Attempto Controlled English for Knowledge Representation

Norbert E. Fuchs

Department of Informatics & Institute of Computational Linguistics
University of Zurich

fuchs@ifi.uzh.ch
http://attempto.ifi.uzh.ch

GF Summer School 2011

Overview

- Languages for Knowledge Representation
- Attempto Controlled English (ACE)
- Translating ACE into First-Order Logic
- Attempto Tools
- Typical Applications of ACE
- Other Controlled Languages

Problem Knowledge Representation

- How should one represent the biological fact that every protein has a terminus?
- some possible representations

first-order logic	$\forall X(protein(X) \rightarrow \exists Y(terminus(Y) \wedge has(X,Y)))$
DL	$Protein \sqsubseteq \exists has.Terminus$

OWL (RDF/XML)	`<owl:Class rdf:ID="Protein">`
	`  <rdfs:subClassOf>`
	`  <owl:Restriction>`
	`  <owl:onProperty rdf:resource="#has"/>`
	`  <owl:someValuesFrom rdf:resource="#Terminus"/>`
	`  </owl:Restriction>`
	`  </rdfs:subClassOf>`
	`</owl:Class>`

UML	
ACE	Every protein has a terminus.

Languages for Knowledge Representation

- formal languages
  - \+ well defined-syntax, unambiguous semantics
  - \+ support automated reasoning
  - – conceptual distance to application domain
  - – incomprehensibility, acceptance problems
- natural language
  - \+ user-friendly: easy to use and understand
  - \+ no extra learning effort
  - \+ high expressiveness, close to application domain
  - – ambiguity, vagueness, incompleteness, inconsistency

Attempto Controlled English (ACE)

- Attempto Controlled English combines pros of formal and natural languages
- ACE is a *controlled* natural language
  - precisely defined, tractable subset of full English
  - automatic, unambiguous translation into first-order logic
- ACE is human *and* machine understandable
  - ACE seems completely natural, but is a formal language
  - one could say that ACE is a first-order logic language with an English syntax
  - ACE texts can be read by anybody who knows English
- ACE *combines* natural language with formal methods
  - provably easier to learn and to use than visibly formal languages
  - automated reasoning with ACE via existing tools

An ACE Appetiser
(Actually Quite a Mouthful of ACE)

...

Every customer has at least two cards and their associated codes. If a customer C approaches an automatic teller and she inserts her own card that is valid carefully into the slot and types the correct code of the card then the automatic teller accepts the card and displays "Card accepted" and C is satisfied. No card that does not have a correct code is accepted. It is false that a customer's card is valid, and is expired or is cancelled. If there is someone X and it is not provable that X is a criminal then the bank can safely assume that X is trustworthy.

...

Important Notice

- The language ACE and the ACE parser do not contain any a priori knowledge of application domains or of formal methods. Users must explicitly define all domain knowledge – for instance definitions, constraints, ontologies – as ACE texts.
- Words occurring in ACE texts are processed by the ACE parser as uninterpreted syntactic elements, i.e. any meaning of these words is solely added by the human writer or reader.

English and ACE

- if you know English then you already know most of ACE, yet …
- … English is both syntactically and semantically much more powerful than ACE
- English is used in human-human communication while ACE is also meant to be used in human-computer communication…
- ... which means that ACE is really a formal language
- ACE simply resembles English syntactically and semantically – and profits on this similarity

English and ACE

- like English, ACE allows syntactically different sentences to express the same meaning (i.e. there is a lot of synonymy)

- unlike English, each ACE text is interpreted in just one way
  - ACE is not ambiguous
  - an ACE text read as an English text can have other meanings

- while English uses syntactic and semantic means to analyse a sentence, ACE uses only syntax

The Language ACE

- vocabulary
- grammar
  - construction rules (syntax)
    - define admissible sentence structures
    - avoid ambiguous or imprecise constructs
  - interpretation rules (semantics)
    - control logical analysis of admissible sentences
    - resolve remaining ambiguities
- style guide (pragmatics)

Vocabulary

- predefined function words (articles, prepositions, ...)
- predefined phrases ('there is a ...', 'it is false that ...')
- user-defined content-words (nouns, verbs, adjectives, adverbs)
- basic full-form lexicon (~100'000 words)
- optionally: user-defined lexicons
- unknown words: guessing of word class
- unknown words: prefixing with word class

  A a:trusted man a:deliberately v:backs-up the n:web-page of the n:pizza-delivery-service.

