not yet fully understood. We, therefore, continue the research on AVATAR, currently focusing mainly on the following aspects.

Tuning the architecture There are several ways of configuring the basic architecture out of which none is obviously better than the others. An example of a configurable option is when exactly in the execution of a saturation algorithm to split a clause, which could as be a soon as the clause is generated or only just before the clause is selected to participate in inferences.

The effect of such options on prover performance is hard to predict, because it is typically connected with intrinsic tradeoffs and certain choices may help on some problems but not on others. We are currently investigating which option value combinations lead to the best performance in the number of solved problems, but also which option values are necessary to solve problems that cannot be solved otherwise.

Nice models Another factor which influences the performance of AVATAR is the quality of models produced by the SAT solver. Although it is not obvious what the best model looks like, it is clear that some models are better than others. For instance, introducing unnecessary assertions may lead to a slowdown in the FO solver without any compensating benefit in obtaining the ultimate contradiction.

We are currently experimenting with partial models and a heuristical minimisation procedure which eagerly removes literals not needed for satisfying the clauses in the SAT solver. Smaller models are expected to speed up the proving process provided the actual minimisation time does not become detrimental.

Theory reasoning As already explained in the original paper [4], the AVATAR architecture naturally allows for a combination of first order reasoning with theories when the SAT solver is replaced by an STM solver. Ground components composed of theory literals can then be made accessible to the SMT solver, allowing it to exclude theory-inconsistent models.

We have already obtained encouraging results for the theory of equality and uninterpreted functions. There is a plan to integrate other theories in the near future. We believe that AVATAR is an important step towards efficient reasoning with both quantifiers and theories, which is one of the key areas in modern applications of theorem provers in program analysis and verification.

References


Reasoning about Dynamic Auctions*

Marco Caminati\textsuperscript{1} \quad Manfred Kerber\textsuperscript{1} \quad Colin Rowat\textsuperscript{2}

\textsuperscript{1} Computer Science, University of Birmingham, UK \quad \textsuperscript{2} Economics, University of Birmingham, UK

Abstract: Auctions are used to sell goods worth trillions of dollars annually. Many of them are well established, frequently used, and hence well tested. However, in some domains new auction schemes are developed and only infrequently used. These can benefit most from formal methods. We report here how verified code can be extracted using Isabelle's extraction mechanism for so-called dynamic auctions, that is, auctions in which participants can bid in several rounds. We use co-recursion in order to reason about and extract code for dynamic auctions in which inputs and computations are interleaved.

1 Introduction

Auctions are an important tool to allocate goods. They come in different shapes, and for important auctions it is not uncommon that new auction schemes are developed. For instance, when western countries sold licenses for G3 frequencies in mobile telecommunication they used different auction schemes, generating proceeds which vastly differed per capita \cite{2}. Auctions such as the G3 frequency auctions may be run only very rarely, possibly only once, but still have a very high monetary value. This makes their smooth running very important.

Dynamic auctions are particularly useful since they support participants in determining their value of the goods during the auction itself. The problem in the verification of dynamic auctions is that the whole process is iterated between the input of the bids, the determination of whether the auction has terminated, and the broadcasting output of all the input bids (respectively, of announcing the winner and the prices to be paid if the auction has actually terminated). In order to represent this interaction, we assume that functionally the inputs are represented by an (a priori potentially) infinite list of bid vectors and that the actual input process is just the identity function with the side effect that the next bid vector is read in. Likewise the process of the output is functionally just the identity function with the side effect that a result computed in the verified part of the system is pretty-printed. Overall the whole approach can be described by the loop as displayed in Figure 1.

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (input) {Read input};
\node [right of=input] (compute) {Compute output};
\node [below of=compute] (termination) {Check termination};
\node [below of=termination] (output) {Pretty print output};
\path (input) edge (compute)
          (compute) edge (termination)
          (termination) edge (output);
\end{tikzpicture}
\caption{Iterated computation and input/output in a dynamic auction}
\end{figure}

In a first approximation we could just extract the parts of the algorithm that check the termination and compute the output. While it would be possible to extract the most critical parts of the algorithm this way, it would not be possible to reason about the algorithm as a whole. For instance, it would not be possible to argue that – under certain assumptions – the algorithms will terminate.

Next we use co-recursion to formalize (in Isabelle/HOL) dynamic auctions. Then we discuss how to extract Scala code.

2 Co-recursion

We want to iterate a given auction scheme, subject to halting conditions to be checked during the auction. If we manage to implement this iteration as an Isabelle function, we get the substantial advantage that both the input and the output Scala routines can be seen as functional arguments to that function. This enables us to prove theorems about the whole auction scheme inside Isabelle. For instance, we assume that the auction does not terminate as long as the unknown input function satisfies some suitable activity rule. We can then formally prove that – under the assumptions of a minimal increment and limited wealth of the bidders – the auction will eventually terminate. While the input/output routines are not extracted from the Isabelle code, we need a minimal wrapper that deals exclusively with reading in input and printing out output. The resulting Scala wrapper will be extremely thin: basically, it will consists only of the calls to the input and output routines, represented as a pair of functions to be passed as arguments to the main Isabelle generated function. This approach allows to reason about the whole algorithm in Isabelle (e.g., about the termination of the loop).

Co-recursion is known to be an ideal tool when dealing with (possibly) infinite loops, as we do; we now illustrate how we exploit co-recursion for our purpose. The basic iterator is defined as follows:

\begin{verbatim}
primcorec conditionalIterates where
  "conditionalIterates f x =
    (if (fst x)
      \dots
    \end{verbatim}

*This work has been supported by EPSRC grant EP/J007498/1.
then (LCons x
   (conditionalIterates f (f x)))
else (LCons x LNil))"

conditionalIterates takes a function \( f \), and returns a lazy list holding in the \( n \)-th place the output of \( f \) at the \( n \)-th iteration, starting from \( x \) as an initial argument of \( f \) (i.e., the output of the 0th iteration). To be able to conditionally interrupt the iteration, the output format of \( f \) must be of type \((\text{bool} \times \{a \}) \Rightarrow \text{bool} \times \{a \}\) \( \llist \), where \( \llist \) is the Isabelle type for a lazy list, and \( \{a \} \) is any type. The bool flag is used only by conditionalIterates to impose termination, while the remaining part of the output (of type \( \{a \} \) in the snippet above) is completely arbitrary. Since this remaining part of the output will contain everything needed for the computation of the auction, we will refer to it as the state data structure in the sequel.

The function \( f \) which is passed to conditionalIterates will be the composition \( \text{output} \circ \text{g} \circ \text{input} \); since it will be passed to \( f \) for the next iteration, \( \text{input} \) must take a variable having the same type as an argument. Besides that, \( \text{output} \) must only print a message for the user: hence the overall type of \( \text{output} \) will be \( \text{String.literal} \times \text{bool} \times \{a \} \Rightarrow \text{bool} \times \{a \} \), with the first component (of type \( \text{String.literal} \)) printed and then discarded: since, besides the printing side effect, \( \text{output} \) does nothing, it is computationally close to the identity function.

Analogously, \( \text{input} \) must only read some input from the user, and pass it on to \( \text{g} \), along with all the arguments it received from the previous iteration. Hence, the type of \( \text{input} \) is \( \text{bool} \times \{a \} \Rightarrow \text{integer} \times \text{bool} \times \{a \} \). The integer component of the output will contain the new bid for the current round, which \( \text{g} \) will then take care to incorporate into the state data structure.

Finally, the Isabelle definition which will be evaluated is the following:

```isar
definition "dynamicAuction input output
   = conditionalIterates (output \circ \text{g} \circ \text{input}) (True, [])"
```

\( \text{g} \) manipulates the state data structure and generates the flag to be passed to conditionalIterates (according to the given auction specification); it also generates the string to be printed by \( \text{output} \), and incorporates the new bid into the state data structure at each iteration.

### 3 Scala code extraction

In the previous section we have seen that from a functional point of view, \( \text{g} \) leaves very little to do to \( \text{input} \) and \( \text{output} \). \( \text{input} \) only augments its arguments by a number and pass the whole object on, and \( \text{output} \) only discards the first component of its argument and passes the rest along. We are mainly interested in the side-effects \( \text{input} \) and \( \text{output} \) can achieve, corresponding to the lines containing \text{readInt} and \text{println} in the following wrapper:

```scala
def input[A](n:(Boolean, List[A])) : (BigInt, (Boolean, List[A])) = {
  val x=readInt;
  return (x,n);
}
```

```scala
def output[A](x: (String, A)) : A = {
  println(fst (x));
  return (snd (x));
}
```

```scala
def main(args: Array[String]) {
  dynamicAuction(input, output);
}
```

We append the lines above to the Scala code generated by Isabelle (which includes \text{dynamicAuction}), and the resulting code can be executed interactively to run an iterative auction from the console.

### 4 Conclusion

In our work we build up on the formalization of co-recursive functions which Paulson did already 30 years ago [3] and has been integrated into the Isabelle system to reason about lazy lists in different contexts that require to deal with potentially unbounded expressions [1].

We showed how the coupling of co-recursion and formal methods permits to generate code to run dynamic auctions, while preserving the possibility of proving theorems about the generated code. Note that the auctions can vary in how complicated the rules are which determine whether bidders are active, bids valid, how the winner is determined, and how much a bidder has to pay. Still the same dynamic co-recursive approach can be used to model the dynamic aspects of the auction. Our approach allows to minimize the hand-written code, in the sense that it consists of the side-effect routine calls alone.

### References


Characterising the workload of SMT solvers for program verification

Andrew Healy\(^1\)  \quad Rosemary Monahan\(^2\)  \quad James F. Power\(^3\)

\(^1\) Dept. of Computer Science, Maynooth University, andrew.healy@nuim.ie
\(^2\) Dept. of Computer Science, Maynooth University, rosemary@cs.nuim.ie
\(^3\) Dept. of Computer Science, Maynooth University, jpower@cs.nuim.ie

Abstract: Through the use of profiling and coverage tools, our research aims to make a useful comparison between the performance of SMT (Satisfiability Modulo Theories) solvers under a program verification workload to that of a generic SMT benchmarking session.

1 Introduction

SMT solvers extend the capabilities of Boolean SAT solvers by combining decision procedures for a range of logics (including uninterpreted functions, linear arithmetic, and bitvectors) using the Nelson-Oppen method. Their increased expressive power has made them a popular back-end for deductive software verification frameworks as well as more traditional constraint-solving application domains such as scheduling, micro-processor verification and model-checking [5].

More often than not, an SMT solver is a highly complex and specialised piece of software which verification systems treat in a “black box” manner. Our research aims to expose the parts of the solver that are exercised by software verification, how much time is spent on these operations, and how the parts of the solver interact during this task. It is our assertion that by comparing our results against equivalent execution profiles obtained from runs of generic SMT benchmarking problems, we can evaluate the relevance of standard SMT benchmarks to the domain of program verification.

2 Motivation

The following are a number motivating factors and questions we would like to answer in this study:

2.1 Can the lack of standard benchmark programs for a comparative evaluation of software verification systems be remedied by generic SMT-LIB benchmarks?

