AceRules: Executing Rules in Controlled Natural Language

Tobias Kuhn

Department of Informatics, University of Zurich, Switzerland
tkuhn@ifi.uzh.ch
http://www.ifi.uzh.ch/cl/tkuhn

Abstract. Expressing rules in controlled natural language can bring us closer to the vision of the Semantic Web since rules can be written in the notation of the application domain and are understandable by anybody. AceRules is a prototype of a rule system with a multi-semantics architecture. It demonstrates the formal representation of rules using the controlled natural language ACE. We show that a rule language can be executable and easily understandable at the same time. AceRules is available via a web service and two web interfaces.

1 Introduction

The idea of the *Semantic Web* [4] is to transform and extend the current World Wide Web into a system that can also be understood by machines, at least to some degree. Ideally, the Semantic Web should become as pervasive as the traditional World Wide Web today. This means that a large part of the society should be able to participate and contribute, and thus we have to deal with the problem that most people feel uncomfortable with technical notations. Controlled natural languages seem to be a promising approach to overcome this problem by giving intuitive and natural representations for ontologies and rules. We present *AceRules* as a prototype of a front-end rule system to be used in the context of the Semantic Web or other environments.

The goal of AceRules is to show that controlled natural languages can be used to represent and execute formal rule systems. *Attempto Controlled English* (ACE) [9, 8] is used as input and output language. ACE looks like English but avoids the ambiguities of natural language by restricting the syntax [15] and by defining a small set of interpretation rules [1]. The ACE parser[1] translates ACE texts automatically into *Discourse Representation Structures* (DRS) [7] which are a syntactical variant of first-order logic. Thus, every ACE text has a single and well-defined formal meaning. Among other features, ACE supports singular and plural noun phrases, active and passive voice, relative phrases, anaphoric references, existential and universal quantifiers, negation, and modality. ACE has successfully been used for different tasks, e.g. as a natural language OWL front-end [17] and as an ontology language for the biomedical domain [19].

---

[1] http://attempto.ifi.uzh.ch/ape

In the following section, we will introduce the AceRules system and we will point out some of the problems involved (Sect. 2). Next, we will explain the interfaces available for AceRules (Sect. 3), and finally we will draw the conclusions (Sect. 4).

2 The AceRules System

AceRules is a multi-semantics rule system prototype using the controlled natural language ACE as input and output language. AceRules is designed for forward-chaining interpreters that calculate the complete answer set. The general approach of AceRules, however, could easily be adopted for backward-chaining interpreters. AceRules is publicly available as a web service and through two different web interfaces (see Sect. 3).

In order to clarify the functionality of AceRules, let us have a look at the following simple program. We use the term *program* for a set of rules and facts.

> If a customer has a card and does not have a code then CompanyX sends a letter
> to the customer.
> Every customer has a card.
> John is a customer.
> John does not have a code.

Submitting this program to AceRules, we get the answer shown below.[2] We use the term *answer* for the set of facts that can be derived from a program.

> John is a customer.
> CompanyX sends John a letter.
> John has a card.
> It is false that John has a code.

As we can see, the program and the answer are both in English. No other formal notations are needed for the user interaction. Even though inexperienced users might not be able to understand how the answer is inferred, they are certainly able to understand input and output and to verify that the output is some kind of conclusion of the input. This is the essential advantage of ACE over other formal knowledge representation languages.

Existing work to use natural language representations for rule systems is based on the idea of verbalizing rules that already exist in a formal representation [13, 16, 20]. In our approach, the controlled natural language is the *main* language that can be translated into a formal representation (parsing) and backwards (verbalizing). It is not necessary that the rules are first formalized in another language.

The rest of this section explains some of the important properties of AceRules, namely its multi-semantics architecture (Sect. 2.1), the representation of negation (Sect. 2.2), and the construction of valid programs (Sect. 2.3).

_____

[2] The courteous interpreter is used here. See Sect. 2.1 for details.

2.1 Multi-Semantics Architecture

AceRules is designed to support various semantics. The decision of which semantics to choose should depend on the application domain, the characteristics of the available information, and on the reasoning tasks to be performed. At the moment, AceRules incorporates three different semantics: courteous logic programs [12], stable models [10], and stable models with strong negation [11].

The original stable model semantics supports only negation as failure, but it has been extended to support also strong negation. Courteous logic programs are based on stable models with strong negation and support therefore both forms of negation. Section 2.2 will take a closer look at this issue.

