REALPRO
General English Grammar
User Manual

August 18, 2000

CoGenTex, Inc.

840 Hanshaw Road, Suite 2
Ithaca, NY, 14850

realpro@cogentex.com

Contents

1 About this Document

This document describes the linguistic representations used and the grammatical coverage provided by REALPRO's general English grammar. More information about the formalisms used in REALPRO, the grammatical theory on which it is based, and standard terminology for English grammar can be found in the References listed in Section 20.

Inputs and outputs for all of the examples used in this document can be found in the folder 'Examples from User Manual', included with the REALPRO distribution. File names are mnemonic, referring to document section and initial fragment of the output sentence. Users may wish to modify the input syntactic representations found in these files, according to the principles given in this manual, to observe REALPRO's modified output.

2 Background: Syntactic Dependency

The input to REALPRO is a syntactic dependency representation for a sentence or sentence fragment. This amounts to a specification for the lexical and syntactic composition of the future sentence in a formalism called a Deep-Syntactic Structure or "**DSyntS**" for short. The formalism is based on the Meaning-Text Theory (MTT) of Igor Mel'cuk and colleagues (see References, Section 20). According to this formalism, a sentence specification has the following salient features:

- Each DSyntS is a *tree* with labeled nodes and labeled arcs.
- A DSyntS is *lexicalized*, meaning that the nodes are labeled with lexemes (uninflected words) from the target language.
- A DSyntS is a *dependency* representation and not a *phrase-structure* representation: there are no nonterminal nodes (such as VPs), and all nodes are labeled with lexemes.
- A DSyntS is a *syntactic* representation, meaning that the arcs of the tree are labeled with syntactic relations such as SUBJECT, rather than conceptual (or "semantic") relations such as "agent".
- The DSyntS is a *deep* syntactic representation, meaning that only meaning-bearing lexemes are represented, and not function words.

This means that REALPRO does not perform the task of lexical choice: the input to REALPRO must specify all meaning-bearing lexemes. Furthermore there is no non-determinism in REALPRO, since the rules are applied in the order in which they are defined, without backtracking. This means that the input to REALPRO fully determines the output, but it represents it at a very abstract level, which is well-suited for interfacing with knowledge-based applications.

3 DSyntS Formalism

3.1 Description

REALPRO takes as input a DSyntS which can be specified either programmatically, or with an ASCII based formalism. This section describes only one of the ASCII based formalism to represent a DSyntS. The input is structured as follows:

- The keyword `DSYNTS:`, followed by
- the specification of a deep-syntactic structure (a DSyntS), followed by
- the keyword `END:`.

A DSyntS is specified as follows:

- A (root) node label, followed optionally by
- a left parenthesis '`(`', an arbitrary (non-null) number of dependent specifications, followed by a right parenthesis '`)`'.

A dependent is specified as follows:

- A specification of a dependency arc,
- followed by a specification of a node.

A dependency arc is specified simply by the arc label. The node specification is explained in Section 4.

3.2 Example

```
// -------------------------------------------------------------------
// This is a test.
// -------------------------------------------------------------------

DSYNTS:

BE [ ]
( I  THIS [ class:demonstrative_pronoun number:sg ]
  II test [class:common_noun article : indef]
)

END:
```

3.3 Notes

- Indentation and line breaking is not relevant. We follow this formatting convention only in order to make the tree-like structure evident.

- Comments can be added before or after the DSyntS specification: i.e. before the keyword `DSYNTS:` or after the keyword `END:`. Comments can consist of any strings except those containing the keyword `DSYNTS:`.
- For testing purposes, the target surface form can be declared before the specification by surrounding it by the keywords `OUTPUT:` and `END:`. This provision is useful during regression testing to automatically check for discrepancies between the target surface form and the realized surface form.
- For the expository purposes of this document, we will often write the lexeme name in upper case when it can be found in the (very limited) REALPRO lexicon provided with this distribution. The inherent features of any such lexeme (e.g., the class:verb and the irregular morphology of the copula BE) can be assumed known by REALPRO. The user need specify between square brackets only the features which may depend on the example (e.g., tense, aspect), and which are not default values. When a lexeme has two or more homonymic forms (e.g., THIS may be a demonstrative article or pronoun), the user must specify the intended lexical class (e.g. "class:demonstrative_pronoun" in the example above). We use lower case here for words which are not in REALPRO's lexicon. For these, the user must specify all information about class, feature values, any irregular morphology, etc. See below for details and examples. Note that the user is not obliged to observe the upper/lower case convention, since all lexeme names (except for proper nouns and acronyms) are converted to lower case before grammar and formatting rules are applied.

4 Nodes

4.1 Description

A node is formally specified as follows:

- A lexeme name (e.g., 'important'), followed by
- A list of features.

A list of features is specified by:

- A left bracket '`[`', followed by
- A possibly empty list of feature-value pairs of the form *feature*`:`*value* (separated by spaces), followed by
- A right bracket '`]`'.

Features are optional if defaults are provided, except that a lexeme which is not in the REALPRO lexicon must have the `class` feature.

At this point in REALPRO's development (August 2000), there is no attempt to supply a broad lexicon of English. The distributed software provides explicit lexical entries only for: (1) important closed-class words (e.g., articles, conjunctions, prepositions, quantifiers), (2) some frequent verbs and nouns having irregular forms, and (3) a few other frequent verbs, nouns, adjectives and adverbs used for testing purposes. Applications using REALPRO as an output generator will therefore need to include some linguistic information about most of the lexical forms appearing in the DSyntSs which the application sends to REALPRO. In particular, each lexeme must be marked for lexical class.

A lexeme not in REALPRO's lexicon is labelled by its root form (uninflected). Most lexemes with regular morphology and regular syntactic behavior are not currently included in the lexicon. Multi-word lexemes use underscores (e.g., '`in_order_to`'), which are converted to spaces in the output.

For open-class words, the feature `class` can have the following values:

Value of **`class:`**	**Example**
adjective	*small, disastrous*
adverb	*really, maybe, surprisingly*
common_noun	*table, map*
proper_noun	*John, Poona, Socks*
verb	*disintegrate, indulge*
symbol	+

For closed-class words, the feature `class` can have the values shown below. Sample entries are given for each class. When the lexeme is calculated from feature values, and not represented as a lexical item in DSyntS, it is written here in lower case.

