Discourse Representation Structures for ACE 6.0

Technical Report ifi-2008.02

Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn

Department of Informatics, University of Zurich
{fuchs,kalju,tkuhn}@ifi.uzh.ch

Abstract

This technical report describes the discourse representation structures (DRS) derived from texts written in version 6.0 of Attempto Controlled English (ACE 6.0). The description is done by an exhaustive set of examples.

Among other things, ACE 6.0 supports modal statements, negation as failure, and sentence subordination. These features require an extended form of discourse representation structures.

The discourse representation structure itself uses a reified, or 'flat' notation, meaning that its atomic conditions are built from a small number of predefined predicates that take constants standing for words of the ACE text as their arguments.

Contents

1 Introductory Notes

This technical report describes the representation of discourse representation structures (DRS) derived from version 6.0 of Attempto Controlled English (ACE 6.0). It uses illustrative ACE examples, but does not describe ACE itself. For a complete description of the ACE language please refer to the Attempto web site [2]. An abstract grammar for ACE 6.0 can be found in [1].

We expect the reader to be familiar with the basic notions of Discourse Representation Theory (DRT) [5] as, for instance, introduced in [3]. Consult [4] for the DRS representation of modality and sentence subordination.

Section 2 introduces the notation. Sections 3 to 12 describe discourse representation structures derived from declarative ACE sentences, and section 13 those derived from ACE questions. Commands in ACE lead to the same DRSs as their declarative counterparts. Therefore, commands are not discussed here.

2 Notation

Using illustrative ACE examples, this report completely describes the language of DRSs derived from ACE texts. For a complete description of the ACE language itself please refer to the relevant documents on the Attempto web site [2].

2.1 Basics

The ACE parser translates an ACE text unambiguously into a DRS representation. The discourse representation structure derived from the ACE text is returned as

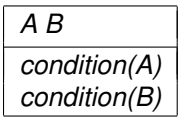
```
drs(Domain,Conditions)
```

The first argument of `drs/2` is a list of discourse referents, i.e. quantified variables naming objects of the domain of discourse. The second argument of `drs/2` is a list of simple and complex conditions for the discourse referents. The list separator ',' stands for logical conjunction. Simple conditions are logical atoms, while complex conditions are built from other discourse representation structures with the help of the logical connectors negation '-', disjunction 'v', and implication '=>'. Furthermore, we use non-standard logical connectors for possibility '<>', necessity '[]', negation as failure '~', and a connector for the assignment of variables to sub-DRSs ':'.

A DRS like

```
drs([A,B],[condition(A),condition(B)])
```

is usually pretty-printed as



2.2 Flat Notation

The discourse representation structure uses a reified, or 'flat' notation for logical atoms. For example, the noun *a card* that customarily would be represented as

```
card(A)
```

is represented here as

```
object(A,card,countable,na,eq,1)
```

relegating the predicate 'card' to the constant 'card' used as an argument in the predefined predicate 'object'.

As a consequence, the large number of predicates in the customary representation is replaced by a small number of predefined predicates. This allows us to conveniently formulate axioms for the predefined predicates.

2.3 Typed Representation

There are two versions of the DRS: the typed one and the untyped one. Each lexicon entry has an optional type argument and these types are carried over to the DRS if the typed representation is used. Otherwise, these types are simply ignored. The types have always the last argument positions of the predicates. These argument positions are missing for the untyped representation. Note that the types have absolutely no effect on the representation, apart from appearing in the typed DRSs.

The types can be arbitrary Prolog atoms. In the built-in lexicon, most types are `na` ("not available"). When you provide your own lexicon, you can define your own types.

For example, we can imagine a scenario where we want to distinguish verbs that define a state (e.g. "sleeps") from verbs that define an event (e.g. "crashes"). In this situation, we can associate each verb with one of the types `state` and `event`. This information could then be used to define axioms or to specify distinct behavior for the two cases.

The next section shows how the two version differ. Afterwards, only the untyped representation is used in this report.

2.4 Predicate Declarations

2.4.1 object

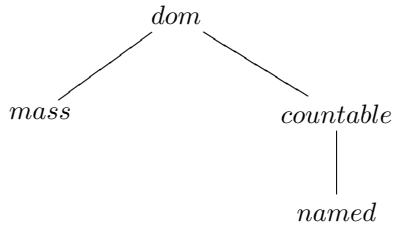
The `object`-predicates stand for objects that are introduced by the different forms of nouns.

typed	`object(Ref,Noun,Quant,Unit,Op,Count,Type,DimType)`
untyped	`object(Ref,Noun,Quant,Unit,Op,Count)`

`Ref` The variable that stands for this object and that is used for references.

`Noun` The noun (mass or countable) or the proper name that was used to introduce the object.

`Quant` This is one of {`dom`,`mass`,`countable`,`named`} and defines the quantisation of the object. The tree structure below shows the hierarchy of these values.



`Unit` If the object was introduced together with a measurement noun (e.g. “2 kg of apples”) then this entry contains the value of the measurement noun (e.g. `kg`). Otherwise, this entry is `na`.

`Op` One of {`eq`,`geq`,`greater`,`exactly`,`na`}. `eq` stands for “equal” and `geq` for “greater or equal”. Note that `leq` and `less` can not appear here but only in the `quantity`-predicate.

`Count` A positive number or `na`. Together with `Unit` and `Op`, this defines the cardinality or extent of the object.

`Type` This is the type that is associated with the noun (mass or countable) or with the proper name.

`DimType` The type that is associated with the measurement noun, if a measurement noun was used. In the case of proper names, coordinated noun phrases, and countable nouns without measurement noun, this has the value `cardinality`. In all the other cases, it is `na`.

2.4.2 property

The `property`-predicates stand for properties that are introduced by adjectives. The references can either be variables or expressions. See section 3.6 for the representation of expressions.

typed	*1-ary*	`property(Ref1,Adjective,Degree,Type)`
	2-ary	`property(Ref1,Adjective,Degree,Ref2,Type)`
	3-ary	`property(Ref1,Adjective,Ref2,Degree,CompTarget,Ref3,Type)`
untyped	*1-ary*	`property(Ref1,Adjective,Degree)`
	2-ary	`property(Ref1,Adjective,Degree,Ref2)`
	3-ary	`property(Ref1,Adjective,Ref2,Degree,CompTarget,Ref3)`

`Ref1` The variable or expression that stands for the primary object of the property (i.e. the subject).

`Ref2` The variable or expression that stands for the secondary object of the property.

`Ref3` The variable or expression that stands for the tertiary object of the property.

`Adjective` The intransitive or transitive adjective.

`Degree` This is one of {`pos`,`pos_as`,`comp`,`comp_than`,`sup`} and it defines the degree of the adjective. Positive and comparative forms can have an additional comparison target ("as rich as ...", "richer than ..."), and for those cases `pos_as` and `comp_than` are used.

`CompTarget` This is one of {`subj`,`obj`} and it defines for transitive adjectives whether the comparison targets the subject ("John is more fond-of Mary than Bill") or the object ("John is more fond-of Mary than of Sue").

`Type` The type that is associated with the adjective.

2.4.3 relation

The `relation`-predicates stand for relations that are introduced by *of*-constructs.

```
relation(Ref1,of,Ref2)
```

`Ref1` A variable that refers to the left hand side object. This variable is always associated with an `object`-predicate.

`Ref2` A variable or expression that stands for the right hand side object.

Note that the second argument is always `of` since no other prepositions can attach to nouns.

2.4.4 predicate

The `predicate`-predicates stand for relations that are introduced by intransitive, transitive, and ditransitive verbs.

typed	*intransitive*	`predicate(Ref,Verb,SubjRef,Type)`
	transitive	`predicate(Ref,Verb,SubjRef,ObjRef,Type)`
	ditransitive	`predicate(Ref,Verb,SubjRef,ObjRef,IndObjRef,Type)`
untyped	*intransitive*	`predicate(Ref,Verb,SubjRef)`
	transitive	`predicate(Ref,Verb,SubjRef,ObjRef)`
	ditransitive	`predicate(Ref,Verb,SubjRef,ObjRef,IndObjRef)`

`Ref` A variable that stands for this relation and that is used to attach modifiers (i.e. adverbs and prepositional phrases).

`Verb` The intransitive, transitive, or ditransitive verb.

`SubjRef` A variable or expression that stands for the subject.

`ObjRef` A variable or expression that stands for the direct object.

`IndObjRef` A variable or expression that stands for the indirect object.

`Type` The type that is associated with the verb.

2.4.5 modifier_adv

The `modifier_adv`–predicates stand for verb phrase modifiers that are introduced by adverbs.

typed	`modifier_adv(Ref,Adverb,Degree,Type)`
untyped	`modifier_adv(Ref,Adverb,Degree)`

`Ref` A variable that refers to the modified verb.

`Adverb` The adverb.

`Degree` This is one of {`pos`,`comp`,`sup`} and defines the degree of the adverb.

`Type` The type that is associated with the adverb.

2.4.6 modifier_pp

The `modifier_pp`–predicates stand for verb phrase modifiers that are introduced by prepositional phrases.

typed	`modifier_pp(Ref1,Preposition,Ref2,Type)`
untyped	`modifier_pp(Ref1,Preposition,Ref2)`

`Ref1` A variable that refers to the modified verb.

`Preposition`**.** The preposition of the prepositional phrase.

`Ref2` A variable or expression that stands for the object of the prepositional phrase.

`Type` This type is always `na`.

2.4.7 has_part

The `has_part`-predicates define the memberships of objects in groups of objects.

`has_part(GroupRef,MemberRef)`

`GroupRef` A variable that refers to a group of objects.

`MemberRef` A variable or expression that stands for the object that is a member of the group.

2.4.8 quantity

The `quantity`-predicate is used for declaring the maximal cardinality or extent of an object. It can appear only alone in the then-part of an implication. It is used for generalised quantifiers like “at most” and “less than”.

```
quantity(Ref,Op,Count)
```

`Ref` A variable that refers to an object.

`Op` One of {`leq, less`} where `leq` stands for “less or equal”.

`Count` A positive number that defines the maximal cardinality or extent of the object.

2.4.9 query

A `query`-predicate points to the object or relation a query was put on.

```
query(Ref,QuestionWord)
```

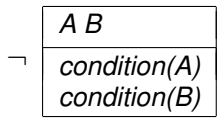
`Ref` A variable that refers to the object or relation of the query.

`QuestionWord` One of {`who,what,which,how,where,when`}.

2.5 Complex Structures

2.5.1 Classical Negation

A negated DRS like



is internally represented as

```
-drs([A,B],[condition(A),condition(B)])
```

The prefix operator `-/1` stands for the logical negation ‘¬’.

2.5.2 Negation As Failure

A DRS that is negated using negation as failure (NAF) is marked with a tilde sign:

$\sim$	*A B*
condition(A)
condition(B)

It is represented as

```
~drs([A,B],[condition(A),condition(B)])
```

The prefix operator `~/1` stands for negation as failure.

