The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0

Technical Report ifi-2004.03

Stefan Hoefler

Department of Informatics, University of Zurich, Switzerland.
shoefler@ifi.unizh.ch

Abstract

Attempto Controlled English (ACE) is a controlled natural language specifically designed for requirements specifications and knowledge representation. ACE is a subset of English with a restricted grammar constrained by a small set of construction and interpretation rules.

In this report, we describe the syntax of ACE, version 4.0, and present an abstract grammar for it in the form of phrase structure rules enhanced by features. The report aims at the implementer who is interested in the linguistic background of ACE. It underlies the current version of the Attempto parser. It is not intended to be a reference manual for the user of ACE.

1 Introduction

1.1 Attempto Controlled English

Attempto Controlled English (ACE) is a language specifically designed for requirements specifications and knowledge representation. ACE is a controlled natural language, i.e. a subset of English with a restricted grammar in the form of a small set of construction and interpretation rules. This means that all ACE sentences are accepted English, but that not all English sentences are allowed in ACE. The restriction of full natural language to a controlled subset is essential for ACE to be suitable for specification and knowledge representation purposes. In brief, ACE allows domain specialists to express knowledge in familiar natural language and to combine this with the rigor of formal specification languages.

Version 3 of ACE was restricted to singular objects only [6, 7]. As with its fourth version, ACE has been extended to plural [10, 11], and (automated) reasoning in ACE has become a prominent line of investigation within the Attempto project [5].

1.2 An abstract grammar for ACE

In this report, we define the syntax of version 4.0 of ACE and present its grammar in the form of phrase structure rules enhanced by features. Note that what we provide is an *abstract* grammar, and as such it does not claim to be formally complete and exhaustively explicit. We have decided on an abstraction level that enables the reader to grasp a concise view of the syntax of the language.

This report aims at the implementer who is interested in the linguistic background of ACE. It underlies the current version of the Attempto parser. It is not intended to be a reference manual for the user of ACE.

We have enhanced the phrase structure rules by features whenever special restrictions are applied to the categories stated in a rule. Other mechanisms that belong to a grammar of natural English have been left out on the chosen level of abstraction: we do not explicitly represent agreement constraints or the proliferation of certain features. Where we felt that it contributes to a proper understanding of the respective syntactic construct, information that is below the abstraction level of our grammar rules is provided in the comments that complement each set of rules.

The Attempto Parsing Engine (APE) contains an implementation of the abstract grammar presented in this report. The core of the parser is a definite clause grammar (DCG) written in Prolog. This DCG is augmented with ProFIT typed feature structures [3, 2]. Since definite clause grammars run into problems when confronted with left-recursion, we have eliminated all incidences of left-recursion in the abstract grammar by introducing intermediate categories as suggested in [1]. The actual implementation, however, uses a more efficient representation for originally left-recursive rules.

The Attempto parser translates ACE sentences into extended discourse representation structures, a notational variant of first-order logic originally introduced in [8, 9]. The form of extended discourse representation structures derived from ACE texts is described in [4].

1.3 Notational conventions

We use some notational conventions in our abstract grammar. The feature values of a constituent are attached to it in square brackets. Required values take a plus sign (+), prohibited values are marked with a minus sign (–). Features which are not discriminatory in the respective rule are not made explicit. Alternatives are separated by a vertical bar (|) and optional elements are put into parentheses.

Gaps left behind by moved constituents are indicated by the SLASH feature (/). A minus sign after the slash (/–) indicates that the respective category is not allowed to contain a gap.

Feature values that are not indicated explicitly in the abstract grammar are not restricted, or they are passed on by feature percolation.

For many rules we provide an example. Parts of the example that are not covered by the respective rule, but are added to complete a sentence or to make the example more illustrative, are put into square brackets. Curly braces are used to clarify the syntactic interpretation of a sentence.

1.4 Feature declaration

We have made explicit use of a limited set of features in the abstract grammar presented in this report. Nominal constituents are marked as nominative (+NOM) or oblique (–NOM), and they can be either pronominal (+PRO) or not (–PRO). Verb forms can be either finite (–INF) or infinite (+INF). The only infinite verb form that exists in ACE is the bare infinitive. If a verbal constituent has the copula as its head, it is marked +COPULA, and if it must not have a copula head –COPULA respectively. Prepositions can be specifically marked as local (+LOC) or temporal (+TEMP). Noun phrases can either be negative (+NEG) or non-negative (–NEG) depending their determiner.

Number, quantification and the WH value are indicated by non-boolean features. A noun phrase can be definite (+DEF), existentially quantified (+EXISTS) or universally quantified (+FORALL). Likewise, a non-universally quantified noun phrase, i.e. one that is either definite or existentially quantified, is marked as –FORALL. Many constituents can have a WH value. Its range includes interrogative (+Q), relative (+R), either (+WH), neither (–WH), non-interrogative (–Q) and non-relative (–R). The number of a constituent is indicated as singular (+SG), mass (+MASS) or plural (+PL) and the respective complementary values.

2 Grammar rules

2.1 ACE texts

An ACE text can either be a specification, a query or an instruction (1). You can also say that ACE is either in the declarative, the interrogative, or the imperative mode. These different modes of ACE texts cannot be mixed since they have different functions: In the declarative mode, rules and facts are added to a knowledge base. Queries ask for specific information from a knowledge base or try to prove a theorem over it. The imperative mode can be regarded as a special type of declarative mode that is used to give instructions. This report focuses on the declarative and interrogative modes only.

(1)
```
ACEText -->
        Specification | Query | Instruction
```

2.2 Specifications

ACE texts entered in the declarative mode are called specifications. Specifications consist of a sentence coordination followed by a period and optionally one ore more subsequent specifications (2).