The SMT-LIB project [1] has been very successful in building a repository of standard programs to assist the SMT community in evaluating their tools. However, there is no corresponding repository of software verification benchmark programs [2]. This is largely due to the diverse range of specification languages and annotated implementation languages that such systems use as input. Yet multiple verification systems use the same SMT solver as the final component in their workflow. We aim to investigate whether this allows for a valid, equivalent evaluation of deductive verification systems.

2.2 Are the best-performing SMT solvers the most suitable for software verification systems?

SMT-solving technologies have evolved rapidly in recent years. Competitions such as SMT-COMP have played an important role in defining the top-performing tools and accelerating their development [4]. As the constraint-solving tasks in these competitions do not specifically focus on verification, it does not necessarily follow that the rankings are applicable for software verification systems. If, for example, a program requires extensive proof by induction to be verified, interactive provers will fare better than the (much faster) automatic tools.

3 Approach

This section outlines our proposed workflow, as illustrated in Figure 1.

SMT solvers: The SMT solvers we will use are CVC4\(^2\) and Z3\(^3\). These particular tools were chosen for their shared characteristics which aid comparison: they are both consistently high-performing solvers written in C++ [4]. Their implementation language is important to consider if we are to use the same profiling tools on both solvers. Both are SMT-LIB v2 compliant and support benchmark problems using all logics.\(^4\) Both are used as back-ends in multiple verification systems.

Input generation: In order to avoid a bias due to a tight integration between verification condition generator and solver, the Why3 platform [3] was chosen for use in the software verification side of this study. In the Why3 workflow, solver-specific drivers are used to convert program specifications (including quantifiers and types) to a series of SMT-LIB assertions that a solver can interpret using its range of supported logics. This approach

\(^{2}\)http://cvc4.cs.nyu.edu/web/  
\(^{3}\)http://z3.codeplex.com  
\(^{4}\)AUFNIRA in SMT-LIB notation: Arrays, Uninterpreted Functions, NonLinear Integer Reals Arithmetic


Figure 1: The workflow for treating verification challenges as benchmarks using the Why3 platform. Note that the .smt2 file produced by Why3 contains all the verification goals for the input program. These must be split and the individual goal files used as input to the SMT prover to avoid errors. The workflow for SMT-LIB benchmarks is simpler as there is only one goal in each file.

4 Work to date

Our work is still in the early stages of developing a suitable experimental set-up for evaluating the workload of the verification benchmarks. We are currently engaged in profiling the debug builds of CVC4 and Z3 in order to gain insight into their operation, and to develop an understanding of how the input workloads correspond to their internal operation.

To this end we are using a fairly standard set of C++ profiling tools: the callgrind tool\(^5\) to generate the call graph, gprof to gather profiling information about time spent on each SMT solver operation, and gcov to record the code coverage data.

We hope that the information provided by these program analysis tools will provide an insight into the relative performance of these SMT solvers, and assist in developing an understanding of their performance for typical program verification tasks.

Acknowledgements

This project is being carried out with funding provided by Science Foundation Ireland under grant number 11/RFP.1/CMS/3068.

References


\(^5\)part of the valgrind suite (http://www.valgrind.org)
Saturation-Based Reasoning for SHI-Knowledge Bases with Applications to Forgetting and Uniform Interpolation

Patrick Koopmann Renate A. Schmidt
University of Manchester
{koopmannp,schmidt}@cs.man.ac.uk

Abstract: We present a new reasoning method for SHI knowledge bases that is based on saturation and resolution, and that can be used for computing uniform interpolants and results of forgetting. The description logic SHI is a non-Horn description logic that supports transitive roles, role hierarchies and inverse roles. The presented method can be used for reasoning in knowledge bases that consist both of a TBox and an ABox. Furthermore, whereas other saturation-based reasoning methods usually only offer completeness for refutation and classification, this calculus is also interpolation complete, that is, it can be used for forgetting and computing uniform interpolants of SHI knowledge bases.

1 Introduction

Ontologies model terminological information for a particular application domain on the basis of concepts and roles. A knowledge base additionally contains an ABox, which stores information about individual objects, similar to a database. Typically such a knowledge base is described using a description logic, which allows to use automated reasoning systems to infer implicit information from the knowledge base. Uniform interpolation, also known under the name forgetting, projection, and variable elimination, restricts the set of concepts and roles in an ontology to a specified signature, such that all entailments in that signature are preserved. It has a wide range of applications for ontology reuse, ontology analysis, privacy and many more, and has received an increased interest in the last years [1, 6]. As shown in [6, 7], it is also a challenging task, and for expressive description logics uniform interpolants can in the worst case be of triple exponential size in the size of the input. For practicality, it is therefore necessary to use a goal-oriented approach for the computation of uniform interpolants. Results in [5, 1] show that saturation-based reasoning is a well-suited technique for this purpose. By using a specific normal form, in [2] a resolution-based method for uniform interpolation is devised that always terminates with a finite result, which might require the use of fixpoint operators in some cases. In [4], this approach is extended for knowledge bases with ABoxes. In this abstract, we present an extension of the reasoning method used in [4] for the description logic SHI, which can be used for computing uniform interpolants in this language.

2 Normalisation

The calculus requires the input to be represented in a specific normal form. Let \( N_d \) be a set of designated concept symbols called definers. A concept literal is a concept of the form \( A, \neg A, \exists R.D \) or \( \forall R.D \), where \( R \) is of the form \( r \) or \( r^{-1} \) (inverse role) and \( D \in N_d \). A TBox literal is of the form \( L(x) \), where \( L \) is a concept literal, and an ABox literal is of the form \( L(a) \), where \( L \) is a concept literal and \( a \) a constant. A TBox clause is a disjunction of TBox literals. An ABox clause is either a disjunction of ABox literals, or a role assertion of the form \( R(a, b) \), where \( R \) is of the forms \( r, r^{-1} \), and \( a, b \) are constants. An RBox clause is a role axiom of the form \( R \sqsubseteq S \) or \( \text{trans}(R) \), where \( R, S \) is of the form \( r \) or \( r^{-1} \). A SHI knowledge base is in SHI normal form if it is a set of clauses. We further assume that for any clause of the form \( R \sqsubseteq S \), \( \text{trans}(R) \) and \( R(a, b) \), there is a corresponding clause of the form \( \text{Inv}(R) \sqsubseteq \text{Inv}(S) \), \( \text{trans}(\text{Inv}(R)) \) and \( \text{Inv}(R)(b, a) \) in the clause set, where \( \text{Inv}(R) \) is defined as \( r^{-1} \) if \( R = r \), and \( r \) if \( R = r^{-1} \).

3 The Calculus

The calculus uses the rules shown in Figure 1. The calculus makes use of a very simple kind of unification on the terms that can occur in a clause: specifically, one can only unify identical terms or variables to constants, since our representation has no Skolem terms and uses maximally one variable per clause. Furthermore, in order to preserve the normal form and for termination reasons, new symbols might have to be introduced during the application of a rule. A new definer \( D_{12} \) representing the conjunction \( D_1 \sqcap D_2 \), as required by the role propagation rule, is introduced by the addition of the two clauses \( \neg D_{12} \sqcup D_1 \) and \( \neg D_{12} \sqcup D_2 \) (see [2] for details).

The first 4 rules, starting from the resolution rule and ending with the role instantiation rule, are directly taken from the calculus introduced in [4] for uniform interpolation of ALC knowledge bases. The next 4 rules, starting with the role hierarchy rule and ending with the transitivity rule, are adaptations of corresponding rules for reasoning in SHIQ presented in [3]. The role inversion rule is new and handles inverse roles. As it turns out, it is sufficient to apply this rule on clauses in the original clause set and on conclusions of the transitivity rule. Such a limitation is also required in order to obtain termination of the calculus, since otherwise an arbitrary number of new definers would
Resolution
\[
C_1 \vee A(t_1) \quad C_2 \vee \neg A(t_2) \\
(\exists \sigma D)\sigma
\]


\[C \vee (\exists R.D)(t) \quad \neg D(x) \quad C\]

\[C_1 \vee (\forall R.D)(t_1) \quad C_2 \vee (Q.R.D_2)(t_2) \\
(\exists \sigma D)\sigma \vee (Q.R.D_2)(\beta t_1)
\]

\[C_1 \vee (\forall R.D)(t_1) \quad R(t_2, b) \\
(\exists \sigma D)\sigma \vee D(b)
\]

\[R_1 \subseteq R_2 \quad R_2 \subseteq R_3 \\
R_1 \subseteq R_3
\]

\[C \vee (\exists R.D)(t) \quad R \subseteq S \\
C \vee (\exists S.D)(t)
\]

\[C \vee (\forall R.D)(t) \quad S \subseteq R \\
C \vee (\forall S.D)(t)
\]

Transitivity
\[
C \vee (\forall R.D)(t) \quad \text{trans}(R) \\
C \vee (\forall R.D')(t) \\
\neg D'(x) \vee D(x) \\
\neg D'(x) \vee (\forall R.D')(x)
\]

Role Inversion
\[
C_1(x) \vee (\forall R.D_1)(x) \\
D_1(x) \vee (\forall \text{Inv}(R).D_2)(x) \\
\neg D_2(x) \vee C_1(x)
\]

where \(\sigma\) is the unifier of \(t_1\) and \(t_2\) if it exists, \(D_{12}\) is a possibly new definer representing \(D_1 \cap D_2\).

Figure 1: The rules of the calculus.

be introduced by this rule. For any other rule introducing a new definer, we can make sure that the number of introduced definers is bounded by \(O(2^n)\), where \(n\) is the number of definers in the input clause set, using the technique described in [2]. By careful adaptation of the completeness proof presented in [1], we can prove the following theorem:

**Theorem 1.** The calculus is sound, terminating and refutationally complete.

We can furthermore prove that the calculus is also interpolation complete. That is, for any knowledge base \(\mathcal{K}\) and any signature \(\mathcal{S}\), we can compute a finite uniform interpolant as follows: (1) transform \(\mathcal{K}\) into set of clauses \(\mathcal{N}\), (2) saturate \(\mathcal{N}\) using the calculus, (3) filter out clauses that are not in \(\mathcal{S} \cup \mathcal{N}_d\) and that have more than one negative definer, (4) transform the resulting clause set into a knowledge base without definers using the technique described in [4]. The resulting uniform interpolant might use greatest fixpoint operators and nominals, since uniform interpolants of \(SHIQ\) knowledge bases cannot always be represented finitely without these [6, 4]. However, the fixpoint operators can always be simulated using helper concepts, or used as a basis for an approximation of the uniform interpolant [2].

**Theorem 2.** The calculus is interpolation complete.

**References**


Debugging of $\mathcal{ALC}$-Ontologies via Minimal Model Generation

Fabio Papacchini  Renate A. Schmidt

School of Computer Science, The University of Manchester, \{papacchf, schmidt\}@cs.man.ac.uk

Abstract: This short abstract revises a procedure for the generation of minimal models for propositional modal logics, and explains how it can be used for $\mathcal{ALC}$-ontology debugging.

1 Introduction

Model generation and minimal model generation are useful for computer science tasks such as fault analysis, model checking and debugging of logical specifications [6]. For this reason, there have been studies on minimal model generation for classical and non-classical logics [1, 4, 5, 2].