Value of the feature **`class:`**	**Sample Lexemes**
article	*those, the, a*
coordinating_conj	*AND, BUT, OR*
demonstrative_pronoun	*THIS, THAT, THESE, THOSE*
numeral	*TWELVE*

particle	*not*
indefinite_pronoun	*ANYTHING*
preposition	*ABOUT*
quantifier	*ALL*
subordinating_conj	*IF*

4.2 Examples of Single-Node DSyntS

```
// ------------------------------------------------------------------
// Mesmerizingly.
// ------------------------------------------------------------------

DSYNTS:

mesmerizingly [ class: adverb]

END:


// ------------------------------------------------------------------
// **&FuN aNd GaMeS&**.
// ------------------------------------------------------------------

DSYNTS:

**&FuN_aNd_GaMeS&** [class:proper_noun]

END:
```

4.3 Notes

- Extra spaces do not matter within the lexeme-feature list combination.
- Ordering of the feature-value pairs does not matter in the list of features.
- By default, the output is formatted as a sentence, with an initial capitalization and a final period. To eliminate the sentence-final period, add `punct:no_dot` to the features of the root verb. See Section 18.1 for details. To avoid initial capitalization, use `caps:none in the root lexeme features`. See Section 17 for details.

5 Nouns

5.1 Description

There are two subtypes of nouns, common nouns and proper nouns.

Nouns have four features: for number, for gender, for case, and for determiner type.

- The feature `number` can have the following values:

`number:`	**Example**	**Default**
sg	*bean*	√
pl	*beans*	

- The feature `gender` can have the following values:

`gender:`	**Example**	**Default**
masc	*boy*	√
fem	*waitress*	
neut	*piano*	
dual	*teacher*	

Specifying `gender:dual` should enable pronominal choices such as "he or she," but this is not currently implemented.

- The feature `case` can have the following values:

`case:`	**Example**	**Default**
nom	*bean, he*	√
gen	*bean's, his*	
obj	*bean, him*	

In English, this feature is not normally used in an input DSyntS for REALPRO (see notes below).

- Definite, indefinite, and demonstrative determiners can be introduced through features. The feature `article` can have the following values:

`article:`	**Example, singular noun**	**Example, plural noun**	**Default**
indef	*a tiara*	*tiaras*	for common nouns
def	*the tiara*	*the tiaras*	
dem-prox	*this tiara*	*these tiaras*	
dem-dist	*that tiara*	*those tiaras*	
no-art	*tiara, Tirana*	*tiaras*	for proper nouns

5.2 Examples

```
// -------------------------------------------------------------------
//  The Yemen.
// -------------------------------------------------------------------

DSYNTS:

Yemen [class:proper_noun article:def]

END:


// -------------------------------------------------------------------
//  Tiaras.
// -------------------------------------------------------------------

DSYNTS:

tiara [class:common_noun number:pl]

END:


// -------------------------------------------------------------------
//  These cars.
// -------------------------------------------------------------------

DSYNTS:

car [class:common_noun article:dem-prox number:pl]

END:
```

5.3 Notes

- The feature combination `article:indef number:pl` (`article:indef` being the default for nouns) yields the bare plural, (*tiaras*). To obtain *some* as a determiner, use lexeme *SOME* as an ATTR to the noun. Note that *SOME* does not have a number, so to obtain *some tiaras*, you need to indicate `number:pl` on the noun. See `Noun/some-duck.dss` and `Noun/some-ducks.dss`.
- In English, the distinction between dative and accusative cases does not exist (overtly). Instead, we use the term "objective" case to cover both (`case:obj`).
- Features for case are added by the grammar as needed. The only times a case feature should be specified in the input to REALPRO is `case:obj` for the "AcI" construction (see Section 12.4).
- For other determiners (numerals, demonstratives -- *those four tiaras*), see Section 6.
- For the possessive construction (*John's tiara*), see Section 7.
- For noun compounds (*diamond tiara*), relate the two nouns using ATTR. You can also specify a single noun node (`diamond_tiara`).
- To append material after nouns, use the APPEND relation. For example, in *my son Desmond*, *Desmond* depends on *son* by an APPEND arc. This arc label can also be used to add parentheses (see Section 18 for details on using parentheses).
- Feature `human` (as most semantic features) is not currently used in REALPRO for English; instead, the relevant distinctions can be made using `gender:dual` or `gender:neut`.

5.4 Shortcomings

- Specification `gender:dual` should yield pronominal choices such as "*his or her*", but it is not currently implemented. Use `gender:masc` or `gender:fem` instead.

6 Determiners

6.1 Description

As mentioned in Section 5, at the deep-syntactic level, the definite, indefinite, and demonstrative articles are specified with the feature `article` which can have one of the following values: `def`, `indef`, `dem-prox`, `dem-dist`, or `no-art`.

Other determiners should be added as ATTR dependents of the noun. They fall into two categories, quantifiers (*all*, *many*, and so on) and numerals (*one*, *six*, and so on). The difference between the two categories is that quantifiers are ordered before articles, while numerals are ordered after articles, and after quantifiers (*all the boys*, *any seven boys*, *the one thing*).

The following quantifiers can be found in the lexicon. The table shows the grammatical number that will be forced on an underspecified head noun by twelve of the eighteen quantifiers. Nouns such as *fish* or *series*, which can be either singular or plural, will take the grammatical number of the quantifier in those cases. In addition, three of the quantifiers used with singular form will induce the feature `uncountable:+` on their head noun. Thus *much* as an ATTR of the noun *fish* in *much fish* will induce the uncountable feature in the noun's specification.

At present, no error message is given when an incompatible quantifier-noun combination is specified in DSyntS, as in *much beans*, and generation may succeed. Future development of the determiner should include error messages.

Quantifier in lexicon	**Grammatical number**	**Induces `uncountable:+ ?`**
A_FEW	pl	
A_LITTLE	sg	**yes**
ALL	none	
ANY	none	
BOTH	pl	
EACH	sg	
EITHER	sg	
EVERY	sg	
FEW	pl	
LITTLE	sg	**yes**
MANY	pl	
MORE	none	
MOST	none	
MUCH	sg	**yes**
NEITHER	sg	
NO	none	
SEVERAL	pl	

SOME	none	

The following numerals can be found in the lexicon. The cardinal numerals can be referred to either by a spelled-out lexeme, or by an integer. The table shows the number agreement that is forced on the head noun by the numeral.