Construction Rules: ACE Texts

- An ACE text is any sequence of anaphorically interrelated declarative, interrogative and imperative sentences.
- declarative sentences
  - describe a state of affairs, a situation, an event, a fact
  - end with a full stop
  - can be simple or composite
- interrogative sentences
  - query the contents of ACE texts
  - end with a question mark
- imperative sentences
  - implement commands
  - end with an exclamation mark

Construction Rules: Simple Sentences

- simple sentences have the structure

  subject + predicate + complements + adjuncts
- complements are the direct and indirect objects
- adjuncts are optional adverbs and optional prepositional phrases
- examples
  - *An old customer waits patiently*.
  - *John's customer inserts a card of Mary*.
  - *(= A card of Mary is inserted by John's customer.)*
  - *A card X and a code Y are valid*.
  - *A customer gives each of two cards to a clerk*.
  - *A customer who is impatient manually inserts at least 2 invalid cards into a slot*.

Construction Rules: *there is* Sentences

- it is possible to create well-formed simple sentences without a verb by using the *there is/are* construct that introduces only an object
  - *There is a customer*.
- no adjuncts or complements are allowed (because there is no verb)
  - **There is a customer in the bank*.
- relative clauses are possible (because a noun is present)
  - *There is a customer who waits*.

Construction Rules: Formulas

- logical formulas constitute another form of simple sentences
  - *10 = 4 + 6.*
  - *5 > 3.*
  - *X >= 13.4.*
- note that formulas are not evaluated by the ACE parser

Construction Rules: Composite Sentences

- composite sentences are recursively built from simpler sentences with the help of predefined constructors
  - coordination
  - negation
  - quantification
  - subordination
  - modality
- example
  - *If a customer inserts a card that is not valid then the automatic teller rejects the card and can display a message.*

Construction Rules: Coordination

- sentences can be coordinated by *and* and *or*
- examples
  - *The screen blinks and John waits.*
  - *X < 4 or X >10.*
- overriding of standard binding order by commas
  - *The screen blinks or John waits and Mary looks at the screen.*
  - *The screen blinks or John waits, and Mary looks at the screen.*

Construction Rules: Negation

- negated existential quantifier
  - *John enters no code*.
- negated universal quantifier
  - *John enters not every code*.
- VP negation
  - *John does not enter a code*.
- negated copula
  - *Some water is not drinkable*.

Construction Rules: Quantification

- existential quantification
  - *There is a card*.
  - *John drinks some water*.
- universal quantification
  - *John enters every card*.
- global existential quantification
  - *There is a code that every clerk checks*.
- global universal quantification
  - *For every code a clerk enters it*.

Construction Rules: Subordination

- ACE knows several forms of subordination
  - conditional sentences
  - sentence subordination
  - modality

Construction Rules: Conditional Sentences

- conditional sentences are built with the help of *if … then*
  - *If John enters a card then the automatic teller accepts it.*
- equivalence of universally quantified and conditional sentences
  - *Every customer enters a card.*

  is equivalent to
  - *If there is a customer then the customer enters a card.*

Construction Rules: Sentence Subordination

- negation
  - *It is false that a customer inserts a card*.
- negation as failure (outside of first-order logic, intended to support translation of ACE into rules and into languages like Prolog)
  - *It is not provable that a customer inserts a card*.
- sentences as objects of verbs
  - *A clerk believes that a customer inserts a card*.
  - *John wants to sleep*.

Construction Rules: Modality

- possibility
  - *A trusted customer can insert a card.*
  - *It is possible that a trusted customer inserts a card.*
- necessity
  - *A trusted customer must insert a card.*
  - *It is necessary that a trusted customer inserts a card.*
- recommendation (outside of first-order logic)
  - *A trusted customer should insert a card.*
  - *It is recommended that a trusted customer inserts a card.*
- admissibility (outside of first-order logic)
  - *A trusted customer may insert a card.*
  - *It is not admissible that a trusted customer inserts a card.*

Ambiguity in English and in ACE

- English is a highly ambiguous language
- ambiguity can occur on several levels
  - syntax: *A man sees a girl with a telescope*. *John has a flat mate*.
  - scope: *Everybody loves somebody*.
  - lexical: *Mary sees a bank*.
- English relies heavily on context to resolve ambiguity
  - *A man sees a girl with a green dress*.
- ACE uses *only* structural information to resolve ambiguity
  - *A man {sees a girl with a telescope.}*
  - *A man {sees a girl with a green dress.}*
- ACE sentences are not ambiguous; however the same sentences can be ambiguous when read as full English

Constraining Ambiguity

- ACE employs three simple means to constrain the ambiguity of natural language
  - some ambiguous constructs are not part of ACE; unambiguous alternatives are available in their place
  - all remaining ambiguous constructs are interpreted deterministically on the basis of a small set of *interpretation rules*
  - users can accept the assigned interpretation, or they must rephrase the input to obtain another one
- here is an ...