None of the two forms of stable models guarantee a unique answer set. Thus, some programs can have more than one answer. In contrast, courteous logic programs generate always exactly one answer. In order to achieve this, priorities are introduced and the programs have to be acyclic.

On the basis of these properties, one should decide which semantics to choose. Since we do not want to restrict AceRules to a certain application or domain, we decided to make the semantics exchangeable.

The three semantics are implemented in AceRules as two interpreter modules. The first interpreter module handles courteous logic programs and is implemented natively.[3] For the stable model semantics with and without strong negation there is a second interpreter module that wraps the external tools *Smodels* [21] and *Lparse* [25].

There are various other semantics that could be supported, e.g. defeasible logic programs [22] or disjunctive stable models [23]. Only little integration effort would be necessary to incorporate these semantics into AceRules.

2.2 Two Kinds of Negation

In many applications, it is important to have two kinds of negation [27]. Strong negation (also called *classical negation* or *true negation*) indicates that something can be proven to be false. Negation as failure (also called *weak negation* or *default negation*), in contrast, states only that the truth of something cannot be proven.

However, there is no such general distinction in natural language. It depends on the context, what kind of negation is meant. This can be seen with the following two examples in natural English:

> If there is no train approaching then the school bus can cross the railway tracks.
> If there is no public transport connection to a customer then John takes the company car.

In the first example (which is taken from [11]), the negation corresponds to strong negation. The school bus is allowed to cross the railway tracks only if the available information (e.g. the sight of the bus driver) leads to the conclusion that

---

[3] The implementation of the courteous interpreter is based on [6].

no train is approaching. If there is no evidence whether a train is approaching or not (e.g. because of dense fog) then the bus driver is not allowed to cross the railway tracks.

The negation in the second sentence is most probably to be interpreted as negation as failure. If one cannot conclude that there is a public transport connection to the customer on the basis of the available information (e.g. public transport schedules) then John takes the company car, even if there is a special connection that is just not listed.

As long as only one kind of negation is available, there is no problem to express this in controlled natural language. As soon as two kinds of negation are supported, however, we need to distinguish them somehow. We found a clean and natural way to represent the two kinds of negation in ACE. Strong negation is represented with the common negation constructs of natural English:

- does not, is not (e.g. John is not a customer)
- no (e.g. no customer is a clerk)
- nothing, nobody (e.g. nobody knows John)
- it is false that (e.g. it is false that John waits)

To express negation as failure, we use the following constructs:

- does not provably, is not provably (e.g. a customer is not provably trustworthy)
- it is not provable that (e.g. it is not provable that John has a card)

This allows us to use both kinds of negation side by side in a natural looking way. The following example shows a rule using strong negation and negation as failure at the same time.

> If a customer does not have a credit-card and is not provably a criminal then the customer gets a discount.

This representation is compact and we believe that it is well understandable. Even for persons that have never heard of strong negation and negation as failure, this rule *makes some sense*, even though they are probably not able to grasp all its semantic properties. Of course, if users want to create or modify rules containing negation then they have to learn first how the two kinds of negation have to be represented in ACE.

2.3 Intelligent Grouping

ACE is an expressive language, in fact more expressive than most rule languages. Thus, some sentences have to be rejected by AceRules because they cannot be mapped to an acceptable rule of the respective rule theory. However, in some situations the formal structure is not directly compliant with the rule theory, but can be translated in a meaningful way into a valid rule representation. This translation we call *intelligent grouping*.

To make this point clear, we present some simple examples using stable model semantics with strong negation. The described procedure can be used in the same

way for the other semantics. Rules of the stable model semantics with strong negation have the form

$$L_0 \leftarrow L_1 \wedge \ldots \wedge L_m \wedge \sim L_{m+1} \wedge \ldots \wedge \sim L_n$$

with $0 \leq m \leq n$ and each L_i being a literal. A literal is an atomic proposition (A_i) or its strong negation ($\neg A_i$). Negations are allowed to be applied only to atomic propositions or — in the case of negation as failure ($\sim$) — to literals. Furthermore the heads of rules must contain nothing but a single literal. These restrictions we have to keep in mind when we translate an ACE text into a rule representation. As a first example, let us consider the following AceRules program:

> John owns a car.
> Bill does not own a car.
> If someone does not own a car then he/she owns a house.