Numeral in lexicon	**Alternate integer representation**	**Number agreement**
ZERO		pl
ONE	1	sg
TWO	2	pl
THREE	3	pl
FOUR	4	pl
FIVE	5	pl
SIX	6	pl
SEVEN	7	pl
EIGHT	8	pl
NINE	9	pl
TEN	10	pl
ELEVEN	11	pl
TWELVE	12	pl

The integer version of the lexeme (but not the full-word version) has a feature `form`, which can take the following values:

`form:`	**Example**	**Default**
`word`	*twelve*	
`roman`	*XII*	
`arabic`	*12*	√

6.2 Examples

```
// -------------------------------------------------------------------
// The four ducks.
// -------------------------------------------------------------------

DSYNTS:

duck [ class:common_noun article:def ]
(
  ATTR  4 [form:word ]
)

END:


// -------------------------------------------------------------------
// All the ducks.
// -------------------------------------------------------------------

DSYNTS:

duck [ class:common_noun article:def ]
(
  ATTR  ALL [ ]
)

END:


// -------------------------------------------------------------------
// More than six ducks.
// -------------------------------------------------------------------

DSYNTS:

duck [ class:common_noun ]
(
  ATTR 6 [ form:word ]
  (
    ATTR more_than [class:adverb]
  )
)

END:

// -------------------------------------------------------------------
// All 14 of the ducks.
// -------------------------------------------------------------------

DSYNTS:

duck [ class:common_noun node-real:- ]
(
  ATTR 14 [ class:numeral ]
  ATTR ALL []
  II duck [ class:common_noun article:def number:pl ]
```

```
   )
)

END:
```

6.3 Notes

- Quantifiers do not remove any articles or other determiners from the head noun, in order to allow *all the women*. However, REALPRO would also generate **all some women* -- it is up to the specification of the input DSyntS to avoid such constructions.

- Numerals remove any indefinite articles from the head noun, but allow for the specification of a definite article (e.g., *the four ducks*).

- An application that sends a DSyntS to REALPRO should normally indicate the intended grammatical number for invariant nouns such as *fish* and *series*, and include the `uncountable:+` feature on any mass noun or other noun to be used in an uncountable sense. The enforcement of noun number and countability features indicated in the tables above will be used to support new features in the grammar at a later date.

- When specifying a quantifier not in the lexicon, use `class:quantifier`. When specifying a numeral not in the lexicon, use `class:numeral`.

- To add *more than...* to a numeral, there are two possible analyses supported by REALPRO. First, you may add *MORE* as an ATTR to the noun, and then add the numeral as a II-argument to the *MORE*. In this case, *than* will be introduced automatically by a grammar rule, and be properly placed before the numeral. A second analysis, preferred by some linguists, is to treat *more than, less than,* etc. as compound degree adverbs (cf. *very, approximately*). It is not necessary to mark the adverb type, since the default positioning of ATTRs works well for degree adverbs. See the third example above. Such degree adverbs can be reused in other contexts (*you are more than welcome, he was less than candid,* etc).

- Complex determiners containing *of* (e.g., *any three of the seven black ducks*) are currently generated by introducing a dummy head noun (here, *duck*) before the *of*, marked with the feature ‘node-real:- ‘ This allows the whole noun phrase to use the normal provision for noun complements, i.e., making what follows *of* a II-argument of the dummy noun (see the last example above; see Section 11 for general information on noun arguments). This device of unrealized head noun can also be used elsewhere (cf. All six were stolen), and some generation applications may trigger the contextual ellipsis of the head noun by control of this feature ‘`node-real`’.

6.4 Shortcomings

Some complex determiners have not been implemented, and this part of the grammar is currently under review. Comments and suggestions are welcome at realpro@cogentex.com.

7 Noun Arguments and the Possessive Construction

7.1 Description

A noun can have arguments, just as a verb, though the meaning of “argument” is of course somewhat different. REALPRO supports up to two arguments for nouns:

- The first argument (arc label I) is realized as the genitive adjunct to the head noun, as in *John’s dream*.

- The second argument (arc label II) is realized as a prepositional adjunct to the head noun with *of*, as in *the destruction of the city*.

In addition, lexical entries can be added in order to have certain nouns realize their arguments in a lexically idiosyncratic way (such as *the desire for justice*).

7.2 Example

```
// -------------------------------------------------------------------
// Our esteemed friend’s two bland definitions of happiness.
// -------------------------------------------------------------------

DSYNTS:

"definition" [ class:common_noun ]
(
  ATTR  "bland" [ class:adjective ]
  I  "friend" [ class:proper_noun ]
  (
    ATTR  "<POSSESSIVE_PRONOUN>" [ number:pl person:1st ]
    ATTR  "esteemed" [ class:adjective ]
  )
  II  "happiness" [ article:no-art class:common_noun ]
  ATTR  "TWO" [ ]
)

END:
```

7.3 Notes

The linguistic analysis of noun arguments is complex, and the implementation in REALPRO is clearly inadequate.

- A first argument on a noun suppresses any other articles (*the*, *a*). However, REALPRO will generate a demonstrative article with a genitive if it is so specified in the input DSyntS (**those John’s books* - which is (marginally) possible in language such as German).

7.4 Shortcomings

- Following the plural -*s*, the genitive should be realized simply as apostrophe (*'*), but is not.

8 Pronouns

8.1 Description

Pronouns are generated by the morphological component based on features present on nodes. There are basically two ways of specifying the use of a pronoun:

- Use a special lexical entry (such as <PRONOUN>). This lexical entry introduces special features and handles the absence of articles. The entry still needs to specify number and person.
- Use a `pro` feature on a noun. In this case, the noun is realized as a pronoun.

The following pronominalizations are handled by the grammar and should not normally require annotation in the input DSyntS:

- Reflexives are added by the grammar in cases in which their appearance is determined grammatically. Specifically, if the 1st and 2nd argument have the same value for the `ref` feature (or the first and third or the second and third), then the latter argument is replaced by a reflexive.
- Relative pronouns are added by the grammar. See Section 16 for details.

Here is a list of the types of pronouns currently supported:

Pronoun Type	**Example**	**Special Lexeme(s)**	**Feature**
demonstrative pronoun	*that*	THIS, THAT, THESE, THOSE	none
indefinite pronoun	*anything*	ANYBODY, ANYONE,…	`none`
personal pronoun	*he, I*	<PRONOUN>	`pro:pro`
possessive pronoun	*his*	<POSSESSIVE_PRONOUN>	`pro:poss`
reflexive pronoun	*themselves*	<REFLEXIVE_PRONOUN>	none
relative pronoun	*which*	none	`pro:rel`