[5] suggests the possibility of using minimal model generation techniques as a complementary debugging notion to the more common notion of ontology debugging such as the one discussed in [7, 3], where an ontology is considered bugged if it is incoherent. The idea in [5] is that even a coherent ontology can be considered faulty when it does not model properly the domain of interest. This can be because aspects and properties expected to hold for the domain of interest do not follow from the ontology. Formally, given an ontology $O$ and a property $\gamma$ that $O$ is supposed to have, if $O \nvdash \gamma$ then $O$ is faulty. In this context procedures for the generation of minimal models, similarly to test-driven software development paradigms, complement the notion of ontology debugging and help to model correctly the domain of interest. Minimal model generation procedures can be used to check whether these properties hold at any stage of the life cycle of the ontology, and then corrected based on the returned models.

This abstract presents a minimal model generation procedure for the description logic $\mathcal{ALC}$.

2 Logic and Minimality Criterion

Syntax and semantics of the description logic $\mathcal{ALC}$ are defined as usual. The minimality criterion used is the same as in [5]. The minimality criterion establishes the minimality of a model by comparing it with other models by means of a relation called subset-simulation. To ease the presentation of subset-simulation, we define a valuation function $V$ as follows.

$$V(a) = \{ A \in \mathcal{N}_C \mid a \in A^T \}$$

Let $I = (\Delta^I, \cdot^I)$ and $I' = (\Delta'^{I'}, \cdot^{I'})$ be two models of an $\mathcal{ALC}$ formula $\phi$. A subset-simulation is a binary relation $S \subseteq \Delta^I \times \Delta'^{I'}$ such that for any two $a \in \Delta^I$ and $a' \in \Delta'^{I'}$, if $aSa'$ then the following hold.

- $V(a) \subseteq V'(a')$, and
- if $(a, b) \in r^I$, then there exist a $b' \in \Delta'^{I'}$ such that $(a', b') \in r'^{I'}$ and $bSb'$.

If there is a full subset-simulation from a model $I$ to a model $I'$, then $I'$ subset-simulates $I$ (i.e., $I \leq I'$).

Subset-simulation is a preorder on models, and it can be used to define the following minimality criterion. A model $I$ of an $\mathcal{ALC}$ formula $\phi$ is minimal modulo subset-simulation iff for any model $I'$, if $I' \leq I$, then $I \leq I'$.

To have a visual idea of the minimality criterion, Figure 1 shows two possible models of a formula. The subset-simulation relationship is represented by directed dashed line. Given the definition of the minimality criterion, the model on the left is considered minimal because it is subset-simulated by the model on the right.

![Figure 1: Example of minimality w.r.t. subset-simulation](image)

As subset-simulation is not anti-symmetric, there can be models that subset-simulate each other, resulting in a symmetry class w.r.t. the preorder. As a result, infinitely many models minimal modulo subset-simulation can belong to the same symmetry class. To avoid the generation of all such models, and because they entail the same positive formulae, we consider a procedure to be minimal model complete if it generates at least one minimal model for each symmetry class of minimal models.

Models minimal modulo subset-simulation spread the positive information into several domain elements, while minimising the valuation function. We believe that this results in minimal models that are easier to understand.

3 Minimal Model Generation Procedure

The procedure for the generation of models minimal modulo subset-simulation for the description logic $\mathcal{ALC}$ is an adaptation of the one proposed in [5]. It is composed of a tableau calculus and a minimality test. For reasons of space we focus only on the modification of the tableau calculus. This is because while the calculus needs to be adapted to handle $\mathcal{ALC}$ ontologies, the minimality test is exactly as in [5]. Table 1 shows the rules of the calculus.

The input of the calculus is assumed to be in negation normal form and all the TBox axioms are supposed to have the form $\neg C \sqcup D$. $\Phi$ represents a disjunction of formulae, $\Phi_D$ a disjunction with no negated basic concept,
and $\Phi^+$ is as $\Phi^+_a$ except that no conjunction is allowed as a disjunct.

The $(\forall)$ and $(\exists)$ rules are the common rules for description logic formulae under role restrictions. The $(TBox)$ rule is used to instantiate TBox axioms with the domain elements constituting the current domain. The $(\alpha)$ rule performs a lazy classification step and, if $\Phi^+_a = \top$, expands a conjunction. Due to the possibility that the property that the ontology is supposed to have can be a set of ABox statements, the calculus needs to be able of handling formulae where concepts belong to different domain elements (for example, $\forall x. C(x) \land B(y)$). For this reason and to have a unified way to deal with such formulae, the $(\forall)$ rule distributes the domain element over a disjunction of concepts. The $(\beta)$ rule is a complement splitting rule that aims to close a branch from which a non-minimal model can be extracted. The $(SBR)$ rule is a selection-based resolution rule. It is the only closure rule of the calculus, aims to remove all the negative information from a disjunction and, together with the $\Phi^+$ and $\Phi^+_a$ restrictions in other rules, reduces the number of inferences resulting in non-minimal models. It is possible to note that the $(SBR)$ rule allows the main premise to have negated role instances. This does not mean that we cover a logic where negation over roles is allowed, but that the calculus needs to handle negated role instances resulting from the negation of ABox statements such as $r(a, b)$.

The calculus in Table 1 is sound and refutationally complete. When augmented with the minimality test, we obtain a procedure that is minimal model sound and complete. Termination can be achieved by dynamic ancestor equality blocking, as proved in [5].

### 4 Conclusion

The abstract presented a procedure for the generation of models minimal modulo subset-simulation for $\mathcal{ALC}$ ontologies. This procedure is just a first step of a work in progress that aims to implement the procedure and its generalisations to more expressive description logics such as $SHI$.

### References


---

### Table 1: Rules of the tableau calculus

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\forall)$</td>
<td>$\frac{r(a, b)}{C(b)}$</td>
</tr>
<tr>
<td>$(\forall r. C)(a)$</td>
<td></td>
</tr>
<tr>
<td>$(TBox)$</td>
<td>$\frac{-C \sqcup D}{(-C \sqcup D)(a)}$</td>
</tr>
<tr>
<td>where $a$ appears on the branch.</td>
<td></td>
</tr>
<tr>
<td>$(\beta)$</td>
<td>$\frac{C(a) \lor \Phi^+}{C(a) \lor \Phi^+}$</td>
</tr>
<tr>
<td>$\text{neg}(\Phi^+) = { (\neg A)(a) \mid A(a) \text{ is a disjunct of } \Phi^+ }$</td>
<td></td>
</tr>
<tr>
<td>$(SBR)$</td>
<td>$A_1(a_1) \ldots A_n(a_n) \ r(b_1, c_1) \ldots r(b_m, c_m) \ u : (\neg A_1)(a_1) \lor \ldots \lor (\neg r(b_m, c_m)) \lor \Phi^+_a$</td>
</tr>
</tbody>
</table>
Abstract: Big ontologies contain large numbers of concept symbols and there are situations where, depending on the applications, parts of them need to be forgotten. Forgetting can be realised using second-order quantifier elimination (SOQE) techniques. In this paper, we present a SOQE-method to forget concept symbols in ontologies specified in the description logic $\mathcal{ALCOI}$, and express the ontologies in terms of the remaining symbols. Forgetting allows one to select the concept symbols that one is interested in, on the one hand; and on the other hand it allows one to create a view of an ontology expressed using only selected set of concept and role symbols.

1 Introduction

This paper is concerned with the forgetting of concept symbols in ontologies specified in the description logic $\mathcal{ALCOI}$. The understanding of forgetting, with respect to ontologies $(\mathcal{O})$, is to define a set $\mathcal{N}$ that contains all the concept symbols to forget in $\Sigma(\mathcal{O})$, namely the non-base symbols, model them as existentially quantified, and apply the second-order quantifiers elimination methods to the axioms in $\mathcal{O}$ to forget these quantifiers.

What remains after forgetting, if it terminates successfully, is a reformulated ontology, with the non-base symbols having been eliminated. To put it in another way, what is returned is an ontology specified in terms of only the symbols $\Sigma(\mathcal{O}) \setminus \mathcal{N}$. There are situations where our method does not succeed or the concept symbols are simply ineliminable. The concept symbols cannot be eliminated does not necessarily mean that they are ineliminable. Another possibility is that they are eliminable, but simply our method is unable to find a solution. The problem of second-order quantifiers elimination is known to be highly undecidable [1], which means that there are no general methods guaranteed that allows for concept forgetting of description logics. Concept forgetting is already not computable for the description logic $\mathcal{EL}$ [2]. Therefore in the area of second-order quantifiers elimination the interest is focussed on understanding in what classes of problems the concept symbols are always eliminable, and developing methods that allow for the elimination of concept symbols in more cases.

2 The Method DSQEL

We present a method for eliminating existential second-order quantifiers for the description logic $\mathcal{ALCOI}$. The method is called DSQEL, which is short for Description logics Second-order Quantifier ELimination.

The Ackermann rule and the purify rule are the two elimination rules in the DSQEL calculus that are specifically used for eliminating concept symbols (see Figure 1). Both of them have to meet particular requirements on the form of the formulae which they apply. $\mathcal{N}$ is a set of $\mathcal{ALCOI}$-formulae, and by $\mathcal{N}_\alpha^\beta$, we mean the set obtained from $\mathcal{N}$ by substituting the expression $\alpha$ for all occurrences of $\beta$ in $\mathcal{N}$.

\begin{align*}
\text{Ackermann:} & \quad \frac{\mathcal{N}, \alpha_1 \cup A, \ldots, \alpha_n \cup A}{\mathcal{N}_\alpha^\beta} \\
& \text{provided} \\
& (i) \quad A \text{ is a non-base symbol,} \\
& (ii) \quad A \text{ does not occur in any of the } \alpha_i, \\
& (iii) \quad A \text{ is strictly maximal wrt. each } \alpha_i, \text{ and} \\
& (iv) \quad \mathcal{N} \text{ is negative with respect to } A.
\end{align*}

\begin{align*}
\text{Purify:} & \quad \frac{\mathcal{N}}{\mathcal{N}_\alpha^\beta} \\
& \text{provided} \\
& (i) \quad A \text{ is a non-base symbol in } \mathcal{N}, \text{ and} \\
& (ii) \quad \mathcal{N} \text{ is negative with respect to } A.
\end{align*}

Figure 1: Elimination rules

The purify rule can be seen as a special case of the Ackermann rule as it eliminates the non-base symbols that occur only negatively, that is, there are no positive occurrences of $A$.

\begin{align*}
\text{Case Splitting:} & \quad \frac{\mathcal{N}, \neg a \cup \beta_1 \cup \ldots \cup \beta_n}{\mathcal{N}, \neg a \cup \beta_1 \ldots \cup \beta_n} \\
& \text{provided} \\
& (i) \quad A \text{ is the largest nb-symbol in } \neg a \cup \beta_1 \ldots \cup \beta_n, \\
& (ii) \quad A \text{ occurs positively in } \beta_1 \cup \ldots \cup \beta_n.
\end{align*}

Figure 2: Case splitting

The rest of the rules in the DSQEL calculus are used to rewrite the formulae so they can be transformed into the form where either the Ackermann rule or the purify rule is applicable.

