RealPro

General English Grammar
User Manual
August 30, 1998

CoGenTex, Inc.
840 Hanshaw Road, Suite 11
Ithaca, NY 14850-1589
Tel: (607) 266 0363
Fax: (607) 266 0364
realpro@cogentex.com

Contents

1 About this Document

This document describes the linguistic resources which make up RealPro's general English grammar. More information about the formalisms used to specify the linguistic resources can be found in the *RealPro Resource Specification Reference Manual*; for more information about the C++ application programmer's interface, see the *RealPro C++ API Reference Manual*.

All of the examples used in this document can be found in the directory of examples, called `DsyntS`, which can be found in the top-level directory of the RealPro distribution. This directory contains several subdirectories which group related examples (for example, `Noun` and `Punctuation`). The files fo rthe examples have descriptive names. This document will refer to these examples using (UNIX-style) pathnames starting in the `DsyntS` directory; so, for example, `Punctuation/double-quotes.dss` will refer to file `DsyntS/Punctuation/double-quotes.dss`.

2 Background: Syntactic Dependency

The input representation to RealPro is a syntactic dependency representation. It is called the Deep-Syntactic Structure or "DSyntS" for short, and is proposed in this form by I. Mel'čuk in his Meaning-Text Theory. This representation has the following salient features:

- The DSyntS is a *tree* with labeled nodes and labeled arcs.
- The DSyntS is *lexicalized*, meaning that the nodes are labeled with lexemes (uninflected words) from the target language.
- The 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.
- The 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 Format

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 the ASCII based formalism to represent a DSyntS. The input is structured as follows:

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

The DSyntS is specified as follows:

- A specification of a node, followed optionally by
- an open parenthesis (`(`), an arbitrary (non-null) number of dependent specifications, followed by a close 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, page 10.

3.2 Example

```
// ---------------------------------------------------------------------
OUTPUT:

This is a test.

END:


DSYNTS:

BE1 [ ]
( I  THIS2 [ number:sg ]
  II TEST2 [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 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 specification is used during regression testing to automatically check for discrepancies between the target surface form and the realized surface form.

3.4 Shortcomings

There are no known shortcomings.

4 Nodes

4.1 Description

A node is specified as follows:

- A specification of a lexeme, followed by
- a list of features.

A lexeme is

- either a lexeme not in the lexicon, or
- a lexeme in the lexicon.

For lexemes in the lexicon, see Section 5, page 13. A lexeme not in the lexicon is specified by its root form (uninflected). Lexemes with regular morphology and regular syntactic behavior typically are not included in the lexicon. The capitalization of the input version is carried over to the output. Several independent words can be combined by underscores (_), which are converted to spaces in the output.

A list of features is specified by:

- An open bracket ([), followed by

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

Features are optional if defaults are provided (see Section 8, page 18), except that a lexeme which is not in the lexicon must have the `class` feature.

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

Value	Example
adjective	small, disastrous
adverb	really, fast
common_noun	table, map
proper_noun	John, Poona, Socks
verb	to play, to indulge
symbol	+

For closed-class words, feature `class` can have the values shown below. Note that all of these lexemes are in fact in the lexicon, and should be specified in that manner (see Section 5, page 13); this table is given for informative purposes only.

Value	Entries in Lexicon
article	those
coordinative_conj	and
demonstrative_pronoun	that
numeral	twelve
particle	not
partitive_pronoun	anything
preposition	about
quantificator	all
subordinative_conj	if

4.2 Examples

First example.

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

DSYNTS:

mesmerizingly [ class: adverb]

END:
```

Second example.

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

DSYNTS:

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

END:
```

4.3 Notes

- The spacing does not matter for the lexeme-feature list combination.
- The 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 avoid the final period, add `punct:no_dot` to the root verb, which eliminates the sentence-final period. See Section 22.1, page 52 for details. To avoid initial capitalization, use `caps:none`. See Section 21, page 51 for details.

4.4 Shortcomings

There are no known shortcomings.

5 The Lexicon

5.1 Description

Words can be specified in the lexicon. This obviates the need for specifying in the input DSyntS information about irregular morphology and irregular syntax. The details of the formalism used to specify lexical entries are described in the *RealPro Resource Specification Reference Manual*.

5.2 Example

The following example gives a specification for the lexeme WORK:

```
LEXEME:        WORK
CATEGORY:      verb
FEATURES:      [ ]
GOV-PATTERN:   [
                 DSYNT-RULE:
                     [ ( WORK  II  X2 ) ]              |  [  ]
                 <-->
                     [ ( WORK completive1    ON1 )
                       ( ON1  prepositional  X2  ) ]   |  [  ]
               ]
MORPHOLOGY:    [   ( [ ] work [ reg ] )   ]
```

This lexical entry specifies government pattern introducing the prepositional lexeme ON1 for the second (II) dependent of WORK: e.g. *A programmer worked [WORK] on [ON1] the project [X2].*

5.3 Notes

- The field `CATEGORY` represents what is called in this document the feature `class`.
- Instead of using the lexical entry above for WORK, it is of course also possible to specify the preposition (ON1) in the input specification to RealPro. The main advantage of specifying the preposition in the lexicon is that in this case, the preposition is not meaning bearing (*he worked on the paper* with the relevant non-spatial meaning does not contrast with, for instance, *he worked under the paper*, which only has a spatial meaning). Ideally, the input to RealPro should only contain meaning-bearing nodes.
- In specifying a lexical entry from the lexicon in a DSyntS, it is not necessary ti use upper case. We do so in this document for clarity.
- Lexical entries need not finish with an integer (e.g., *CAN1*). This is only necessary if there (potentially) are different lexemes with the same label.

5.4 Shortcomings

There are no known shortcomings.

6 DSynt-Grammar

6.1 Description

The deep-syntactic grammar (DSynt-Grammar) contains the rules used to transform a DSyntS into a SSyntS (Surface Syntactic Structure). Further details of the formalism used to specify the DSynt-Grammar are described in the *RealPro Resource Specification Reference Manual*.

6.2 Example

```
// ------------------------------------------------------------------
// A programmer[Y] worked[X] ...
// ------------------------------------------------------------------

DSYNT-RULE:

        [ ( X  I  Y ) ]           |  [ ( X [ class:verb ] ) ]
   <-->
        [ ( X predicative Y ) ]   |  [ ]
```

This rule states that the first (I) dependent (Y) of the verb (X) — i.e., the subject — should be carried over to the SSyntS as a dependent in the PREDICATIVE relation to the verb.

6.3 Notes

In the DSynt-Grammar, the ordering of the rules is relevant; for a given transformation, RealPro applies the first rule whose pattern matches. Consequently, more restrictive rules should be specified before more general ones.

6.4 Shortcomings

There are no known shortcomings.

7 SSynt-Grammar

7.1 Description

The surface-syntactic grammar (SSynt-Grammar) contains rules to transform a SSyntS into a linearized deep-morphological structure (DMorphS). To facilitate the implementation in RealPro, the SSynt-Grammar has been divided in two grammars, one specifying how to linearize a governor and its

dependents, and the other specifying how to linearize the dependents themselves. These grammars are presented in next two sections. Further details on these grammars and their formalisms are given in the *RealPro Resource Specification Reference Manual*.

7.2 SSynt-Grammar for Governor-Dependents Linearization

7.2.1 Description

The SSynt-Grammar for linearizing a governor and its dependents (SSynt-Grammar Governor-Dependent) specifies the ordering between a governor and its syntactic dependents.

7.2.2 Example

```
// ------------------------------------------------------------------
//  A programmer[Y] worked[X] ...
// ------------------------------------------------------------------

SSYNT-RULE:

     [ ( X  predicative  Y ) ] | [ ( Y [ number:?n person:?p ] )
                                   ( X [ class:verb ] ) ]
   <-->
     [ ( Y  <  X ) ]           | [ ( X [ class:verb number:?n person:?p ] ) ]
```

This rule states that a dependent Y standing in the PREDICATIVE relation to the verb X which governs it should precede this verb in the linearization; this rule also ensures that the verb agrees in number and person with X.

7.2.3 Notes

In the SSynt-Grammar Governor-Dependent, the ordering of the rules is relevant; for a given transformation, RealPro applies the first rule whose

pattern matches. Consequently, more restrictive rules should be specified before more general ones.

7.2.4 Shortcomings

To be supported by the current version of RealPro, a rule rule regarding governor-dependent linearization must be between one governor and one and only one dependent.

7.3 SSynt-Grammar for Dependents Linearization

7.3.1 Description

The SSynt-Grammar for linearizing dependents (SSynt-Grammar Dependent-Dependent) specifies the ordering between the syntactic dependents of a given governor.

7.3.2 Example