(2)
```
Specification -->
        SentenceCoord Period (Specification)
```

2.2.1 Sentence coordination

Sentences[1] can be coordinated by *and* and *or*. *And* refers to the logical conjunction, while *or* denotes the logical disjunction. The logical conjunction has a higher precedence than the disjunction. Both connectors are right-associative. The expression

A or B and C or D

is therefore ordered like

$A \vee ((B \wedge C) \vee D)$

To enable more combinations, we have introduced comma-*and* and comma-*or*. These expressions reverse the order of precedence. To achieve the order

$A \vee (B \wedge (C \vee D))$

we can write

A, or B, and C or D

A sentence coordination in general consists of a sentence coordination of a lower level (thus ensuring right-associativity) optionally followed by the respective connector and a sentence coordination of the same level (3–6).

(3)
```
SentenceCoord -->
        SentenceCoord_1 (Comma or SentenceCoord)
```

(4)
```
SentenceCoord_1 -->
        SentenceCoord_2 (Comma and SentenceCoord_1)
```

(5)
```
SentenceCoord_2 -->
        SentenceCoord_3 (or SentenceCoord_2)
```

[1]Note that we use the term ‘sentence’ only in the declarative mode. In the interrogative mode we use the term ‘question’ instead.

(6) `SentenceCoord_3 -->`

```
        TopicalizedSentence (and SentenceCoord_3)
```

Example: *A customer enters a card and a clerk enters a code and for every code that the customer enters the card is valid [.]*

2.2.2 Topicalized sentences

To avoid scope ambiguities ACE uses the 'principle of surface order' which makes the scope of a quantifier uniquely predictable from the quantifier's position in the sentence. This interpretation principle says that the relative scope of a quantifier corresponds to its surface position. The scope opens at the textual position of the quantified noun phrase and extends to the end of the sentence. If sentences are coordinated, the scope of a quantifier extends only to the end of the sentence conjunct/disjunct containing the quantifier [7].

The sample sentence commonly used to illustrate scope amguities

Every man loves a woman.

is therefore unambiguously interpreted in ACE as

$$\forall x : (man(x) \rightarrow \exists y : (woman(y) \land loves(x, y)))$$

We may, however, want to express the reverse interpretation

$$\exists y : (woman(y) \land \forall x : (man(x) \rightarrow loves(x, y)))$$

To achieve the latter interpretation, we have introduced two ACE constructions that allow the user to tropicalize a quantifier (i.e. move it to the front of the sentence) and give it a wider scope. The global quantifiers *there is/there are* (10) and *for every/for each* (11, 12) expand their scope over the whole sentence. In ACE, the above interpretation can thus be expressed as

There is a woman such that every man loves her.

or

There is a woman that every man loves.

The two readings of the sentence

A man loves every woman.

which are

$\exists x : (man(x) \land \forall y : (woman(y) \rightarrow loves(x, y)))$ and
$\forall y : (woman(y) \rightarrow \exists x : (man(x) \land loves(x, y)))$

can be made explicit in ACE as

There is a man who loves every woman.[2] and
For every woman a man loves her.

A topicalized sentence can start with an existential topic (7) or a universal topic (8). It needs, however, not be topicalized at all but can just be an ordinary composite sentence (9)

(7) `TopicalizedSentence -->`

```
        ExistentialTopic (such that SentenceCoord)
```

Example: *There is a card such that the code of the card is valid [.]*

[2]This corresponds to the ACE reading of *A man loves every woman.*

(8) `TopicalizedSentence -->`
```
        UniversalTopic SentenceCoord
```
Example: *For every code there is a card such that the code belongs to it [.]*

(9) `TopicalizedSentence -->`
```
        CompositeSentence
```

Topics consist of a global quantifier enhanced by a noun phrase coordination (i.e. a noun phrase or a coordination of noun phrases) and a VP coordination (a verb phrase or a coordination of verb phrases) or by an N' respectively. An existential topic takes the existential global quantifier *there is/there are* and an NP coordination that must not be in an oblique case (+NOM), nor universally quantified (–FORALL), nor interrogative or relative (–WH).

(10) `ExistentialTopic -->`
```
        ExistentialGlobalQuantor NPCoord[+NOM,-FORALL,-WH]/-
```
Examples: *There is a card [which is valid.] There are a card and a code [such that the code is the code of the card.]*

With these restrictions, we prohibit sentences like

* *There is him.*
* *There is every customer.*
* *There is who?*
* *There is which card?*

A universal global quantifier, on the other hand, is only followed by an N' and not by a complete NP (**for every some code*). This N' must be in the nominative case and it must not be plural (**for every codes*). Since *every* cannot be used with mass nouns (**every money*), we use *every* in combination with countable nouns and *all* in combination with mass nouns in ACE. The agreement between the universal global quantifier is not made explicit in rule (11).

(11) `UniversalTopic -->`
```
        UniversalGlobalQuantor N'[+NOM,-PL]
```
Examples: *For every card [there is a code.] For all money there is a bank.*

A universal topic can also start with a distributive global quantifier *for each of.* The distributive global quantifier must be followed by a complete plural NP coordination in nominative case (12).

(12) `UniversalTopic -->`
```
        DistributiveGlobalQuantor NPCoord[+NOM,+PL]/-
```
Examples: *For each of the customers [a clerk enters a code.] For each of a customer and a clerk [some code is valid.]*

2.2.3 Composite sentences

A composite sentence can either be a conditional sentence (13), a negated sentence (14) or just an ordinary simple sentence (15).

(13) `CompositeSentence -->`
```
        ConditionalSentence
```

(14) `CompositeSentence -->`
```
        NegatedSentence
```

(15) `CompositeSentence -->`
```
          Sentence
```

A conditional sentence consists of an antecedent, introduced by *if*, and a consequent, introduced by *then* (16). Both, the antecedent and the consequent must be complete sentence coordinations.

(16) `ConditionalSentence -->`
```
          if SentenceCoord then SentenceCoord
```

Example: *If a card is valid then its code is valid and a customer enters the card [.]*

In ACE, sentence negation is introduced through the fixed expression *it is not the case that.* Thus, a negated sentence consists of this expression followed by a sentence coordination.