Constraining Ambiguity: Structural Ambiguity

- ... example:
  - input 1: *A customer inserts a card that is valid and has a code.*
  - *paraphrase 1:* *A card X is valid. A customer Y inserts the card X. The customer Y has a code Z.*
  - *input 2:* *A customer inserts a card that is valid and that has a code.*
  - *paraphrase 2:* *A card X is valid. A customer Y inserts the card X. The card X has a code Z.*

Interpretation Rules: Structural Ambiguity

- prepositional phrases modify the verb not the noun
  - *A customer {enters a card with a code}.*
- relative clauses modify the immediately preceding noun
  - *A customer enters {a card that carries a code}.*
- to express coordination within a relative clause the relative pronoun has to be repeated
  - *A customer inserts {a card that is valid and that has a code}.*

Interpretation Rules: Scope Ambiguity

- textual position of a quantifier opens its scope that extends to the end of the sentence, or in a coordination to the end of the respective coordinated phrase
  - *{A customer types {every code}}.* ∃∀
  - *{Every customer types {a code}}.* ∀∃

Constraining Ambiguity: Lexical Ambiguity

- verbs are highly ambiguous, since the same verb can appear as intransitive, transitive and ditransitive, and furthermore can occur with and without phrasal particles and prepositions as integral constituents

- to constrain this type of lexical ambiguity ACE expects that the phrasal particle of a phrasal verb (look up, drop out, shut down) and the preposition of a prepositional verb (look at, apply for) are hyphenated to the verb
  - *A steward waits-on the table.* (vs. *The food waits on the table.*)
  - *John looks-up an entry.* (vs. *John looks up the alley.*)
  - *What does John apply-for?* (vs. *John applies for the second time.*)

Summary of ACE's Disambiguation

- advantages of constructive disambiguation
  - automatic and efficient disambiguation
  - no use of contextual knowledge, domain knowledge, ontologies
  - simple, systematic, general, easy to learn interpretation rules
  - reliable, reproducible and thus intelligible behaviour
- open problems
  - rules do not always lead to natural interpretation
  - sometimes result in stilted English
  - Can we control all ambiguities with this strategy?
  - Does strategy scale up to a larger fragment of ACE?

Anaphoric References

- ACE texts are interrelated by anaphoric references, i.e. references to textually preceding noun phrases
- anaphoric references can be made by
  - proper names: *John - John*
  - reflexive and non-reflexive pronouns: *a card - it*, *John - himself*
  - definite noun phrases: *a card - the card*, *a red card - the red card*, *a man who waits - the man who waits*
  - variables: *a card X - the card X*, *a card X - X*
- *John has a customer. John inserts his card and types a code X. Bill sees X. He inserts his own card and types the code.*

Interpretation Rules: Anaphoric References

- proper names like *John* always denote the same object and thus serve as their own anaphoric references
- in all other cases resolution of anaphoric references is governed by
  - accessibility
  - recency
  - specificity
  - reflexivity

Interpretation Rules: Accessibility

- a noun phrase is not accessible if it occurs in a negated sentence
  - *John does not enter a card. *It is correct.*
- a noun phrase is not accessible if it occurs in a conditional sentence
  - *Every customer has a card. *It is correct.*
- but a noun phrase in the *if*-part of a conditional sentence is accessible in the *then*-part
  - *If a customer has a card then he enters it.*
- a noun phrase in a disjunction is only accessible in subsequent disjuncts
  - *A customer enters a card or drops it. *It is dirty.*

Interpretation Rules: Definite Noun Phrases

- if the anaphor is a definite NP then it is resolved with the most recent and most specific accessible noun phrase that agrees in gender and number
- example
  - *There is a blue ball. There is a red ball. John sees the ball. Mary sees the blue ball.*
- pragmatics often requires using a definite noun phrase that is not meant anaphorically
  - if a definite NP cannot be resolved then it is interpreted as an indefinite noun phrase introducing a new object
  - *John goes to the bank.* (= *John goes to a bank.*)