2.5.3 Implication and Disjunction

In a DRS, all variables are existentially quantified unless they occur in the precondition of an implication. The implication

A
condition(A)

$\Rightarrow$

B
condition(B)

is internally represented as

```
drs([A],[condition(A)]) => drs([B],[condition(B)])
```

The disjunction

A
condition(A)

$\vee$

B
condition(B)

is likewise internally represented as

```
drs([A],[condition(A)]) v drs([B],[condition(B)])
```

The predicates `=>/2` and `v/2` are defined as infix operators.

2.5.4 Possibility and Necessity

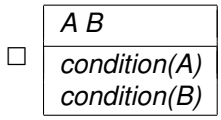
Possibility and necessity are modal extensions for DRSs. Consult [4] for details about such modal constructs and their representations in first-order logic. Possibility is represented with a diamond sign

$\Diamond$	*A B*
condition(A)
condition(B)

and is internally represented as

```
<>drs([A,B],[condition(A),condition(B)])
```

Necessity is represented with a box sign



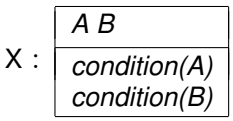
and is internally represented as

```
[]drs([A,B],[condition(A),condition(B)])
```

The prefix operators `<>/1` and `[]/1` are used to represent possibility and necessity, respectively.

2.5.5 Sentence Subordination (*that*-Subordination)

For sentences like 'John believes that Mary sleeps' we need an extended DRS syntax. For that reason we introduce a new notation that allows us to attach labels to sub-DRSs. Consult [4] for details.



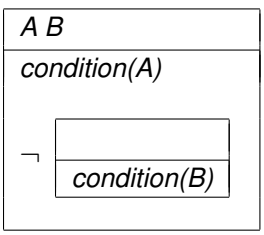
This is internally represented as

```
X:drs([A,B],[condition(A),condition(B)])
```

The infix operator `:/2` is used to attach labels to sub-DRSs.

2.5.6 Nesting

In nested discourse representation structures, a DRS can occur as an element of the conditions list of another DRS. Therefore



is represented as

```
drs([A,B],[condition(A),-drs([],[condition(B)])])
```

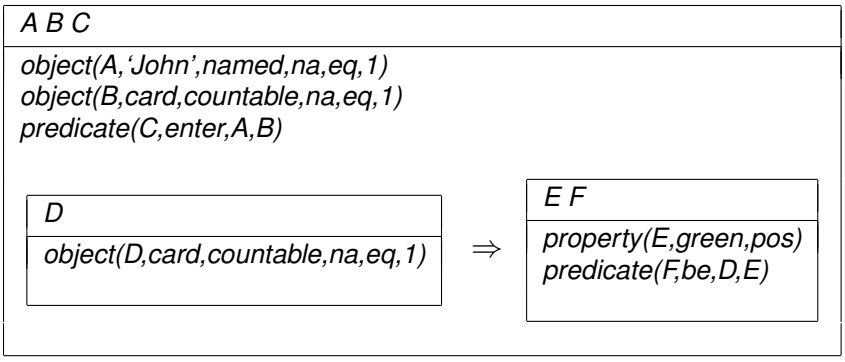
2.6 Sentence Numbers

Logical atoms occurring in `drs/2` are actually written as `Atom-I` (using an infix operator `-/2`) where the number `I` refers to to the sentence from which `Atom` was derived.

The example text

```
John enters a card. Every card is green.
```

the DRS of which is



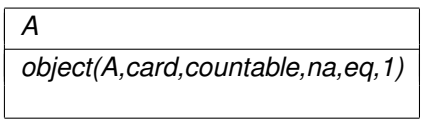
will thus internally be represented as

```
drs([A,B,C],[object(A,'John',named,na,eq,1)-1,
object(B,card,countable,na,eq,1)-1,predicate(C,enter,A,B)-1,
drs([D],[object(D,card,countable,na,eq,1)-2])=>
drs([E,F],[property(E,green,pos)-2,predicate(F,be,D,E)-2])]).
```

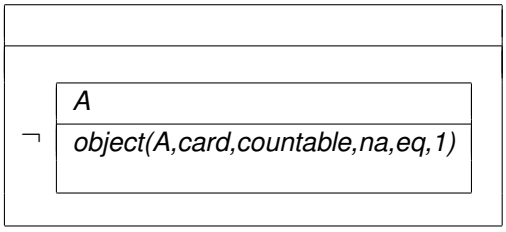
3 Noun Phrases

3.1 Singular Countable Noun Phrases

a card

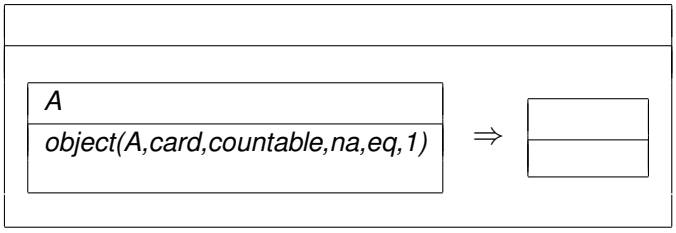


no card

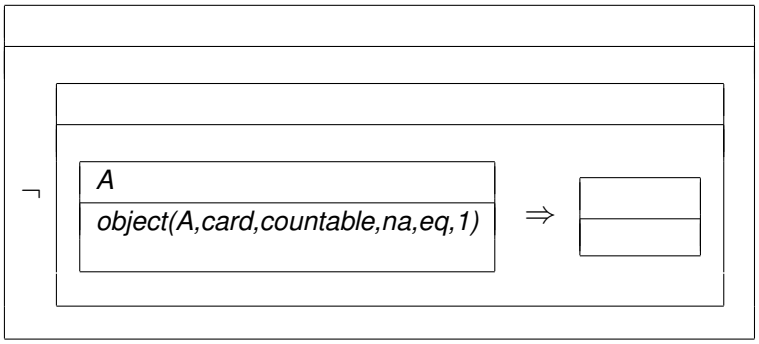


Note that the representation of "*no card*" depends on the context (see section 9.1.1).

every card

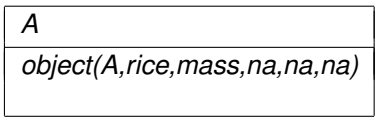


not every card

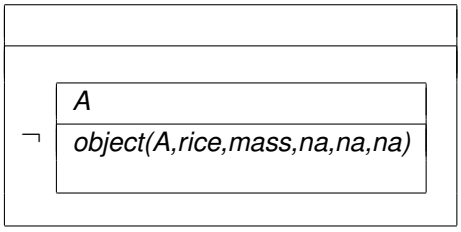


3.2 Mass Nouns

some rice

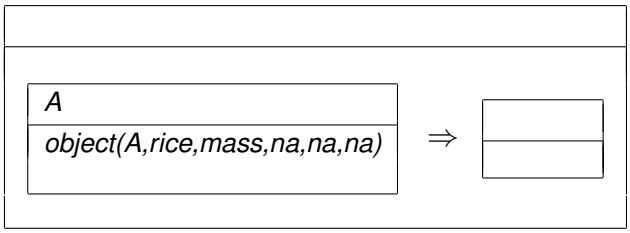


no rice

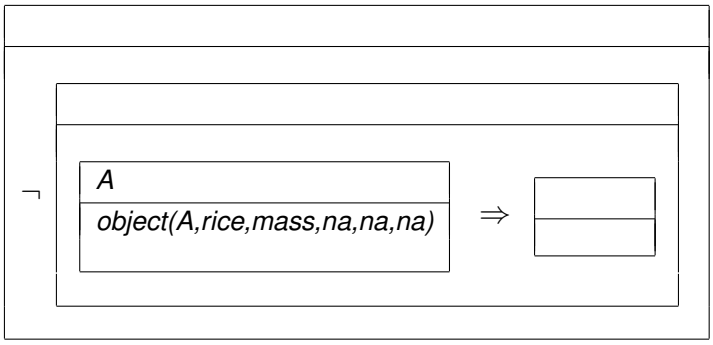


Note that the representation of "*no rice*" depends on the context (see section 9.1.1). Furthermore, the determiner *no* is ambiguous between countable and mass. For nouns that can be countable or mass, e.g. *money*, preference to countable is given. Mass reading can be forced by using sentential negation, e.g. *It is false that some money is omnipotent*.

all rice

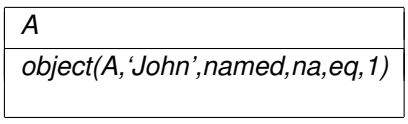


not all rice



3.3 Proper Names

John



3.4 Plural Noun Phrases

some cards

A
*object(A,card,countable,na,**geq,2**)*

***2** cards*

A
*object(A,card,countable,na,**eq,2**)*

***five** cards*

A
*object(A,card,countable,na,**eq,5**)*

3.5 Indefinite Pronouns

someone / somebody

A
object(A,somebody,countable,na,eq,1)

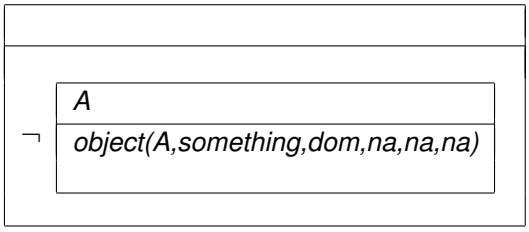
something

A
object(A,something,dom,na,na,na)

no one / nobody

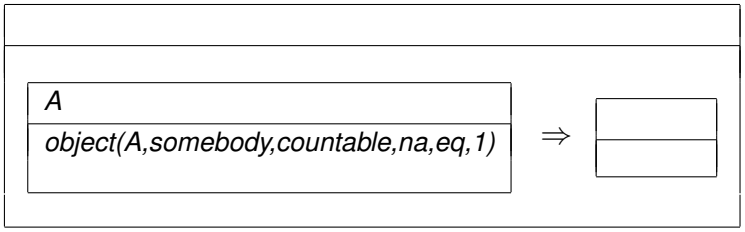
¬ *A* : *object(A,somebody,countable,na,eq,1)*

nothing

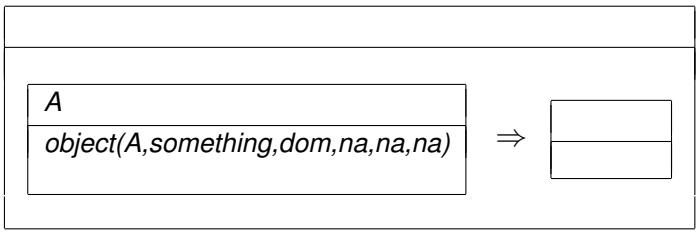


Note that the representations of "*no one*", "*nobody*", and "*nothing*" depend on the context (see section 9.1.1).

