Making sense of conflicting (defeasible) rules in the controlled natural language ACE: design of a system with support for existential quantification using skolemization*

Martin Diller
TU Wien, Austria
mdiller@kr.tuwien.ac.at

Adam Wyner
Swansea University, Wales
a.z.wyner@swansea.ac.uk

Hannes Strass
Leipzig University, Germany
strass@informatik.uni-leipzig.de

Abstract

We present the design of a system for making sense of conflicting rules expressed in a fragment of the prominent controlled natural language ACE, yet extended with means of expressing defeasible rules in the form of normality assumptions. The approach we describe is ultimately based on answer-set-programming (ASP); simulating existential quantification by using skolemization in a manner resembling a translation for ASP recently formalized in the context of $\exists$-ASP. We discuss the advantages of this approach to building on the existing ACE interface to rule-systems, ACERules.

1 Introduction

Attempto Controlled English (ACE) (Fuchs et al., 2008) is a prominent controlled natural language (CNL) for knowledge representation (KR). Apart of its appropriateness for basic KR, ACE's attraction comes, first of all, from its ties with formal logic: ACE texts have an unambiguous translation to first order logic (FOL). Secondly, there are several open-source tools for ACE; the main one being the parser APE, which translates ACE texts to FOL via discourse representation structures (DRSs) (Blackburn and Bos, 2005) and also verbalises DRSs.

Our interest is in the adaptation of ACE for handling conflicting information expressed in the form of strict and defeasible rules. In (Diller et al., 2017) we outlined a methodology for this task, which we more recently dubbed the EMIL ("Extracting Meaning out of Inconsistent Language") pipeline. The pipeline starts out with rules expressed in ACE (ACE rules), yet extended with means of expressing defeasible rules; currently, in the form of normality assumptions ("it is usual that"). We transform ACE rules to defeasible theories that can be evaluated via the direct stable semantics as defined in (Strass and Wyner, 2017). We verbalise possible manners of making sense of the rules (the "stable sets"), again, by making use of ACE.

In (Diller et al., 2017) we report on an adaptation of the main existing open-source interface to rule systems for ACE, ACERules (Kuhn, 2007), for our purposes[1]. At the back-end we used the ASP encodings for the direct stable semantics reported on in (Strass and Wyner, 2017). Here we motivate and

*This work has been funded by the Austrian Science Fund (FWF) projects I2854 and W1255-N23; also by the German Research Foundation (DFG) project 389792660 - TRR 248.

[1]This line of study has been continued via the PENG^{ASP} system (Guy and Schwitter, 2017), which as far as we are able to tell, inherits many of the features (while also improving on several others; e.g. in one of the more recent iterations, using a bi-directional grammar for specifying and verbalising ASP-programs (Schwitter, 2018)) of ACERules (in particular, that it does not offer explicit support of existential quantification) yet is not open-source nor publicly available for experimentation.

sketch the design of an alternative implementation of the EMIL pipeline. We, first of all, target shortcomings we found in the transformations that ACERules carries out for making APE parses ammenable to handling by rule systems. We refer to these in Section 2 and our alternative approach in Section 3. The main difference between the approach we present here and that based on ACERules is that ACERules attempts to remove existential variables whenever possible and filters-out ACE texts composed of rules it cannot handle. We here, on the other hand, simulate rules with existential quantification using skolemization in a manner resembling the procedure formalised in the context of ∃-ASP (Garreau et al., 2015) (see Section 3.1). A second difference is that we use dynamic encodings of the direct-stable-semantics to ASP, optimised for defeasible theories with function symbols (Section 3.2). For an extended presentation we refer to Chapter 4 of (Diller, 2019).

2 ACERules and its transformations

AceRules builds on APE to provide an ACE interface to formal rule systems. The system works by first checking the parses from APE , filtering those that amount to what we, following Garreau et al.[2], will call ∃-rules[3]. In their most general form, these have the form

$$b_1, \ldots, b_m, \triangle(n_1^1, \ldots, n_{u_1}^1), \ldots, \triangle(n_1^s, \ldots, n_{u_s}^s) \triangleright H$$

