Book Title
Book Editors
IOS Press, 2003

Argue tuProlog: A Lightweight Argumentation Engine for Agent Applications [1]

Daniel Bryant [a,2], Paul J. Krause [a] and Gerard A. W. Vreeswijk [b]
[a] *Department of Computing, University of Surrey, GU2 7XH, UK*
[b] *Department of Information and Computing Sciences, Universiteit Utrecht, 3508 TA Utrecht, The Netherlands*

Abstract. Argumentation is becoming increasingly important in the design and implementation of autonomous software agents. We believe that agents engaged in decision-making and reasoning should have access to a general purpose argumentation engine that can be configured to conform to one of a range of semantics. In this paper we discuss our current work on a prototype light-weight Java-based argumentation engine that can be used to implement a non-monotonic reasoning component in Internet or agent-based applications.

Keywords. Argumentation Engine, Reasoning in Agents, Argument games.

1. Introduction

Agents are often cited as a key enabling technology for the next-generation of online services, such as large-scale electronic commerce [1] and Service-Oriented Computing [2]. In order to be effective agents will often need to reason about what is to be done, i.e. perform practical reasoning [3], but in these situations, consisting of large-scale open multi-agent systems, classical logic-based approaches to reasoning and decision-making are often unsuitable [4]. Accordingly, agents may benefit from the use of argumentation, a process based on the exchange and valuation of interacting arguments, to support the process of practical reasoning.

In this paper we discuss our current work on a light-weight Java-based argumentation engine that can be used to implement a non-monotonic reasoning component in Internet or agent-based applications. The core engine has been built using tuProlog [5,6], an existing open-source Prolog engine, as its foundation, which followed the same design principles that we require for our intended domain of application. Although our ultimate goal is to create a general purpose argumentation engine that can be configured to conform to one of a range of semantics, the current version of the engine implements

[1]This work was partially supported by the EU IST/STReP ASPIC project, Grant 002307, and an EPSRC PhD Studentship.

[2]Correspondence to: Daniel Bryant, Department of Computing, University of Surrey, Guildford, GU2 7XH. UK. Tel: 0044 (0) 1483 682263; E-mail: d.bryant@surrey.ac.uk

the argumentation-based framework presented in [4] (allowing our engine to generate arguments and counter arguments over an inconsistent knowledge base, determine the acceptability of arguments and construct proofs using an argument game approach to constructing proofs of acceptance [7]), and also standard PROLOG inference (allowing us to prototype a variety of metainterpretters that support other forms of argumentation.)

The motivation behind this paper is primarily to illustrate that a practical Internet-ready/agent-based implementation of argumentation is now viable. Our implementation, which we named "Argue tuProlog" (AtuP), will be made available later this year in SourceForge [8] under an Open Source licence. We have quite deliberately focused on this as an empirical application of a theoretical model of argumentation, and do not address theoretical issues directly (although we will return to some of the outstanding issues in the concluding section). This paper is structured as follows: In Section 2 we provide motivation for our work and also introduce tuProlog, the foundation of our engine. Section 3 introduces the ASPIC argumentation framework and in section 4 we discuss how we have implemented this in our engine. We conclude the paper with an overview of the planned future work. A fundamental message of this paper, and we will return to this in our final words, is that we take seriously the need for sound empirical evidence for the applicability of argumentation.

2. The Motivation for Argue tuProlog

There has been much recent work on argumentation-based engines, notably Vreeswijk's *IACAS* [9], Rock and colleagues *Deimos* [10] and García and Simari's *DeLP* [11] (and later an extension to this work, *P-DeLP*, by Chesñevar and colleagues [12]). However, none of these engines implement support for more than one form of argumentation semantics. Good practical reasoning is complex with respect to the argument schemes it can use and only in limited and well-defined domains of decision-making does it make sense to use a single scheme of practical reasoning [3]. Accordingly, it is our belief that agents engaged in reasoning should have access to a general purpose argumentation engine that can be configured to conform to one of a range of semantics.