The highlight of the DSQEL calculus is the new case splitting inference rule (see Figure 2). It has shown to be advantageous to decreasing the search space and improving the success rate.

We also introduced some simplification rules, which are helpful in that, they transform more expression so that infer-


<table>
<thead>
<tr>
<th>Input</th>
<th>Experiment</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corpura</td>
<td>Ontologies No.</td>
<td>Axioms Avg.</td>
</tr>
<tr>
<td>1 – 1000</td>
<td>224</td>
<td>652</td>
</tr>
<tr>
<td>1001 – 4000</td>
<td>53</td>
<td>3091</td>
</tr>
<tr>
<td>≥ 4001</td>
<td>15</td>
<td>6506</td>
</tr>
<tr>
<td>Total</td>
<td>292</td>
<td>1407</td>
</tr>
</tbody>
</table>

Table 1: Evaluation results

\[
\frac{C \sqcup \forall R^* \exists \forall R^* \ldots \forall R^* \forall R^* \neg (C \sqcup D)}{C \sqcup \forall R \exists \neg (C \sqcup \forall R^* \neg)} \quad \text{provided}
\]
\[(i) \text{ } C \text{ and } D \text{ are arbitrary concepts, and }
(ii) \sigma^i \leq \sigma^n \text{ for } 1 \leq i \leq n\]

Figure 3: The generalised factoring

In order to evaluate how the DSQEL method behaves on real-life ontologies, we implemented it in Java using the OWL API and applied the implementation to a set of ontologies from the NCBO BioPortal, a large repository of biomedical ontologies.

Since DSQEL handles expressivity as far as ALCOI, the ontologies for our evaluation were restricted to their ACCOl-fragments and axioms outside of the scope of ACCOl were dropped from the ontologies. In this way, we used 292 ontologies from the repository for our evaluation. The selected ontologies were divided into three groups with each of them containing the ontologies with their numbers of concept symbols ranging from 1 to 1000, from 1001 to 4000, and more than 4000, respectively. We ran the experiments on each ontology 100 times and took the results on an average basis to explore how forgetting was influenced by the number of the concept symbols. To show the difficulty of the forgetting problems, and how well our method behaved, we considered an extreme case where we tried to forget all concept symbols from each ontology. A timeout of 1000 seconds was used to challenge our implementation. Evaluation results are shown in Table 1.

4 Related Work

There are a number of methods to elimination of second-order quantifiers, such as SCAN, DLS, DLS*, SQEMA, and MSQEL [1]. SCAN handles formulae only in clausal form and involves unskolemisation, which is not generally decidable. DLS and DLS* also involve unskolemisation. SQEMA and MSQEL avoid unskolemisation, but work only on modal formulae. Koopmann and Schmidt presented a method for computing uniform interpolants of ALC-ontologies, which uses fixpoint operators to make uniform interpolants finitely representable [3]. The method was further extended to handle ALC, SHQ, and ALC with ABoxes.

DSQEL contributes in three respects. It is the first method for forgetting concept symbols from ontologies specified in the description logic ALCOI. It inherits from MSQEL the consideration of elimination orders, which has been empirically shown to decrease the execution time of the forgetting. It also incorporates the case splitting rule and generalised simplification rules, making it applicable to a wider range of problems.

5 Conclusion

We have presented a second-order quantifiers elimination method that allows for forgetting concept symbols in ontologies specified in the description logic ALCOI. It was adapted from MSQEL, a SOQE-method specifically designed for modal logics, and further developed by incorporating new rules. The adaptation was motivated for the purpose of applying the second-order quantifier elimination techniques to the area of knowledge representation, where description logics provide important logical formalisms. The new rules, including case splitting, and the generalised simplification rules, enrich DSQEL with the ability to solve elimination problems over new classes of formulae where existing methods are unable to handle.

References


A Tool for the Formal Verification of Quantum Optical Computing Systems

Sidi Mohamed Beillahi, Mohamed Yousri Mahmoud and Sofiène Tahar

Electrical and Computer Engineering Dept., Concordia University, Montréal, Québec, Canada

{beillahi, mo_solim, tahar}@ece.concordia.ca

Abstract: Quantum optics systems provide a promising platform for universal quantum computing, since they link quantum computation and quantum communication in the same framework. Thanks to their high capability, quantum computers are considered good candidates to replace classical cryptography and supercomputing systems which require a robust and accurate verification process. In this paper, we introduce a tool for the formal verification of quantum computing systems based on the soundness and accuracy of high-order-logic theorem proving augmented by a set of decision procedures to automate the proof process.

1 Introduction

Quantum computers promise to increase greatly the efficiency of solving problems such as factoring large integers, combinatorial optimization and quantum physics simulation. Compared to traditional computers, quantum computers allow performing optimization calculations exponentially faster. It has been proved that it is indeed possible to create universal quantum computers with linear optics (e.g., beam splitters, phase shifters), single photons, and photon detection, resulting of what is called Linear Optical Quantum Computing (LOQC) systems [3]. Despite major progresses in the area, several challenges remain unresolved. For example, the verification of quantum devices and algorithms did not yet receive enough attention, particularly when quantum devices and algorithms are used in some high computing domains (i.e., aerospace, cybersecurity, biomedical...). In order to verify quantum computers which exploit the laws of quantum physics we have to express these laws in mathematical forms. Generally, quantum devices and protocols are verified using traditional techniques (e.g., numerical methods, lab simulations...) which are uncertain, and costly. In lab simulation, in order to measure the effect of different initial conditions or parametric variation over the quantum circuit operation, it is necessary to perform exhaustive tests. However, even by doing this, there is no guarantee of the full correctness of the results, because it will be necessary to test the system for an infinite number of operating conditions. We believe that there is a dire need of an accurate framework to build high assurance quantum systems. In the last decades, formal methods based techniques have proven to be an effective approach to analyze physical systems, thanks to their inherent states, beam splitters, phase shifters, and flip gates which are the counterparts of classical gates (i.e., OR, AND, XOR) in quantum physics. The next step is to build a verification tool, based on the same underlying theory, to reason about larger circuits composed of the above universal gates. Recently, a work on the formalization of linear quantum optics was proposed in [7] using process calculus. However, process calculus is mostly used for the verification of concurrent systems such as quantum cryptography. Moreover, the models expressed using this technique are quite complex to understand and proof steps are lengthy and not readable, especially for large circuits. Another work was proposed in [8] to model and analyse quantum systems using the CWB-NC model checker. Though, the CWB-NC can only be used for verifying and modeling finite-state concurrent systems. In [4], a comprehensive linear algebra library was formalized in the HOL Light theorem prover [1], using this library, coherent states, beam splitters, phase shifters, and flip gates were formalized in higher-order-logic [5]. In our work, we will build on top of this formalization, for the sake of developing a tool for formalizing quantum systems in a more sound and expressive fashion.

2 Quantum Linear Optics

Linear optical networks, i.e., passive networks constructed from beam splitters, phase shifters and mirrors, can be used to build universal quantum computing system. In LOQC systems, the qubit is usually taken as a single photon that can be in two different modes, \( |0\rangle_L \equiv |1\rangle \otimes |0\rangle \equiv |1, 0\rangle \) and \( |1\rangle_L \equiv |0\rangle \otimes |1\rangle \equiv |0, 1\rangle \) which is called dual rail. When the two modes represent the internal polarization degree of freedom of the photon \( |0\rangle_L = |H\rangle \) and \( |1\rangle_L = |V\rangle \), we call it a polarization qubit. Knill, Laflamme, and Milburn [2] showed that linear optics combined with adaptive measurements is considered a viable proposal for building a universal quantum computer. Although logical gates can only be implemented probabilistically, it has been proved that they can be rendered deterministic (near-deterministic) by making use of ancillary resources, mea-
measurements, cluster-state techniques, and feed-forward [3]. The controlled-phase or CZ gate can be constructed in linear optics using two nonlinear sign (NS) gates [3] and the Fredkin gate can be constructed using quantum linear swap gate and beam splitters. On the other hand, the NS and swap gates can be built using linear optics elements.

3 Proposed Tool Structure

The proposed structure, given in Figure 3, outlines the main idea to graphically model quantum optical systems and to formally prove that they satisfy certain system specifications. The whole framework can be decomposed into two major parts. First, the graphical interface and the connection with HOL, where the quantum circuit will be drawn by connecting the chosen components from available quantum blocks to obtain the final circuit. Also the quantum circuit specification should be represented or given as parameters of the circuit. Consequently, for the connection between HOL and the interface, we will use Java Script Object Notation (JSON). JSON has been proposed as parser in [6] to implement a graphical interface connected to HOL, to analyze web services. The parser will then express the system model and specifications as HOL predicates. Next, the interface can execute HOL scripts to conduct the formal proof of the given theorem, by applying developed rules and tactics in HOL to provide effective automation. These tactics are automated in order to reduce the painful manual interaction often required with interactive (higher-order-logic) theorem proving. Once the formal verification completed, the parser will take the proof result to be visualised in the interface.

The second part is composed of a library for quantum optics, which mainly contains quantum universal gates and most common applications of these gates. Note that, our formalization will be based on the formalization of linear space, and basic quantum components (e.g., phase shifter, beam splitter and interferometer) [3]. Using this we are going to formalize the notions of linear quantum optics. Next, we will conduct the formal verification of frequently used quantum universal gates such as the Fredkin and CZ gates. Finally, using these gates we can investigate some widely used quantum applications (i.e., Shor’s and Grover’s algorithms, and Benes network). By then, we will have a complete HOL library of quantum optics and some common applications in the field.

4 Conclusion

We introduced a new alternative for the verification of quantum optical systems. An overview of our proposed tool and quantum linear optics were presented. We are currently working on the formalization of quantum universal gates and the implementation of the first components of the tool to conduct the formal verification of quantum systems.

References

CLProver++: An Ordered Resolution Prover for Coalition Logic

Ullrich Hustadt\(^1\)  Paul Gainer\(^1\)  Clare Dixon\(^1\)  Cláudia Nalon\(^2\)  Lan Zhang\(^3\)

\(^1\) Department of Computer Science, University of Liverpool  
{uhustadt, sgpgaine, cldixon}@liverpool.ac.uk  
\(^2\) Department of Computer Science, University of Brasilia  
nalon@unb.br  
\(^3\) Information School, Capital University of Economics and Business, China  
lan@cueb.edu.cn

Abstract: We present CLProver++, a theorem prover for Coalition Logic a non-normal modal logic for reasoning about cooperative agency. CLProver++ is based on recent work on an ordered resolution calculus for Coalition Logic. We provide an overview of this calculus, give some details of the implementation of CLProver++ and present an evaluation of the performance of CLProver++.