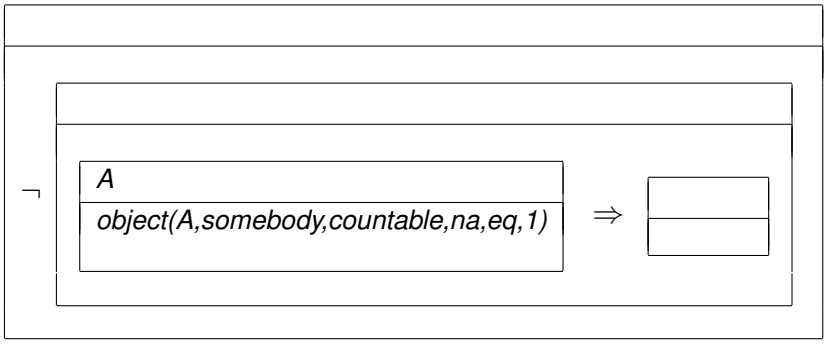
everyone / everybody



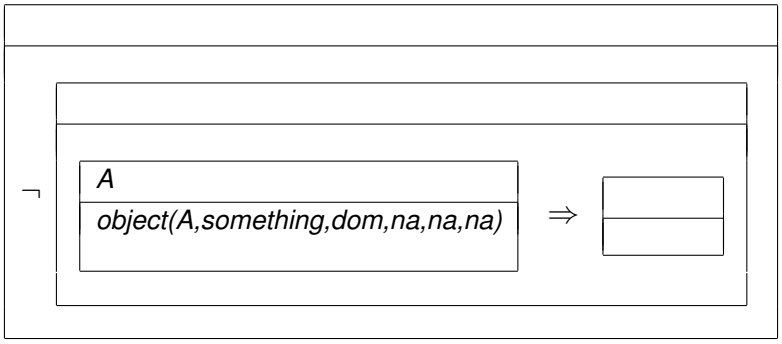
everything



not everyone / not everybody



not everything



3.6 Expressions

3.6.1 Atomic Expressions

*A number is **14**.*

A B
object(A,number,countable,na,eq,1) *predicate(B,have,A,**int(14)**)*

***3.5** is greater than **2.3**.*

A B
*property(A,great,comp_than,**real(2.3)**)* *predicate(B,be,**real(3.5)**,A)*

*”**abcd**” is entered by John.*

A B
object(A,‘John’,named,na,eq,1) *predicate(B,enter,A,**string(abcd)**)*

3.6.2 Compound Expressions

A value is ***(1 + 2) / X * 4****.*

A B C
object(A,'X',named,na,eq,1) *object(B,value,countable,na,eq,1)* *predicate(C,be,B,****expr(*,expr(/,expr(+,int(1),int(2)),A),int(4))****)*

"abc" & "123" *is a valid password.*

A B
object(A,password,countable,na,eq,1) *property(A,valid,pos)* *predicate(B,be,****expr(&,string(abc),string('123'))****,A)*

3.6.3 Lists and Sets

3 is the first element of ***[3,4.5,"ab",John,1+2]****.*

A B C
object(A,'John',named,na,eq,1) *predicate(B,be,int(3),C)* *relation(C,of,****list([int(3),real(4.5),string(ab),A,expr(+,int(1),int(2))])****)* *property(C,first,pos)* *object(C,element,countable,na,eq,1)*

{***3,6,[1,2]***} *contains 6.*

A
*predicate(A,contain,****set([int(3),int(6),list([int(1),int(2)])])****,int(6))*

3.7 Generalised Quantors

In the case of "exactly" and if the generalised quantor implies a minimality condition then the DRS representation is flat.

*John has **at least 2** cards that are valid.*

A B C D E
object(A,'John',named,na,eq,1) *object(B,card,countable,na,**geq,2**)* *property(C,valid,pos)* *predicate(D,be,B,C)* *predicate(E,have,A,B)*

*John has **more than 2** cards that are valid.*

A B C D E
object(A,'John',named,na,eq,1) *object(B,card,countable,na,**greater,2**)* *property(C,valid,pos)* *predicate(D,be,B,C)* *predicate(E,have,A,B)*

*John has **exactly 2** cards that are valid.*

A B C D E
object(A,'John',named,na,eq,1) *object(B,card,countable,na,**exactly,2**)* *property(C,valid,pos)* *predicate(D,be,B,C)* *predicate(E,have,A,B)*

If the generalised quantor implies a maximality condition then an implication is introduced in the DRS. The then-part of the implication contains only a `quantity`-predicate. This is necessary because we need to capture the scope of the maximality restriction.

*John has **at most 2** cards that are valid.*

A
object(A,'John',named,na,eq,1) *B C D E* *object(B,card,countable,na,na,na)* *property(C,valid,pos)* *predicate(D,be,B,C)* *predicate(E,have,A,B)* ⇒ ***quantity(B,leq,2)***

*John has **less than 2** cards that are valid.*

A
object(A,'John',named,na,eq,1)

B C D E
object(B,card,countable,na,na,na) *property(C,valid,pos)* *predicate(D,be,B,C)* *predicate(E,have,A,B)*

$\Rightarrow$

quantity(B,less,2)

3.8 Noun Phrase Conjunction

a customer and a clerk

A B C
object(A,customer,countable,na,eq,1) *object(B,clerk,countable,na,eq,1)* *has_part(C,A)* *has_part(C,B)* *object(C,na,countable,na,eq,2)*

3.9 Measurement Noun Phrases

***2 kg of** apples*

A
*object(A,apple,countable,**kg,eq,2**)*

***2 kg of** rice*

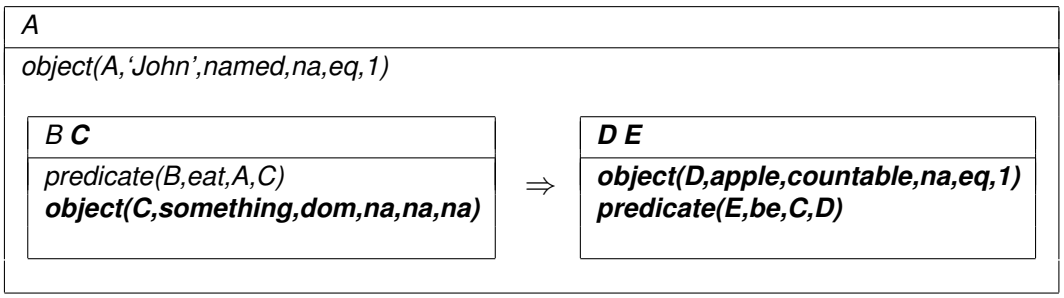
A B
*object(A,rice,mass,**kg,eq,2**)*

*John's weight is **80 kg**.*

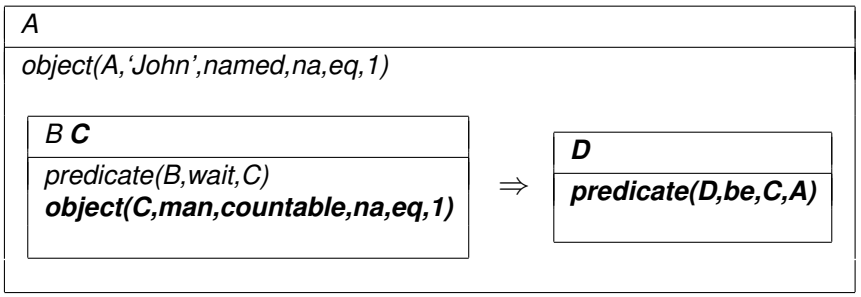
A B C
object(A,'John',named,na,eq,1) *relation(B,of,A)* *object(B,weight,countable,na,eq,1)* *predicate(C,be,B,**int(80,kg)**)*

3.10 Nothing But

*John eats **nothing but apples**.*



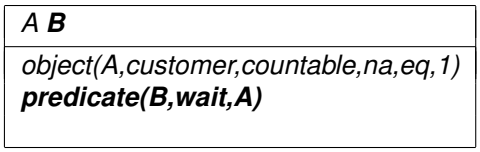
***No man but John** waits.*



4 Verb Phrases

4.1 Intransitive Verbs

*A customer **waits**.*



4.2 Transitive Verbs

The following two sentences are parsed identically.

*John **enters** a card.*
*A card **is entered by** John.*

*A B **C***
object(A,'John',named,na,eq,1) *object(B,card,countable,na,eq,1)* ***predicate(C,enter,A,B)***

4.3 Ditransitive Verbs

The following four sentences are parsed identically.

*A clerk **gives** a password **to** a customer.*
*A clerk **gives** a customer a password.*
*A password **is given to** a customer **by** a clerk.*
*A customer **is given** a password **by** a clerk.*

*A B C **D***
object(A,clerk,countable,na,eq,1) *object(B,password,countable,na,eq,1)* *object(C,customer,countable,na,eq,1)* ***predicate(D,give,A,B,C)***

4.4 Copula

4.4.1 Copula and Intransitive Adjectives

*A customer **is important**.*

*A **B C***
object(A,customer,countable,na,eq,1) ***property(B,important,pos)*** ***predicate(C,be,A,B)***

*A customer **is as important as** John.*

*A B **C D***
object(A,'John',named,na,eq,1) *object(B,customer,countable,na,eq,1)* ***property(C,important,pos_as, A)*** ***predicate(D,be,B,C)***

A customer ***is more important****.*

A ***B C***
object(A,customer,countable,na,eq,1) ***property(B,important,comp)*** ***predicate(C,be,A,B)***

A customer ***is more important than*** *John.*

A B ***C D***
object(A,'John',named,na,eq,1) *object(B,customer,countable,na,eq,1)* ***property(C,important,comp_than, A)*** ***predicate(D,be,B,C)***

A customer ***is most important****.*

A ***B C***
object(A,customer,countable,na,eq,1) ***property(B,important,sup)*** ***predicate(C,be,A,B)***

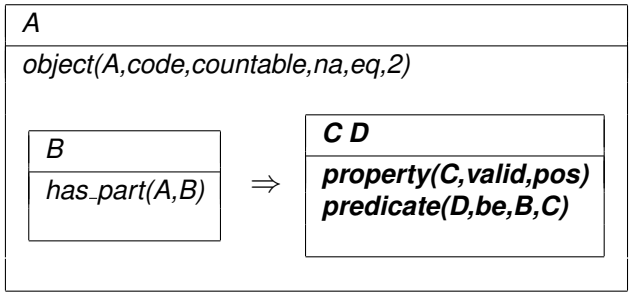
A card ***is valid and correct****.*

A ***B C***
object(A,card,countable,na,eq,1) ***property(B,valid,pos)*** ***property(B,correct,pos)*** ***predicate(C,be,A,B)***