where $\triangleright$ is $\rightarrow$ ("strict implication") and H is of the form $h_1, \ldots, h_t$ or $\neg(h_1, \ldots, h_t)$. Also, $h_1, \ldots, h_t$, $b_1, \ldots, b_m$, $n_1^1, \ldots, n_{u_1}^1, \ldots, n_1^s, \ldots, n_{u_s}^s$ are atoms with constants and variables as usual in logic programs (Brewka et al., 2011), $m, s \geq 0$, and $t, u_1, \ldots, u_s \geq 1$. Moreover, $\triangle \in \{\texttt{not}, \texttt{not}\neg, \neg\}$ with $\neg$ standing for strong negation and not standing for negation-as-failure. Variables in $n_1^1, \ldots, n_{u_1}^1$ to $n_1^s, \ldots, n_{u_s}^s$ but not in $b_1, \ldots, b_m$ are interpreted as existentially quantified. The same for variables in the head H but not in $b_1, \ldots, b_m$. In the implementation from (Diller et al., 2017), dACERules, we modified APE to recognise "it is usual that" as a form of subordination on the par with modal constructs. We thus also allow $\triangleright$ to be $\Rightarrow$ ("defeasible implication"). On the other hand, we disallow negation-as-failure.

ACERules (and thus dACERules) filters DRSs corresponding to ∃-rules rather than normal rules (i.e. $u_i = 1$ for $0 \leq i \leq s$, $t = 1$, and there are no existentially quantified variables) because most meaningful examples of ACE rules require the additional resources provided by ∃-rules. The reason is not only that ∃-rules are natural, but also due to the flat form of logical atoms used by APE (Fuchs et al., 2013). I.e. a sentence such as "Mary gladly gives John a present" is represented as

$$\exists A, B\big(object(A, present, countable, na, eq, 1) \wedge modifier\_adv(B, gladly, pos) \wedge predicate(B, give, named(Mary), A, named(John))\big)$$

where e.g. nouns, verbs, and adverbs are "wrapped" into the special atoms "object", "predicate", "modifier_adv". These have additional arguments encoding semantic information. Critical for our purposes is that only noun phrases and verb phrases introduce referents (quantified variables); the remaining predicates make use of the referents introduced by noun and verb phrases. In particular, verb phrases have a Neo-Davidsonian event-theoretic semantics (Parsons, 1990), allowing to attach modifiers stemming from adverbs and prepositional phrases to the referents introduced by verb phrases (in the example "B").

In order to make ACE texts parsed as ∃-rules ammenable to processing by rule systems , ACERules implements a series of transformations of the ∃-rules; the main ones being predicate condensation and grouping (only the latter is discussed in (Kuhn, 2007)). Predicate condensation merges atoms for verb phrases and their modifiers, grouping them in a single atom $pred\_mod$; the referent for the verb is removed. E.g. for the example above the result of predicate condensation is:

$$pred\_mod(give, named(Mary), A, named(John), [modifier\_adv(gladly, pos)])$$

The problem with this transformation is that it modifies the semantics of the parse given by APE: the "diamond inference pattern" holding between a verb phrase plus modifiers and its components (see Parsons (1990)) is broken. So e.g. the output given by ACERules for the text (adapted from Parsons (1990))

[2] Although they do not consider strong negation.

[3] ACERules also transforms rules with double implication to equivalent ∃-rules.

```
Brutus unhesitatingly stabs Caeser in the back with a knife.
If Brutus stabs Caeser then Brutus is a traitor.
```

does not include the assertion that Brutus is a traitor[4] since ACERules is unable to relate "Brutus stabs Caeser" with "Brutus unhesitatingly stabs Caeser in the back with a knife".

After postprocessing condensed atoms for the copula "be" and the preposition "of" (see (Diller, 2019)), the main transformation carried out by ACERules is "grouping", which amounts to aggregating atoms appearing in the heads of $\exists$-rules or negated (via $\neg$, not $\neg$, or not) in the bodies of rules when possible. Atoms appearing in other parts of the parse of APE, yet which match some of the grouped atoms, are likewise grouped together ("matching phase"). Existentially quantified variables in the grouped atoms are removed subject to the to the restriction that the variable is not used elsewhere in the parse, outside of the group of atoms. An example of rules that can be treated by grouping are (from (Kuhn, 2007)):

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

Here atoms for "owns a car" and "owns a house" are merged in a single atom (denoting "owns-a-car" and "owns-a-house") and the variables referring to the car and house respectively are removed. If after "John owns a car" one adds the sentence "Mary sees the car" grouping fails as reference to a car independent of someone owning it is needed. See (Kuhn, 2007) and (Diller, 2019) for further details.

Apart from groups of atoms being treated as lists rather than sets in ACERules[5], we found the checks for grouping to succeed to be too liberal. As an example, the matching phase only considers groups of atoms that match exactly, while often also sub-groups need to be considered. E.g., consider:

```
John owns a car.
Every car is an automobile.
John does not own an automobile.
```

ACERules groups atoms for "John owns an automobile" and then is unable to relate "owning a car" with "owning an automobile" concluding that John owns a car but not an automobile.