1 Introduction

Coalition Logic CL was introduced by Pauly [5] as a logic for reasoning about what groups of agents can bring about by collective action. CL is a propositional modal logic over a countably infinite set \(\Pi\) of propositional symbols and a non-empty, finite set \(\Sigma \subseteq \mathbb{N}\) of agents with modal operators of the form \([A]\), where \(A \subseteq \Sigma\). The formula \([A]\varphi\), where \(A\) is a set of agents and \(\varphi\) is a formula, can be read as the coalition of agents \(A\) can bring about \(\varphi\). Formally, the semantics of CL formulae is given by Concurrent Game Models (CGMs), see [3]. The satisfiability problem of CL is PSPACE-complete [5]. Various decision procedures for Coalition Logic exist including tableaux and unrefined resolution calculi.

In the following we present a new decision procedure for CL based on ordered resolution, briefly describe its implementation and present its evaluation.

2 Ordered Resolution for CL

Our ordered resolution calculus does not operate on CL formulae, but on formulae of Vector Coalition Logic (VCL) in a clausal normal form.

Let \(|\Sigma| = k\). A coalition vector \(\vec{c}\) is a \(k\)-tuple such that for every \(a\), \(1 \leq a \leq k\), \(\vec{c}[a]\) is either an integer number not equal to zero or the symbol \(*\) and for every \(a, a'\), \(1 \leq a < a' \leq k\), if \(\vec{c}[a] \neq 0\) and \(\vec{c}[a'] \neq 0\) then \(\vec{c}[a] = \vec{c}[a']\).

The set \(\text{WFF}^{\text{VCL}}\) of Vector Coalition Logic (VCL) formulae is inductively defined as follows: (i) if \(p\) is a propositional symbol in \(\Pi\), then \(p\) and \(\neg p\) are VCL formulae; (ii) if \(\varphi\) is a propositional formula and \(\psi\) is a VCL formula, then \((\varphi \rightarrow \psi)\) is a VCL formula; (iii) if \(\varphi_i\), \(1 \leq i \leq n\), \(n \in \mathbb{N}\), are VCL formulae, then so are \((\varphi_1 \land \ldots \land \varphi_n)\), also written \(\bigwedge_{i=1}^n \varphi_i\); and \((\varphi_1 \lor \ldots \lor \varphi_n)\), also written \(\bigvee_{i=1}^n \varphi_i\); and (iv) if \(\vec{c}\) is a coalition vector and \(\varphi\) is a VCL formula, then so is \(\vec{c}\varphi\). The semantics of \(\text{WFF}^{\text{VCL}}\) formulae is given by Concurrent Game Models extended with choice functions (CGM\(_{\text{CP}}\)) that give meaning to coalition vectors. Intuitively, a coalition vector represents the choices made by each agent. Each number represents a choice function that selects an agent’s move (action) depending on the current world and, possibly, the moves of other agents.

A coalition problem in \(\text{DSNF}^{\text{VCL}}\) is a tuple \((\mathcal{I}, \mathcal{U}, \mathcal{N})\) such that \(\mathcal{I}\) is a set of initial clauses, and \(\mathcal{U}\) is a set of global clauses, are finite sets of propositional clauses \(V_{j=1}^m l_j\), and \(\mathcal{N}\), the set of coalition clauses, consists of VCL formulae of the form \(\bigwedge_{i=1}^m l_i' \rightarrow \vec{c}\bigvee_{j=1}^n l_j\) where \(m, n \geq 0\) and \(l_i', l_j\), for all \(1 \leq i \leq m\), \(1 \leq j \leq n\), are literals such that within every conjunction and every disjunction literals are pairwise different, and \(\vec{c}\) is a coalition vector.

Intuitively, initial clauses are true at one distinguished world and, possibly, the moves of other agents. Let \(\vec{c}_1\) and \(\vec{c}_2\) be two coalition vectors of length \(k\). The coalition vector \(\vec{c}_2\) is an instance of \(\vec{c}_1\) and \(\vec{c}_1\) is more general than \(\vec{c}_2\), written \(\vec{c}_1 \subseteq \vec{c}_2\), if \(\vec{c}_2[i] = \vec{c}_1[i]\) for every \(i\), \(1 \leq i \leq k\), with \(\vec{c}_1[i] \neq \ast\). We say that a coalition vector \(\vec{c}_3\) is a common instance of \(\vec{c}_1\) and \(\vec{c}_2\) if \(\vec{c}_3\) is an instance of both \(\vec{c}_1\) and \(\vec{c}_2\). A coalition vector \(\vec{c}_3\) is a merge of \(\vec{c}_1\) and \(\vec{c}_2\), denoted \(\vec{c}_1 \downarrow \vec{c}_2\), if \(\vec{c}_3\) is a common instance of \(\vec{c}_1\) and \(\vec{c}_2\), and for any common instance \(\vec{c}_4\) of \(\vec{c}_1\) and \(\vec{c}_2\) we have \(\vec{c}_4 \subseteq \vec{c}_3\). If there exists a merge for two coalition vectors \(\vec{c}_1\) and \(\vec{c}_2\) then we say that \(\vec{c}_1\) and \(\vec{c}_2\) are mergeable.

An atom ordering is a well-founded and total ordering \(\succ\) on the set \(\Pi\). The ordering \(\succ\) is extended to literals such that for each \(p \in \Pi\), \(\neg p \succ p\), and for each \(q \in \Pi\) such that \(q \succ p\) then \(q \succ \neg p\) and \(\neg q \succ \neg p\). A literal \(l\) is maximal with respect to a propositional disjunction \(C\) iff for every literal \(l'\) in \(C\), \(l' \not\succ l\).

The ordered resolution calculus \(\text{RES}^{\text{CL}}\) is then given by the rules shown in Figure 1.

**Theorem 1** Let \(\varphi\) be a CL formula. Then there is a coalition problem \(C\) in \(\text{DSNF}^{\text{VCL}}\) that is satisfiable if and only
3 CLProver++

CLProver++ [2] is a C++ implementation of the resolution based calculus RES\textsubscript{CL} described in Section 2. CLProver++ also implements unit propagation, pure literal elimination, forward subsumption and backward subsumption. Clauses in a coalition problem are split into a set Wo of worked-off clauses and set Us of usable clauses. The main loop of the prover heuristically selects a clause G from Us, moves it to Wo and performs all inferences between G and clauses in Wo. The set New of newly derived clauses is subject to forward subsumption and the remaining clauses in New may optionally be used to backward subsume clauses in Us and Wo. Feature vector indexing [6], a non-perfect indexing method, is used to store Us and Wo, and to retrieve a superset of candidates for subsumption or resolution efficiently.

To evaluate the performance of CLProver++ we have compared it with CLProver and TATL (September 2014 version). CLProver [4] is a prototype implementation in SWI-Prolog of the calculus RES\textsubscript{CL}. It also implements forward subsumption but uses no heuristics to guide the search for a refutation. TATL [1] is an implementation in OCaml of the two-phase tableau calculus by Goranko and Shkatov for ATL [3], that can also be used to decide the satisfiability of CL formulae.

We have used two classes \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \) of randomly generated CL formulae for the evaluation that are available from the CLProver++ website [2]. \( \mathcal{B}_1 \) consists of twelve sets \( S_i \), \( 1 \leq i \leq 12 \), of 100 formulae each, with each formula in \( S_i \) having length 100 \( \times \) i. \( \mathcal{B}_2 \) consists of 12 sets \( S_i \), \( 1 \leq i \leq 12 \), of 100 formulae in conjunctive normal form with i conjuncts of the form \( (\neg \neg l_i)^{l_i} \) with elements of each conjunct generated randomly.

Figures 2 and 3 show the total runtime of each of the provers on each of the sets in \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \), respectively. Execution of a prover on a formula was stopped after 1000 CPU seconds. The time to transform a formula into a coalition problem is not included, but is negligible. Overall, CLProver++ outperforms all other systems by a large margin.

References


Development of the Logic Programming Approach to the Intelligent Monitoring of Anomalous Human Behaviour
(Extended Abstract)

Alexei A. Morozov¹,²
Alexander F. Polupanov¹

¹ Kotel’nikov Institute of Radio Engineering and Electronics of RAS, Mokhovaya 11, Moscow, Russia
² Moscow State University of Psychology & Education, Sretenka 29, Moscow, Russia
morozov@cplire.ru, sashap55@mail.ru

Abstract: A research software platform is developed that is based on the Actor Prolog concurrent object-oriented logic language and a state-of-the-art Prolog-to-Java translator for experimenting with the intelligent visual surveillance. We demonstrate an example of the application of the method to the monitoring of anomalous human behaviour that is based on the logical description of complex human behaviour patterns and special kinds of blob motion statistics. The logic language is used for the analysis of graphs of tracks of moving blobs; the graphs are supplied by low-level analysis algorithms implemented in a special built-in class of Actor Prolog. The blob motion statistics is collected by the low-level analysis procedures that are of the need for the discrimination of running people, people riding bicycles, and cars in a video scene. The first-order logic language is used for implementing the fuzzy logical inference based on the blob motion statistics.

Human activity recognition is a rapidly growing research area with important application domains including security and anti-terrorist issues. Recently logic programming was recognized as a promising approach for dynamic visual scenes analysis (see surveys of logic-based recognition systems in [9, 7]). The idea of the logic programming approach is in usage of logical rules for description and analysis of people activities. Knowledge about object co-ordinates and properties, scene geometry, and human body constraints is encoded in the form of certain rules in a logic programming language and is applied to the output of low-level object / feature detectors.

The distinctive feature of our approach to the visual surveillance logic programming is in application of general-purpose concurrent object-oriented logic programming features, but not in the development of a new logical formalism. We use the Actor Prolog object-oriented logic language [2, 4, 5] for implementation of concurrent stages of video processing. A state-of-the-art Prolog-to-Java translator is used for efficient implementation of logical inference on video scenes. Special built-in classes of the Actor Prolog language were developed and implemented for the low-level video storage and processing.

We have created a set of blob motion metrics based on the windowed coefficient of determination of the temporal changes of the length of the contour of the blob. They are supposed to be reliable when images of moving objects are noised and / or fuzzy, that is necessary for real applications (see an example of a video scene to be analyzed in Fig. 1).

Figure 1: An example of BEHAVE [11] video with a case of a street offence: one group attacks another.

The coefficient of determination $R^2$ indicates the proportionate amount of variation in the given response variable $Y$ explained by the independent variables $X$ in a linear regression model. Thus, the $R^2$ metrics is supposed to be useful for discrimination of vehicles and running pedestrians, when the $X$ variable is the time and the $Y$ variable is the area or the length of the contour of the moving blob. In the general case, vehicles will be characterised by bigger values of the $R^2$ metrics than running persons, because the contour of the running person changes permanently in the course of his motion when he waves his arms and moves his legs.

We use a windowed modification of the $R^2$ metrics, that is, the trajectory of a moving blob is characterised by a set of instantaneous values of the $R^2$ metrics computed in each point of the trajectory. Suppose $t_B$ is the beginning time point of the trajectory and $t_E$ is the end time point. Thus, the windowed $R^2$ metrics is a set:

$$wR^2 = \{ R^2_{t,w} \}$$

where $t$ is the time ($t \in \{ t_B + w/2 \ldots t_E - w/2 \}$) and $w$ is the width of the widow (the neighbourhood of $t$) to be used for computation of the $R^2$ value.
We use two statistical metrics that characterise the motion of the blob, namely, the mean of the \( wR^2 \) distribution and the bias-corrected skewness of the \( wR^2 \) distribution. We have implemented these metrics in Java in the Vision standard package of the Actor Prolog language and use them for experimenting with the intelligent visual surveillance.