2 codes ***are valid****.*

A ***B C***
object(A,code,countable,na,eq,2) ***property(B,valid,pos)*** ***predicate(C,be,A,B)***

*Each of 2 codes **is valid**.*



4.4.2 Copula and Transitive Adjectives

*John **is fond-of** Mary.*

*A B **C D***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* ***property(C,'fond-of',pos,B)*** ***predicate(D,be,A,C)***

*John **is as fond-of** Mary **as** Bill.*

*A B C **D E***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* *object(C,'Bill',named,na,eq,1)* ***property(D,'fond-of',B,pos_as,subj,C)*** ***predicate(E,be,A,D)***

*John **is as fond-of** Mary **as of** Sue.*

*A B C **D E***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* *object(C,'Sue',named,na,eq,1)* ***property(D,'fond-of',B,pos_as,obj,C)*** ***predicate(E,be,A,D)***

John ***is more fond-of*** *Mary.*

A B ***C D***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* ***property(C,'fond-of',comp,B)*** ***predicate(D,be,A,C)***

John ***is more fond-of*** *Mary* ***than*** *Bill.*

A B C ***D E***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* *object(C,'Bill',named,na,eq,1)* ***property(D,'fond-of',B,comp_than,subj,C)*** ***predicate(E,be,A,D)***

John ***is more fond-of*** *Mary* ***than of*** *Sue.*

A B C ***D E***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* *object(C,'Sue',named,na,eq,1)* ***property(D,'fond-of',B,comp_than,obj,C)*** ***predicate(E,be,A,D)***

John ***is most fond-of*** *Mary.*

A B ***C D***
object(A,'John',named,na,eq,1) *object(B,'Mary',named,na,eq,1)* ***property(C,'fond-of',sup,B)*** ***predicate(D,be,A,C)***

4.4.3 Copula and Noun Phrase

John ***is*** *a rich customer.*

A B ***C***
object(A,'John',named,na,eq,1) *property(B,rich,pos)* *object(B,customer,countable,na,eq,1)* ***predicate(C,be,A,B)***

4.4.4 Copula and Prepositional Phrase

*John **is** in the bank.*

*A **B** C*
object(A,'John',named,na,eq,1)) ***predicate(B,be,A)*** *modifier_pp(B,in,C)* *object(C,bank,countable,na,eq,1)*

4.5 Coordinated Verb Phrases

4.5.1 Verb Phrase Conjunction

A screen flashes and blinks.

A B C
object(A,screen,countable,na,eq,1) *predicate(B,flash,A)* *predicate(C,blink,A)*

4.5.2 Verb Phrase Disjunction

A screen flashes or blinks.

A
object(A,screen,countable,na,eq,1)

B	$\vee$	*C*
predicate(B,flash,A)		*predicate(C,blink,A)*

5 Modifying Nouns and Noun Phrases

5.1 Adjectives

*An **important** customer waits.*

A B
object(A,customer,countable,na,eq,1) ***property(A,important,pos)*** *predicate(B,wait,A)*

*A **more important** customer waits.*

A B
object(A,customer,countable,na,eq,1) ***property(A,important,comp)*** *predicate(B,wait,A)*

*The **most important** customer waits.*

A B
object(A,customer,countable,na,eq,1) ***property(A,important,sup)*** *predicate(B,wait,A)*

*A **rich and old** customer waits.*

A B
object(A,customer,countable,na,eq,1) ***property(A,rich,pos)*** ***property(A,old,pos)*** *predicate(B,wait,A)*

5.2 Variables

*A **customer X** greets a clerk. The clerk is happy. **X** is glad.*

***A** B C D E F G*
object(A,customer,countable,na,eq,1)
object(B,clerk,countable,na,eq,1)
predicate(C,greet,A,B)
property(D,happy,pos)
predicate(E,be,B,D)
property(F,glad,pos)
*predicate(G,be,**A**,F)*

Note: Variables do not appear in the DRS. They only establish anaphoric references.

5.3 Relative Sentences

5.3.1 Simple Relative Sentences

*A customer enters a card **which is valid**.*

*A B C **D E***
object(A,customer,countable,na,eq,1)
object(B,card,countable,na,eq,1)
property(C,valid,pos)
predicate(D,be,B,C)
predicate(E,enter,A,B)

*Every card **the code of which is correct** is valid.*

*A **B C D***
object(A,card,countable,na,eq,1)
property(B,correct,pos)
predicate(C,be,D,B)
relation(D,of,A)
object(D,code,countable,na,eq,1)

$\Rightarrow$

E F
property(E,valid,pos)
predicate(F,be,A,E)

Everything ***which eats*** *is animate.*

A ***B***		*C D*
object(A,something,dom,na,na,na) ***predicate(B,eat,A)***	⇒	*property(C,animate,pos)* *predicate(D,be,A,C)*

John ***who is a clerk*** *waits.*

A B ***C D***
object(A,'John',named,na,eq,1) *predicate(B,wait,A)* ***object(C,clerk,countable,na,eq,1)*** ***predicate(D,be,A,C)***

There is a card X. X ***which a customer possesses*** *is valid.*

A ***B C*** *D E*
object(A,card,countable,na,eq,1) ***object(B,customer,countable,na,eq,1)*** ***predicate(C,possess,B,A)*** *property(D,valid,pos)* *predicate(E,be,A,D)*

5.3.2 Relative Sentence Conjunction and Disjunction

A customer enters a card ***which is green and which is valid****.*

A B ***C D E F*** *G*
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* ***property(C,green,pos)*** ***predicate(D,be,B,C)*** ***property(E,valid,pos)*** ***predicate(F,be,B,E)*** *predicate(G,enter,A,B)*

A customer enters a card ***which is green or which is red****.*

A B C
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***D E*** | ***F G*** ***property(D,green,pos)*** | ***property(F,red,pos)*** ***predicate(E,be,B,D)*** $\vee$ ***predicate(G,be,B,F)***

5.4 *of*-Prepositional Phrases

The surface ***of*** *the card has a green color.*

A B C D
object(A,surface,countable,na,eq,1) *object(B,card,countable,na,eq,1)* ***relation(A,of,B)*** *object(C,color,countable,na,eq,1)* *property(C,green,pos)* *predicate(B,have,A,C)*

5.5 Possessive Nouns

Possessive nouns are introduced by a possessive pronoun or a Saxon genitive. While possessive nouns are equivalent to *of* PPs, Saxon genitives in general are not because of the scoping rules of quantifiers:

- a man's dog (1 man with 1 dog) $=$ a dog of a man (1 man with 1 dog)
- every man's dog (several men each with 1 dog) $\neq$ a dog of every man (1 dog of several men)

*The customer'****s*** *card is valid.*

A B C D
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* ***relation(B,of,A)*** *property(C,valid,pos)* *predicate(D,be,B,C)*

Note: There are no recursive Saxon genitives. “A customer’s card” is in ACE, but “A customer’s card’s code” is not.

*There is a customer. **His** code is correct.*

A B C D
object(A,customer,countable,na,eq,1) *object(B,code,countable,na,eq,1)* ***relation(B,of,A)*** *property(C,correct,pos)* *predicate(D,be,B,C)*

6 Modifying Verb Phrases

6.1 Adverbs

The following two sentences are parsed identically.

*A customer **quickly** enters a card.*
*A customer enters a card **quickly**.*

A B C
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***modifier_adv(C,quickly,pos)***

The following two sentences are parsed identically.

*A customer **more quickly** enters a card.*
*A customer enters a card **more quickly**.*

A B C
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***modifier_adv(C,quickly,comp)***

The following two sentences are parsed identically.

*A customer **most quickly** enters a card.*
*A customer enters a card **most quickly**.*

A B C
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***modifier_adv(C,quickly,sup)***

6.2 Prepositional Phrases

*John enters a card **in a bank**.*

*A B C **D***
object(A,'John',named,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***object(D,bank,countable,na,eq,1)*** ***modifier_pp(C,in,D)***

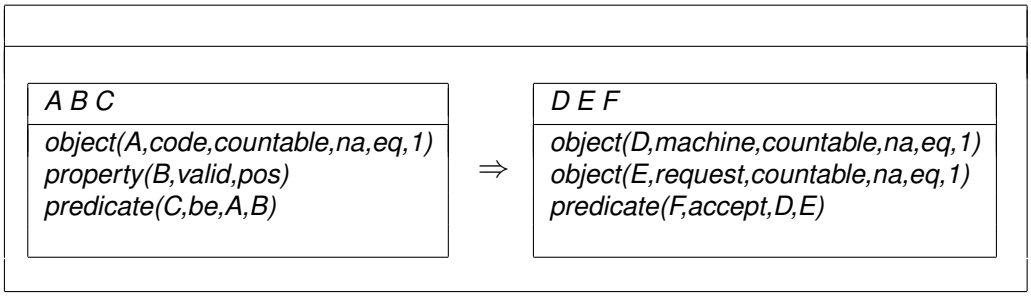
*A customer enters a card **quickly and manually in a bank in the morning**.*

*A B C **D E***
object(A,customer,countable,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***modifier_adv(C,quickly,pos)*** ***modifier_adv(C,manually,pos)*** ***object(D,bank,countable,na,eq,1)*** ***modifier_pp(C,in,DD)*** ***object(E,morning,countable,na,eq,1)*** ***modifier_pp(C,in,E)***

7 Composite Sentences

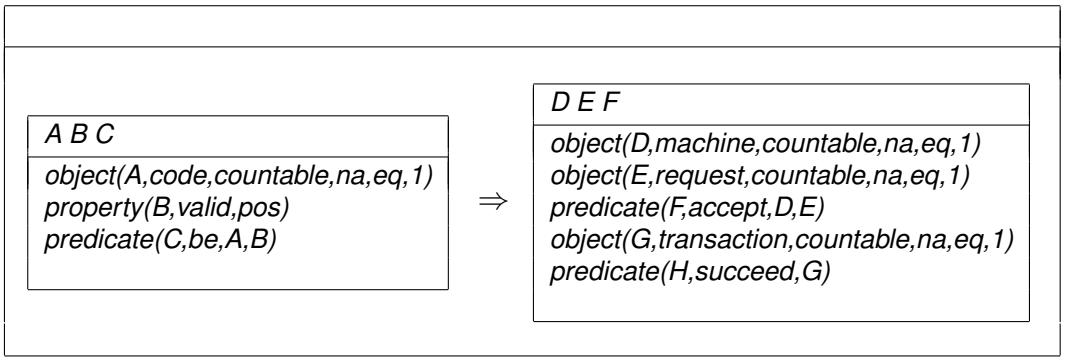
7.1 Conditional Sentences

If the code is valid then the machine accepts the request.