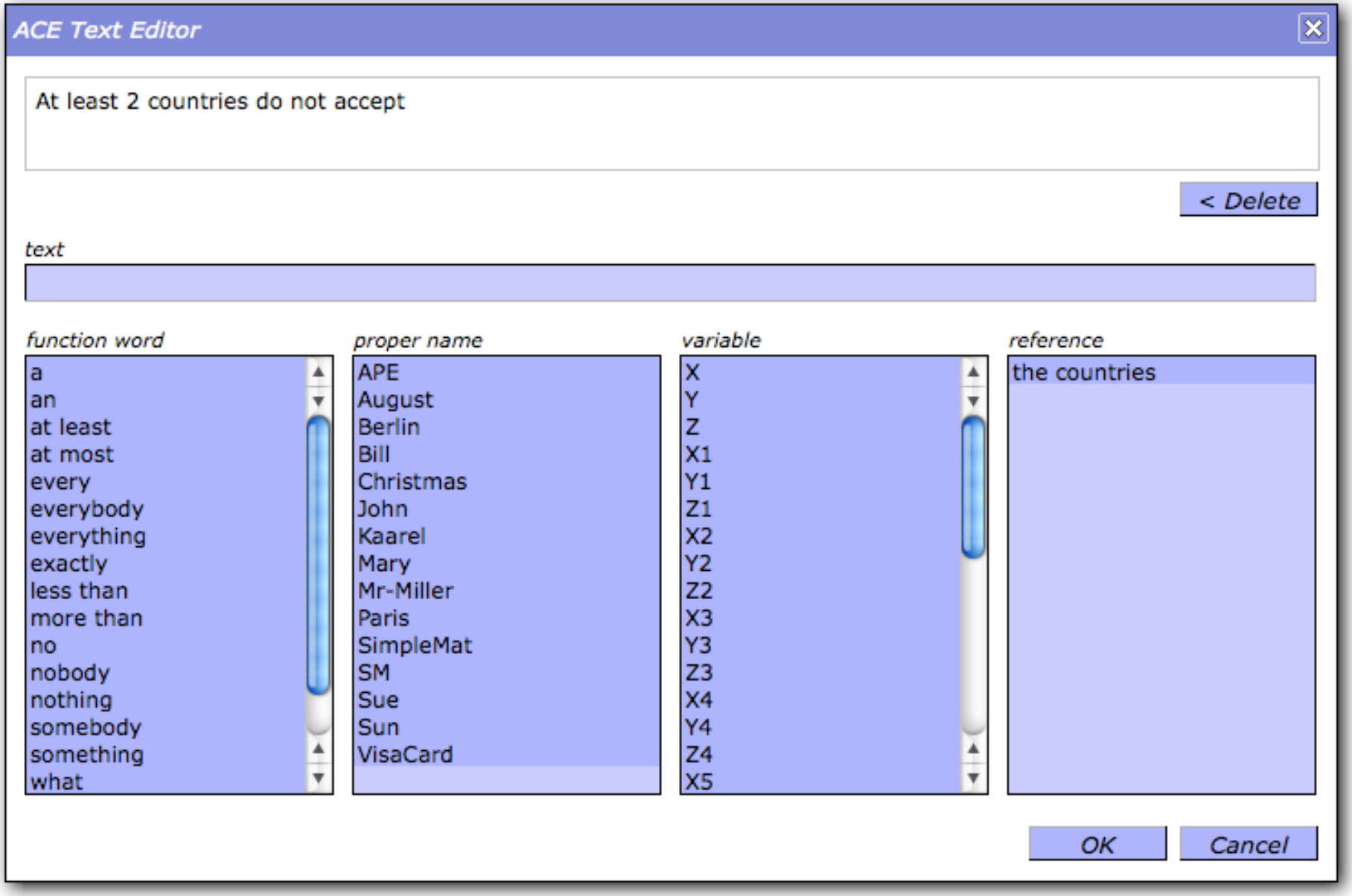
Very Brief Style Guide

- While the ACE parser will unravel any syntactically correct sentence, however complex, you may have problems to do so …
- … thus keep it simple.
- Remember that your text is the only source of information; there is no hidden knowledge.

ACE Editor

- APE requires users to learn and to recall the ACE construction and interpretation rules which is certainly possible as we know from teaching ACE, but …

- ACE Editor is an experimental predictive editor that helps users to construct syntactically and lexically correct ACE texts by just clicking on words and word classes
- alternatively users can freely input ACE texts


ACE Text Editor
At least 2 countries do not accept
< Delete
text
function word
a
an
at least
at most
every
everybody
everything
exactly
less than
more than
no
nobody
nothing
somebody
something
what
proper name
APE
August
Berlin
Bill
Christmas
John
Kaarel
Mary
Mr-Miller
Paris
SimpleMat
SM
Sue
Sun
VisaCard
variable
X
Y
Z
X1
Y1
Z1
X2
Y2
Z2
X3
Y3
Z3
X4
Y4
Z4
X5
reference
the countries
OK
Cancel

From ACE to First-Order Logic

- input: ACE text
- target: Extended Discourse Representation Structure (DRS)
  - uses syntactic variant of language of standard first-order logic
  - internal representation as term *drs(Referents, Conditions)* where *Referents* is a set of quantified variables and *Conditions* a set of logical conditions for *Referents*
- Attempto Parsing Engine (APE)
  - Definite Clause Grammar enhanced with feature structures (Prolog with ProFIT)
  - implements construction and interpretation rules
  - can be called locally, via web-service, via web-interface

Attempto Tools

APE (ACE parser)

The ACE parser APE can be used through APE Webclient, or directly, via a APE Webservice.