Our prototype argumentation engine has been built using tuProlog [6] as its foundation. tuProlog is a Java-based Prolog engine which has been designed from the ground up as a thin and light-weight engine that is easily deployable, dynamically configurable and easily integrated into Internet or agent applications [5]. There are a number of advantages to using tuProlog as a foundation for our engine. Firstly, the development of tuProlog itself followed the same design principles that we require for our intended domain of application. Secondly, we are building on top of a mature code-base so that much of the functionality that is common to both argumentation and Prolog-type inference can be relied on with a high-degree of confidence. Thirdly, this ensures that in the absence of defeasible rules, our engine defaults to standard Prolog inference.

Utilising the Prolog inference provided by the tuProlog engine we can implement a series of metainterpretters for a variety of forms of argumentation. However, this way of implementing an argumentation engine has both a serious performance overhead and a less than ideal interface. In order to avoid these problems and produce an argumentation engine that fully conforms to the spirit of a light-weight Internet enabled tool, we are re-engineering tuProlog by implementing a series of core argumentation algorithms in

Java, effectively pushing the functionality of the algorithms down into the core engine. The first algorithm we have implemented in our engine is presented in [4].

3. The Acceptability of Arguments

In [4] a framework for argument games is presented that is concerned with establishing the acceptability of arguments. Argument games between two players, a proponent (PRO) and opponent (OPP), can be interpreted as constructing proofs of acceptance utilising a dialectical structure [7]. The proponent and opponent share the same (possibly inconsistent) knowledge base and the proponent starts with a a main claim to be "proved". The proponent attempts to build an admissible set to support the claim and endeavors to defend any argument against any attack coming from the opponent. The proponent wins the game (proving acceptability of the claim) if all the attacking arguments have been defeated, and the opponent wins if they can find an attacking argument that cannot be defeated. In [4] a prototype web-based implementation (coded in RUBY) of the framework algorithms, entitled "Argumentation System" (AS), is also presented.

4. The Implementation of our Engine

4.1. Overview

AtuP is currently implemented in Java and presented as a self-contained component that can be integrated into a range of applications by utilising the well defined application programming interface (API) provided. The API exposes key methods to allow an agent or Internet application developer to access and manipulate the knowledge base (including the ability to define numerical values indicating the degree of belief of each proposition), to construct rules, specify and execute queries (establishing whether a claim can be supported using the knowledge base) and analyse results (determining the support for a claim and the acceptability of arguments).

4.2. Language

As with the original AS, AtuP accepts formulae in an extended first-order language and returns answers on the basis of the semantics of credulously preferred sets (as defined in [4]). The language of AtuP is constituted of atoms, terms and rules (see Section 2.7.3 in [4] for further details) and can be considered as a conservative extension of the basic language of Prolog, enriched with numbers that quantify degree of belief. As AtuP is built on top of an existing Prolog engine, the engine naturally accepts Prolog programs.

In AtuP the numerical input values in (0, 1] represent the degree of belief (DOB), or the credibility, of a proposition [4]. As stated in [4], the DOB is currently provided to allow experimentation with different methods of argument evaluation and is not intended to express probabilities or represent values from other numerical theories to reason with uncertain or incomplete information. However, in our earlier work [13] we explored the integration of argumentation with a number of numerical calculi, such as the semi-qualitative/ordinal possibilities, and "probability of provability" for a fully numeric scale. In future work we plan to enhance arguments with possibilities as discussed in,

for example, Amgoud [14] or Chesñevar [12]. This provides us with a computationally efficient model with a well-founded semantics. We will then progress to explore the integration of a numerical calculus that has a sound probabilistic semantics.