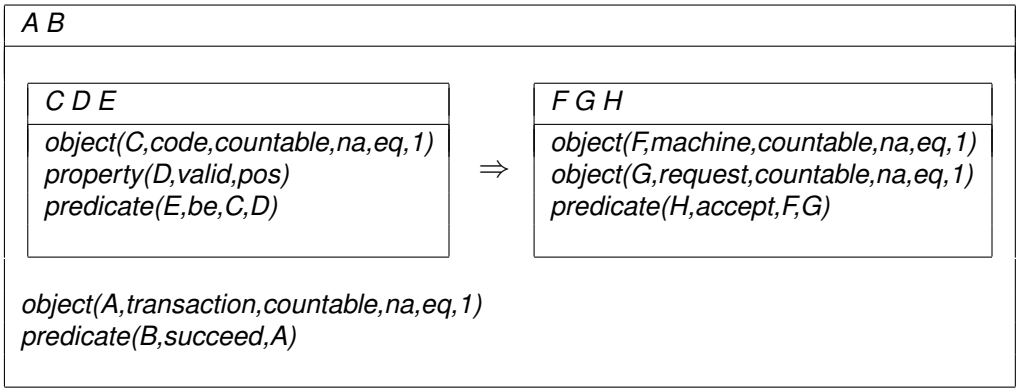


Conditional sentences always take wide scope. Narrow scope requires starting a new sentence.

If the code is valid then the machine accepts the request and the transaction succeeds.



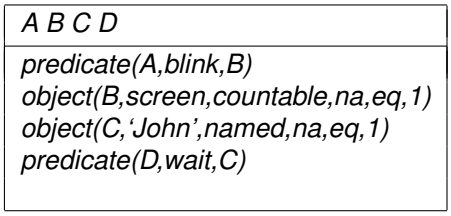
If the code is valid then the machine accepts the request. The transaction succeeds.



7.2 Coordinated Sentences

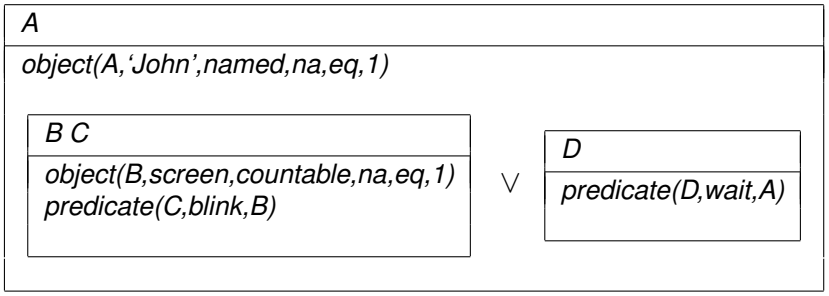
7.2.1 Sentence Conjunction

The screen blinks and John waits.



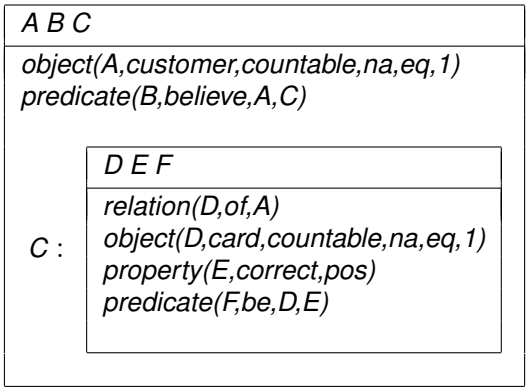
7.2.2 Sentence Disjunction

A screen blinks or John waits.



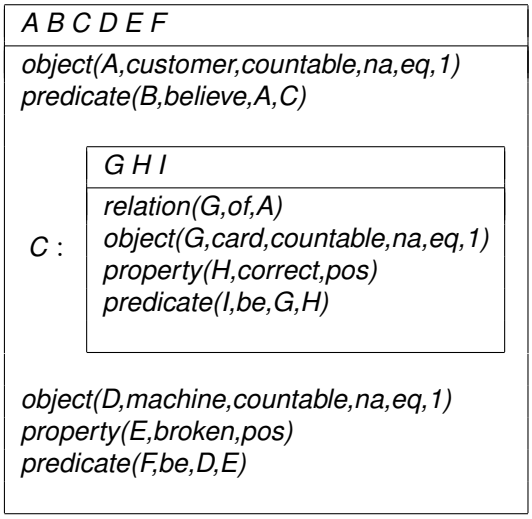
7.3 Sentence Subordination

A customer believes ***that*** *his own card is correct.*

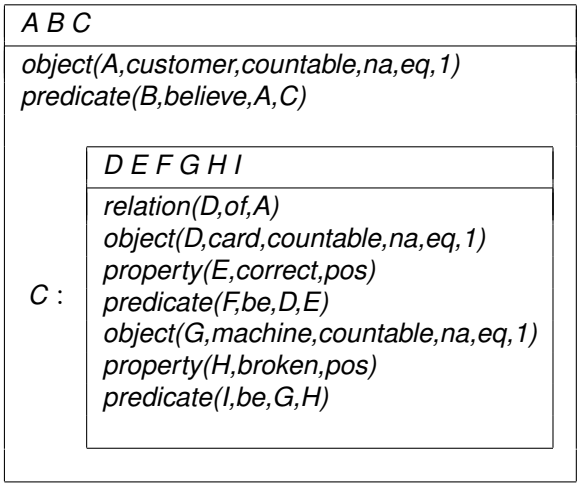


Sentence subordination takes narrow scope unless the word "*that*" is repeated.

*A customer believes **that** his own card is correct and the machine is broken.*



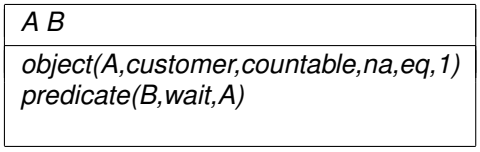
*A customer believes **that** his own card is correct and **that** the machine is broken.*



7.4 Positive Sentence Marker

For consistency reasons, we support the sentence-initial phrase "*It is true that ...*". It does not make much sense for normal sentences but it is useful for macros (see section 11).

***It is true that** a customer waits.*



7.5 Formulas

10 = 4 + 6.

formula(int(5),=,int(3))

5 > 3.

formula(int(5),>,int(3))

X >= 13.4.

object(A,X,named,na,eq,1) *formula(A,>=,real(13.4))*

3 < 4 and 3 =< 5.

formula(int(3),<,int(4)) *formula(int(3),=<,int(5))*

8 Quantified Sentences

8.1 Existential Quantification

***A** card ... / **There is a** card.*

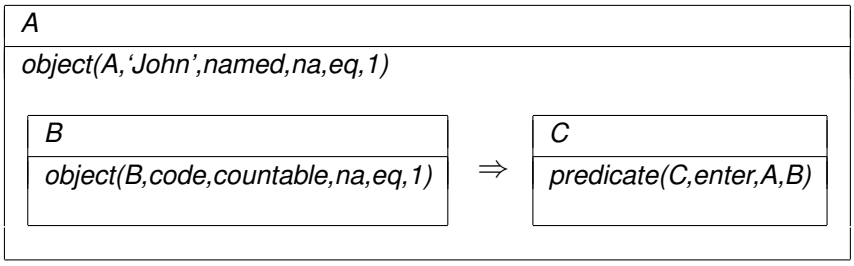
A
object(A,card,countable,na,eq,1)

*John enters **a** card.*

A B C
object(A,'John',named,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)*

8.2 Universal Quantification

*John enters **every** code.*

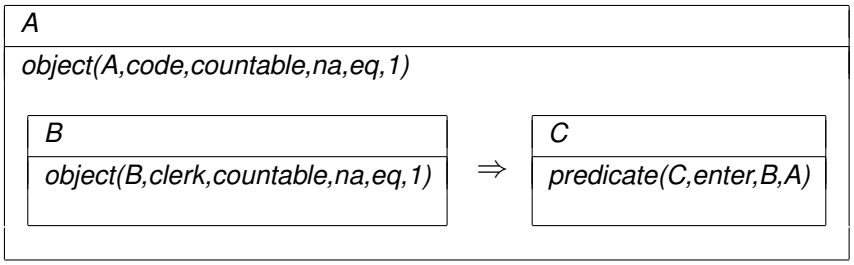


8.3 Global Quantification

8.3.1 Global Existential Quantification

The following two sentences are parsed identically.

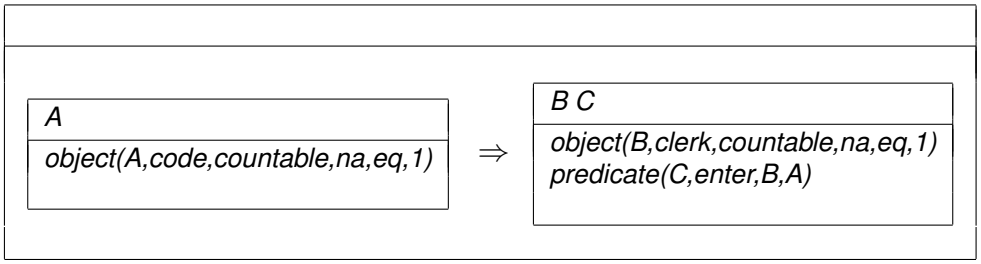
***There is** a code **such that** every clerk enters it.*
***There is** a code **that** every clerk enters.*



8.3.2 Global Universal Quantification

The following two sentences are parsed identically.

***For every** code a clerk enters it.*
***For every** code there is a clerk such that he enters it.*



9 Negation

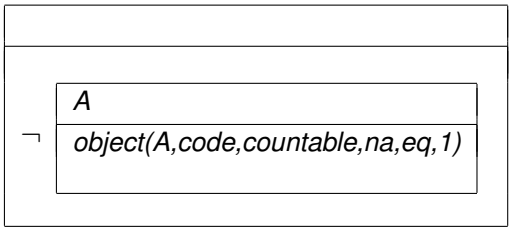
Unless stated otherwise, we talk about classical negation. For negation as failure see subsection 9.4.

9.1 Quantor Negation

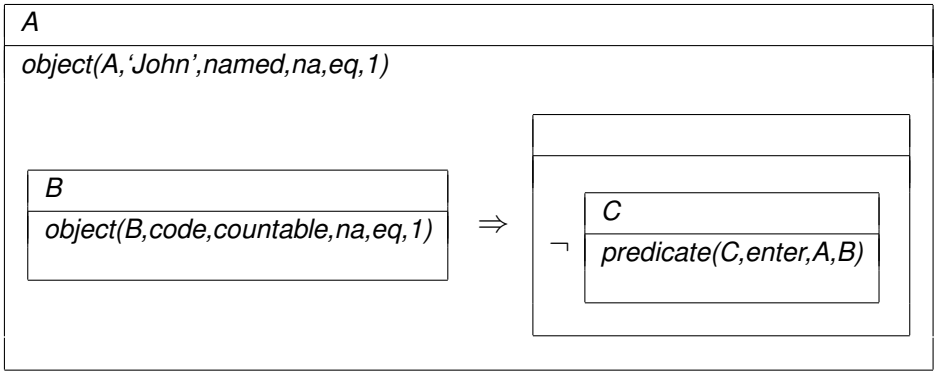
9.1.1 Negated Existential Quantor

Note that negated existential quantors can produce different DRS representations, depending on the context. Within "*there is ...*", a negated sub-DRS is created. Otherwise, we get an implication with a negated sub-DRS on the right hand side.