(17) `NegatedSentence -->`
```
          it is not the case that SentenceCoord
```

Example: *It is not the case that a customer enters a card [.]*

A simple declarative sentence consists of an NP coordination. The NP coordination is its subject, and it has to be in nominative case and must not be relative or interrogative (–WH). The VP coordination too must not contain any interrogative or relative arguments or modifiers, and its verb form must be finite (18).

(18) `Sentence -->`
```
          NPCoord[+NOM,-WH]/- VPCoord[-WH,-INF]/-
```

Example: *A customer enters a green card into a slot [.]*

2.3 Queries

A query consists of a topicalized question followed by a question mark (19). Optionally, it can be preceded by a specification.

(19) `Query -->`
```
          (Specification) TopicalizedQuestion QuestionMark
```

2.3.1 Topicalized questions

A topicalized question can start with an existential question topic (23–24), optionally continued by *such that* followed by an ordinary declarative sentence coordination (20). The existential question topic in example (20) comprises *is there a card.*

(20) `TopicalizedQuestion -->`
```
          ExistentialQuestionTopic (such that SentenceCoordination)
```

Example: *Is there a card such that the code of the card is valid [?]*

Likewise, a topicalized question can start with an universal topic followed by another topicalized question (21). Note that this is a recursive rule. In example (21), the universal topic is *for every card.*

(21) `TopicalizedQuestion -->`
```
          UniversalTopic TopicalizedQuestion
```

Example: *For every card does some customer own it [?]*

Last, a topicalized question need not have a topic at all but can just be an ordinary question (22).

(22) `TopicalizedQuestion -->`
```
          Question
```

An existential question topic can either consist of an interrogative NP followed by the existential global question quantifier *is there* (23), or it starts with the existential global question quantifier followed by an NP (24). Like in its declarative counterpart (10), this NP must not stand in an oblique case (+NOM) nor must it be universally quantified (–FORALL), relative or interrogative (–WH). In this way we prevent sentences like

> **Is there every card?*
> **Is there which card?*

Furthermore, an NP containing an interrogative element must not be coordinated, preventing sentences like **A card and what are valid?* or * *Who and who enters a card?*

(23) `ExistentialQuestionTopic -->`
```
          NP[+NOM,+Q] ExistentialGlobalQuestionQuantor
```
Example: *Which code is there [?]*

(24) `ExstentialQuestionTopic -->`
```
          ExistentialGlobalQuestionQuantor NPCoord[+NOM,-FORALL,-WH]/-
```
Examples: *Is there a card [such that its code is valid?] Are there a code and a card [?]*

2.3.2 Questions

There are two types of questions in ACE: Yes/no-questions (26–27) and WH-questions (28–34).

(25) `Question -->`
```
          YesNoQuestion | WhQuestion
```

Yes/no-questions either start with an auxiliary (26) or the copula *is/are* (27). Since we do not have any complex tenses or continuous forms in ACE, *do* and *does* are the only two auxiliary forms (105). By 'copula' we always denote the forms of *be* (107). Note that the VP of example (27) consists of a gap plus *valid*. The gap has been left behind by the copula, which has been moved to the front of the sentence. This is indicated by the SLASH feature.

(26) `YesNoQuestion -->`
```
          Aux NPCoord[-WH,+NOM]/- VPCoord[+INF,-COPULA,-WH]/-
```
Example: *Does a customer enter a code [?]*

(27) `YesNoQuestion -->`
```
          Copula/- NPCoord[-WH,+NOM]/- VP[+COPULA,-WH]/Copula
```
Example: *Is a card valid [?]*

WH-questions can take various forms (28–34). The only WH-question without inversion occurs when one asks for the subject of the sentence (28). In WH-questions that ask for an object or an adjunct, the interrogative constituent is either followed by an auxiliary (29, 31, 33) or by the copula (30, 32, 34). In these questions, the interrogative constituent can be an NP (29, 30), a prepositional phrase (31, 32) or an adverb coordination (33, 34). The verbal phrases of these questions contain a gap for the constituent that was moved to the front of the sentence (i.e. the constituent that is asked for). The verb of these verbal phrases stands in the infinitive form. In copula sentences (30, 32, 34), the copula has also been moved to the front and is therefore missing within the VP of that sentence. Interrogative nominal phrases are never coordinations (hence NP instead of NPCoord).

(28) `WhQuestion -->`
```
          NP[+Q,+NOM] VP/-
```
Example: *Which customer / Who enters a card [?]*

(29) `WhQuestion -->`

```
        NP[+Q,-NOM] Aux NPCoord[+NOM]/- VP[-COPULA,+INF]/NPCoord
```

Example: *Which code / What does a customer enter [?]*

(30) `WhQuestion -->`

```
        NP[+Q,-NOM] Copula/- NPCoord[+NOM]/- VP[+COPULA]/NPCoord,Copula
```

Example: *What is a customer interested in [?]*

(31) `WhQuestion -->`

```
        PP[+Q]/- Aux NPCoord[+NOM]/- VP[-COPULA,+INF]/PP
```

Example: *Into what does a customer enter a card [?]*

(32) `WhQuestion -->`

```
        PP[+Q]/- Copula/- NPCoord[+NOM]/- VP[+COPULA]/PP,Copula
```

Example: *In what is a customer interested [?]*

(33) `WhQuestion -->`

```
        AdverbCoord[+Q]/- Aux NPCoord[+NOM]/- VP[-COPULA,+INF]/AdverbCoord
```

Example: *Where does a customer enter a card [?]*

(34) `WhQuestion -->`

```
        AdverbCoord[+Q]/- Copula/- NPCoord[+NOM]/- VP[+COPULA]/AdverbCoord,Copula
```

Example: *When is a card valid [?]*

2.4 Verb phrases

2.4.1 Verb phrase coordination

Verb phrases can be either coordinated (35) or single (36). For reasons of simplicity, we do not represent the distinct precedence of *and* and *or* (as described in 2.2.1) in rule (35). ‘Coord’ denotes either of them.