The ACE parser transforms this text into its logical representation[4]

$$owns(john, X)$$
$$car(X)$$
$$\neg(owns(bill, Y) \wedge car(Y))$$
$$\neg(owns(A, B) \wedge car(B)) \rightarrow (owns(A, C) \wedge house(C))$$

which is not yet compliant with the rule theory. It contains complex terms inside of a negation and in the head of a rule. But considering the initial text, we would expect this example to be acceptable. In fact, it was just formalized in an inappropriate way. This is the point where the intelligent grouping is applied. If we aggregate some of the predicates then we end up with a simpler representation that has a correct rule structure:

$$owns\_car(john)$$
$$\neg owns\_car(bill)$$
$$\neg owns\_car(X) \rightarrow owns\_house(X)$$

This transformation is based on a set of grouping patterns that are collected in a first step, and then these patterns are used to perform the actual grouping. For our example, the following two patterns have been used:

$$owns(X_1, I_1), car(I_1) \quad \Rightarrow owns\_car(X_1)$$
$$owns(X_2, I_2), house(I_2) \Rightarrow owns\_house(X_2)$$

In such patterns, there can be two kinds of placeholders: Each X_i stands for any variable or atom, and each I_i stands for a variable that does not occur outside of the group. This allows us to omit the variables I_i after the transformation. From a more intuitive point of view, we can say that the phrases "owns a car" and "owns

---

[4] Throughout this article we will use a simplified form of the logical representation. For a more precise description, consult [7].

a house" are considered as atomic propositions. This means that the car and the house do not have an independent existence, and thus references to these objects are not allowed. If this restriction is violated then a consistent transformation into a valid rule structure is not possible. For example, the program

> Bill does not own a car.
> John owns a car X.
> Mary sees the car X.

that leads to the logical representation

$$\neg(owns(bill, A) \wedge car(A))$$
$$owns(john, B)$$
$$car(B)$$
$$sees(mary, B)$$

cannot be translated into a valid rule structure. An error message has to be raised in such cases informing the user that the program has an invalid structure. It has still to be evaluated how hard it is for normal users to follow this restriction and how often such situations actually occur. We are considering to develop authoring tools [3] that automatically enforce these restrictions.

Concerning the grouping step, one might think that the text was just translated into a too complex representation in the first place and that the parser should directly create a grouped representation. The following program shows that this is not the case:

> John owns a car.
> The car contains a suitcase.
> If someone X owns something that contains something Y then X owns Y.

It is transformed by the ACE parser into:

$$owns(john, H)$$
$$car(H)$$
$$contains(H, S)$$
$$suitcase(S)$$
$$owns(Z, X) \wedge contains(X, Y) \rightarrow owns(Z, Y)$$

In this case, we need the more fine-grained representation and no grouping has to be done since the program is already in a compliant form. This and the first example start both with the sentence "John owns a car", but in the end it has to be represented differently. Thus, the grouping is *intelligent* in the sense that it must consider the whole program to find out which predicates have to be grouped.

Another important property of the grouping step is that the transformation has to be reversible. Before verbalizing an answer set, we need to ungroup the predicates that were grouped before.

Altogether, the intelligent grouping gives us much flexibility. A sentence like "John owns a car" is treated as an atomic property of an object (John) or as a relation between two objects (John and a car), whichever makes sense in the respective context.

3   Interfaces

There are different existing interfaces that use controlled natural languages, e.g. query interfaces like *Querix* [18] or *LingoLogic* [26]. Other interfaces can also be used as editors like *ECOLE* [24], *GINO* [5], or *AceWiki*[5], but none of them concerns rule systems. There are applications like *NORMA* [14] which can verbalize formal rules, but no tool exists that uses a controlled natural language as a full-blown rule language.

AceRules comes with three interfaces. A webservice [2] facilitates the integration of the AceRules functionality into any other program. Furthermore, there are two web interfaces for human interaction. One is a technical interface[6] that is intended for advanced users that are interested in the technical background of the system. The main web interface[7] aims at end-users who are not familiar with formal notations. For the rest of this section, we will take a closer look at this main interface of which Fig. 1 shows a screenshot.

We claim that a Semantic Web interface for end-users (in the sense of an editor interface) should fulfill the following three properties which partly overlap.

1. Technical notations should be hidden. The users should not see any technical language (e.g. any XML-based language), but instead there should be a well-understandable and intuitive representation. Novice users should be able to understand the semantic representations after a very short learning phase.
2. The interface should guide the users during modification and creation of the formal representations. The users should not need to read any manuals or other documentation before they can start using the system, but they should be able to learn how to interact with the system while using it.
3. The users should be supported by a context-sensitive and comprehensive help feature. Especially in the case of errors, the users should be led immediately to a corresponding help article. These articles should be concise suggestions how to solve the problem.