*There is **no** code.*

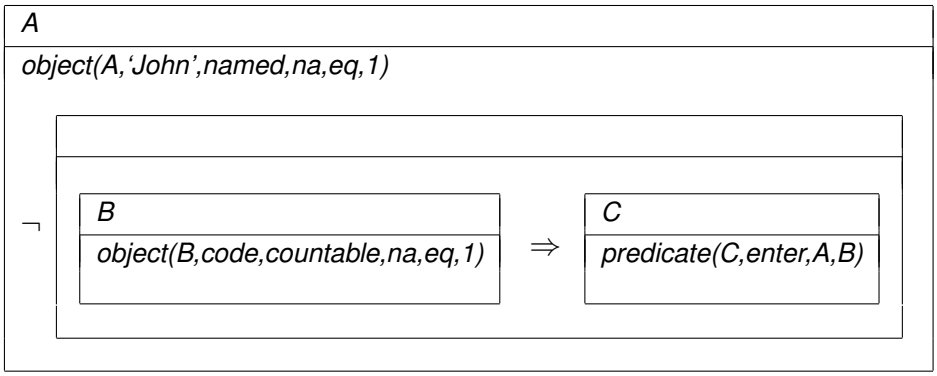


*John enters **no** code.*



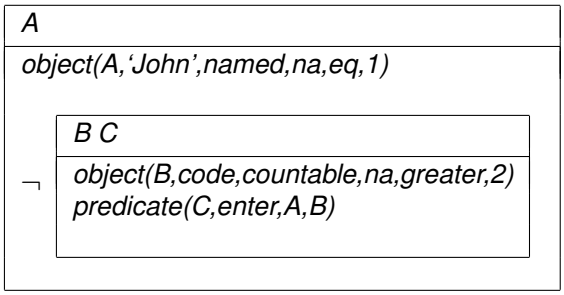
9.1.2 Negated Universal Quantor

*John enters **not every** code.*

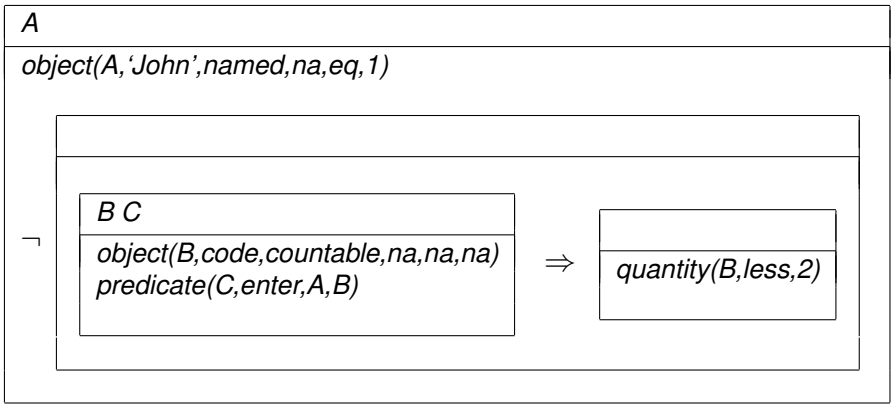


9.1.3 Negated Generalised Quantors

*John enters **not more than** 2 codes.*

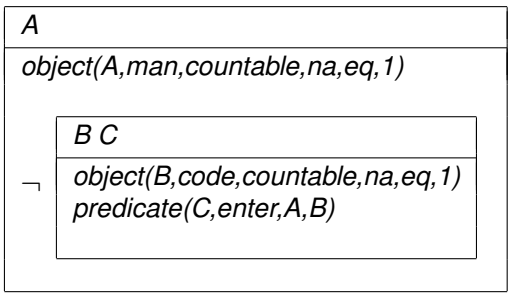


*John enters **not less than** 2 codes.*

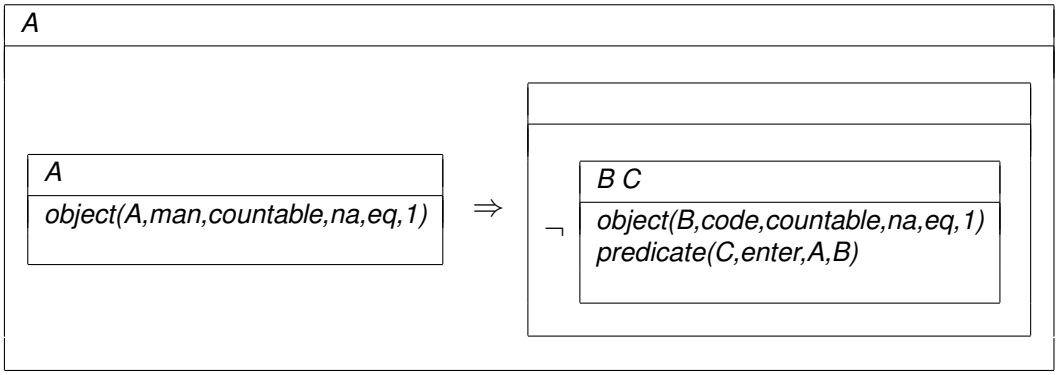


9.2 Verb Phrase Negation

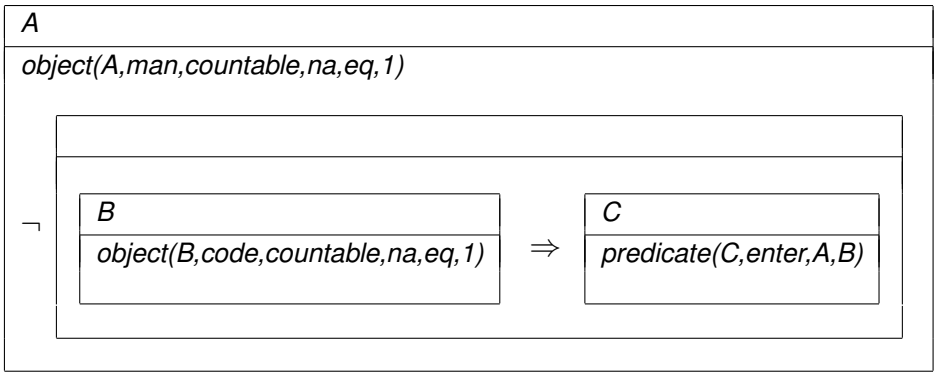
*A man **does not** enter a code.*



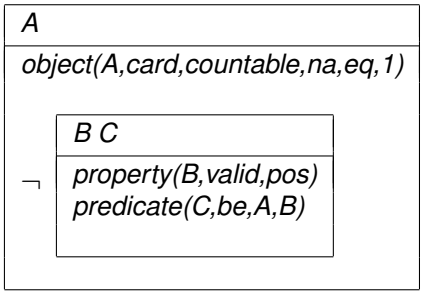
*Every man **does not** enter a code.*



*A man **does not** enter every code.*

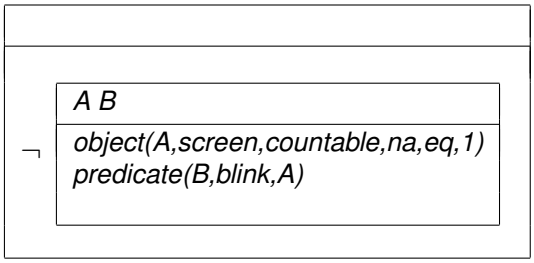


*A card is **not** valid.*

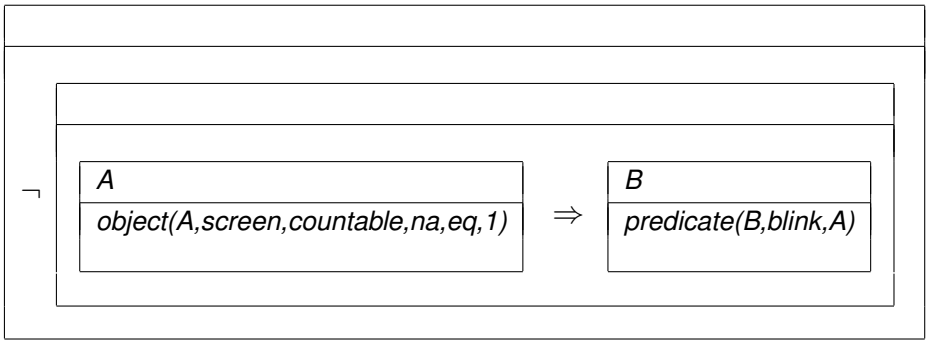


9.3 Sentence Negation

***It is false that** a screen blinks.*

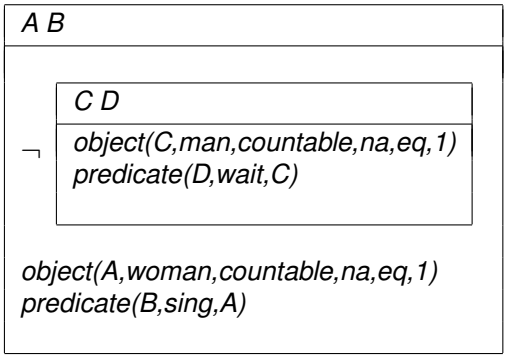


***It is false that** every screen blinks.*

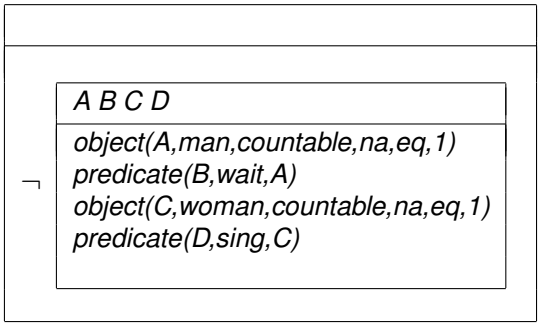


Sentence negation takes narrow scope, but wide scope can be triggered by repeating the *that* complementizer. Compare the following two examples.

It is false that *a man waits and a woman sings.*



It is false that *a man waits and* ***that*** *a woman sings.*



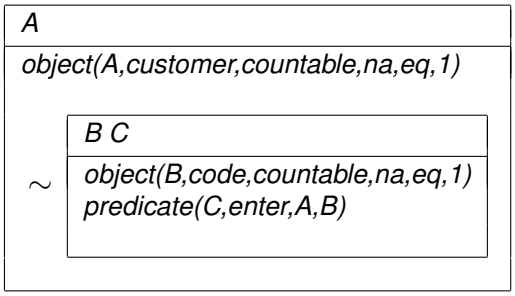
9.4 Negation as Failure

There are two ways to express negation as failure (NAF). First, one can use the construct "*... not provably ...*" for verb phrase negation. Second, the predefined phrase "*It is not provable that ...*" can be used for sentence negation.