A verb phrase coordination does automatically take the feature value –COPULA. And, of course, both coordinated verb phrases have to agree in number and verb form (either finite or infinite). As mentioned in the introduction of this report, feature percolation and agreement is not made explicit in our abstract grammar.

(35) `VPCoord[-COPULA] -->`

```
        VP Coord VPCoord
```

Example: *[A customer] enters a card and waits [.]*

(36) `VPCoord -->`

```
        VP
```

Usually, in an declarative context, a verb phrase consists of a V’ only (37). However, if the verb phrase is negated, it is preceded by an auxiliary (or the copula respectively) and the negation *not* (38, 39). Note that the copula has been moved out of the VP in (39). There is also a variant without auxiliary for questions (40). In all cases, the verb form has to be infinite; the VP as a whole, however, is only infinite in rule (40).

(37) `VP -->`

```
        V’
```

(38) `VP[-INF,-COPULA] -->`
```
        Aux not V'[+INF,-COPULA]
```
Example: *[A customer] does not enter a card into a slot [.]*

(39) `VP[-INF,+COPULA] -->`
```
        Copula/- not V'[+COPULA]/Copula
```
Example: *[A card] is not valid [.]*

(40) `VP[+INF,-COPULA] -->`
```
        not V'[+INF,-COPULA]
```
Example: *[Who does] not enter a card into a slot [?]*

A verb (and its complements) can optionally be modified by an adverb coordination preceding the verb and/or by verbal modifiers following the verb and its complements (41). Such verbal modifiers (adjuncts) can be adverb coordinations, prepositional phrases or adverbial prepositional phrases (42). Note that the Kleene star is used in rule (41). It indicates that none, one or several verbal modifiers can follow the verb and its complements.

(41) `V' -->`
```
        (AdverbCoord/-) ComplV VModifier*
```
Example: *[A customer] quickly and hastily enters a card manually into a slot in a bank on Tuesday during afternoon.*

(42) `VModifier -->`
```
        AdverbCoord | PP | AdverbialPP
```

2.4.2 Subcategorization of verbs

In ACE, verbs are subcategorized into intransitive verbs, transitive verbs, ditransitive verbs and the copula. For the first three subcategories, separate rules are needed for phrasal and non-phrasal verbs.

Intransitive verbs take no complement (43, 44). Intransitive phrasal verbs are accompanied by a ‘phrasal particle’ (44).

(43) `ComplV -->`
```
        IntransitiveV
```
Example: *[A customer] waits [.]*

(44) `ComplV -->`
```
        PhrasalIntransitiveV PhrasalParticle
```
Example: *[A clerk] gives in [.]*

Transitive verbs take one complement (45–47). Transitive phrasal verbs can have two patterns: the phrasal particle can stand either between the verb form and the complement (46) or after the complement (47). In the former, the complement cannot be a pronoun:

A clerk gives away a code. (46)
**A clerk gives away it.* (46)
A clerk gives a code away. (47)
A clerk gives it away. (47)

(45) `ComplV -->`
```
        TransitiveV Compl
```
Example: *[A customer] inserts a card [.]*

(46) `ComplV -->`
```
        PhrasalTransitiveV PhrasalParticle Compl[-PRO]
```
Example: *[A clerk] gives away a code [.]*

(47) `ComplV -->`
```
        PhrasalTransitiveV Compl PhrasalParticle
```
Example: *[A clerk] gives a code away [.]*

Ditransitive verbs take two complements (48, 49). With ditransitive phrasal verbs, the phrasal particle stands between the two complements (49).

(48) `ComplV -->`
```
        DitransitiveV Compl Compl
```
Example: *[A clerk] gives a card to a customer [.]*

(49) `ComplV -->`
```
        PhrasalDitransitiveV Compl PhrasalParticle Compl
```
Example: *[A clerk] puts an error down to a customer [.]*

The copula always takes a complement (50). This complement underlies constraints which are different from the ones that apply to complements of non-copula verbs: refer to rules (51) and (52).

(50) `ComplV -->`
```
        Copula CopulaCompl
```
Example: *[A code] is valid [.]*

Non-copula verbs can take NP coordinations and prepositional phrases (both have to be non-relative) as complements (51). NP coordinations have to be in the oblique case (*Mary sees him*) and must not be nominative (**Mary sees he*).

(51) `Compl -->`
```
        NPCoord[-R,-NOM] | PP[-R]
```
Examples: *[A customer enters] a card [.] [A clerk gives] a code to a customer [.]*

The copula can take adjective phrase coordinations (*is taller than his brother*), noun phrase coordinations that are non-relative and non-pronominal (*is a customer*), or prepositional phrases (*is in the garden*) as its complement (52).

(52) `CopulaCompl -->`
```
        APCoord | NPCoord[-PRO,-R,-NOM] | PP[-R]
```
Examples: *[A card is] valid [.] [John is] a customer [.] [A card is] in a slot [.]*

2.5 Noun phrases

A noun phrase coordination can consist of the intermediate category that we call 'unmarked NP coordination' either preceded (53) or not by the distributive marker *each of* (54). A distributive noun phrase coordination is always singular (+SG) as a whole, but its distributive marker can only precede a non-negative unmarked NP coordination that is plural and non-universally quantified (53). Noun phrases like **each of no customer*, **each of a customer* or **each of every customer* do not make sense. We use the distributive marker *each of*, as well as its global counterpart *for each of* (12), to resolve plural ambiguities as described in [4, 10].

Unmarked NP coordinations that are not preceded by a distributive marker have no restrictive feature values (54).

(53)
```
NPCoord[+SG]/- -->
          DistributiveMarker UnmarkedNPCoord[-NOM,-FORALL,-NEG,+PL]
```
Examples: *each of some customers, each of them*

(54)
```
NPCoord/- -->
          UnmarkedNPCoord
```
Examples: *some customer(s), he, they, a card that is valid*

Finally, a noun phrase coordination can also consist of a measurement noun phrase (55), or be a gap (56) if it has been moved to the front of the sentence in a question.