```
// ------------------------------------------------------------------
//  Will[Z] the programmer[Y] work[X]?
// ------------------------------------------------------------------

SSYNT-RULE:

        [ ( X  predicative  Y )
          ( X  auxiliary    Z ) ]   | [ ( X [ class:verb invert:+ ] ) ]
   <-->
        [ ( Z  <  Y ) ]             |  [ ]
```

This rule states that the AUXILIARY dependent Z of a verb X bearing the feature INVERT:+ should precede the dependent Y standing in the PREDICATIVE relation to X.

7.3.3 Notes

In the SSynt-Grammar Dependent-Dependent, the ordering of the rules is relevant; for a given transformation, RealPro applies the first rule whose pattern matches. Consequently, more restrictive rules should be specified before more general ones.

7.3.4 Shortcomings

To be supported by the current version of RealPro, a rule regarding dependent-dependent linearization must be between one governor and two and only two dependents.

8 Defaults

8.1 Description

Default features are added to lexemes when the DSyntS is read. These features are specified in the file `defaults.dat` in the directory `LKB`.

8.2 Example

This is the standard default file supplied with RealPro.

```
DEFAULT: verb        [ tense:pres mood:ind ]

DEFAULT: common_noun [ noun number:sg person:3rd gender:neut article:indef ]

DEFAULT: proper_noun [ noun number:sg person:3rd gender:neut article:no-art ]
```

8.3 Notes

- The user can change the file `defaults.dat` freely (the required syntax is self-explanatory). Defaults can be added to any lexical class.

- The term "standard defaults" refers to the defaults set in the file `defaults.dat` with the delivered system.
- In this document, the standard defaults are listed in the sections for the relevant lexical classes.

8.4 Shortcomings

There are no known shortcomings.

9 Nouns

9.1 Description

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

Nouns have four types of features: for number, for gender, for case, and for determiners.

- Feature `number` can have the following values:

Value	Example	Default
sg	bean	√
pl	beans	

- Feature `gender` can have the following values:

Value	Example	Default
masc	boy	√
fem	waitress	
neut	piano	
dual	teacher	

  Specification `gender:dual` should yield pronominal choices such as "his or her" but is not currently implemented.

- Feature `case` can have the following values:

Value	Example	Default
nom	bean, she	√
gen	bean's, her	
obj	bean, him	

  This feature is not usually used in an input DSyntS for RealPro.

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

Value	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

9.2 Example

First example (`Noun/the-yemen.dss`):

```
OUTPUT:
The Yemen.
END:

DSYNTS:

Yemen [class:proper_noun article:def]

END:
```

Second example:

```
// ------------------------------------------------------------------
// Some tiaras.
// ------------------------------------------------------------------

DSYNTS:

tiara [class:common_noun number:pl]

END:
```

Third example:

```
OUTPUT:
These cars.
END:

DSYNTS:

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

END:
```

9.3 Notes

- The feature combination `article:indef number:pl` yields the bare plural, (*tiaras*). (Note that since `article:indef` is the default, simply `number:pl` also yields the bare plural.) 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 obtaon *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 case features should be specified in the input to REALPRO are `case:gen` in the possessive construction (see Section 11, page 26 — not yet implemented) and `case:obj` for the "AcI" construction (see Section 16, page 37).
- For other determiners (numerals, demonstratives — *those four tiaras*), see Section 10, page 22.
- For the possessive construction (*John's tiara*), see Section 11, page 26.
- For compound nouns (*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 22 for details on using parentheses).
- Feature `human` is not used in REALPRO for English; instead, the relevant distinctions can be made using `gender:dual` or `gender:neut`.

9.4 Shortcomings

- Specification `gender:dual` should yield pronominal choices such as "his or her" but is not currently implemented.
- `case:gen` is not currently implemented.
- Proper nouns are currently not inflected at all.

10 Determiners

10.1 Description

As mentioned in Section 9, page 19, 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*, *several*, and so on). The difference between the two categories is that quantifiers are ordered before articles, while numerals are ordered after articles (*all the boys*, *any seven boys*, *the one thing*).

The following quantifiers can be found in the lexicon. The table shows the number agreement that is forced on the head noun by the quantifier.

Entry in lexicon	Number agreement
ALL	pl
ANY	none
BOTH	pl
EACH	sg
MORE3	pl
MOST2	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 quantifier.

Entry in lexicon	Alternate integer representation	Number agreement
ZERO	0	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
SEVERAL	none	pl

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

Value	Example	Default
word	twelve	
roman	XII	
+	12	$\surd$

10.2 Examples

First example:

```
OUTPUT:
More than six ducks.
END:

DSYNTS:

duck [ class:common_noun ]
(
```

```
  ATTR  MORE2 [ ]
  (
    II  6 [ form:word ]
  )
)

END:
```

Second example:

```
OUTPUT:
The several ducks.
END:

DSYNTS:

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

END:
```

Third example:

```
OUTPUT:
All the ducks.
END:

DSYNTS:

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

END:
```

10.3 Notes

- Quantifiers do not remove any articles or other determiners from the head noun, in order to allow *many a woman* or *all the women*. However, RealPro would also generate **many the woman* and **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 (*the four ducks*).
- When specifying a quantifier not in the lexicon, specify `class:quantificator` (note the non-standard name for the class). When specifying a numeral not in the lexicon, use `class:numeral`.
- To add *more than...* to a numeral, add *MORE2* as an ATTR to the noun, and then add the numeral as a II to the *MORE2*. See the first example above.

10.4 Shortcomings

The complex interaction among determiners (*fewer than five of those many nice young linguists*) has not been fully implemented. In particular, the appearance of *of* in certain combinations (*most of the 56 men*) is not handled automatically by the grammar. It is not clear how to force it, either.

11 The Possessive Construction

11.1 Description

Nominal constructions with a possessor, such as *John's tiara*.

11.2 Example

Forthcoming.

11.3 Notes

This construction will be implemented soon.

11.4 Shortcomings

This construction is not yet implemented.

12 Pronouns

12.1 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 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 actant have the same value for the `ref` feature (or the first and third or the second and third), then the lower argument is replaced by a reflexive.
- Relative pronouns are added by the grammar. See Section 20, page 48 for details.

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

Type	Example	Special Lexeme	Feature
demonstrative pronoun	that	THIS2, THAT2	none
partitive pronoun	anything	none	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`

12.2 Examples

First example:

```
Item missing.
```

Second example:

```
OUTPUT:
John sees himself.
END:


DSYNTS:

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

END:
```

Third example:

```
OUTPUT:
The psychiatrist revealed the patient to herself.
```

```
END:

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:
```

12.3 Notes

- If the second and third actant are co-referential, RealPro always generates the third actant 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).

12.4 Shortcomings

- While the category "partitive pronoun" exists in the lexicon, it does not have any special syntactic behavior.

13 Adjectives

13.1 Description

Adjectives can get attached to nouns in two manners:

- By the ATTR arc, in which case they appear pre-nominally (the usual case).

- By the DESC-ATTR arc, in which case they appear post-nominally, set off by commas.

13.2 Example

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

DSYNTS:

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

END:
```

13.3 Notes

ATTR and DESC-ATTR arcs are repeatable.

13.4 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.

14 Verbs

14.1 Description

This is an exhaustive list of verbal features.

- Feature `tense` can have the following values.

Value	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:

Value	Example	Default
act	John likes Mary	√
pass	John is liked	

For information on the argument structure in passive voice, see Section 15.3, page 36.

- Feature `aspect` can have the following values:

Value	Example	Default
simple	John eats beans	√
cont	John is eating beans	

- Feature `taxis` can have the following values:

Value	Example	Default
nil	John likes Mary	√
perf	John has liked Mary	

- Feature `mood` can have the following values:

Value	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:

Value	Example	Default
nil	John likes Mary	√
neg	John does not like Mary	

- Feature `question` can have the following values:

Value	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 17, page 40.