8.2 Examples

```
// ------------------------------------------------------------------
// I would do anything for her.
// ------------------------------------------------------------------

DSYNTS:
DO [ mood:cond ]
(
  I  "<PRONOUN>"  [ number:sg person:1st ]
  II anything     [ class:indefinite_pronoun  person:3rd  number:sg  ]
  (
    ATTR  FOR  [ ]
    (
       II girl [ class:common_noun gender:fem article:def pro:pro ]
    )
  )
)
END:

// ------------------------------------------------------------------
// John sees himself.
// ------------------------------------------------------------------

DSYNTS:

see [ class:verb ]
(
  I  John     [ class:proper_noun ref:J1 ]
  II John     [ class:proper_noun ref:J1 gender:masc ]
)

END:



// ------------------------------------------------------------------
// The psychiatrist revealed the patient to herself.
// ------------------------------------------------------------------

DSYNTS:

reveal [ class:verb tense:past ]
(
  I   psychiatrist  [ class:common_noun article:def ]
  II  patient       [ class:common_noun ref:P1 article:def ]
  III patient       [ class:common_noun ref:P1 article:def gender:fem ]
)

END:


// ------------------------------------------------------------------
// Almost anything else would be something new.
// ------------------------------------------------------------------

DSYNTS:

BE [ mood:cond ]
```

```
  (   I  "ANYTHING" [ class:"indefinite_pronoun" ]
       (
         ATTR  "almost" [ class:"adverb" ]
         ATTR  "else" [ class:"adjective" ]
        )
     II  "SOMETHING" [class:"indefinite_pronoun"]
      (
         ATTR  "new" [ class:"adjective" ]
        )
   )

END:
```

8.3 Notes

- If the second and third arguments are co-referential, REALPRO always generates the third argument as a *to*-phrase (irrespective of the `rheme` feature), with a reflexive pronoun.
- The grammar overrides any indications of article on a pronoun to be pronominalized (as in the third example above).
- Indefinite pronouns (e.g., *anyone, everybody, nothing, something*, etc.) can have post-posed adjectives (e.g., *something important, nobody else*). REALPRO will post-pose any adjectives linked to an indefinite pronoun (feature value '`pro:indef`') by an ATTR arc (see the final example above).

9 Adjectives

9.1 Description

Adjectives can be attached to nouns in DSyntS in two ways:

- By the ATTR arc, in which case they appear pre-nominally (the usual case).
- By the DESC-ATTR arc, in which case they will appear post-nominally, set off by commas.

9.2 Example

```
// ----------------------------------------------------------------------
// Two eggs, small.
// ----------------------------------------------------------------------

DSYNTS:

egg [class:common_noun article:no-art number:pl]
(
  ATTR TWO
  DESC-ATTR small [class:adjective]
)
```

```
END:
```

9.3 Notes

ATTR and DESC-ATTR arcs are repeatable.

9.4 Shortcomings

The comparative (*bigger*) and the superlative (*biggest*) have not yet been implemented through features. They can of course be obtained simply by specifying them in the input. There is also no provision for ordering among adjectives according to semantic type. Therefore the order given in DSyntS becomes the order used for the output sentence. Future work will allow for the optional typing of adjectives to allow REALPRO to determine the correct order.

10 Verbs

10.1 Description

This is an exhaustive list of verbal features.

- Feature `tense` can have the following values.

`tense:`	**Example**	**Default**
`pres`	*John likes Mary.*	√
`past`	*John liked Mary.*	
`future`	*John will like Mary.*	

Feature `tense` is not meaningful in conjunction with a non-finite mood (see below). REALPRO will usually ignore the value of the `tense` feature.

- Feature `voice` can have the following values:

`voice:`	**Example**	**Default**
`act`	*John likes Mary.*	√
`pass`	*John is liked.*	

For information on the argument structure in passive voice, see Section 11.3.

- Feature `aspect` can have the following values:

`aspect:`	**Example**	**Default**
`simple`	*John eats beans.*	√
`cont`	*John is eating beans.*	

- Feature `taxis` can have the following values:

`taxis:`	**Example**	**Default**
`nil`	*John likes Mary*	√
`perf`	*John has liked Mary*	

- Feature `mood` can have the following values:

`mood:`	**Example**	**Default**
`ind`	*John likes Mary.*	√
`cond`	*John would like Mary.*	
`imp`	*Call Mary.*	
`inf`	*John like Mary.*	
`inf-to`	*For John to like Mary (would be a problem).*	
`pres-part`	*John liking Mary (is a problem).*	
`past-part`	*Given the book, (Mary disappeared).*	

The subjunctive (*lest John cause trouble* in the present, *if John were mistaken* in the past) is currently not supported. (However, the present subjunctive is always morphologically identical to the bare infinitive in English.)

The combination of imperative and question is not supported.

Subject-auxiliary inversion only happens with finite auxiliaries.

- Feature `polarity` can have the following values:

`polarity:`	**Example**	**Default**

`nil`	*John likes Mary*	√
`neg`	*John does not like Mary*	

- Feature `question` can have the following values:

`question:`	**Example**	**Default**
-	John likes Mary	√
+	Does John like Mary	

This feature is used in the input DSyntS only to specify yes/no questions. For information on *wh*-questions, see Section 13.

10.2 Example

```
// -------------------------------------------------------------------
// John does not love Mary.
// -------------------------------------------------------------------

DSYNTS:

love [ class:verb tense:pres polarity:neg ]
( I  John [ class:proper_noun ]
  II Mary [ class:proper_noun ]
)

END:
```

10.3 Notes

- These verbal features can be combined in any way, though the non-finite moods (infinitive with or without *to* and the present and past participles) and the conditional mood do not have tense, and do not choose past, present, or future. (Specifications of tense in such cases will be ignored.)

- For the imperative (`mood:imp`), the generator will not automatically remove the subject, in order to allow for constructions such as *You be on time!*. Furthermore, the exclamation mark is not generated automatically either. See Section 18.

- Modal auxiliaries such as *can* and *may* are not recursive in English (**I can may come*), and therefore they should be attached as an ATTR to the main verb. Put differently, in *John can play tennis*, *John* and *tennis* are arguments I and II, respectively, of *play*.

  The following is a complete list of modal auxiliaries which have entries in the lexicon: *CAN*, *MAY*, *MUST* and *SHOULD*. These modal auxiliaries need not be given any feature list. Other modal auxiliaries should be specified as `class:verb modal-aux` (note the space) in their feature list. An example is shown below.

- The copula *be* is treated as the head of its clause, with the subject as argument I, and the adjective, noun, adverb or prepositional phrase which is predicated of the subject as argument II.
- For information on the argument structure in passive voice, see Section 11.3.
- For information on *wh*-questions, see Section 13.

10.4 Shortcomings

- The subjunctive is not handled.

11 Clauses and Sentences

11.1 Description

Clauses and sentences are constructed by giving arguments to verbs. The arguments of the verb are labeled I, II, III, and IV. Argument I always corresponds to what is usually called the "subject", and II to IV correspond to objects of decreasing proximity to the verb (direct object, indirect object, additional complement).