(55)
```
NPCoord/- -->
          MeasurementNP
```
Example: *more than 50 piles of cards*

(56)
```
NPCoord/NPCoord -->
          []
```

If an unmarked NP coordination is an actual coordination of noun phrases, then it is automatically assigned plural number (+PL). Interrogative, relative, negative or universally quantified NPs cannot be coordinated in ACE; hence the –WH, –NEG and –FORALL restriction in the LHS and RHS of rule (57). Simple (not coordinated) noun phrases do not underlie such restrictions (58).

(57)
```
UnmarkedNPCoord[+PL,-FORALL,-NEG,-WH] -->
          NP[-FORALL,-NEG,-WH] and UnmarkedNPCoord[-FORALL,-NEG,-WH]
```
Example: *a card and a code and a customer*

(58)
```
UnmarkedNPCoord -->
          NP
```

In general, a noun phrase consists of a specifier followed by an N' (59–61). Rule (59) describes the most general type of noun phrases. It includes purely declarative noun phrases like *some code* and noun phrases with a relative or interrogative specifier like *whose code* or *which code.* In none of these noun phrases, the N' can be interrogative or relative (hence the –WH feature). Note that the rule does not show the percolation of the WH feature and neither the agreement constraints between the specifier and the N'.

(59)
```
NP[-PRO] -->
          Specifier N'[-WH]
```
Examples: *a card X, some card, the card that is valid, the card of John, every card, whose card, which card*

If N' contains a relative element, the specifier must not be interrogative or relative (60). If N' contains an interrogative element, the specifier can contain an interrogative element too, but it cannot be relative (61).

(60)
```
NP[+R,-PRO] -->
          Specifier[-WH] N'[+R]
```
Example: *[a customer] the card of whom [is valid enters a code.]*

(61) `NP[+Q,-PRO] -->`

```
          Specifier[-R] N'[+Q]
```

Example: *the card of which customer, the card of whom, which card of which customer*

Pronouns (62), proper names and variables (63) are also self-contained noun phrases. Refer to rules (130–149) in the appendix for a list of ACE pronouns.

Proper names always start with a capital letter. They can be used anaphorically, and they are in the lexicon. Variables also start with a capital letter, but they are not in the lexicon. Variables can be used anaphorically when used in an apposition or as a self-contained NP. They must, however, first be introduced in an apposition (refer to section 2.7).

(62) `NP[+PRO] -->`

```
          Pronoun
```

(63) `NP[-PRO,-WH] -->`

```
          ProperName | Variable
```

Nouns can be optionally modified by a preceding adjective coordination and/or, following the noun, an apposition coordination, an of-PP and/or a relative clause coordination (64). The order of the modifiers after the noun is fixed. Note that in example (64), the relative clause *which is valid* cannot refer to *a customer*, because it does not agree with its gender (human vs. non-human), and can thus only refer to *card*.

(64) `N' -->`

```
          (AdjectiveCoord) N (ApposCoord) (of-PP) (RelativeClauseCoord)
```

Example: *a red and blue card X of a customer which is valid*

A measurement noun phrase consists of a number phrase (*2, at least 3*) followed by a measurement noun (*kg, liters, boxes*), the preposition *of* and an N' (65). Refer to rule (67) for number phrases.

(65) `MeasurementNP -->`

```
          NumberP MeasurementNoun of N'
```

Examples: *4 kg of apples, at most 3 liters of milk*

2.6 Specifiers

In ACE, specifiers can be determiners, possessive NP coordinations or number phrases (66). A number phrase consists of a number which is optionally preceded by a generalized quantifier (67, 121).

(66) `Specifier -->`

```
          Determiner | PossessiveNPCoord | NumberP
```

Examples: *the, every, some, the clerk's, a clerk's customer's, at least 3, 5*

(67) `NumberP -->`

```
          (GeneralizedQuantor) Number
```

Examples: *3, at most 6, not less than 4*

2.7 Appositions

Appositions stand immediately after the nouns to which they are attached (64). Appositions can be coordinated (68). However, the number of coordinated appositions and the number of the noun they refer to must match exactly. Therefore, *2 codes X and Y* is a correct noun phrase in ACE, whilst **3 codes X and Y* is not.

(68) ```
ApposCoord -->
        Apposition (and ApposCoord)
```
Example: *[2 codes] X and Y [are valid.]*

An apposition can be a variable or a quoted string (69). Both, variables and quoted strings, are not in the lexicon. But unlike variables, quoted strings cannot be used anaphorically. Variables always start with an upper case letter, whilst quoted strings stand in double quotes.

(69) ```
Apposition -->
        Variable | QuotedString
```

2.8 Possessive noun phrases

The specifier of a noun phrase can be a possessive noun phrase coordination. Possessive noun phrase coordinations are either genitive noun phrase coordinations followed by a saxon genitive marker (70), or possessive pronoun coordinations (71). Saxon genitives and possessive pronouns can not be coordinated with each other in ACE: expressions like **a customer's and her card* are not allowed. On the other hand, the saxon genitive marker refers to all conjunctions of a genitive NP coordination: *a customer and a clerk's contract* denotes a contract that belongs to both the customer and the clerk.

(70) ```
PossessiveNPCoord -->
        GenitiveNPCoord SaxonGenitiveMarker
```
Example: *A customer and a clerk's [contract is valid.]*

(71) ```
PossessiveNPCoord -->
        PossessivePronounCoord
```
Examples: *his [card], their [card], his and her [card]*

Genitive noun phrases differ from ordinary NP coordinations. They cannot take distributive markers, and they cannot be modified by *of*-PPs or relative clauses, because in such cases the saxon genitive marker could refer to the wrong noun phrase.

*\{*each of the customer*\}*'s code*
*\{*a customer of a bank*\}*'s code*
*\{*a customer who enters a card*\}*'s code*

Genitive noun phrases either consist of a specifier and a genitive N' (73) or of a proper name (74). They can be coordinated with other genitive noun phrases (72).