14.2 Example

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

DSYNTS:

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

END:
```

14.3 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 22, page 52.

- 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: *CAN1*, *MAY1*, *MUST1* and *SHOULD1*. 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 actant I, and the adjective, noun, or prepositional phrase which is predicated of the subject as argument II.
- For information on the argument structure in passive voice, see Section 15.3, page 36.
- For information on *wh*-questions, see Section 17, page 40.

14.4 Shortcomings

- The subjunctive is not handled.

15 Clauses and Sentences

15.1 Description

Clauses and sentences are constructed by giving arguments to verbs. The arguments of the verb are labeled I, II, III, and IV. 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 anticipatory *it* (*It bothers me that she is there*), and *there* insertion with existential-type verbs (*There appeared three geese in the study*). Feature `extrapo` can have the following values:

Value	Example	Subject must be	Default
-	Geese are in the garden That she is here bothers me	clause or NP	√
there	There are geese in the garden	NP	
i	It bothers me that she is there	Clause	

15.2 Examples

```
OUTPUT:
John loves Mary.
END:

DSYNTS:

love [ class:verb tense:pres inflection:reg ]
( I  John [ class:proper_noun ]
  II Mary [ class:proper_noun ]
)

END:
```

Second example:

```
OUTPUT:
John tells Mary a story.
END:

DSYNTS:

tell [ class:verb tense:pres inflection:reg ]
( I  John [ class:proper_noun ]
  III Mary [ class:proper_noun ]
  II story [ class:common_noun article:def ]
)

END:
```

Third example:

```
OUTPUT:
Have there not been firefighters available in this city?
END:
```

```
DSYNTS:
BE1 [ class:verb extrapo:there polarity:neg question:+ taxis:perf ]
(
  I  firefighter [ class:common_noun number:pl article:no-art ]
  ATTR  IN1 [ ]
  (
    II  city [ class:common_noun article:dem-prox ]
  )
  II  available [ class:adjective ]
)
END:
```

For examples involving extraposition of sentential subjects, see Section 16.2, page 37.

15.3 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 actant. To generate *John was killed by the car*, the *by the car* must be specified as an adverbial clause (see Section 18, page 42).
- As a default, the third actant (marked by III) is realized as an indirect object rather than as a prepositional object (see the second example above), unless the third actant 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 (see Section 5, page 13).
- Verbs can be specified in the lexicon for having a strongly governed preposition introducing one or more of their actants (*They discriminate against foreigners*). If there is no entry in the lexicon, the preposition must be added in the DSyntS.

15.4 Shortcomings

- Relation IV is not presently supported.
- For *there*-insertion clauses, if the number of the DSyntS subject (i.e., the I argument) is not marked in the input DSyntS, it will default to singular no matter what numerals or quantifiers have been specified.

16 Embedded Clauses

16.1 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.

16.2 Examples

First example:

```
OUTPUT:
I saw John eating beans.
END:

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 article:no-art ]
  )
```

```
)
END:
```

Second example:

```
OUTPUT:
I told Mary that John eats beans.
END:

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 article:no-art ]
     ATTR THAT3
  )
)
END:
```

Third example:

```
OUTPUT:
I told Mary that John eats beans.
END:

DSYNTS:
BE1 [ class:verb mood:cond extrapo:I ]
(
  I  see [ class:verb extrapo:+ mood:inf-to extrapo:+ ]
  (
    I  John [ class:proper_noun ref:J1 ]
    II  John [ class:proper_noun ref:J1 gender:masc ]
```

```
  )
  II  horrible [ class:adjective ]
)
END:
```

Fourth example:

```
OUTPUT:
It bothers me that John can not see himself.
END:

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  THAT3 [ ]
    ATTR  CAN1 [ ]
  )
  II Mary [ class:proper_noun ]
)

END:
```

16.3 Notes

- Sentential subjects are treated in the same manner as sentential objects – just make them the first actant (I). If the sentential subject has MOOD:TO-INF and it has a first actant, then a *for* is automatically inserted for the first actant.
- To extrapose a sentential subject and replace it by an expletive *it*, use `extrapo:i` on the main verb (not on the verb of the sentential subject).

See (15), page 34.

- 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 THAT3. See the second example above.

16.4 Shortcomings

- The example given above is in fact an "AcI" (or "raising-to-object" or "ECM") construction, meaning that the embedded subject is in accusative 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.

17 *Wh*-Questions

17.1 Description

Wh-questions are questions that involve at least one *wh* word. To generate such a sentence, use `pro:wh` on the argument or adjunct that is to be a *wh*-word.

17.2 Examples

First example:

```
OUTPUT:
Who likes John?
END:
```

```
DSYNTS:

like [ class:verb ]
(
  I   Manfred [ class:proper_noun gender:masc pro:wh ]
  II  John    [ class:proper_noun ]
)