The feature `extrapo` governs the realization of the basic sentence structure. Currently, two variants are supported: extraposition of the subject with impersonal *it* (*It bothers me that she is here*), and *there* insertion with existential-type verbs (*There appeared three geese in the study*). Feature `extrapo` can have the following values:

`extrapo:`	**Example**	**Subject must be…**	**Default**
`-`	*Geese are in the garden* (subject is NP)	clause or NP	√
	That she is here bothers me (subject is clause)		
`there`	*There are geese in the garden*	NP	
`i`	*It bothers me that she is there*	Clause	

11.2 Examples

```
// ---------------------------------------------------------------------
// John loves Mary.
// ---------------------------------------------------------------------

DSYNTS:

love [ class:verb ]
( I  John [ class:proper_noun ]
  II Mary [ class:proper_noun ]
)

END:


// ---------------------------------------------------------------------
// John tells Mary a story.
// ---------------------------------------------------------------------

DSYNTS:

tell [ class:verb ]
( I  John [ class:proper_noun ]
  III Mary [ class:proper_noun ]
  II story [ class:common_noun ]
)

END:


// ---------------------------------------------------------------------
// Have there not been firefighters available in this city?
// ---------------------------------------------------------------------

DSYNTS:
BE [ class:verb extrapo:there polarity:neg question:+ taxis:perf ]
(
  I  firefighter [ class:common_noun number:pl ]
  ATTR  IN [ ]
  (
    II  city [ class:common_noun article:dem-prox ]
  )
  II  available [ class:adjective ]
)
END:
```

Note that the three examples above do not specify tense on the verb, which allows the default present tense to apply. For examples involving extraposition of sentential subjects, see Section 12.2.

11.3 Notes

- The notion of “subject” is purely syntactic, not semantic. Thus, in a passive sentence such as *John was killed by the car*, John is the syntactic subject and hence gets the arc label I.

- An agent in a passive clause is not a syntactic argument. To generate *John was killed by the car*, the *by the car* must be specified as an adverbial clause (see Section 14).
- As a default, the third argument (marked by III) is realized as an indirect object rather than as a prepositional object (see the second example above), unless the third argument is marked `rheme:+`, in which case it is realized by default as a prepositional object with *to* (see the third example above). These defaults can be overridden by entries in the lexicon.
- Verbs can be specified in the lexicon for having a strongly governed preposition introducing one or more of their arguments (*They discriminate against foreigners*). If there is no entry in the lexicon, the preposition must be added in the DSyntS.

11.4 Shortcomings

- Argument IV is not presently supported.

12 Embedded Clauses

12.1 Description

Embedded clauses are formed simply by adding the embedded clause as an argument to the matrix verb. Depending on whether or not the matrix verb also has a nominal object, this will be as second or third argument (using II or III, respectively). The verb form of the embedded clause's main verb must be explicitly marked on the verb.

12.2 Examples

```
// -----------------------------------------------------------------
// I saw John eating beans.
// -----------------------------------------------------------------

DSYNTS:
SEE [ class:verb tense:past morpheme:saw inflection:inv ]
(
  I  I   [ class:personal_pronoun  person:1st number:sg ]
  II eat [ class:verb mood:pres-part ]
  (
     I    John   [ class:proper_noun ]
     II   bean   [ class:common_noun number:pl ]
  )
)
END:


// -----------------------------------------------------------------
// I told Mary that John eats beans.
// -----------------------------------------------------------------
```

```
DSYNTS:
tell [ class:verb tense:past morpheme:told inflection:inv ]
(
  I  I   [ class:personal_pronoun  person:1st number:sg ]
  II  Mary [ class:proper_noun ]
  III eat [ class:verb ]
  (
     I    John   [ class:proper_noun ]
     II   bean   [ class:common_noun number:pl ]
     ATTR THAT
  )
)
END:


// ---------------------------------------------------------------------
// It would be horrible for John to see himself.
// ---------------------------------------------------------------------

DSYNTS:
BE [ class:verb mood:cond extrapo:I ]
(
  I  see [ class:verb extrapo:+ mood:inf-to ]
  (
    I  John [ class:proper_noun ref:J1 ]
    II  John [ class:proper_noun ref:J1 gender:masc ]
  )
  II  horrible [ class:adjective ]
)
END:


// ---------------------------------------------------------------------
// It bothers Mary that John can not see himself.
// ---------------------------------------------------------------------

DSYNTS:

bother [ class:verb extrapo:i ]
(
  I  see [ class:verb polarity:neg ]
  (
    I  John [ class:proper_noun ref:J1 ]
    II  John [ class:proper_noun ref:J1 gender:masc ]
    ATTR  THAT [ ]
    ATTR  CAN [ ]
  )
  II Mary [ class:proper_noun ]
)

END:
```

12.3 Notes

- Sentential subjects are treated in the same manner as sentential objects - just make them the first argument (I). If the sentential subject has `mood:to-inf` and it has a first argument, then a *for* is automatically inserted for the first argument.

- To extrapose a sentential subject and replace it by an impersonal *it*, use `extrapo:i` on the main verb (not on the verb of the sentential subject). See (11).
- You must indicate the verb form of the embedded clause using the `mood:` feature.
- Complementizers and subordinating conjunctions such as *that* should be added as ATTR dependents to the embedded main verb. The lexical entry for the complementizer *that* in the lexicon is simply THAT. See the second and fourth examples above.

12.4 Shortcomings

- The first example given above is in fact an "AcI" (or "raising-to-object" or "ECM") construction, meaning that the embedded subject is in objective case *I saw him eating beans*). Currently, the case must be marked manually in the input specification.
- Raising verbs (raising-to-subject) such as *to seem* are not currently handled correctly. Instead, they can be treated like control verbs.

13 *Wh*-Questions

13.1 Description

Wh-questions are questions that involve at least one *wh* word. One *wh*-word is fronted while the rest remain *in situ*. If the question is a matrix question (i.e., in a main clause) and the *wh* word is not the subject, a tensed auxiliary must precede the subject.

To generate such a sentence, use feature `wh:+` on the argument or adjunct that is to be a *wh*-word. This feature is either obtained from the lexicon, or can be added explicitly.

The following *wh*-words are in the lexicon and can be used in a DSyntS. They are specified as `wh:+`.

wh-word in REALPRO lexicon	**`class:`**
HOW	adverb
WHAT	article
WHEN	adverb
WHERE	adverb
WHEREFORE	adverb

WHICH	article
WHOSE	article
WHY	adverb
<WH_PRONOUN>	common_noun

To specify the pronominal forms *who*, *whom*, *what*, use <WH_PRONOUN> and specify feature `gender`. (The distinction between *who* and *whom* is made by the grammar in most cases - see Section 5.3 for details.)