(72) ```
GenitiveNPCoord -->
        GenitiveNP (and GenitiveNPCoord)
```

(73) ```
GenitiveNP -->
        Specifier GenitiveN'
```

(74) ```
GenitiveNP -->
        ProperName
```

The noun of a genitive N' can optionally be preceded by an adjective coordination and followed by an apposition coordination (75).
```

(75) `GenitiveN' -->`
```
            (AdjectiveCoord) N (ApposCoord)
```

Note that rule (73) introduces left-recursion: a specifier can be a possessive noun phrase coordination again. Since this is an indirect type of left-recursion, ranging over several rules (73, 66, 70), its elimination is not as straight-forward as for immediately left-recursive rules. We have replaced rules (70–74). The reformulated rules (76–82) do not reflect any linguistic evidence and are therefore presented without any further comment.

(76) `PossessiveNPCoord -->`
```
            GenitiveNPCoord | PossessivePronounCoord
```

(77) `GenitiveNPCoord -->`
```
            GenitiveSpecifier GenitiveN' GenitiveTail
```

(78) `GenitiveNPCoord -->`
```
            ProperName GenitiveTail
```

(79) `GenitiveTail -->`
```
            SaxonGenitiveTail | GenitiveCoordTail
```

(80) `SaxonGenitiveTail -->`
```
            SaxonGenitiveMarker (GenitiveN' SaxonGentiveTail)
```

(81) `GenitiveCoordTail -->`
```
            and GenitiveNPCoord
```

(82) `GenitiveSpecifier -->`
```
            Determiner | PossessivePronounCoord | Number
```

Two rules remain to complete the rule set for possessive noun phrase coordinations. The saxon genitive marker is either an apostrophe followed by an *s* as in *a customer's card* or a bare apostrophe as in *5 customers' contract* (150, appendix).

A possessive pronoun coordination consists of a pronoun which is optionally followed by *and* and another possessive pronoun coordination (83).

(83) `PossessivePronounCoord -->`
```
            PossessivePronoun (and PossessivePronounCoord)
```

2.9 Prepositional Phrases

There are three types of prepositional phrases in ACE: ordinary prepositional phrases (84), *of*-PPs (86) and adverbial prepositional phrases (87).

Ordinary prepositional phrases consist of a preposition followed by an NP coordination (84). Note that the prepositional phrase as a whole is always non-pronominal (–PRO). This is relevant for a prepositional phrase containing a pronoun as its NP which is the complement of a transitive phrasal verb. Rule (46) states that only non-pronominal complements can stand after the phrasal particle and prohibits sentences like

A customer looks up a code.
**A customer looks up it.*

If the complement is a prepositional phrase, it can stand in the end position even if its NP is a pronoun:

A boy looks up to a famous actor.
A boy looks up to him.

This means that the pronominal feature is not passed on from an NP to its dominating PP and prepositional phrases are always non-pronominal.

When a prepositional phrase has been moved to the front of a question, it leaves a gap behind (85). Note that, in an implementation, rule (85) can get into an infinite loop, if it is parsed in combination with rule (41) above and its gap feature is not properly restricted. An infinite number of empty verb modifiers (adjuncts) would be generated in this case.

(84)
```
PP[-PRO] -->
        Preposition NPCoord
```

Examples: *into a slot, to a clerk, with a card and a code*

(85)
```
PP/PP -->
        []
```

Although an ordinary prepositional phrase can, of course, take the preposition *of* if demanded, we have introduced a separate type of prepositional phrases called *of*-PP (86). These prepositional phrases do only occur as noun-modifiers (64) and have a semantic interpretation that differs from the one assigned to ordinary prepositional phrases that serve as complements or adjuncts [4].

(86)
```
of-PP[-PRO] -->
        of NPCoord/-
```

Example: *[The customer] of the bank [enters a code.]*

Finally, adverbs can also be combined with prepositions. This option is restricted to expressions of time or location: *until there*, *by then* (87). The adverb of such a prepositional phrase can also be coordinated: *until now and then.*

(87)
```
AdverbialPP -->
        Preposition[+LOC | +TEMP] AdverbCoord/-
```

Examples: *until here, by then, until now and then*

2.10 Relative Clauses

Rule (64) above defines at which position relatives clauses occur within an N'. Relative clauses can be coordinated (88).[3]

(88)
```
RelativeClauseCoord -->
        RelativeClause (Coord RelativeClauseCoord)
```

Example: *[a card] that is green and that is valid*

A relative clause can either be introduced by the complementizer *that* (89, 91) or by a relative NP coordination or PP (90, 92, 93). The constituent to which a relative clause refers can either be its subject (89–91), or it can be a complement or adjunct (92–93). If it is a complement or an adjunct, like in rules (92–93), then it has been moved from the VP of the relative clause and has left a corresponding gap, namely an NP coordination in an oblique case or a PP, which cannot be interrogative or relative. Furthermore, it is followed by the subject of the relative clause (a non-relative NP coordination in nominative case).

(89)
```
RelativeClause -->
        that VPCoord[-INF]/-
```

Example: *[A customer] that enters a card [waits.]*

(90)
```
RelativeClause -->
        NPCoord[+R,+NOM]/- VPCoord[-INF]/-
```

Example: *[A customer] the card of whom is valid [waits.].*

[3]Note that, like in rule (35), the distinct precedence of *and* and *or* are not made explicit.