All results of the low-level analysis are received by the logic program in a form of Prolog terms describing the list of connected graphs. The connected graph of tracks is a list of underdetermined sets [2] denoting separate edges of the graph. The nodes of the graph correspond to the points where tracks cross, and the edges are pieces of tracks between such points.

Here is an example of a logic program output. The logic program checks the graph of tracks and looks for the following pattern of interaction among several persons: “If two or more persons meet somewhere in the scene and one of them runs after the end of the meeting, the program should consider this scenario as a kind of a running away and a probable case of a sudden attack or a theft.” So, the program has to alarm if this kind of sub-graph is detected in the total connected graph of tracks. In this case, the program will mark all persons in the inspected graph by yellow rectangles and outputs the “Attention!” warning in the middle of the screen (see Fig. 2).

Figure 2: The logical inference has found a possible case of a street offence in the graph of blob trajectories. All probable participants of the conflict are marked by yellow rectangles. Multicoloured lines denote tracks of the blobs. The program estimates the velocity of the blobs and depicts it by different colours. Direct blue lines depict possible links between the blobs.

Even a simple video surveillance logic program has to contain a lot of elements, including video information gathering, low-level image analysis, high-level logical inference control, and reporting the results of intelligent visual surveillance; we have to emphasise that the logic programming approach allows one to implement all stages of the video data processing using the single logic language that is a prominent step in the approach to the intellectual visual surveillance.

We have created a research software platform based on the Actor Prolog concurrent object-oriented logic language and an open source Java library of Actor Prolog built-in classes [6]. It is intended to facilitate the study of the intelligent monitoring of anomalous people activities, the logical description and analysis of people behaviour (see Web Site [8]).

We acknowledge a partial financial support from the Russian Foundation for Basic Research, grant No 13-07-92694.

References


Abstract. In this paper we present a concept of an agent-based strategy to allocate services on a Cloud system without overloading nodes and maintaining the system stability with minimum cost. As a research base we specify an abstract model of cloud resources utilization, including multiple types of resources and service migration costs. We also present an early version of simulation environment and a prototype of agent-based load balancer.

Introduction. Modern day applications are often designed in such a way that they can simultaneously use resources from different computer environments. System components are not just properties of individual machines and in many respects they can be viewed as though they are deployed in a single application environment. The sheer physical size of the system itself means that thousands of machines may be involved. In recent years the most advanced technologies offer cloud solutions (e.g.: Amazon Web Services, Google App Engine). Clouds are typically more cost-effective than single computers of comparable speed and usually enable applications to have higher availability than a single machine. A few well-known examples of services backed up by cloud computing are Dropbox, Gmail, Facebook or Youtube.

Software as a service. This elasticity of resources, without paying a premium for a large-scale usage, is unprecedented in the history of IT (Fox et al, 2009). This makes the software even more attractive as a service and is shaping the way software is built today. Companies no longer need to be concerned about maintaining a huge IT infrastructure just so they have enough computing power for those critical hours when their service is in highest demand. Instead companies can simply rent thousands of servers for a few hours. However, it introduces a new set of challenges and problems, which need to be solved. The cloud systems are usually made up of machines with very different hardware configurations and different capabilities. These systems can be rapidly provisioned as per the user's requirements thus resource sharing is a necessity. Resource management has been an active research area for a considerable period of time and those systems usually feature specialized load balancing strategies.

Focus on stability. Currently existing distributed load balancers (e.g.: Maui and Moab (Etison and Tsafrir, 2005), LoadLeveler (Kannan, 2001), Mesos (Hindman et al., 2011)) mainly focus on features like: performance, responsiveness, locality, fairness, etc. However, the primary purpose of commercial Cloud systems is to keep third party operations working continuously and with a minimal disturbance (i.e.: ‘stability’). Therefore, our resources utilization model focuses on maintaining system stability with minimum cost. Our model consists of nodes and services where the load balancer task is to keep a good load balance through resource vector comparisons. We assume in the majority of problem instances, the system already has capacity to provide all required resources to current services, although the system need to detect that existing resources are insufficient. The main challenge is to allocate and re-allocate services properly, so no single node is overloaded.

Our research will focus mainly on providing system stability combined with optimal minimal cost. Other features, such as fairness and performance will be considered to be a secondary objective. In considering what is actually constituted as a ‘service’ in a Cloud environment an example may be seen in a popular Cloud environment such as Amazon’s EC2, where applications are deployed within the full Operating System Virtual Machine (VM). This schema has many benefits such as the almost complete separation of execution contexts (although services might still share the same hardware) and complete control over local system environment parameters.

Resource utilization model. Let us define:

- \( \psi = \{t_1, t_2, \ldots, t_k\} \) as a set of all different kinds of resources.
- \( a : \psi \times \eta \rightarrow \mathbb{N} \cup \{0\} \) as a fixed available resources on the nodes. \( a_i(n) \) is the available level (integer value) of a resource \( i \) on the node \( n \).
- \( r : \psi \times \tau \rightarrow \mathbb{N} \cup \{0\} \) as a fixed required resources for services. \( r_i(t) \) is the required level (integer value) of a resource \( i \) of service \( t \).
- \( c : \tau \rightarrow \mathbb{N} \cup \{0\} \) as a service migration cost function. \( c(t) \) means cost incurred migrating service executables and its state and preparing service environment.

We also specify \( \Lambda = (\tau, \eta, \psi, a, r, c) \) as a problem space and system as a pair \((\Lambda, \mu)\). In the d-resource system optimization problem, we receive a set \( \tau \) of \( f \) mobile services \( \tau = \{t_1, t_2, \ldots, t_k\} \) and a set \( \eta \) of \( m \) fixed nodes \( \eta = \{n_1, n_2, \ldots, n_m\} \). We call \( \mu : \tau \rightarrow \eta \) as a service assignment function, where each service has to be assigned to the node.

For every node \( n \in \eta \) we define a set \( A_n = \{t \in \tau : \mu(t) = n\} \) of all services assigned to the node \( n \). We also define \( f : \psi \times \eta \rightarrow \mathbb{N} \cup \{0\} \) as remaining resources on the nodes: \( f_i(n) = a_i(n) - \sum_{t \in A_n} r_i(t) \).

We consider system \((\Lambda, \mu)\) as stable, if:
5. \( f_i(n) \geq 0, \text{i.e.: } \sum_{n \in \Lambda_i} r_i(n) \leq a_i(n) \), for every \( n \in \eta, i \in \Psi \).

otherwise the system \((\Lambda, \mu)\) is overloaded.

Each service \( t \) is initially assigned by service assignment function \( \mu_0 \) to some node \( \eta \). During the system transformation \((\mu_0 \rightarrow \mu_1)\) service \( t \in \tau \) can be reassigned to any different node \( n \in \eta \). The process of moving the service to a different node is referred to as service migration and this feature generates a service reassigning cost:

\[
c_i(\mu_0 \rightarrow \mu_1)(t) = \begin{cases} 
0, & \mu_0(t) = \mu_1(t) \\
c(t), & \mu_0(t) \neq \mu_1(t)
\end{cases}
\]

Every system transformation process \((\mu_0 \rightarrow \mu_1)\) has its system transformation cost:

\[
c(\mu_0 \rightarrow \mu_1) = \sum_{t \in \tau} c_i(\mu_0 \rightarrow \mu_1)(t)
\]

Consider initial service assignment \( \mu_0 \); service assignment \( \mu^* \) is optimal for \( \mu_0 \), if \( \mu^* \) renders system \((\Lambda, \mu^*)\) stable and:

\[
c(\mu_0 \rightarrow \mu^*) \leq c(\mu_0 \rightarrow \mu_1) \text{ for every stable system } (\Lambda, \mu).
\]

N.b.: when \((\Lambda, \mu_0)\) is stable for initial service assignment \( \mu_0 \), the system transformation cost equals \( 0 \) as it is considered optimal.

**Agent-based load balancer.** In order to correctly assign new services to nodes as well as to provide adequate execution environments to existing ones, we have designed a load-balancing system, where autonomous agents continuously negotiate between themselves the placement of tasks.

Every node is represented by an agent. An agent has several tasks:

- Periodically update central database about currently available and utilized resources on his node
- Accept or reject service placement requests from other agents
- Keep track of all services currently being executed or scheduled to be executed
- Select and evict services, which consume too much resources and find an alternative node and negotiate with its agent evicted service migration

In current implementation decision algorithms used in every of the above tasks are simplistic. The agent reacts only when the level of any of utilized resources exceeds available levels and selects the most resource-consuming service for eviction. Then agent tries to find alternative node for this service and negotiate service migration.

**Conclusions.** In our current prototype, we have based implementation on functional programming language Scala with actors framework from Akka library. Such setup is quite flexible and easy to test on single machine as well as in distributed environment. From early experiments, we have noted that agent-based load balancer system is very unpredictable, difficult to control and might provide a wide spectrum of results. Nevertheless, thanks to distributed nature on agents and computing scheduling decisions on multiple nodes in parallel, we can avoid ‘a-blocking-head-of-queue’ problem, which is prevalent in ‘monolitic’ schedulers (Schwarzkopf et al., 2013).

**References**

Discovering new ontologies and lemmas through concept blending and analogical reasoning

Meenakshi Kesavan
School of Computing, University of Dundee, UK
m.kesavan@dundee.ac.uk

Abstract. Examination and application of ontologies is being tried in many fields. Previously the interest was mainly on the development of ontologies. Now the subject is taking a shift towards how to use ontologies to improve machine intelligence. My aim is to manipulate ontologies to discover new concepts, relationships and rules by making use of the subject called "Concept Blending".

Keywords: Theorem Proving, Machine Learning, Concept Blending, Semantic Web, Reasoning, Logic, Analogy

Introduction

This can proceed in two different ways. First, this idea can be tested with some existing standard ontologies. Second, it can be tested by abstracting the concepts from the domain of interest be developing ontologies from the scratch.

This can also be extended by incorporating statistical machine learning to find and analyse the clusters which can be evaluated by the similarity in ontology concepts. The clusters obtained through the process of ontology mapping can be compared with the clusters obtained through the process of analogy mapping. Also, this can be scaled to other domains of interest. Further extension would be to try this method for lemma discovery in theorems, to create target ontologies from source ontologies by analogy and to automate the ontology development process and evaluating how close it is to manually developed ontology. This topic will draw interdisciplinary research results from Semantic web, Formal methods, Theorem proving, Reasoning and Cognitive science.
References


3. Marco Schorlemmer; Alan Smaill; Kai-Uwe Kühnberger; Oliver Kutz; Simon Colton; Emilios Cambouropoulos; Alison Pease, “COINVENT: Towards a Computational Concept Invention Theory ,” Fifth International Conference on Computational Creativity (ICCC 2014), Ljubljana, Slovenia (2014)

Capturing Temporal Aspects in Ontologies

Jared Leo Uli Sattler Bijan Parsia

The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
{jared.leo,bijan.parsia,ulrike.sattler}@manchester.ac.uk

Abstract: Representing temporal aspects in ontologies is a known problem, and is often required to faithfully represent the knowledge in certain ontologies, particularly those in the life science domain. We surveyed BioPortal, a repository for life-science ontologies, to determine the extent of temporal modelling, before introducing a family of new temporal description logics, motivated by the problem at hand. We evaluate the new logics and current proposals against a set of temporal requirements gathered from the survey.