Altogether, these three properties ensure that the Semantic Web interface has a shallow learning curve which we consider to be crucial for the success of the Semantic Web.

AceRules uses ACE as input and output language. No other notations are needed. Thus, AceRules fully satisfies the first condition. Furthermore, AceRules has a help feature which is shown in a browser-internal window. There is a help article for every error that can occur. If an error has occurred, then the user is directed to the respective article. Thus, AceRules fulfills the third condition as well.

AceRules gives also some help for the modification of existing programs and for the creation of new sentences. Nevertheless, we have to admit that it can only partially fulfill the second condition. For a real guidance of the user, a predictive

---

[5] `http://attempto.ifi.uzh.ch/acewiki`

[6] `http://attempto.ifi.uzh.ch/acerules_ti`

[7] `http://attempto.ifi.uzh.ch/acerules`
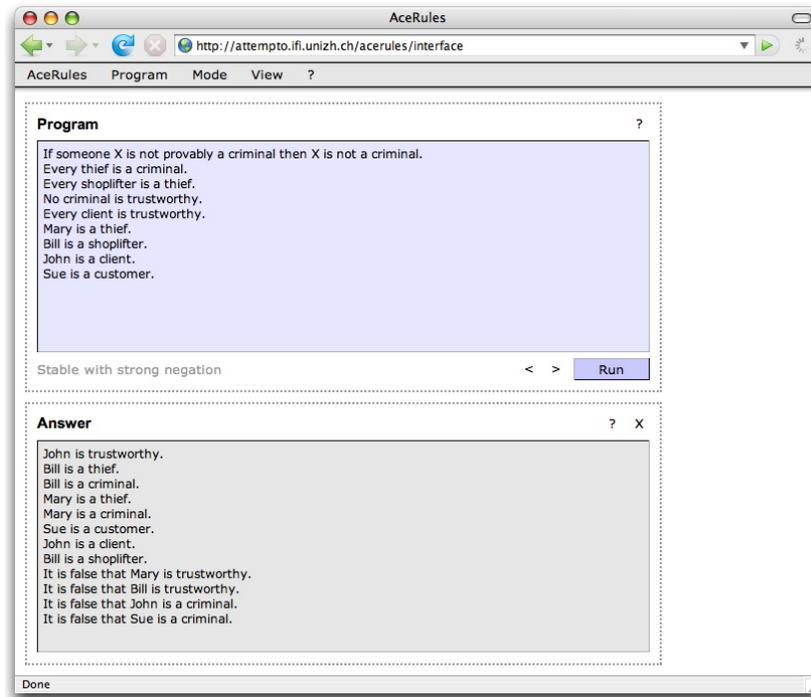
Fig. 1. The figure shows a screenshot of the AceRules web interface. The upper text box is the input component and contains the program to be executed. The result of the program is then displayed in the text box below

authoring tool [3] would be needed, as provided by ECOLE and AceWiki. Such an authoring tool guides the user step by step through the creation of a sentence and makes it impossible to create syntactically incorrect representations.

4   Conclusions

We demonstrated that it is possible to use controlled natural languages for the formal representation of rule systems. Negation as failure and strong negation can be used side by side in a natural way. We introduced intelligent grouping as a method of transforming formal structures into valid rule representations. The AceRules web interface proves that a controlled natural language like ACE is well suited for the communication with the user and that no other formal language is needed.

AceRules is still a prototype and not yet ready for the use in real world applications. For example, the underlying theories do not support procedural attachments or arithmetics that would probably be needed in a real world environment. We believe that ACE and AceRules can be extended to support such

constructs in a natural way. On the interface level, a predictive authoring tool would be very helpful. We referred to existing tools that demonstrate how this could be done.