(91) `RelativeClause -->`

```
        that NPCoord[-R,+NOM]/- VPCoord[-INF]/NPCoord[-WH,-NOM]
```

Example: *[A card] that a customer enters [is valid.]*

(92) `RelativeClause -->`

```
        NPCoord[+R,-NOM]/- NPCoord[-R,+NOM]/- VPCoord[-INF]/NPCoord[-WH,-NOM]
```

Example: *[A card] the code of which a customer enters [is valid.]*

(93) `RelativeClause -->`

```
        PP[+R]/- NPCoord[-R,+NOM]/- VPCoord[-INF]/PP[-WH]
```

Example: *[A bank] in which a customer enters a code [is open.]*

2.11 Adjective phrases

Adjective phrase coordinations (94, 95) have a predicative usage only, i.e. they function as the complement of the copula, only (50, 52). Attributive usage, modifying a noun (64), is only possible with bare adjective coordinations (101).

If adjective phrases are coordinated, they are not allowed to contain any universally quantified or negated noun phrase (94). Like this, we prevent sentences like

**A customer is rich and interested in no money.*
**A customer is rich and interested in every code.*

Such sentences would lead to a contradictory doubling of the predicate *be* in the logical representation and must therefore be prohibited. The restriction, of course, does not hold for non-coordinated adjective phrases (95).

(94) `APCoord[-FORALL,-NEG] -->`

```
        APgrad[-FORALL,-NEG] and APCoord[-FORALL,-NEG]
```

Examples: *valid and correct, interested in some money and rich*

(95) `APCoord -->`

```
        APgrad
```

Examples: *valid, interested in no money, richer than every customer*

A graded adjective phrase is either a comparative adjective phrase followed by *than* and a noun phrase coordination (96, 97) or a bare non-comparative adjective phrase (98). To guarantee that a graded adjective phrase contains no universally quantified or negated noun phrases, it is assigned the feature values –FORALL and –NEG (97). One consequence of this is that any graded adjective phrase which is not likewise restricted has to bear the feature values +FORALL and +NEG, no matter if it contains such noun phrases or not (96).

(96) `APgrad[+FORALL,+NEG] -->`

```
        AP[+Comparative] than NPCoord/-
```

Example: *[A customer is] more interested in a/some/every/no card than a/some/every/no clerk [.]*

(97) `APgrad[-FORALL,-NEG] -->`

```
        AP[+Comparative,-FORALL,-NEG] than NPCoord[-FORALL,-NEG]/-
```

Example: *[Customer A is] more interested in a card than customer B [.]*

(98) `APgrad -->`
```
            AP[-Comparative]
```
Example: *[Customer A is] interested in a card [.]*

In ACE, we distinguish between intransitive and transitive adjectives. An adjective phrase can be either an intransitive adjective (99) or a transitive adjective followed by a prepositional phrase (100). Transitive adjective phrases receive the feature values which indicate if they contain a negative or universally quantified noun phrase from their prepositional phrase.

(99) `AP[-FORALL,-NEG] -->`
```
            IntransitiveAdjective
```
Examples: *correct, green, valid*

(100) `AP -->`
```
            TransitiveAdjective PP
```
Examples: *fond of Mary, interested in an account*

In attributive position (64), only non-comparative intransitive adjectives or coordinations of them (101) can occur.

(101) `AdjectiveCoord -->`
```
            IntransitiveAdjective[-Comparative] (and AdjectiveCoord)
```
Example: *[a] green and valid [card]*

2.12 Adverbs

An adverb coordination can either be a coordination of one or more non-interrogative (–Q) adverbs (102), a single interrogative adverb (103), or a gap left behind when an adjunct was moved to the front of a question (104).

(102) `AdverbCoord[-Q] -->`
```
            Adverb[-Q] (and AdverbCoord[-Q])
```
Example: *quickly and hastily and manually*

(103) `AdverbCoord[+Q] -->`
```
            Adverb[+Q]
```
Examples: *when, how, where*

(104) `AdverbCoord/AdverbCoord -->`
```
            []
```

Rule (104), like rule (85) above, can get a parser into an infinite loop if it is called from rules (41/42) and its gap feature is not sufficiently restricted.

3 Conclusion

In this report, we have presented an abstract grammar of the syntax of version 4.0 of Attempto Controlled English, a controlled natural language specifically designed for requirements specification and knowledge representation. It comprises phrase structure rules for the declarative and interrogative part of ACE. ACE may feel artificial and slightly stilted at some places, but we have shown in this report, how fixed expressions and carefully constructed syntax rules contribute substantially to the undertaking of reducing natural English to a controlled, formal subset.

The version of ACE described in the presented abstract grammar is enhanced with several computational applications, foremost the Attempto parser (APE), which translates ACE texts into discourse representation structures, and the Attempto reasoner (RACE), which performs automated logical deductions on ACE texts.

It has become evident that a controlled natural language like ACE can combine the advantages of natural and formal languages and thus substantially contribute to the fields of requirements engineering and knowledge representation, and to a successful interaction and communication between domain specialists and software engineers.

References

[1] Michael A. Covington. *Natural Language Processing for Prolog Programmers.* Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

[2] Gregor Erbach. Multi-Dimensional Inheritance. In H. Trost, editor, *Proceedings of KONVENS '94*, Vienna, 1994. Springer.

[3] Gregor Erbach. ProFIT: Prolog with Features, Inheritance and Templates. Technical report, Saarland University, Saarbrücken, 1994.

[4] Norbert E. Fuchs, Stefan Hoefler, and Uta Schwertel. Discourse Representation Structures in Attempto Controlled English. Technical report, Department of Informatics, University of Zurich, 2004.

[5] Norbert E. Fuchs and Uta Schwertel. Reasoning in Attempto Controlled English. In *Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, Lecture Notes in Computer Science, Hannover, 2003. Springer.

[6] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English - Not Just Another Logic Specification Language. In Pierre Flener, editor, *Logic-Based Program Synthesis and Transformation*, number 1559 in Lecture Notes in Computer Science, Manchester, UK, June 1999. Eighth International Workshop LOPSTR'98, Springer.

[7] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich, August 1999.

[8] Hans Kamp. A Theory of Truth and Semantic Representations. In J. A. G. Goenendijk, T. M. V. Janssen, and M. B. J. Stokhof, editors, *Formal Methods in the Study of Natural Language; Part 1*, pages 277–322. Amsterdam, 1981.