Within AtuP there are two types of different rules, namely those with an empty antecedent (called beliefs) and those with a non-empty antecedent (called rules). Every expression of the form t b. is a rule where t is a term and b indicates a degree of belief. Examples of beliefs include `a 0.8.` and `flies(sylvester) 0.1`. Every expression of the form t :- $t_1, ..., t_n$ b. is also a rule, provided t, t_i are terms, $n > 0$, and b denotes the DOB. Examples of rules include `flies(X) :- bird(X) 0.8.` and `a :- c,d 1.0.` A query is an expression of the form ?- $t_1, ...t_n$. where $n > 0$. It is possible to include more queries in the input, but since we are usually only interested in one goal proposition, this is not typical.

4.3. Algorithms

If ?- t. is a query then AtuP's main goal is to try and find an argument with conclusion t and then try to construct an admissible set (using the algorithm presented in [4]) around that argument. In AtuP every search for arguments for a particular query is encapsulated within another internal instance of an engine. Using multiple internal instances of an engine allows us to keep track of which participant (PRO or OPP) is conducting the current query and also to pause the "dialogue" at any time for further analysis. Once the first argument, say A, is found, the first engine is suspended and A is returned to AtuP. AtuP then tries to find an attacker of A. Thus for every sub-conclusion s of A, a separate engine is instantiated to search for arguments against s. If one of these remains undefeated (which is defined within [4]), then A is defeated, else A remains undefeated.

4.4. Getting Results

When AtuP has finished determining the support for a claim and the acceptability of associated arguments the engine generates a trace of the argument game dialogue (shown in the window on the right of Figure 1). In addition to providing an API to allow agent developers to utilise our engine we have also modified the existing tuProlog graphical user interface to facilitate off-line experimentation with the engine (as shown in Figure 1). We have also developed the core engine using Sun Microsystem's NetBeans integrated development environment in which we have installed the latest version of NetBeans Profiler [15], a fully functional application profiling tool. This allows us to simulate deployment of our engine within a variety of realistic scenarios, and to monitor and analyse such data as CPU usage, memory usage, program loop/branch counting, thread profiling and other basic Java Virtual Machine (JVM) behaviour. We are currently in the process of setting up several large-scale knowledge bases, and when this is complete the profiling tool will facilitate our ultimate goal of obtaining empirical evaluations of the performance of a range of argumentation models.

5. What did argumentation ever do for us?

The above question is easier to answer of the Romans than it is of argumentation. Our current work is a first step in trying to set up some real-world experiments that will help

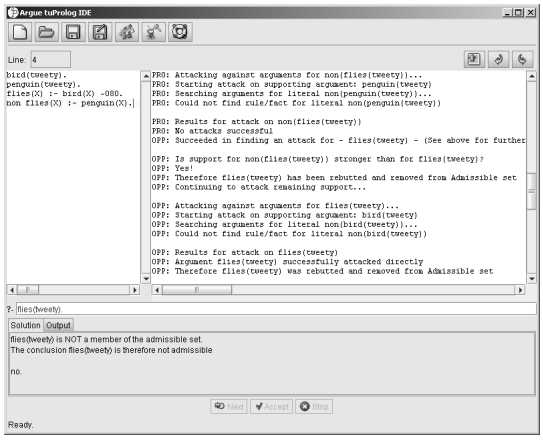

Figure 1. Screenshot of "Argue tuProlog" GUI. The left window allows manipulation of the knowledge base, the bottom window allows query entry and displays the results and the right window shows an argument game trace after a query has been executed.

us answer our question. In this paper we have presented a prototype light-weight Java-based argumentation engine which is capable of facilitating automated reasoning and decision-making, and is suitable for deployment into Internet and agent applications. We have also discussed the integration of an argumentation-based framework for determining the acceptability of arguments, as presented in [4], into our engine.

Our immediate next step is to set up some large-scale knowledge bases that will enable us to obtain empirical evaluations of the performance of the engine. This work is in hand now, and we expect to be able to report the results within the next three months. At that point we will feel confident to release the engine into the wider community, together with a clear definition of its scope and limitations. As well as gaining empirical data on the applicability and performance of a specific instance of an argumentation engine, we will also be evaluating a series of enhancements. As far as possible we are aiming towards implementing a general purpose argumentation engine that can be configured to conform to one of a range of semantics. Our basic position is that we have no prior disposition towards any one model of argumentation. Instead, our plan is to explore a range of models to provide an independent evaluation of their expressive power, performance and scalability.