Related documentation:

- APE Webclient Help
- APE Webservice Specification
- ACE Lexicon Specification

RACE (ACE reasoner)

The ACE reasoner RACE allows users to do deduction on ACE texts, for example consistency checks and query answering.

AceRules

AceRules is a forward-chaining rule system based on ACE, i.e. the rules and the answer sets are written in ACE. It supports multiple semantics (called 'modes'): courteous logic programs, stable models, and stable models with strong negation. There is a public webservice for AceRules. See the document AceRules Webservice for more information. Based on this webservice, there are two web interfaces: the AceRules interface as a demonstration for normal users and the AceRules technical interface for developers.

AceWiki

AceWiki is a semantic wiki using ACE. Unlike most other semantic wikis, the semantics are contained directly in the article texts and not in some form of annotations.

ACE View

ACE View is an ontology and rule editor that uses ACE in order to create, view and edit OWL 2 ontologies and SWRL rulesets.

OWL verbalizer

OWL verbalizer converts OWL 2 ontologies (expressed in XML Serialization) into ACE.

ACE Editor

The ACE Editor demonstrates how editing of ACE texts can be done in a convenient way. The ACE Editor is not a finished tool but rather a general basis to create domain-specific tools on top of it.

Externally developed tools

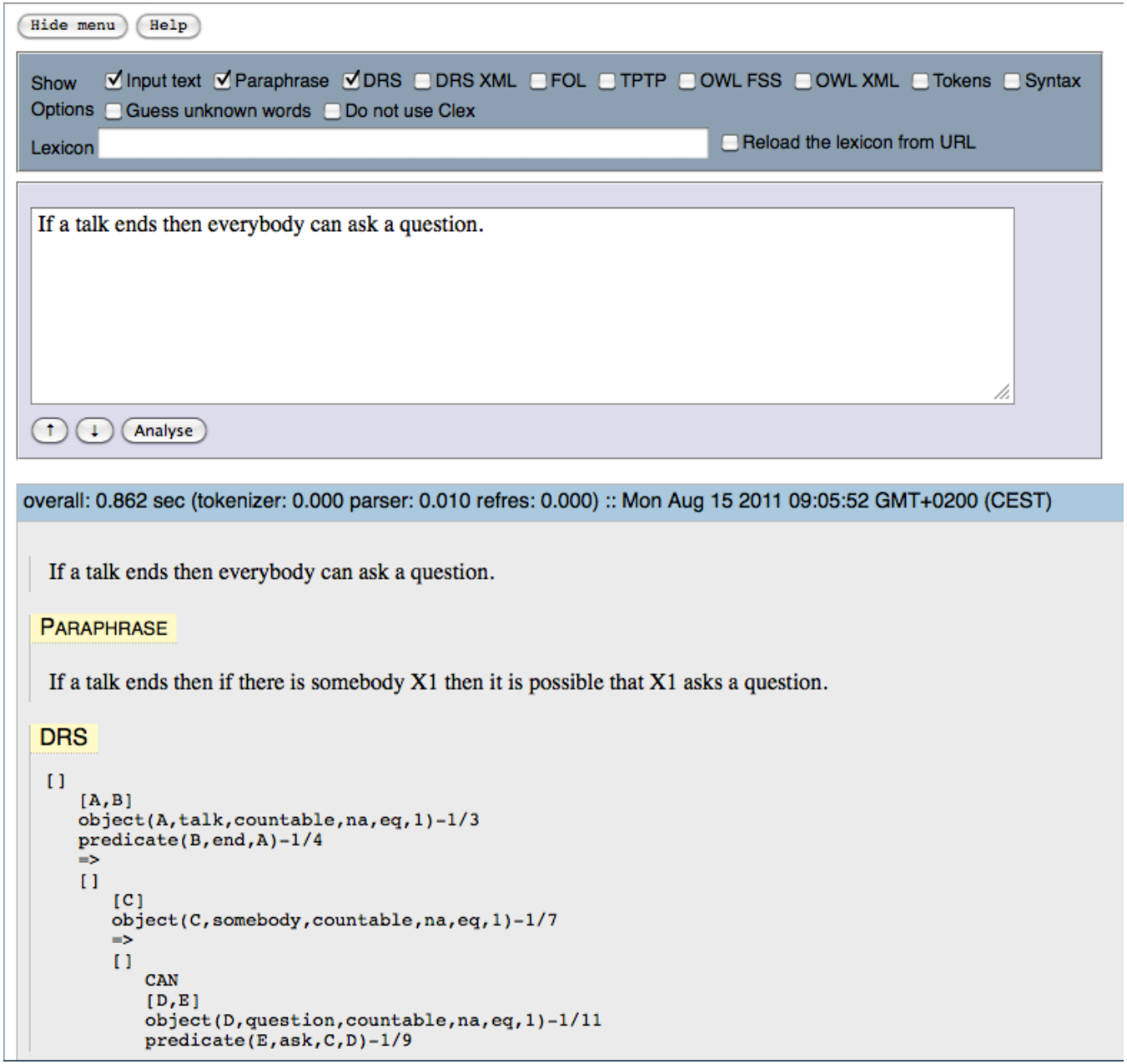
David Hirtle (University of Waterloo) developed TRANSLATOR that converts an ACE text into RuleML.