The matching phase in ACERules also does not consider groups matching in terms of the atoms but differing in generality. An example of a text where this is necessary is:

```
Bill does not own a vehicle.
If Bill does not own a vehicle then he does not own a car.
If someone does not own a car then he/she owns a motorcycle.
If someone owns a motorcycle then he/she owns a vehicle.
```

ACERules is unable to relate the group of atoms for "Bill owns a vehicle" and the more general (because of the use of an indefinite pronoun) "he/she owns a vehicle" concluding that it is both true and false that John owns a vehicle.

More minor issues we found in our study of ACERules are that transformations introduced for indefinite pronouns blur the distinction between inanimate objects and persons[6] and post-processing of condensed atoms for the copula "be" favours an intersective reading of adjectives[7].

3 A system with support for existential quantification using skolemization

We have shown in Section 2 that several of the transformations implemented in ACERules, particularly predicate-condensation and grouping, introduce significant deviations from the semantics of ACE rules induced by APE. Moreover, even in the current rather liberal implementation of grouping, the texts ACERules can handle are limited; especially, often existential quantification is unavoidable.

[4]While, for instance, the first order reasoner for ACE, RACE (Fuchs, 2010), finds a proof of "Brutus is a traitor" from the same text. Another issue with predicate condensation, which is more likely a bug than an intended feature of ACERules, is that modifiers in the *pred_mod* predicates are aggregated into ordered lists, while their semantics (at least as given by the parse by APE) would require them to be aggregated into sets.

[5]This bug is documented in the source-code of ACERules.

[6]Thus, e.g., from "there is a table" and "everybody likes Mary" ACERules concludes that "the table [X1] likes Mary".

[7]For instance (example adapted from https://www3.nd.edu/~jspeaks/courses/2012-13/43916/handouts/13-modifiers.pdf; accessed on 28.11.2018) ACERules reads the subsective adjective "tall" intersectively in the discourse composed of the sentences "Bob is a tall midget", "Bob is a basketball-player", and "if Bob is a tall basketball-player then he plays for the NBA"; concluding that Bob plays for the NBA.

3.1 From strict and defeasible ∃-rules to normal rules

The most obvious manner of supporting existential quantification in rules is by using skolemization in normal-rules to simulate the latter; for ASP this approach has recently been formalised in the context of ∃-ASP (Garreau et al., 2015). Although the original definitions for defeasible theories from (Strass and Wyner, 2017) do not allow function symbols these can be incorporated without further ado; in fact, the ASP-encodings reported on in (Strass and Wyner, 2017) support them. Having function symbols, we can thus compile strict and defeasible ∃-rules (without `not`) to normal defeasible theories in a manner similar to that proposed in (Garreau et al., 2015) for transforming ∃-rules into normal ASP programs. Differences are due to the fact that we have strong negation rather than negation-as-failure (in (Garreau et al., 2015) strong negation is not considered) and defeasible implication in addition to strict implication.

The translation of strict ∃-rules with positive elements in the body and head is exactly as in (Garreau et al., 2015); defeasible rules introduce the issue of deciding the scope of "⇒". Consider, for instance:

```
It is usual that a ferry that starts in Vienna services Bratislava.
```

In our translation the rule gets replaced by the unary assumption with an auxilliary atom:

$$\Rightarrow x\_auxPH1()$$

accompanied by auxilliary rules introducing skolem constants (e.g. x_sk1) for the objects and verbs[8]:

$$x\_auxPH1() \triangleright_1 object(x\_sk1(), ferry)$$
$$x\_auxPH1() \triangleright_2 predicate(x\_sk2(), start, x\_sk1())$$
$$x\_auxPH1() \triangleright_3 modifier\_pp(x\_sk2(), in, vienna)$$
$$x\_auxPH1() \triangleright_4 predicate(x\_sk3(), service, x\_sk1(), bratislava)$$

Having $\triangleright_i$ being $\rightarrow$ for each i ($1 \leq i \leq 4$) amounts to interpreting the scope of $\Rightarrow$ to be over the entire phrase "a ferry that starts in Vienna services Bratislava". We currently implement this option, but a more satisfactory reading may be that there usually is a ferry that either starts in Vienna or services Bratislava (and, typically, both). This option can be encoded using further auxilliary atoms (see (Diller, 2019)).