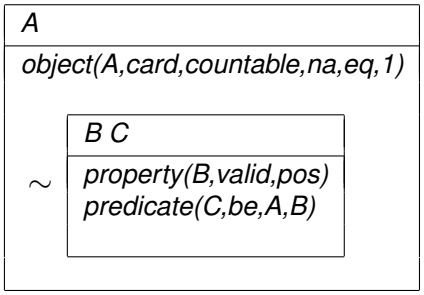
9.4.1 Verb Phrase Negation for NAF

The construct "*... not provably ...*" can be used for all the cases of verb phrase negation as explained in section 9.2.

A customer does ***not provably*** *enter a code.*

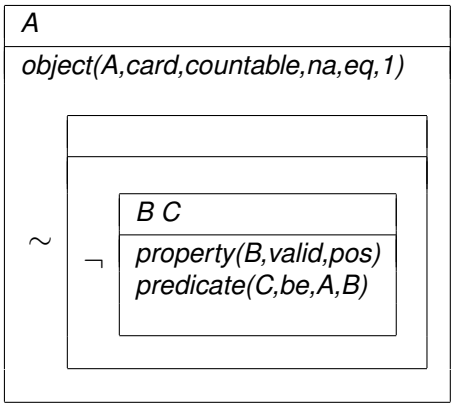


*A card is **not provably** valid.*



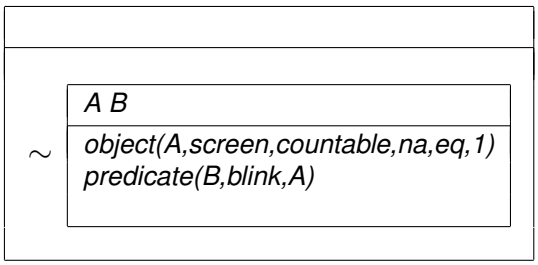
Furthermore, classical negation can be directly nested inside of negation as failure.

*A card is **not provably** not valid.*



9.4.2 Sentence Negation for NAF

***It is not provable that** a screen blinks.*



Concerning scoping, it behaves like the classical sentence negation ("*It is false that ...*") explained in section 9.3.

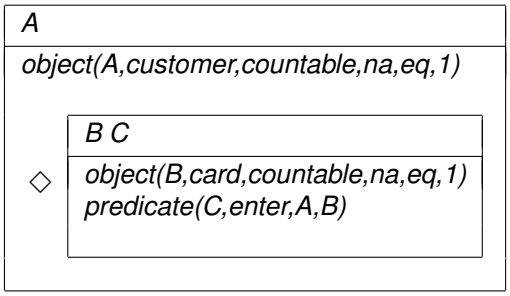
10 Modality

Each of the two forms of modality (possibility and necessity) can be represented in two different ways. First, we can use the modal auxiliary "*can*" or "*must*", respectively. Second, we can use the sentence-initial phrase "*It is possible that...*" or "*It is necessary that ...*", respectively. Negation of these constructs is also allowed (see below for details).

Note that "*a customer can enter a card*" is not equivalent to "*it is possible that a customer enters a card*" (see below).

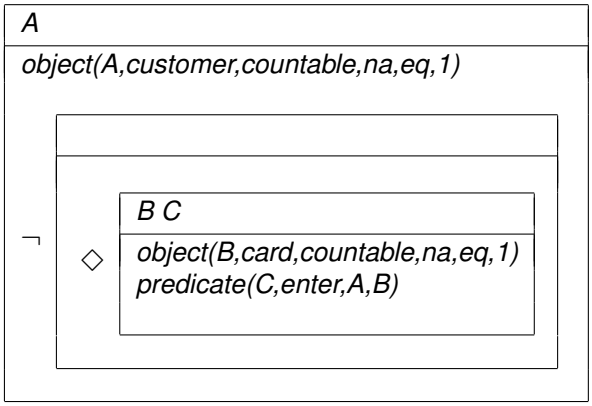
10.1 Possibility

*A customer **can** enter a card.*

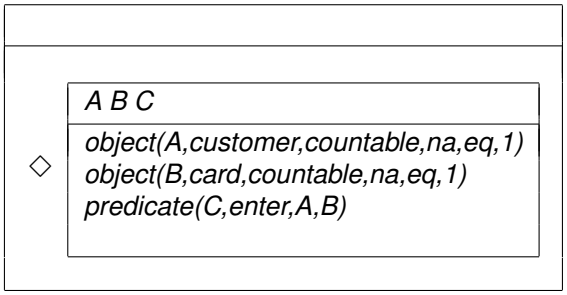


The following three sentences are equivalent.

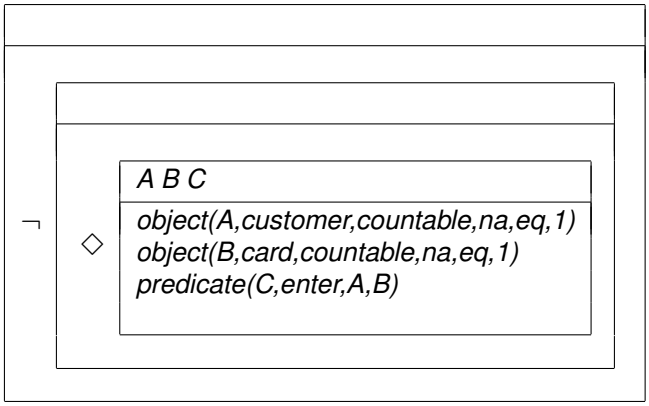
*A customer **can't** enter a card.*
*A customer **cannot** enter a card.*
*A customer **can not** enter a card.*



It is possible that *a customer enters a card.*



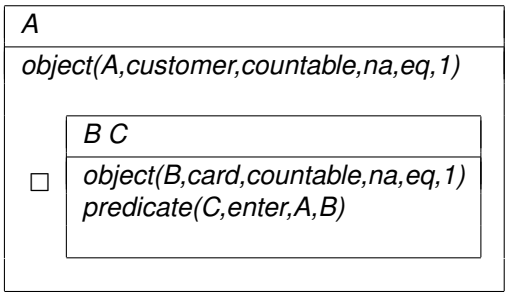
It is not possible that *a customer enters a card.*



10.2 Necessity

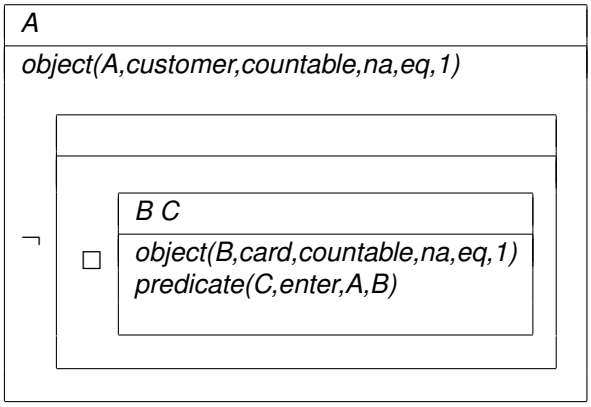
The two synonyms "*must*" and "*has to*" can be used.

A customer ***must*** *enter a card.*
A customer ***has to*** *enter a card.*

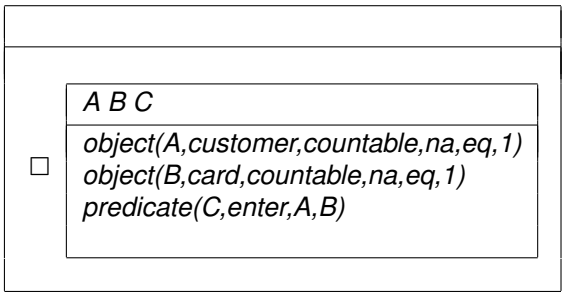


For the negation, only "*does not have to*" is allowed.

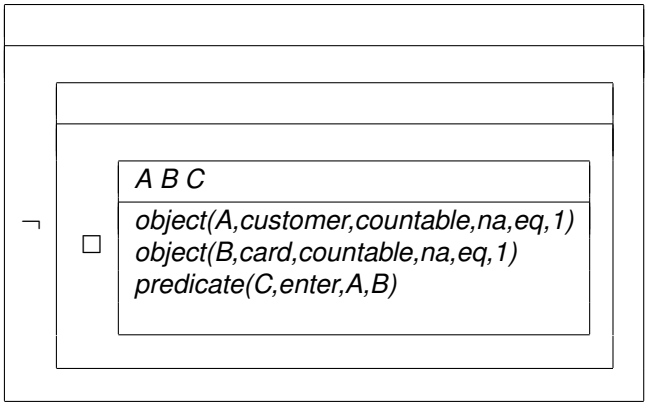
*A customer **does not have to** enter a card.*



***It is necessary that** a customer enters a card.*



***It is not necessary that** a customer enters a card.*



11 Macros

11.1 Macro Definitions

A macro definition has the form "*Proposition [V]: [S]*" where *[V]* is a new variable and *[S]* is a valid ACE Sentence. As long as there is no reference to this macro, no conditions are added to the DRS.

Proposition P: *A customer waits.*

(no conditions)

Alternatively, we can use macros of the form "*Fact [V]: [S]*". In this case the contained conditions are added to the top-most level of the DRS, even if there are no references.

Fact P: *A customer waits.*

A B
object(A,customer,countable,na,eq,1) *predicate(B,wait,A)*

This is equivalent to "*Proposition P: A customer waits. It is true that P.*".