Acknowledgement

References

1. *ACE 5 Interpretation Rules*. Attempto Documentation, 13 December 2006, `http://attempto.ifi.uzh.ch/site/docs/ace_interpretationrules.html`
2. *AceRules Webservice*. Attempto Documentation, 15 March 2007, `http://attempto.ifi.uzh.ch/site/docs/acerules_webservice.html`
3. *Authoring Tools for ACE*. Attempto Documentation, 23 March 2007, `http://attempto.ifi.uzh.ch/site/docs/authoring_tools.html`
4. Tim Berners-Lee, James Hendler, and Ora Lassila. *The semantic web*. Scientific American, 2001
5. Abraham Bernstein, Esther Kaufmann. *GINO - A Guided Input Natural Language Ontology Editor*. 5th International Semantic Web Conference (ISWC 2006), 144-157, Springer, 2006
6. Marc Dörflinger. *Interpreting Courteous Logic Programs*, Diploma Thesis. Department of Informatics, University of Zurich, 2005
7. Norbert E. Fuchs, Stefan Hoefler, Kaarel Kaljurand, Tobias Kuhn, Gerold Schneider, Uta Schwertel. *Discourse Representation Structures for ACE 5*, Technical Report ifi-2006.10. Department of Informatics, University of Zurich, 2006
8. Norbert E. Fuchs, Kaarel Kaljurand, Gerold Schneider. *Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces*. The 19th International FLAIRS Conference (FLAIRS'2006), 2006
9. Norbert E. Fuchs, Uta Schwertel, Rolf Schwitter. *Attempto Controlled English — Not Just Another Logic Specification Language*. In *Logic-Based Program Synthesis and Transformation*, Eighth International Workshop LOPSTR'98, Lecture Notes in Computer Science 1559, Springer, 1999
10. Michael Gelfond, Vladimir Lifschitz. *The stable model semantics for logic programming*. In *Proceedings of the 5th International Conference on Logic Programming*, 1070-1080, MIT Press, 1988
11. Michael Gelfond, Vladimir Lifschitz. *Classical negation in logic programs and disjunctive databases*. In *New Generation Computing*, 9:365-385, 1990
12. Benjamin N. Grosof. *Courteous Logic Programs: Prioritized Conflict Handling For Rules*. IBM Research Report RC 20836. Technical report, IBM T.J. Watson Research Center, 1997

13. Terry Halpin. *Business Rule Verbalization.* In Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (editors), *Proceedings of Information Systems Technology and its Applications*, 3rd International Conference ISTA 2004, Lecture Notes in Informatics, 2004

14. Terry Halpin, Matthew Curland. *Automated Verbalization for ORM 2.* In Robert Meersman, Zahir Tari, Pilar Herrero et al. (editors), *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, Lecture Notes in Computer Science 4278, 1181-1190, Springer, 2006

15. Stefan Hoefler. *The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0*, Technical Report ifi-2004.03. Department of Informatics, University of Zurich, 2004

16. Mustafa Jarrar, Maria Keet, Paolo Dongilli. *Multilingual verbalization of ORM conceptual models and axiomatized ontologies.* Technical report, Vrije Universiteit Brussel, 2006

17. Kaarel Kaljurand, Norbert Fuchs. *Bidirectional mapping between OWL DL and Attempto Controlled English.* In *Fourth Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, 2006

18. Esther Kaufmann, Abraham Bernstein, Renato Zumstein. *Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs.* 5th International Semantic Web Conference, 2006

19. Tobias Kuhn, Loïc Royer, Norbert E. Fuchs, Michael Schroeder. *Improving Text Mining with Controlled Natural Language: A Case Study for Protein Interactions.* In Ulf Leser, Barbara Eckman, and Felix Naumann (editors), *Proceedings of the 3rd International Workshop on Data Integration in the Life Sciences 2006* (DILS'06), Lecture Notes in Bioinformatics, Springer, 2006

20. Sergey Lukichev, Gerd Wagner. *Verbalization of the REWERSE I1 Rule Markup Language, Deliverable I1-D6.* Technical report, REWERSE, 2006

21. I. Niemelä and P. Simons. *Smodels — an implementation of the stable model and well-founded semantics for normal logic programs.* In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, Lecture Notes in Artificial Intelligence, volume 1265, 420-429, Dagstuhl, Germany, 1997

22. Donald Nute. *Defeasible Logic.* In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, 353-395, Oxford University Press, 1994

23. Teodor C. Przymusinski. *Stable Semantics for Disjunctive Programs.* In *New Generation Computing*, volume 9, number 3/4, 401-424, 1991

24. Rolf Schwitter, Anna Ljungberg, David Hood. *ECOLE: A Look-ahead Editor for a Controlled Language.* In *Proceedings of EAMT-CLAW03, Controlled Language Translation*, Dublin City University, 141-150, 2003

25. Tommi Syrjänen. *Lparse 1.0 User's Manual.* 2000

26. Craig W. Thompson, Paul Pazandak, Harry R. Tennant. *Talk to Your Semantic Web.* In *IEEE Internet Computing*, 9(6):75-79, 2005

27. Gerd Wagner. *Web Rules Need Two Kinds of Negation.* In *Principles and Practice of Semantic Web Reasoning*, Lecture Notes in Computer Science, volume 2901/2003, 33-50, Springer, 2003