[9] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Kluwer, Dordrecht/Boston/London, 1993.

[10] Uta Schwertel. Controlling plural ambiguities in Attempto Controlled English. In *Proceedings of the 3rd Internaltional Workshop on Controlled Language Applications*, Seattle, Washington, 2000.

[11] Uta Schwertel. *Plural Semantics for Natural Language Understanding – A Computational Proof-Theoretic Approach.* PhD thesis, University of Zurich, Zurich, 2003.

Appendix: Function words

(105) `Aux --> do | does`

(106) `Coord --> and | or`

(107) `Copula --> is | are`

(108) `Determiner[-WH,+DEF,-NEG] --> the`

(109) `Determiner[-WH,+EXISTS,+SG,-NEG] --> a(n)`

(110) `Determiner[-WH,+EXISTS,-NEG] --> some`

(111) `Determiner[-WH,+EXISTS,+NEG] --> no`

(112) `Determiner[-WH,+FORALL,-NEG,+SG] --> every | each`

(113) `Determiner[-WH,+FORALL,-NEG,+MASS] --> all`

(114) `Determiner[-WH,+FORALL,+NEG,+SG] --> not every | not each`

(115) `Determiner[-WH,+FORALL,+NEG,+MASS] --> not all`

(116) `Determiner[+Q,+DEF,-NEG] --> which`

(117) `DistributiveGlobalQuantor --> for each of`

(118) `DistributiveMarker --> each of`

(119) `ExistentialGlobalQuantor --> there is | there are`

(120) `ExistentialGlobalQuestionQuantor --> is there | are there`

(121)
```
GeneralizedQuantor -->
          at most | at least | more than | less than |
          not more than | not less than
```

(122) `PossessivePronoun[-WH,+DEF,-NEG,+SG] --> his | her | his/her`

(123) `PossessivePronoun[-WH,+DEF,-NEG,-PL] --> its`

(124) `PossessivePronoun[-WH,+DEF,-NEG,+PL] --> their`

(125) `PossessivePronoun[-WH,+DEF,-NEG,+SG] --> his own | her own | his/her own`

(126) `PossessivePronoun[-WH,+DEF,-NEG,-PL] --> its own`

(127) `PossessivePronoun[-WH,+DEF,-NEG,+PL] --> their own`

(128) `PossessivePronoun[+Q,+DEF,-NEG] --> whose`

(129) `PossessivePronoun[+R,+DEF,-NEG] --> whose`

(130) `Pronoun[-WH,+DEF,-NEG,-PL] --> it`

(131) `Pronoun[-WH,+DEF,-NEG,+NOM,+SG] --> he | she | he/she`

(132) `Pronoun[-WH,+DEF,-NEG,-NOM,+SG] --> him | her | him/her`

(133) `Pronoun[-WH,+DEF,-NEG,+NOM,+PL] --> they`

(134) `Pronoun[-WH,+DEF,-NEG,-NOM,+PL] --> them`

(135) `Pronoun[-WH,+DEF,-NEG,-NOM,-PL] --> itself`

(136) `Pronoun[-WH,+DEF,-NEG,-NOM,+SG] --> himself | herself | himself/herself`

(137) `Pronoun[-WH,+DEF,-NEG,-NOM,+PL] --> themselves`

(138) `Pronoun[-WH,+EXISTS,-NEG,+SG] --> someone | somebody`

(139) `Pronoun[-WH,+EXISTS,-NEG,-PL] --> something`

(140) `Pronoun[-WH,+EXISTS,+NEG,+SG] --> no one | nobody`

(141) `Pronoun[-WH,+EXISTS,+NEG,-PL] --> nothing`

(142) `Pronoun[-WH,+FORALL,-NEG,+SG] --> everyone | everybody`

(143) `Pronoun[-WH,+FORALL,-NEG,-PL] --> everything`

(144) `Pronoun[-WH,+FORALL,+NEG,+SG] --> not everyone | not everybody`

(145) `Pronoun[-WH,+FORALL,+NEG,-PL] --> not everything`

(146) `Pronoun[+Q,+DEF,-NEG] --> what | who`

(147) `Pronoun[+Q,+DEF,-NEG,-NOM] --> whom`

(148) `Pronoun[+R,+DEF,-NEG] --> which | who`

(149) `Pronoun[+R,+DEF,-NEG,-NOM] --> whom`

(150) `SaxonGenitiveMarker --> ' | 's`
Examples: *[a customer]'s [card], [5 customers]' [contract]*

(151) `UniversalGlobalQuantor[+SG] --> for every | for each`

(152) `UniversalGlobalQuantor[+MASS] --> for all`

Addenda

After the publication of the abstract grammar of ACE 4.0, the syntax of ACE was extended by some minor rules. These rules are listed in the addenda and constitute later stages of the ACE language. At first instance, they are not implemented in the Attempto Parsing Engine (APE).

Comparative Adjectives: Comparison of Prepositional Objects

Rules (153) and (154) are complementary to rules (96) and (97). They allow for another type of comparison. Compare the respective sample sentences:

A customer is more interested in a card than a clerk. (96, 97)
A customer is more interested in a card than in a code. (153, 154)

Evidently, rules (153) and (154) only apply if the comparative adjective is transitive, i.e. if it takes a prepositional object. Furthermore, agreement of prepositions has to be made sure. Sentences like the ones below are ruled out:

**A customer is taller than in a bank.*
**A customer is more interested in a card than at a bank.*

Note that the agreement of prepositions is not made explicit in the abstract grammar.

(153)
```
APgrad[+FORALL,+NEG] -->
        AP[+Comparative] than PP/-
```
Example: *[A customer is] more interested in a/some/every/no card than in a/some/every code [.]*

(154)
```
APgrad[-FORALL,-NEG] -->
        AP[+Comparative,-FORALL,-NEG] than PP[-FORALL,-NEG]/-
```
Example: *[A customer is] more interested in a card than in a code [.]*