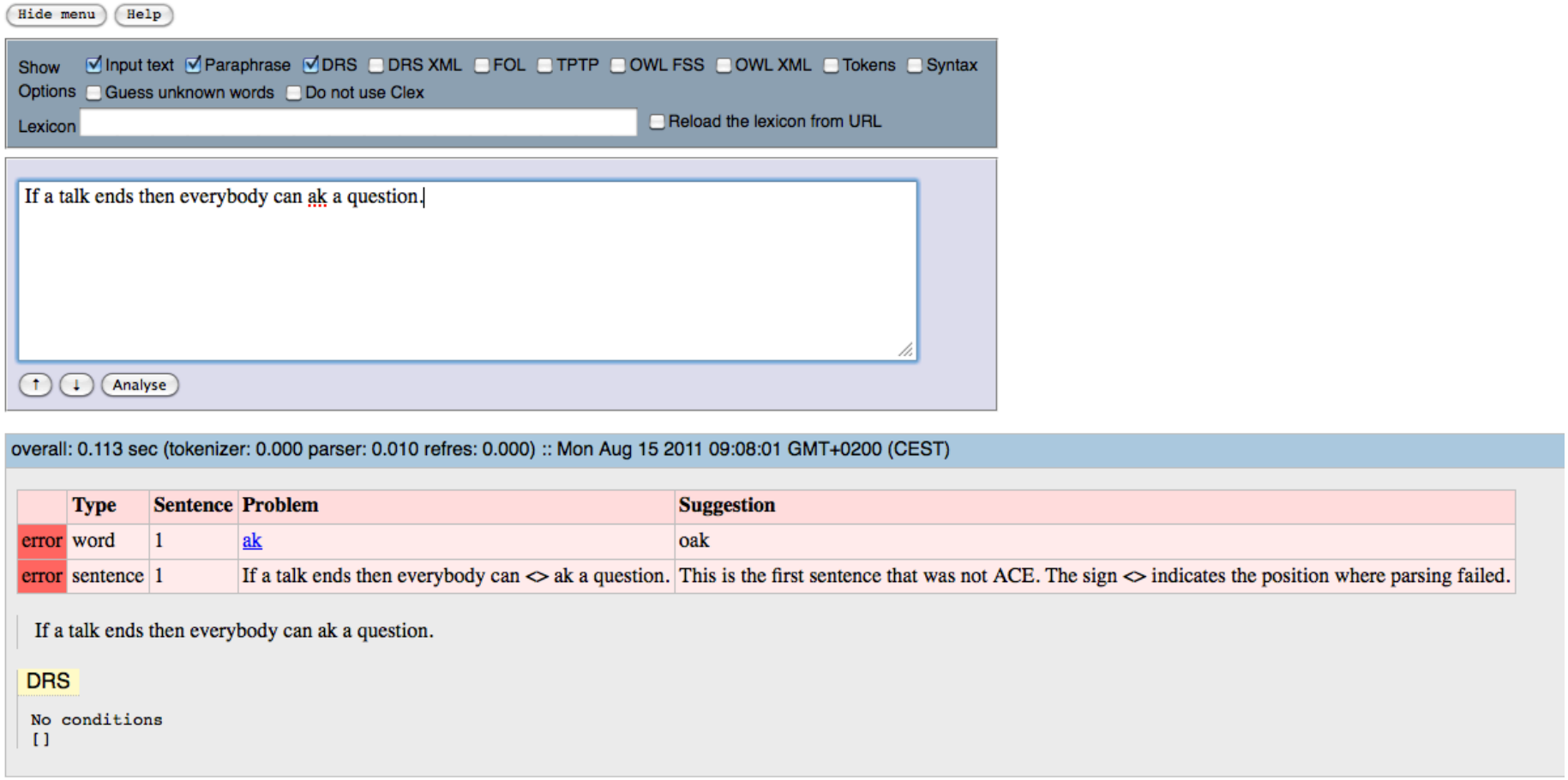
Juri Luca De Coi (University of Hannover and University of Bologna) translates policy rules expressed in ACE into the Protune policy language and has been developing several tools for this purpose.