13.2 Examples

```
// ------------------------------------------------------------------
// Who likes John?
// ------------------------------------------------------------------

DSYNTS:

like [ class:verb ]
(
  I   <WH_PRONOUN> [ gender:masc ]
  II  John    [ class:proper_noun ]
)

END:


// ------------------------------------------------------------------
// Whom does Mary like?
// ------------------------------------------------------------------

DSYNTS:
like [ class:verb ]
(
  I  Mary [ class:proper_noun ]
  II  "<WH_PRONOUN>" [ gender:fem ]
)

END:


// ------------------------------------------------------------------
// The authorities are wondering who gave books to whom.
// ------------------------------------------------------------------

DSYNTS:

wonder [ class:verb aspect:cont ]
(
  II  give [ class:verb tense:past morpheme:gave inflection:inv]
  (
    I  Mary [ class:proper_noun gender:fem pro:wh ]
    III  John [ class:proper_noun pro:wh gender:masc rheme:+ ]
    II  book [ class:common_noun number:pl article:no-art ]
  )
  I  authority [ class:common_noun number:pl article:def ]
)

END:


// ---------------------------------------------------------------
// Under which bridge did Maria-Luz sleep?
// ---------------------------------------------------------------


DSYNTS:
"sleep" [ class:verb tense:past ]
(
  I  "Maria-Luz" [ class:proper_noun ]
  ATTR  "UNDER" [ wh:+ ]
  (
    II  "bridge" [ class:common_noun ]
    (
      ATTR  "WHICH" [ ]
    )
  )
)
END:
```

13.3 Notes

- The grammar automatically determines the need for an auxiliary based on the grammatical function of the fronted *wh*-word and the embedded/matrix status of the verb. It also fronts at most one *wh*-word.

- Internally, there are two feature specifications related to *wh*: `wh:+` and `pro:wh`. The feature `wh:+` is a syntactic one, which regulates the word order of *wh* words with respect to others in the sentence. The feature `pro:wh` is morphological and generates the *wh* forms for nouns (but not for adverbs or articles). While <WH_PRONOUN> introduces both features at once, they can be used separately as well. The third example above works because subject *wh* constituents behave like non-*wh* constituents when used in embedded contexts.

- In order to obtain echo questions with *wh*-words *in situ* (such as *You gave books to whom?*), you can use the `pro:wh` feature on a common noun (whose lexeme is irrelevant). You must also specify `article:no-art` and gender:masc (or gender:fem) on the noun, and in order to obtain a question mark for the echo question, you must add `end:punct:question-mark`.

- Only direct dependents of the verb (in DSyntS) can behave syntactically as *wh* constituents. In order to obtain *Under which bridge did Mary-Luz sleep?*, with the prepositional phrase fronted, the preposition *under* needs to be marked `wh:+`.

13.4 Shortcomings

- Only arguments labeled I, II, or III are handled, but not IV.
- It is currently impossible to get strongly governed prepositions (those introduced from the lexicon) to front, as in *From whom did you inherit your fortune?* (assuming that *from* introduces the third argument). Instead, the prepositions must be present in the DSyntS.
- REALPRO does not yet allow any non-projective constructions, and thus the following cases cannot be generated: *Of what did you take a picture?*, *What did Mary say that John likes?*. *What bridge did you sleep under?*, and so on.

14 Adjuncts to a Clause

14.1 Description

Adjuncts to a clause such as adverbs, adverbial phrases, prepositional phrases, and adjunct clauses are related to the verb they modify by the ATTR relationship.

There are four positions for adjuncts: sentence-initial, immediately pre-verbal, "post-verbal" and sentence-final. As a default, prepositional and clausal adjuncts appear in sentence-final position (*John ate beans while waiting for Nancy*), while unmarked adverb adjuncts appear immediately pre-verbal (*John often eats beans*). (More precisely, the position is that immediately preceding the main verb of the clause, whether finite or not, except in the case of copular or existential *be*, in which case the default position is immediately post-verbal (*Denise is often in the garden*).)

The position of the adverbial phrase can be controlled through the use of the feature `position` marked on the head of the adverbial phrase. It can have the following values.

`position:`	**Example**	**Used by Default**
`sent-initial`	Often John eats beans	for adverbs with feature `adv-type:sentential`
`pre-verbal`	John often eats beans	for all unmarked adverbs
`post-verbal`	John ate voraciously	for adverbs with feature `adv-type:manner` when II-argument is empty or is a clause
`sent-final`	John eats beans often	for unmarked clauses and prepositional phrases; for adverbs with feature `adv-type:place` or `adv-type:time`, in that relative order

In addition, there are features that refer to the information status (theme, rheme, and so on) of phrases. Currently, there are two options:

- `starting_point`, with value `+`, positions an adverb or adverbial phrase in sentence-initial position (*Often, John eats beans*). Thus, `starting_point:+` implies (by default) `position:sent-initial.` The latter feature, however, is normally used to indicate the default position of a lexical item or phrase, while the former should be used to represent the communicative structure of a particular sentence, which my override the default choices.

- `rheme`, with value `+`, positions the adverbial phrase in sentence-final position (*John eats beans often*). Thus, `rheme:+` implies `position:sent-final by default`. The latter feature is given to adverbials of time and place, whereas the former feature carries communicative information for the given sentence, which may override the defaults.

14.2 Examples

```
// ------------------------------------------------------------------
// John often eats beans.
// ------------------------------------------------------------------

DSYNTS:

eat [ class:verb ]
(
  I    John   [ class:proper_noun ]
  II   bean   [ class:common_noun number:pl ]
  ATTR often  [ class:adverb ]
)
END:


// ------------------------------------------------------------------
// Often, John eats beans.
// ------------------------------------------------------------------

DSYNTS:

eat [ class:verb ]
(
  I    John   [ class:proper_noun ]
  II   bean   [ class:common_noun number:pl ]
  ATTR often  [ class:adverb starting_point:+ ]
)
END:


// ------------------------------------------------------------------
// John eats beans often.
// ------------------------------------------------------------------

DSYNTS:

eat [ class:verb tense:pres ]
(
  I    John   [ class:proper_noun ]
  II   bean   [ class:common_noun number:pl article:no-art ]
  ATTR often  [ class:adverb rheme:+ ]
)
END:


// ----------------------------------------------------------------------
// If you had money I would do anything for you.
// ----------------------------------------------------------------------

DSYNTS:

DO [ mood:cond ]
(
  I  I        [ class:personal_pronoun number:sg person:1st ]
  II anything [ class:indefinite_pronoun  person:3rd  number:sg  ]
     (
        ATTR  FOR
              (
                 II you [ class:personal_pronoun number:sg person:2nd ]
              )
     )
  ATTR HAVE [ tense:past starting_point:+ ]
       (
          I you  [ class:personal_pronoun number:sg person:2nd ]
          II money [ class:common_noun article:no-art]
          ATTR IF
       )
)
END:


// ----------------------------------------------------------------------
// Admittedly, she is really sleeping very soundly there now.
// ----------------------------------------------------------------------

DSYNTS:

sleep [ class:verb aspect:cont ]
(
  I    Maria-Luz   [ class:proper_noun gender:fem pro:pro]
  ATTR there  [ class:adverb adv-type:place ]
  ATTR now  [ class:adverb adv-type:time ]
  ATTR soundly  [ class:adverb adv-type:manner ]
  (
    ATTR very [class:adverb ]
   )
  ATTR really  [ class:adverb ]
  ATTR admittedly  [ class:adverb adv-type:sentential ]
)
END:
```