1 Introduction

Description Logics (DLs) are a collection of knowledge representation languages that can be used to express the knowledge of an application domain in a precise and structured way. The most common use of DLs are to formalise ontologies, which have applications within the semantic web and the life science domain. Although very expressive DLs exist such as SROIQ, which is currently employed as the logical basis for OWL 2, there are still limiting factors over what we can express. Many health care and life science concepts involve temporal aspects, for example, any sort of representation of organism development. It would be beneficial to have some sense of time encoded into the underlying logic, allowing us to represent and query knowledge that can change over time. Focussing on stage-based and developmental ontologies, the temporal information that is usually present is either based on objects changing (developing) over a timeline, or representing sequences of stages. DLs do not have a built in notion of time, and as a step towards solving this problem, proposals for adding temporal aspects to DLs have been considered. Such attempts usually include combining the semantics of a Temporal Logic (TL) with the semantics of a DL, such as LTL_DL and CTL_DL (based on the combinations of the standard TLs LTL and CTL) to form temporal description logics (TDLs). Such combinations are based on sequences of usual DL interpretations, acting as a timeline (see survey [3] for further details) allowing us to view elements at different time points. Other attempts include combinations with other temporal dimensions such as Halpern and Shohams logic of time intervals, or extending (as opposed to combining) the semantics of DLs by adding in concrete temporal data values such as the DL tOWL. Although the expressivity of these logics are extremely high, they have yet to be adopted by current ontologies as a means for a better representation of their knowledge. They all provide different temporal extensions to DLs, but may have to make compromises w.r.t expressivity to ensure decidability; a principle example being that no current TDL can admit rigidity whilst staying decidable.

2 Requirements Analysis

BioPortal [4] is a repository for ontologies in the biomedical domain, with more than 300 ontologies to date. We recently surveyed BioPortal to determine the extent and variation of stage-based modelling. Methodology: The corpus of ontologies we used in the survey was a snapshot of BioPortal (September 2014), created using the NCBO BioPortal v4.0 REST services, containing 331 files that are parseable with the OWL API. We filtered the dataset by identifying those ontologies that contained any keywords related to stage-based development. We used all of the OBO-Relation terms from the OBO Foundry that contain time information in their formal definition, as well as other commonly used developmental phrases. Results: 198/331 contained at least one of the keywords in at least 1 entity or annotation. Each ontology out of the 198 was then manually reviewed to identify those that had some type of developmental pattern encoded. We found that a significant amount (24%) of BioPortal contains ontologies that could benefit from a suitable TDL. From the data gathered, we produced a set of requirements of temporal representations that a TDL would need to meet in order to faithfully represent these ontologies; 1. Discrete Timeline: have an explicit representation of a linear sequential timeline based on a set a discrete time points. 2. Property Change: be able to make statements about an element’s properties changing over time. 3. Rigidity: have a controlled form of rigidity over the timeline to represent relationships between individuals holding over a specified sequence of time points. 4. Time point referencing: be able to relate concepts and roles to specific time points along the timeline. 5. Sequence Repetition: be able to model a developmental pattern that can occur at multiple stages. 6. Uncertainty: be able to model a level of uncertainty against the timeline when specific bounds are given. 7. Past, Present & Future: refer to an objects past, present and future properties.

3 Background & Contribution:

There have been many methods proposed of adding temporal aspects to DLs. These include temporal query answering, temporal data access, time stamping, incorporating Allen’s interval relations through means of a concrete domain, using reification and combining standard TLs such as LTL or CTL [3] with DLs. As expressive and powerful as these logics are, we believe LTL_DL combinations are most suited to the problem at hand, since they focus directly on
representing time at a concept or axiom level, and provide a suitable dimension of time to model change effectively. We focus our attention on evaluating $LTL_{EC}$, against $EL_1$ and $EL_{[x]}$, both of which were introduced by Leo et al, as an attempt to tackle the problem at hand. $LTL_{EC}$ and $EL_{[x]}$ share a similar semantics, both are infinite sequences of normal DL interpretations which can be mapped to either $\mathbb{Z}$ or $\mathbb{N}$. The semantics of $EL_1$ is also based on sequences of DL interpretations, however the sequences are finite, which are mapped to a sequential subset of $\mathbb{Z}$. To access the temporal dimension, $LTL_{EC}$ allows standard $EL$ concept descriptions to be labelled with the temporal operators from $LTL$ \((\bigcirc, \bigodot, \square, U, W)\), whereas $EL_1$ and $EL_{[x]}$ allow $EL$ concept and role names to be labelled with time point intervals of the form \([i, j]\) or \([x + i, x + j]\) resp. where \(i, j \in \mathbb{Z}, i \leq j\) and \(x\) is a variable. To demonstrate the expressiveness of each logic consider the following formulæ:

\[
A \sqsubseteq \square (\exists R. \bigcirc B) \sqcap \square C \\
A_{[0,0]} \sqsubseteq \exists R_{[1,1]}. B_{[1,1]} \\
A_{[x,x]} \sqsubseteq \exists R_{[x+1,x+1]}. B_{[x+1,x+2]}
\]

The interpretation for each axiom is as follows: (1) For all time points \(i\), all instances of \(A\) at time \(i\), have an \(R\) successor at time \(i + 1\) to an instance that is a \(B\) at some time point \(> i + 1\), and is an instance of \(C\) for every time point \(> i + 1\). (2) Every instance of \(A\) at time \(0\) has an \(R\) successor at time \(1\), to an instance of \(B\) at time \(1\). (3) For all time points \(i\), all instances of \(A\) at time \(i\), have an \(R\) successor at time \(i + 1\) to an instance that is a \(B\) at times \(i + 1\) and \(i + 2\). Table 1 shows the TRs met by each logic.

The main reasoning problem in $EL$ is computing subsumption between all atomic concepts occurring in an $EL$ TBox (classification). Computing classification in $EL$ is $PTIME$ [1], whereas classification in $LTL_{EC}$ with rigid roles (one of the most desired features) is undecidable [3]. In contrast, the same reasoning problem in $EL_1$ remains in $PTIME$ [2]. For $EL_{[x]}$, we have provided a decision procedure for subsumption w.r.t TBoxes referencing a unidirectional timeline, i.e. representing information in either future time or past time only. $EL_{[x]}$ with a unidirectional timeline has a loose upper complexity bound of $EXPSPACE$. This is a positive result since we encode a localised version of rigidity with temporal roles, and remain decidable.

### 4 Outlook

We will continue to extend the expressivity of $EL_{[x]}$ in both the temporal domain and the DL domain. We plan to show complexity results in the general TBox case (bidirectional temporal referencing). We hope to extend $EL_1$ to $ALC_1$, and extend $EL_{[x]}$ to $ALC_{[x]}$. There are also similarities between $EL_{[x]}$ and $LTL_{EC}$ with regards to temporal operators and variable intervals. We know we can encode the $O$ operator in $LE_{[x]}$ quite easily. For example, the $LTL_{EC}$ axiom $A \sqsubseteq \bigcirc \bigcirc B$ can be encoded as $A_{[x,x]} \sqsubseteq B_{[x+2,x+2]}$. It is easy to see that in this trivial case the 2 axioms are semantically equivalent if we adopt the same timeline in both structures. Other similarities should be further investigated.

<table>
<thead>
<tr>
<th>$L/\text{TR}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SROIQ$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$LTL_{EC}$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$LTL_{EC}(RR)$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$EL_1$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$EL_{[x]}$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

Table 1: Evaluation of Temporal Requirements against current TDLs (RR : rigid roles, $\rightarrow$ : unidirectional)

We are currently extending our requirements by further surveying BioPortal for more temporal information and plan to provide metrics on exactly how many ontologies would benefit from each requirement. These metrics include ontologies that use either unidirectional or bidirectional temporal referencing, those that require a timeline isomorphic to $\mathbb{Z}$ or $\mathbb{N}$ and those that need global or localised rigidity, amongst other crucial metrics. We are also in the process of communicating with the authors of the ontologies, to see exactly what they would like to model, provided they had a suitable representation, to further evaluate and compare our contributions and current proposals, in search for a suitable TDL. Once this is done, we intend to recode an ontology using a suitable TDL to hopefully show a faithful representation with additional temporal entailments, when compared to the original static encoding.

We also have an implementation for classification of $EL_1$ and $EL_{[x]}$ ontologies, compatible with the OWL API. We are currently looking into the computational feasibility in practice, which we can further evaluate once we have an ontology encoded into the language.

### References


Using Herbrand Bases for Building a Smart Semantic Tree Theorem Prover

Nourah Shenaiber  Manfred Kerber  Volker Sorge

School of Computer Science, University of Birmingham, UK,
N.A.Shenaiber@cs.bham.ac.uk
M.Kerber@cs.bham.ac.uk
V.Sorge@cs.bham.ac.uk

1 Introduction

Traditionally, semantic trees have played an important role in proof theory for validating the unsatisfiability of sets of clauses. More recently, they have also been used to implement more practical tools for verifying the unsatisfiability of clause sets in first-order predicate logic. The method ultimately relies on the Herbrand Base, a set used in building the semantic tree. The Herbrand Base is used together with the Herbrand Universe, which stems from the initial clause set in a particular theorem. When searching for a closed semantic tree, the selection of suitable atoms from the Herbrand Base is very important and should be carried out carefully by educated guesses in order to avoid building a tree using atoms which are irrelevant for the proof. In an effort to circumvent the creation of irrelevant ground instances, a novel approach is investigated in this dissertation. As opposed to creating the ground instances of the clauses in \( S \) in a strict syntactic order, the values will be established through calculations which are based on relevance for the problem at hand. This idea has been applied and accordingly tested with the use of the Smart Semantic Tree Theorem Prover (SSTTP), which provides an algorithm for choosing prominent atoms from the Herbrand Base for utilisation in the generation of closed semantic trees. Part of our study is an empirical investigation of this prover performance on first-order problems without equality, as well as whether or not it is able to compete with modern theorem provers in certain niches. The results of the SSTTP are promising in terms of finding proofs in less time than some of the state-of-the-art provers. However, it can not compete with them in terms of the total number of the solved problems.

2 SSTTP Calculus

The SSTTP calculus starts with a clause set \( \Phi \), a candidate set and a Smart Herbrand Base set \( SHB \). The intention of the letter is generate a set of ground atoms that will be useful in building a closed semantic tree.

Initially the candidate set is \( \Phi \) (same as the clause set) and the \( SHB \) is empty. So, the input is of the form \([\Phi, \Phi, \emptyset]\) and, after the derivation rules, we will garner proof of the clause set if all the leaves of the tree are closed; otherwise, the algorithm continues until it runs out of resources (time or space) or no rule is applicable. If there is no proof rule applicable then the \( SHB \) becomes an actual model of the input clause set.