Geoff Sutcliffe (University of Miami) added ACE input to his TPTP tools.

Attempto Parsing Engine (APE)

- for syntactically correct ACE texts outputs an analysis of the text
  - DRS
  - tokens
  - syntax trees
  - paraphrase in ACE
  - various logical translations of the DRS
- for erroneous ACE texts
  - detects syntactic errors and unknown words in the text
  - generates error and warning messages indicating the location and the possible causes of the errors, and suggesting remedies


Hide menu
Help
Show
Input text
Paraphrase
DRS
DRS XML
FOL
TPTP
OWL FSS
OWL XML
Tokens
Syntax
Options
Guess unknown words
Do not use Clex
Lexicon
Reload the lexicon from URL
If a talk ends then everybody can ask a question.
Analyse
overall: 0.862 sec (tokenizer: 0.000 parser: 0.010 refres: 0.000) :: Mon Aug 15 2011 09:05:52 GMT+0200 (CEST)
If a talk ends then everybody can ask a question.
PARAPHRASE
If a talk ends then if there is somebody X1 then it is possible that X1 asks a question.
DRS
[]
[A,B]
object(A,talk,countable,na,eq,1)-1/3
predicate(B,end,A)-1/4
=>
[]
[C]
object(C,somebody,countable,na,eq,1)-1/7
=>
[]
CAN
[D,E]
object(D,question,countable,na,eq,1)-1/11
predicate(E,ask,C,D)-1/9


Hide menu
Help
Show Input text Paraphrase DRS DRS XML FOL TPTP OWL FSS OWL XML Tokens Syntax
Options Guess unknown words Do not use Clex
Lexicon
Reload the lexicon from URL
If a talk ends then everybody can ak a question.
Analyse
overall: 0.113 sec (tokenizer: 0.000 parser: 0.010 refres: 0.000) :: Mon Aug 15 2011 09:08:01 GMT+0200 (CEST)
Type Sentence Problem Suggestion
error word 1 ak oak
error sentence 1 If a talk ends then everybody can <> ak a question. This is the first sentence that was not ACE. The sign <> indicates the position where parsing failed.
If a talk ends then everybody can ak a question.
DRS
No conditions
[]

Example DRS Representation
(Pretty-Printed APE Output)

Every company that buys at least 2 standard machines gets a discount.

PARAPHRASE

If a company X1 buys at least 2 standard machines then the company X1 gets a discount.

DRS

```
[]
   [A,B,C]
   object(A,company,countable,na,eq,1)-1/2
   object(B,machine,countable,na,geq,2)-1/9
   property(B,standard,pos)-1/8
   predicate(C,buy,A,B)-1/4
   =>
   [D,E]
   object(D,discount,countable,na,eq,1)-1/12
   predicate(E,get,A,D)-1/10
```

Pretty Printed Example DRS

Every company that buys at least 2 standard machines gets a discount.

```
[]
   [A, B, C]
   object(A, company, countable, na, eq, 1)-1/2
   object(B, machine, countable, na, geq, 2)-1/9
   property(B, standard, pos)-1/8
   predicate(C, buy, A, B)-1/4
   =>
   [D, E]
   object(D, discount, countable, na, eq, 1)-1/12
   predicate(E, get, A, D)-1/10
```

Properties of DRS Representation

Only Predefined Relation Symbols

Every company that buys at least 2 standard machines gets a discount.

```
[]
  [A, B, C]
  object(A, company, countable, na, eq, 1)-1/2
  object(B, machine, countable, na, geq, 2)-1/9
  property(B, standard, pos)-1/8
  predicate(C, buy, A, B)-1/4
  =>
  [D, E]
  object(D, discount, countable, na, eq, 1)-1/12
  predicate(E, get, A, D)-1/10
```

Properties of DRS Representation

Predicates as Arguments

Every company that buys at least 2 standard machines gets a discount.

[]
  [A, B, C]
  object(A, company, countable, na, eq, 1)-1/2
  object(B, machine, countable, na, geq, 2)-1/9
  property(B, standard, pos)-1/8
  predicate(C, buy, A, B)-1/4
  =>
  [D, E]
  object(D, discount, countable, na, eq, 1)-1/12
  predicate(E, get, A, D)-1/10

Properties of DRS Representation

Quantity Information

Every company that buys at least 2 standard machines gets a discount.

[]
  [A, B, C]
  object(A, company, countable, na, eq, 1)-1/2
  object(B, machine, countable, na, geq, 2)-1/9
  property(B, standard, pos)-1/8
  predicate(C, buy, A, B)-1/4
  =>
  [D, E]
  object(D, discount, countable, na, eq, 1)-1/12
  predicate(E, get, A, D)-1/10

Properties of DRS Representation

Sentence/Token Indices

Every company that buys at least 2 standard machines gets a discount.