14.3 Notes

- The positioning of the adjunct to a clause is independent of the type of adjunct (adverbial, prepositional, clausal). The default position (immediately pre-verbal) is not always the best with all types of adverbial phrases. For example, a prepositional phrase is usually not placed pre-verbally: *?John has in Paris eaten brains*. However, they do not appear to be ungrammatical in that position: *John has in the past eaten brains*.

- For adverbial clauses, the subordinating conjunction (including *if*) should be treated as an ATTR of the adverbial clause's main verb. The main verb of the adjunct clause itself should be the ATTR of the main clause. See the fourth example above. The following subordinating conjunctions are in the lexicon: *EVEN_IF*, *IF*, *THAN*, *THAT*, *THEN*.

- When using a subordinating conjunction which is not in the lexicon, use feature `class:subordinating_conj`.

- In an *if ...then* construction, the *then* clause is the main clause and the *if* clause the adjunct clause. Use *IF* and *THEN*. Note that the *then* is only possible if the *if* clause is preposed (`starting_point:+` on the main verb of the *if* clause). This relationship is not enforced by REALPRO.

- For a *more ... than* construction, the clause containing *more* is the main clause and the *than* clause is the adjunct clause. Use *MORE* marked with class:adverb for cases such as *She worked more than he thought he would*, and *MORE* with class:quantifier for cases like *More children arrived than Billie-Jean had expected*. Note that the complex syntactic dependencies between the *more* and possible gaps in the *than* clause are not modeled in REALPRO (*More articles were written than Mary had thought that Mona could file without reading*, but **More articles were written than Mary regretted that Mona had read*). The DSyntS must be carefully constructed to generate the correct gappings. Note that VP ellipsis is not currently handled (*Lyn wrote more papers than Steve did*).

- No punctuation is added in any of the three positions. To add commas around (i.e., before or after) an adverbial phrase, use `between:punct:comma`. See Section 18.1 for details.

- The mapping from information status (theme, rheme, and so on) to word order and other linguistic means of expression is a complex task which is not part of the tasks of REALPRO. Future releases will have a separate module which performs this task.

14.4 Shortcomings

- Positioning of adverbial phrases in English is a notoriously difficult problem, and the current treatment is only a beginning. In particular, it is possible in English to add adverbial phrases between auxiliaries: *John has often been admired*. This is not currently supported.

- In *more ... than* constructions, the complex syntactic dependencies between the *more* and possible gaps in the *than* clause are not modeled in REALPRO (see 18.1).

15 Coordination

15.1 Description

To coordinate nodes X and Z with a conjunction Y, use the relation COORD between X and Y, and the relation II between Y and Z.

15.2 Example

```
// ----------------------------------------------------------------------
// John laughed but Mary smacked the butler and the maid.
// ----------------------------------------------------------------------

DSYNTS:

laugh [ class:verb tense:past ]
( I      John [ class:proper_noun ]
  COORD  BUT [ ]
         ( II  smack [ class:verb tense:past ]
               (  I   Mary    [ class:proper_noun ]
                  II  butler  [ class:common_noun article:def ]
                      ( COORD AND  [ ]
                              (  II  maid [ class:common_noun article:def ]
                               )
                      )
               )
             )
)
END:
```

15.3 Notes

- If two verbs are coordinated, with shared post-verbal arguments and adjuncts (*John bought and ate beans all year in Paris*), then these arguments and adjuncts should be dependents of the second (lower) verb and not of the first (higher) verb.
- REALPRO will determine that a constituent coordinated with *and* has plural agreement behavior.

16 Relative Clauses

16.1 Description

To form a relative clause, add a complete, well-formed finite clause as a dependent to a nominal node.

- Use the ATTR arc to obtain a restrictive relative clause (*I saw the man who was drinking* *a Martini*); use the DESC-ATTR arc to obtain a non-restrictive relative clause (*I saw the man, who was drinking a Martini*).

- There must be a feature `ref` specified for the hosting noun and the first or second argument of the relative clause. The value of the feature must be the same; it can be any arbitrary string.

16.2 Examples

```
// ----------------------------------------------------------------
// I saw Fred, who was drinking a Martini.
// ----------------------------------------------------------------

DSYNTS:
SEE [ class:verb tense:past morpheme:saw inflection:inv ]
(
  I  I   [ class:personal_pronoun number:sg person:1st  ]
  II Fred [ class:proper_noun gender:masc ref:r1 ]
     (
       DESC-ATTR drink [ class:verb tense:past aspect:cont ]
            (
               I Fred [ class:proper_noun gender:masc ref:r1 ]
               II martini [ class:common_noun ]
            )
     )
)
END:


// ----------------------------------------------------------------
// I saw the guys who were drinking Martinis.
// ----------------------------------------------------------------

DSYNTS:
SEE [ class:verb tense:past morpheme:saw inflection:inv ]
(
  I  I   [ class:personal_pronoun number:sg person:1st  ]
  II guy [ class:common_noun article:def number:pl gender:masc ref:r1 ]
     (
       ATTR drink [ class:verb tense:past aspect:cont ]
            (
               I guy [class:common_noun gender:masc article:def ref:r1
                        number:pl pro:pro]
               II martini [ class:common_noun number:pl ]
            )
     )
)
END:
```

16.3 Notes

- You must indicate all relevant features on both nodes in the tree. For example, if we omitted the feature NUMBER:PL in the lower *bloke* node in Example 2, REALPRO would generate **I saw the blokes, who was drinking Martinis*.