Turning to ∃-rules with negated atoms in the head, consider the sentence "if someone owns a car then he/she does not own a house". In our translation we once more replace the head of the ∃-rule obtained from the `APE` parse with an auxilliary atom:

$$object(A, somebody), object(B, car), predicate(C, own, A, B) \rightarrow x\_auxNH1(A, B, C)$$

and add rules encoding the meaning of the auxilliary atom:

$$object(D, house), x\_auxNH1(A, B, C), pName(E) \rightarrow \neg predicate(E, own, A, D)$$
$$predicate(E, own, A, D), x\_auxNH1(A, B, C) \rightarrow \neg object(D, house)$$

Here the use of the special atom $pName$ (which collects all variables standing for verbs) is optional, but used for the first rule to be safe (i.e. all variables occurring in the head occur in the body). If there are more than two atoms appearing negated in the head of a rule we need to apply the illustrated translation recursively. Note also that the treatment of defeasible rules with negative heads is exactly analogous to that of strict rules. The reason is that $\Rightarrow$ inherits the scope from $\neg$ in this case.

A conceptually more intricate case is when negation occurs in atypical manner in the bodies of rules, e.g. for the sentence "if someone does not own a car then he/she owns a house". Here there are several options. The most straightforward, following more or less (Garreau et al., 2015), is to put the burden of proof on the existential assertion; i.e. by default no one owns a car. This option, which is the one we currently implement, can be encoded as follows (omitting the auxilliary rules for x_auxPH1) :

$$object(A, somebody), \neg x\_auxPB1(A) \rightarrow x\_auxPH1(A)$$
$$object(B, car), predicate(C, own, A, B) \rightarrow x\_auxPB1(A)$$
$$object(A, somebody) \Rightarrow \neg x\_auxPB1(A)$$

One can also put the burden of proof on the negation of the existential assertion (see (Diller, 2019)), but arguably more in line with the framework of (Strass and Wyner, 2017) is to reason by cases; i.e.

[8] For readability we use a simpler representation of the atoms to that of `APE`.

consider for everyone both the possibility that the he/she owns a car and that he/she does not. One simple encoding of this option is as follows:

$$object(A, somebody), \neg x\_auxPB1(A) \rightarrow x\_auxPH1(A)$$
$$object(A, somebody) \Rightarrow \neg x\_auxPB1(A)$$
$$object(A, somebody) \Rightarrow x\_auxPB1(A)$$
$$x\_auxPB1(A) \rightarrow object(x\_sk1(A), house)$$
$$x\_auxPB1(A) \rightarrow predicate(x\_sk2(A), own, A, x\_sk1(A))$$
$$\neg x\_auxPB1(A), predicate(C, own, A, B) \rightarrow \neg object(B, house)$$
$$\neg x\_auxPB1(A), object(B, house), pName(C) \rightarrow \neg predicate(C, own, A, B)$$

3.2 Dynamic ASP encoding for defeasible theories with variables

The ASP encoding for evaluating (normal) defeasible theories[9] via the direct stable semantics we used in the implementation of the EMIL pipeline reported on in (Diller et al., 2017) is static: only the part specifying the defeasible theory changes with the input. The module for the semantic evaluation uses ASP-disjunction and remains fixed. The encoding is thus complexity-sensitive for propositional defeasible theories (the complexity of the latter and the data complexity of disjunctive ASP is Σ_2^P-complete).

For theories with variables the encoding nevertheless has the disadvantage that defeasible theories need to be specified essentially as facts and hence the grounding (transformation of defeasible theories with variables to theories without variables; see (Strass and Wyner, 2017)) needs to be generated explicitly (in most cases) while this is usually not the case for ASP programs with variables (Kaufmann et al., 2016). This is even more a problem when using function symbols, which introduce the possibility of infinite groundings. For instance, the defeasible theory $\{\rightarrow o(a), o(X) \rightarrow p(f(X)), p(X) \rightarrow q(f(X))\}$ has the unique stable set $\{a, p(f(a)), q(f(a))\}$, which as an ASP program is computed in under one second[10] by e.g. the ASP-solver `clingo` (5.3.0) (Gebser et al., 2018). When evaluating the theory in the context of the encodings from (Strass and Wyner, 2017) via `clingo` there are memory errors ("std::bad_alloc") after 48.566 seconds.

For the mentioned reason, we developed alternative dynamic (both the data and program change with the input), yet structure-preserving ASP encodings for evaluating defeasible theories with variables via the direct-stable-semantics for our new implementation of the EMIL pipeline. Such encodings allow us to piggyback on the grounding developments for any ASP grounder (+ solver) we wish to experiment with. Moreover, the encodings are to non-disjunctive ASP. We refer to (Diller, 2019) for details.