Argumentation is inherently computationally challenging. As a reminder of just one point, the consistency of a set of first-order logical formulae is undecidable. Yet, most or all definitions of an argument refer to the selection of a "consistent subset" of formulae. We need to investigate theoretical approaches to easing this blocking issue for implementations (e.g. [16]), and indeed we have a parallel strand of research that is targeting this. However, we also feel that it is important to make publicly available our and other "pragmatic" implementations of argumentation, together with some large-scale benchmarking knowledge bases. This way we can also stimulate scientific evaluations of what is practically possible now, and really find out what argumentation can do for us.

Acknowledgements

We would like to gratefully thank Mariam Tariq for use of her earlier implementation work and also the tuProlog team at the University of Bologna for their enthusiastic support of our work. We also offer our thanks to the partners from the ASPIC project for useful discussions. Finally, we would like to greatly thank the anonymous reviewers for their insightful and very helpful comments.

References

[1] C. Guilfoyle, J. Jeffcoate, and H. Stark. *Agents on the Web: Catalyst for E-Commerce*. Ovun Ltd. London, 1997.

[2] M. P. Papazoglou. Service-Oriented Computing: Concepts, characteristics and directions. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 3, Washington, DC, USA, 2003. IEEE Computer Society.

[3] R. Girle, D. Hitchcock, P. McBurney and B. Verheij. Decision support for practical reasoning. In C. C. Reed and T. J. Norman, editors, *Argumentation Machines*, pages 56–83. Kluwer Academic Publishers, The Netherlands, 2004.

[4] L. Amgoud, M. Caminada, S. Doutre, H. Prakken, and G. Vreeswijk. Draft formal semantics for ASPIC system. Technical Report ASPIC Deliverable 2.5, 2005.

[5] E. Denti, A. Omicini, and A. Ricci. Multi-paradigm java-prolog integration in tuProlog. *Sci. Comput. Program.*, 57(2):217–250, 2005.

[6] E. Denti, A. Omicini, and A. Ricci. tuProlog: A light-weight prolog for internet applications and infrastructures. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2001.

[7] H. Jakobovits and D. Vermeir. Dialectic semantics for argumentation frameworks. In *International Conference on Artificial Intelligence and Law*, pages 53–62, 1999.

[8] SourceForge.net. *Welcome to SourceForge.net*. available from http://sourceforge.net/, 2006. Last accessed: 10 May 2006.

[9] G. A. W. Vreeswijk. IACAS: An implementation of Chisholm's principles of knowledge. In *The proceedings of the 2nd Dutch/German Workshop on Nonmonotonic Reasoning, Utrecht.*, pages 225–234, 1995.

[10] M. J. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller. Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(4):483–501, 2001.

[11] A. J. Garcia and G. R. Simari. Defeasible logic programming: an argumentative approach. *Theory Pract. Log. Program.*, 4(2):95–138, 2004.

[12] C. I. Chesnevar, G. R. Simari, T. Alsinet, and L. Godo. A logic programming framework for possibilistic argumentation with vague knowledge. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 76–84, Arlington, Virginia, United States, 2004. AUAI Press.

[13] P. Krause, S. Ambler, M. Elvang-Goransson, and J. Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11:113–131, 1995.

[14] L. Amgoud and H. Prade. Using arguments for making decisions: a possibilistic logic approach. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 10–17, Arlington, Virginia, United States, 2004. AUAI Press.

[15] Sun Microsystems. *NetBeans Profiler*. available from http://profiler.netbeans.org/, 2006. Last accessed: 10 May 2006.

[16] A. Hunter. Approximate arguments for efficiency in logical argumentation. In *Proceedings of NMR-06*, 2006. *In press*.