- The lexeme given to the lower of the two co-referential nodes is actually irrelevant. We have used the same lexeme as on the higher of the two nodes since this clarifies the situation best.

- To obtain a “reduced relative” clause (a passive relative clause in which the relative pronoun and the passive auxiliary are omitted, such as *the guys attacked by Mary*), specify `mood:past-part` on the verb and do not include the argument that is omitted. For example:

```
// ---------------------------------------------------------------------
// The guys attacked by Mary.
// ---------------------------------------------------------------------

DSYNTS:

II  guy [ class:common_noun article:def number:pl gender:masc ref:r1 ]
(
  ATTR  attack [ class:verb mood:past-part ]
  (
    ATTR  BY [ ]
    (
      II  Mary [ class:proper_noun ]
    )
  )
)
END:
```

16.4 Shortcomings

- A serious bug in the current version is that the lower node cannot be labelled with a lexeme in the lexicon. If the node is labeled with a lexeme in the lexicon, then the relative pronoun will not be generated. This will be fixed in the next release.
- The co-referential noun must be an immediate dependent of the verb of the relative clause -- there is no “pied piping” to obtain *the guy whose tiara was stolen* or *a situation up with which I will not put.*

17 Capitalization

17.1 Description

- Use `caps:none` on a node to keep the word generated from that node from being capitalized (for example, if it appears in sentence-initial position).
- Use `caps:words` to capitalize all words generated from the subtree rooted in the annotated node.
- Use `caps:word` to capitalize just the word generated from the annotated node.

17.2 Example

```
// -------------------------------------------------------------------
// this is a test.
// -------------------------------------------------------------------

DSYNTS:

BE [ caps:none ]
( I  THIS [ number:sg ]
  II TEST [ ]
)

END:
```

18 Punctuation

18.1 Description

By default, the system generates a sentence with a final period, unless feature `question:+` is used (Section 10), in which case the sentence ends with a question mark. This behavior can be overridden by adding the following features to the root node of the DSyntS representing the sentence:

- `punct:no_dot` eliminates the sentence-final period (e.g., for titles).
- `end:punct:question-mark` ends the sentence with a question mark (‘?’).
- `end:punct:exclamation-point` ends the sentence with an exclamation mark (‘!’).
- `end:punct:semicolon` ends the sentence with a semicolon (‘;’).
- `end:punct:ellipsis-dots` ends the sentence with suspension points (‘...’).

Furthermore, a bullet can be placed in front of a sentence:

- `begin:punct:bullet` begins the sentence with a bullet (‘*’).

In addition, parentheses, brackets, or quotes can be placed around an output sentence:

- `between:punct:parenthesis` puts parentheses (‘(‘ and ‘)’) around the sentence.
- `between:punct:square-bracket` puts square brackets (‘[‘ and ‘]’) around the sentence.
- `between:punct:double-quote` puts double quotes (‘ “ ’ and ‘ “ ’) around the sentence.
- `between:punct:single-quotes` puts single quotes (‘ ‘ ’and ‘ ’ ’) around the sentence.

These features can also be used at other nodes in a DSyntS. The parentheses, brackets, or quotes are then placed around the text string generated by the subtree dominated by the annotated node.

Comma punctuation within a sentence is handled by the grammar. Additional commas can be added using the following features:

- `between:punct:comma` puts commas (‘,’) around the string that is generated from the subtree rooted in the annotated node.
- `leftmost:punct:comma` puts a single comma after the word immediately preceding the text string generated by the subtree dominated by the annotated node.
- `rightmost:punct:comma` puts a single comma after the last word of the text string generated by the subtree dominated by the annotated node.

The same commands, with `comma` replaced by `dash`, can be used to generate dashes.

18.2 Example

```
// ---------------------------------------------------------------------
// (John loves Mary.)
// ---------------------------------------------------------------------

DSYNTS:

love [ class:verb between:punct:parenthesis ]
( I  John [ class:proper_noun ]
  II Mary [ class:proper_noun ]
)

END:
```

18.3 Notes

- The system automatically does “point absorption.” If several “point” punctuation marks (period, semicolon, colon, dash, comma) coincide in the same location, the point with the highest precedence is chosen. The priority hierarchy is as follows: period precedes semicolon precedes colon precedes dash precedes comma.
- Adding feature `rightmost:punct:colon` to the root node will not have any effect since the point absorption mechanism will favor the period over the colon. Instead, use feature `end:punct:colon`.
- The system automatically transposes quotes and periods in sentence final position, following standard convention.

18.4 Shortcomings

- Point absorption also happens with brackets and parentheses: *He was nice (but slow,) and*

I thanked him.

19 HTML Annotations

19.1 Description

To add an HTML annotation α (HTML tag with or without attributes) to just one node, add the feature `sgml:α` to this node (eg. `sgml:B, sgml:”A HREF=url”,` …).

To add an HTML annotation α (HTML tag with or without attributes) to a subtree rooted by a given node, add the feature `between:sgml:α` to this node (eg. between:`sgml:B, between:sgml:”A HREF=url”,` …).

19.2 Example

```
// ----------------------------------------------------------------------
// This is <A HREF=http://www.cogentex.com> CoGenTex. </A>
// ----------------------------------------------------------------------

DSYNTS:

BE [ ]
( I  THIS [ number:sg ]
  II CoGenTex [class:proper_noun
               sgml:”A HREF=http://www.cogentex.com"
              ]
)

END:
```

19.3 Notes

- Surround the HTML annotation α (HTML tag with or without attributes) with double quotes if the tag has attributes. Eg.

  `sgml:”A HREF=http://www.cogentex.com"`

- Prefix each double quote character appearing in the HTML annotation α (HTML tag with or without attributes) with a forward slash character \ . Eg.

  `sgml:”A HREF=http://\”www.cogentex.com\”"`

20 References

The following documents may be helpful in understanding Meaning-Text Theory, Dependency Syntax and the REALPRO grammar implementation for English:

- *Dependency Syntax: Theory and Practice.* Igor Mel'cuk. State University of New York Press, 1988. [gives an overview of Meaning-Text theory, with examples from a wide variety of languages]

- *Surface Syntax of English: A Formal Model within the Meaning-Text Framework.* Igor Mel'cuk and Nikolaj Pertsov. John Benjamins Publishing Company, 1987. [provides an extensive description of the various grammatical constructions of English at the level of surface syntactic structure (SSyntS); also includes a useful overview of the Meaning-Text framework, with examples of English DSyntSs]

- *Collins COBUILD English Grammar.* Harper Collins Publishers, 1990. [used as an auxiliary resource for standard terminology pertaining to English grammatical phenomena]

21 Index