The calculus is refutationally sound and complete: \( \Phi \) is a theorem if and only if from \([\Phi, \Phi, \emptyset]\) we can be built a tree using the SSTTP calculus rules so that each leaf is closed (that is, of the form \([\Box, *, *]\)).

In the rules, we represent a clause set \( \Phi \) as \((-l|C, l'|C', \Phi')\), Where \(-l|C \) and \( l'|C'\) are two clauses from \( \Phi \) and \( \Phi' \) is the rest of clauses from \( \Phi \). Moreover, we denote by \( l|C \) a clause \( D \), such that \( l \) is a literal of \( D \) whilst \( C \) is the clause obtained by removing \( l \) from \( D \). And \( s \) is an atom of the \( SHB \) set.

The SSTTP calculus consists of three rules: \( SSoS \), Build tree and Close. The rules are schematically depicted in the table on top of the next page. All the three rules take three input sets and give three output sets. The first two sets are clause sets and the third one is a set of atoms.

The \( SSoS \) rule describes the generating procedure of the Smart Herbrand Base (\( SHB \)) for a given set of clauses. In this rule, we have three inputs: \( \Phi \) which is the clause set, \(-l|C, l'|C', \Phi'\) is the candidate clause set that will be used to generate the Smart Herbrand Base and \( SHB \) is the set of the Smart Herbrand Base atoms. Initially, the first and the second inputs are the same (the original clause set) and the \( SHB \) is empty. Through this rule, we will use only the second clause set and keep the first clause set invariant. Then, check the applicability of the resolution rule in the second clause set. If the resolution rule is applicable then we delete the literals \( l \) and \( l' \) from their clauses \( C \) and \( C' \). After that, the algorithm positions \( C \) and \( C' \) together with \( \Phi \) in one set and inserts \( l \) in the \( SHB \) after applying \( \sigma \) to it.

The Build tree rule describes the derivation process of the semantic tree theorem prover for a given set of clauses by splitting along the two literals \( l \) and \( l' \) into two different clauses. Then, resolve \(-l|C \) with \([s]|\sigma \) to get \([C]|\sigma \) in the left branch and resolve \( l'|C' \) with \([s]|\sigma \) to get \([C']|\sigma' \) in the right branch.

Close rule describes the close procedure of the semantic tree theorem prover. If \( \Phi \) contains the empty clause \( \Box \), then this node is a leaf node and it is closed.
3 Heuristics

To improve efficiency of our proof procedure we have experimented with four different heuristic strategies that were partially inspired by existing heuristics in other theorem provers, mainly based on the resolution principle.

3.1 SSTTP-h1: Unit heuristic

The unit heuristic strategy used in the SSTTP prover is similar to the unit-preference strategy used in DPLL and resolution to reduce the size of the literals. The unit heuristic strategy collects all unit clauses from the base set of the given input problem, and incorporates them within the SHB before generating atoms using the SSTTP calculus. This will help the SSTTP prover to close at least one node immediately after each level of the semantic tree. Hence, the complexity of the semantic tree will be kept minimal. Therefore, this strategy looks promising for the efficiency of the SSTTP prover in terms of the size of the semantic tree and of the processing time.

3.2 SSTTP-h2: Degree order heuristic

In order to reduce the search spaces in first-order automated deduction, sort strategies are useful with a view to the re-arrangement of clauses. The degree order heuristic use a selection sort technique in order to sort the clauses inside the semantic tree nodes by their degree, that is defined as a number of literals inside a given clause. SSTTP-h2 applies a sort function that orders the clauses inside each node of the semantic tree by ascending order. That is, it first lists all clauses with degree 1, e.g. all unit clauses, then of degree 2 and so on. The heuristic assists the SSTTP prover to allocate atoms within the SHB that reduce the size of the clauses first. Accordingly, if the atom that is used to split the tree is generated from a unit clause, then at least one of the branches within the semantic tree will closed in one level as the unit heuristic is carried out. Moreover, if the atom that is used to split the tree is generated from a clause of degree 2, then the clause resulting from the split is a unit clause. This technique reduces the size of the semantic tree in each level, thereby hopefully contributing to the efficiency of the SSTTP prover.

3.3 SSTTP-h3: Impact heuristic

The impact heuristic strategy estimates how useful an SHB atom is in producing a closed semantic tree by calculating an impact number for each atom inside the SHB. The impact number of an atom in the SHB is calculated as the number of literals inside the clause set matching the atom irrespective of variable names. The heuristic chooses an atom with maximum impact number for the next split of the semantic tree, which ideally generates a node that will have more new clauses that give more choice to close the semantic tree in less time and space.

3.4 SSTTP-h4: Meta duplicate elimination heuristic

This heuristic deals with the meta variables that occur within the SHB in order to eliminate redundancy. It minimizes the number of proof steps required by removing the duplicated atoms, i.e., those only differing in meta variables, from the SHB. When the SSTTP algorithm generates the SHB atoms that contain variables, it replaces these variables by meta variables assigned to an empty cell until they are grounded during the building of the semantic tree (that is, we postpone the instantiation of variables until information is available with what they should be instantiated). Therefore, when in the SHB there is a duplication of these atoms, it is redundant to use them before they are ground.

4 Results

We have experimented with our implementation of the SSTTP calculus on all eligible theorems in the TPTP. The following table summarises the results with respect to the pure calculus and the different heuristic strategies.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Input</th>
<th>SSTTP</th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>1</td>
<td>65 k</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AB</td>
<td>2</td>
<td>20 k</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CAM</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COL</td>
<td>4</td>
<td>19</td>
<td>16</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>COM</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>FLD</td>
<td>6</td>
<td>24</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>GED</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GRP</td>
<td>8</td>
<td>42</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>KRS</td>
<td>9</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LAT</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>LCL</td>
<td>11</td>
<td>157</td>
<td>12</td>
<td>11</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>MG1</td>
<td>12</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>MS</td>
<td>13</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NUM</td>
<td>14</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PLA</td>
<td>15</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PUZ</td>
<td>16</td>
<td>40</td>
<td>9</td>
<td>11</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>386</td>
<td>52</td>
<td>56</td>
<td>53</td>
<td>57</td>
<td></td>
</tr>
</tbody>
</table>

References

Semantic Understanding of Mathematical Formulae in Documents

Randa Almomen  
School of Computer Science,  
University of Birmingham, UK,  
R.Almomen@cs.bham.ac.uk  
V.Sorge@cs.bham.ac.uk

1 Introduction

The correct semantic interpretation of mathematical formulae that recognised in documents is very important in several areas such as improving the precision of systems that translate documents indirectly into speech or to other formats. Also its importance increase to improve the accessibility of maths documents and improve precision of existing maths search systems.

For instance, the formula

\[ f(a + b) \]  

(1)

could be understood as if \( f, a \) and \( b \) are variables and therefore

\[ f(a + b) = f \times a + f \times b \]  

(2)

Another possibility is that \( f \) is a function to be applied to the variables \( a \) and \( b. \) Indeed both semantic meanings are perfectly legal, but it depends on the context or the document the maths formula has been extracted from. Also, if we have the formula

\[ H \leq G \]  

(3)

and we know that the context of this formula is the domain of Group Theory, then we can say that \( H \) is a subgroup of a group \( G \) but not \( H \) is less than or equal to \( G. \)

In this abstract we present our attempt to close the gap between what an expert mathematician can interpret and what a machine can interpret. Therefore, we are developing an approach to determine the semantics of mathematical formulae by analysing both the mathematical formulae and their context. This general goal can be divided into several goals as to start with the extraction of basic semantic information such as maths entity type from the representation of maths formula in a mark-up language. Secondly, the analyses of maths context to interpret each maths entity within that context. and finally, to find a way to combine the results of the previous two goals and represent the extracted semantic information.

2 Overview

In order to develop an approach that determines the semantic of mathematical formulae, the analysis is carried out on both the mathematical formulae and their context in several steps. To ease the start we put some restrictions that can be released afterwards. These restrictions are:

- To start working on a particular maths domain (Elementary Number Theory) and then we can extend the work to other domains.
- To start with single component of maths formulae and later may move to compounds component of formulae.

We started from the source code (the \( \LaTeX \) format) of a corpus of mathematical documents in the domain of Elementary Number Theory. The documents are collected randomly from the e-prints arXiv [1]. The selected maths documents then processed following the steps:

1. Extracting basic semantic information from the representation of maths formula.

   This has been done by the following steps:
   - converting the source code of the documents (which are in \( \LaTeX \) format) into xml files using \( \LaTeX \) XML [3].
   - As the basic semantic information for each maths symbol in the xml files stored in the attributes of that symbol, we extracted each maths symbol with its attributes.

2. Analysing maths context to interpret each maths entity within that context.

   This step consists of three main steps:
   - Annotate the documents manually with the correct interpretation of maths entity. This will be done by two expert mathematicians in addition to me.
   - determine the target formulae which are the single component maths formulae.
   - Analyse the context automatically to interpret each target maths entity. This will be done by; firstly, abstracting over the maths expressions to get it through the parsers. Secondly, process the documents using the Natural Language Toolkit (NLTK) [2] to find different segmentation of the context. Thirdly, determine the sentences which contain embedded single component maths expressions in particular type structures; i.e the sentences which carry the type information in a form of an explicit declaration. Finally, analyse the determined sentences to find the correct interpretation of the target maths entity.
Table 1: An example of explicit declaration of a maths expression

<table>
<thead>
<tr>
<th>In LaTeX file</th>
<th>Let $n$ be an odd prime.</th>
</tr>
</thead>
<tbody>
<tr>
<td>In XML file</td>
<td>Let $&lt;\text{Math mode}=&quot;\text{inline}&quot; \text{id}=&quot;S3.Thmdefn4.pl.ml&quot; \text{tex}=&quot;n&quot; \text{text}=&quot;n&quot;&gt;\text{XMath}\text{XMTok \text{role}=&quot;UNKNOWN&quot;&gt;n&lt;/XMTok&gt;\text{XMath}&gt;\text{/Math}&gt;\text{be an odd prime.}</td>
</tr>
<tr>
<td>text from nltk</td>
<td>Let ‘math.111’ be an odd prime.</td>
</tr>
<tr>
<td>In annotated XML file</td>
<td>$&lt;\text{Math mode}=&quot;\text{inline}&quot; \text{id}=&quot;S3.Thmdefn4.pl.ml&quot; \text{tex}=&quot;n&quot; \text{text}=&quot;n&quot;&gt;\text{XMath}\text{XMTok \text{role}=&quot;odd prime&quot;&gt;n&lt;/XMTok&gt;\text{XMath}&gt;\text{/Math}&gt;$</td>
</tr>
</tbody>
</table>

Table 1 shows an example of a maths expression which is declared explicitly in the documents and how its corresponding LaTeXML format is annotated.

3. Analysing the use of maths entities semantically in a particular mathematical domain.

4. Developing a framework to represent the interpretation of maths formulae in a way that can be used by other existing systems such as maths searching systems.

The Evaluation: To be able to evaluate the result of our work, it has been compared with a ground truth of number of representative documents which was built previously.

References