4 Discussion and future work

We have outlined an alternative design for a system for making sense of conflicting rules in the CNL ACE. The main component is a translation from (defeasible) $\exists$-rules to normal rules which can be seen as a form of meaning-preserving grouping, subsuming a form of predicate-condensation. In particular, the latter does not break the relation between verbs and their modifiers (via the use of auxilliary rules and skolemization) (see the examples in Section 3.1). The second component is the dynamic ASP encoding optimised for evaluating defeasible theories with variables.

We have an implementation working[11]. Immediate future work is to add support for generating arguments from defeasible theories and experimenting with ASP grounders and solvers. More future plans are incorporation of means of restricting existential variables whenever possible (in the spirit of `ACERules`) and/or restrictions to ensure finite groundability. Further long term goals are investigation of alternative means of supporting existential quantification, support of different forms of adding defeasibility to ACE, and enhancing the natural language understanding capabilities of our system.

[9] `https://github.com/hstrass/defeasible-rules`

[10] On a 4 GB openSUSE (42.3) machine with 4 Intel Core processors (3.30 GHz).

[11] We have been able to successfully run (in under one minute) most test-cases (with some modifications in case of there being negation-as-failure as well as priorities over rules) available for `AceRules` (around 40 of them). The implementation is available at `https://www.dbai.tuwien.ac.at/proj/grappa/emil/`. An upcoming version will include treatment of the copula "be" and the preposition "of"; in the first case modifying and in the second case incorporating the treatment of `ACERules` (see (Diller, 2019)).

References

Blackburn, P. and J. Bos (2005). *Representation and Inference for Natural Language: A First Course in Computational Semantics*. CSLI Publications.

Brewka, G., T. Eiter, and M. Truszczynski (2011). Answer set programming at a glance. *Commun. ACM 54*(12), 92–103.

Diller, M. (2019). *Realising argumentation using answer set programming and quantified boolean formulas*. Ph. D. thesis, Vienna University of Technology. Submitted.

Diller, M., A. Wyner, and H. Strass (2017, September). Defeasible AceRules: A prototype. In C. Gardent and C. Retoré (Eds.), *Proceedings of the 12th International Conference on Computational Semantics (IWCS 2017)*.

Fuchs, N. E. (2010). First-Order Reasoning for Attempto Controlled English. In M. Rosner and N. E. Fuchs (Eds.), *Proceedings of the 2nd International Workshop on Controlled Natural Language (CNL 2010)*, Volume 7175 of *Lecture Notes in Computer Science*, pp. 73–94. Springer.

Fuchs, N. E., K. Kaljurand, and T. Kuhn (2008). Attempto Controlled English for knowledge representation. In *Tutorial lectures of the 4th International Summer School on the Reasoning Web (Reasoning Web 2008)*, pp. 104–124.

Fuchs, N. E., K. Kaljurand, and T. Kuhn (2013). Discourse Representation Structures for ACE 6.7. Technical report. Available at `http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_67.pdf`.

Garreau, F., L. Garcia, C. Lefèvre, and I. Stéphan (2015). ∃-ASP. In *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, Volume 1517 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gebser, M., R. Kaminski, B. Kaufmann, P. Lühne, P. Obermeier, M. Ostrowski, J. Romero, T. Schaub, S. Schellhorn, and P. Wanko (2018). The Potsdam Answer Set Solving Collection 5.0. *KI 32*(2-3), 181–182.

Guy, S. and R. Schwitter (2017). The PENG ASP system: architecture, language and authoring tool. *Language Resources and Evaluation 51*(1), 67–92.

Kaufmann, B., N. Leone, S. Perri, and T. Schaub (2016). Grounding and solving in answer set programming. *AI Magazine 37*(3), 25–32.

Kuhn, T. (2007). AceRules: Executing Rules in Controlled Natural Language. In *Proceedings of the 1rst International Conference on Web Reasoning and Rule Systems (RR 2007)*, Volume 4524 of *Lecture Notes in Computer Science*, pp. 299–308. Springer.

Parsons, T. (1990). *Events in the Semantics of English: A Study of Subatomic Semantics* (1s ed.). Cambridge, MA, USA: MIT Press.

Schwitter, R. (2018). Specifying and Verbalising Answer Set Programs in Controlled Natural Language. *TPLP 18*(3-4), 691–705.

Strass, H. and A. Wyner (2017, February). On Automated Defeasible Reasoning with Controlled Natural Language and Argumentation. In R. Barták, T. L. McCluskey, and E. Pontelli (Eds.), *Proceedings of the 2nd International Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS 2017)*.