11.2 References to Macros

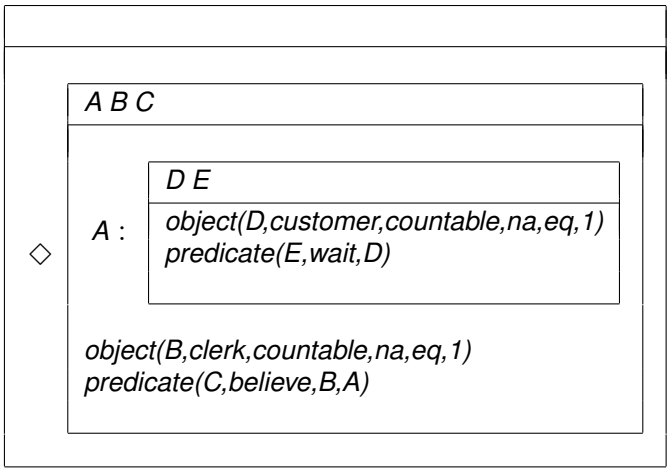
There are multiple possibilities to refer to a macro (we assume that there is a preceding macro definition with the variable *P*):

- It is true/false that P.
- It is (not) possible/necessary that P.
- It is not provable that P.
- John believes that P. *(or any other sentence subordination)*

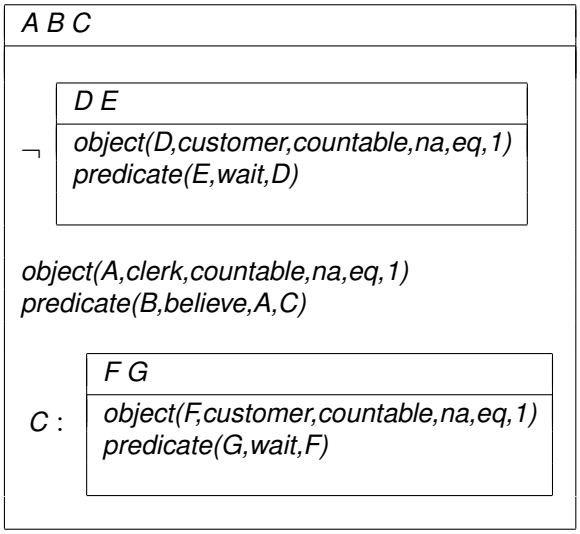
During parsing, each occurrence of a macro is replaced by its definition. Thus, the DRS does not contain any information about macros.

Two examples:

Proposition P: *A customer waits.*
Proposition Q: *A clerk believes that* ***P****.*
It is possible that ***Q****.*



Proposition P: *A customer waits.*
It is false that ***P****.*
A clerk believes that ***P****.*



12 Plural Interpretations

In this section, we present the eight readings of the natural English sentence

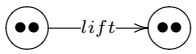
2 girls lift 2 tables.

which can be expressed in ACE. For background information on the disambiguation of plurals consult [6] and [7]. The numbers refer to [6]. Note that reading 4 has two interpretations 4a and 4b and that reading 5 is identical to reading 1.

In ACE, a plural noun phrase has a default collective reading. To express a distributive reading, a noun phrase has to be preceded by the marker *each of*. The relative scope of a quantifier corresponds to its surface position. We use *there is/are* and *for each of* to move a quantifier to the front of a sentence and thus widen its scope.

12.1 Reading 1

$girls$ $tables$

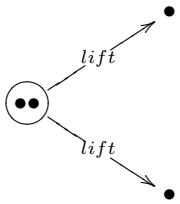


2 girls lift 2 tables.

A B ***C***
object(A,girl,countable,na,eq,2) *object(B,table,countable,na,eq,2)* ***predicate(C,lift,A,B)***

12.2 Reading 2

$girls$ $tables$



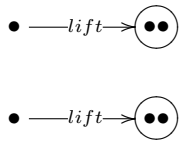
2 girls lift each of 2 tables.

A B
object(A,girl,countable,na,eq,2) *object(B,table,countable,na,eq,2)*

C		***D***
has_part(B,C)	$\Rightarrow$	***predicate(D,lift,A,C)***

12.3 Reading 3

$girls$ $tables$



Each of 2 girls lifts 2 tables.

A
object(A,girl,countable,na,eq,2)

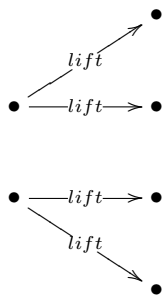
B
has_part(A,B)

$\Rightarrow$

C ***D***
object(C,table,countable,na,eq,2)
predicate(D,lift,B,C)

12.4 Reading 4a

$girls$ $tables$



Each of 2 girls lifts each of 2 tables.

A
object(A,girl,countable,na,eq,2)

B
has_part(A,B)

$\Rightarrow$

C
object(C,table,countable,na,eq,2)

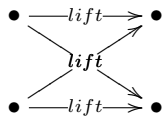
D
has_part(C,D)

$\Rightarrow$

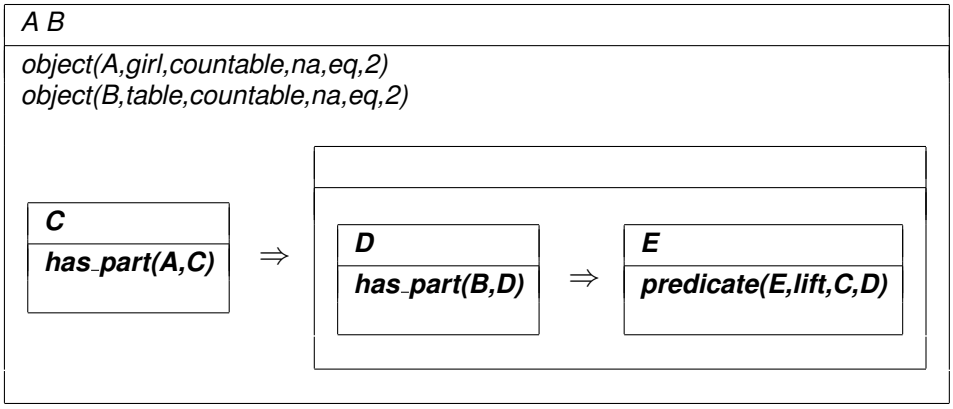
E
predicate(E,lift,B,D)

12.5 Reading 4b

$girls$ $tables$



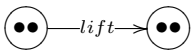
There are 2 girls and there are 2 tables such that each of the girls lifts each of the tables.



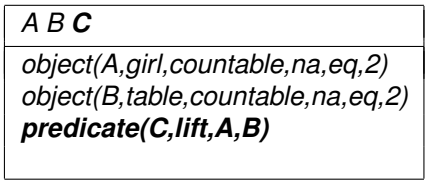
12.6 Reading 5

Reading 5 is identical to reading 1.

$girls$ $tables$

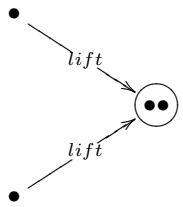


There are 2 tables such that 2 girls lift the tables.



12.7 Reading 6

$girls$ $tables$



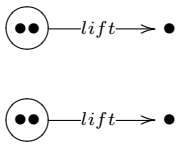
There are 2 tables such that each of 2 girls lifts the tables.

A B
object(A,table,countable,na,eq,2) *object(B,girl,countable,na,eq,2)*

C	⇒	***D***
has_part(B,C)		***predicate(D,lift,C,A)***

12.8 Reading 7

$girls$ $tables$



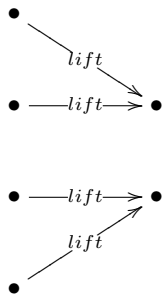
For each of 2 tables 2 girls lift it.

A
object(A,table,countable,na,eq,2)

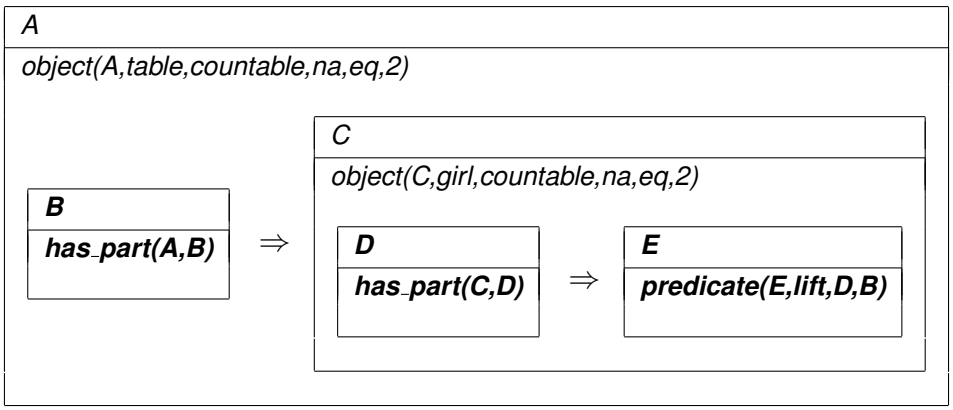
B	⇒	*C* ***D***
has_part(A,B)		*object(C,girl,countable,na,eq,2)* ***predicate(D,lift,C,B)***

12.9 Reading 8

$girls$ $tables$



For each of 2 tables each of 2 girls lifts it.

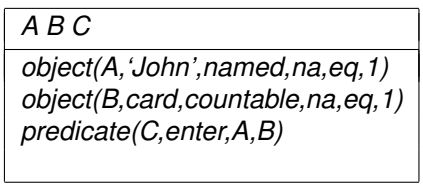


13 Questions

13.1 Yes/No-Questions

Yes/no-questions ask for the existence of a state of affairs. These questions are translated exactly as their declarative counterparts.

Does John enter a card?



Is the card valid?

A B C
object(A,card,countable,na,eq,1) *property(B,valid,pos)* *predicate(C,be,A,B)*

13.2 Who/What/Which-Questions

Who/what/which-questions ask for the subjects or the objects of sentences. These questions are translated as their declarative counterparts but contain additional conditions for the query words.

***Who** enters **what**?*

A B C
query(A,who) ***query(B,what)*** *predicate(C,enter,A,B)*

***Which** customer enters a card?*

A B C
query(A,which) *object(A,customer,countable,na,eq,1)* *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)*

13.3 How/Where/When-Questions

How/where/when-questions ask for details of an action. Concretely they ask for the verb modifications introduced by adverbs and prepositional phrases. They are translated as their declarative counterparts but contain an additional condition for the respective query word.

***How** does John enter a card?*

A B C
object(A,'John',named,na,eq,1) *object(B,card,countable,na,eq,1)* *predicate(C,enter,A,B)* ***query(C,how)***

Where *does John wait?*

A B
object(A,'John',named,na,eq,1) *predicate(B,wait,A)* ***query(B,where)***

When *does John wait?*

A B
object(A,'John',named,na,eq,1) *predicate(B,wait,A)* ***query(B,when)***

References

[1] ACE 6.0 Syntax Report. 2008.
http://attempto.ifi.uzh.ch/site/docs/ace/6.0/syntax_report.html

[2] Attempto project. Attempto website, 2008.
http://attempto.ifi.uzh.ch/site

[3] Patrick Blackburn and Johan Bos. *Working with Discourse Representation Structures*, volume 2nd of *Representation and Inference for Natural Language: A First Course in Computational Linguistics*. September 1999.

[4] Johan Bos. Computational Semantics in Discourse: Underspecification, Resolution, and Inference. *Journal of Logic, Language and Information*, 13(2):139–157, 2004

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

[6] Uta Schwertel. Controlling Plural Ambiguities in Attempto Controlled English. In *Proceedings of the 3rd International Workshop on Controlled Language Applications*, Seattle, Washington, 2000

[7] Uta Schwertel. *Plural Semantics for Natural Language Understanding — A Computational Proof-Theoretic Approach*. PhD thesis, University of Zurich, 2004