[]
  [A, B, C]
  object(A, company, countable, na, eq, 1)-1/2
  object(B, machine, countable, na, geq, 2)-1/9
  property(B, standard, pos)-1/8
  predicate(C, buy, A, B)-1/4
  =>
  [D, E]
  object(D, discount, countable, na, eq, 1)-1/12
  predicate(E, get, A, D)-1/10

Attempto Tools

APE (ACE parser)

The ACE parser APE can be used through APE Webclient, or directly, via a APE Webservice.

Related documentation:

- APE Webclient Help
- APE Webservice Specification
- ACE Lexicon Specification

RACE (ACE reasoner)

The ACE reasoner RACE allows users to do deduction on ACE texts, for example consistency checks and query answering.

AceRules

AceRules is a forward-chaining rule system based on ACE, i.e. the rules and the answer sets are written in ACE. It supports multiple semantics (called 'modes'): courteous logic programs, stable models, and stable models with strong negation. There is a public webservice for AceRules. See the document AceRules Webservice for more information. Based on this webservice, there are two web interfaces: the AceRules interface as a demonstration for normal users and the AceRules technical interface for developers.

AceWiki

AceWiki is a semantic wiki using ACE. Unlike most other semantic wikis, the semantics are contained directly in the article texts and not in some form of annotations.

ACE View

ACE View is an ontology and rule editor that uses ACE in order to create, view and edit OWL 2 ontologies and SWRL rulesets.

OWL verbalizer

OWL verbalizer converts OWL 2 ontologies (expressed in XML Serialization) into ACE.

ACE Editor

The ACE Editor demonstrates how editing of ACE texts can be done in a convenient way. The ACE Editor is not a finished tool but rather a general basis to create domain-specific tools on top of it.

Externally developed tools

David Hirtle (University of Waterloo) developed TRANSLATOR that converts an ACE text into RuleML.

Juri Luca De Coi (University of Hannover and University of Bologna) translates policy rules expressed in ACE into the Protune policy language and has been developing several tools for this purpose.

Geoff Sutcliffe (University of Miami) added ACE input to his TPTP tools.

Relevant Documentation

- *ACE in a Nutshell* is a short overview of the ACE language
- *ACE Lexicon Specification* describes the allowed content words
- *ACE Construction Rules* lists the rules that determine which sentences belong to ACE
- *ACE Interpretation Rules* lists the rules that remove the ambiguity from the ACE sentences
- *ACE Troubleshooting Guide* describes how to use ACE, including how to avoid pitfalls
- *ACE Syntax Report* contains an abstract syntax of ACE
- *DRS Report* describes the DRS language

- this and more documentation is found at attempto.ifi.uzh.ch/site/docs/

Some Applications of ACE

- *specifications that we developed*: automated teller machine, Kemmerer's library data base, Schubert's Steamroller, data base integrity constraints, Kowalski's subway regulations etc.
- *natural language interfaces*: model generator EP Tableaux (Munich), FLUX agent/robot control (Dresden), MIT's process query language (Zurich), RuleML (New Brunswick), TPTP (Miami)
- *medicine*: clinical reports (Uppsala), clinical practice guidelines (Yale)
- *semantic web*: business & policy rules, translation into and from web-languages, protein ontology (EU Network of Excellence REWERSE)
- *documentation*: annotations of web-pages in CNL (Macquarie)
- *future*: MOLTO extended

Other Controlled Languages

- http://en.wikipedia.org/wiki/Controlled_natural_language
- https://sites.google.com/site/controllednaturallanguage/
- research
  - Schwitter (Macquarie): Processable English (PENG)
  - Sowa (VivoMind): Common Logic Controlled English (CLCE)
  - Pratt-Hartmann (Manchester): E2V
  - Dolbear et al. (Ordnance Survey): Rabbit
  - ...
- industry
  - Clark et al (Boeing): Computer-Processable Language (CPL)
  - ...

To Take Home

- ACE is (a superset of) a first-order logic language with the syntax of a subset of English – thus human *and* machine understandable
- ACE does not introduce a division of labour between people who understand formal languages and those who don't – and thus eliminates a major communication problem
- ACE is ontologically neutral, i.e. does not require a priori world knowledge or a domain ontology – though both can be expressed in ACE
- ACE is neutral with regard to particular applications or methods
- over time ACE has been extended by many language features – mostly motivated by applications and by projects – but it remains a knowledge representation language and is not an attempt to ultimately cover full natural language

Attempto Web-site

http://attempto.ifi.uzh.ch/site/