END:
```

Second example:

```
OUTPUT:
The authorities are wondering who gave books to whom.
END:

DSYNTS:

wonder [ class:verb ]
(
  II  give [ class:verb ]
  (
    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:
```

17.3 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.

17.4 Shortcomings

- Adjuncts are currently not handled, only arguments labeled I, II, or III.
- Also, prepositions present at DSyntS are not handled even if they are marked as an argument and their argument is marked `pro:wh`.
- Genitive *wh*-words (*whose*) are not handled.
- For echo questions with *wh*-words *in situ* (such as *You gave books to whom?*), do not use the `pro:wh` feature, but instead specify the word as a noun, and add a question mark (see Section 22, page 52).

18 Adjuncts to a Clause

18.1 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 three positions for adjuncts: sentence-initial, immediately pre-verbal, and sentence-final. As a default, prepositional and clausal adjuncts appear in sentence-final position (*John ate beans while waiting for Nancy*), while adverbial 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 (*Désirée is often in the garden*).)

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

Value	Example	Default
sent-initial	Often John eats beans	
pre-verbal	John often eats beans	for adverbs
sent-final	John eats beans often	for clauses and prepositional phrases

Note: feature values `sent-initial` *and* `sent-final` *are not yet implemented. Please use the features* `starting_point` *and* `rheme` *described below.*

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 the adverbial phrase in sentence-initial position (*Often, John eats beans*). Thus, `starting_point:+` is a synonym for `position:sent-initial`.
- `rheme`, with value `+`, positions the adverbial phrase in sentence-final position (*John eats beans often*). Thus, `rheme:+` is a synonym for `position:sent-final`.

18.2 Examples

First example:

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

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 ]
)
END:
```

Second example:

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

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 starting_point:+ ]
)
END:
```

Third example:

```
// ------------------------------------------------------------------
// 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:
```

Fourth example:

```
OUTPUT:
If you had money I would do anything for you.
```

```
END:

DSYNTS:
DO [ mood:cond ]
(
  I  I        [ class:personal_pronoun number:sg person:1st ]
  II anything [ class:partitive_pronoun  person:3rd  number:sg  ]
     (
        ATTR  FOR1
              (
                 II you [ class:personal_pronoun number:sg person:2nd ]
              )
     )
  ATTR HAVE1 [ tense:past starting_point:+ ]
       (
          I you  [ class:personal_pronoun number:sg person:2nd ]
          II money [ class:common_noun article:no-art]
          ATTR IF
       )
)
END:
```

18.3 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 preverbally: *?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*, *THAN1*, *THAT3*, *THEN1*.

- When using a subordinating conjunction which is not in the lexicon, use feature `class:subordinative_conj` (note the non-standard terminology).
- In an *if ... then* construction, the *then* clause is the main clause and the *if* clause the adjunct clause. Use *IF* and *THEN1*. Note that the *then* is only possible if the *if* clause is preposed (`starting_point:+` on the main verb of the *if* clause). This 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 *MORE1* if an adverb (*She worked more than he though he would*) and *MORE3* if a quantifier (*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 22.1, page 52 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.

18.4 Shortcomings

- For feature `position`, values `sent-initial` and `sent-final` are not yet implemented.
- 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 (*More articles were written than Mary had thought that John could file without reading*, but **More articles were written than Mary regretted that John had read*).

19 Coordination

19.1 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`.

19.2 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 AND2  [ ]
                              (  II  maid [ class:common_noun article:def ]
                               )
                      )
               )
           )
```

```
)
END:
```

19.3 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.

19.4 Shortcomings

There are no known shortcomings.

20 Relative Clauses

20.1 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 descriptive 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.

20.2 Examples

Example 1:

```
OUTPUT:
I saw the man who was drinking a Martini.
END:

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

Second Example:

```
OUTPUT:
I saw the blokes who were drinking Martinis.
END:

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

```
                I bloke [class:common_noun gender:masc article:def ref:r1
                         number:pl pro:pro]
                II Martini [ class:common_noun article:no-art number:pl ]
            )
    )
)
END:
```

20.3 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 blokes attacked by Mary*), specify `mood:past-part` on the verb and do not include the actant that is omitted. For example:

```
OUTPUT:
The blokes attacked by Mary.
END:

DSYNTS:
II  bloke [ class:common_noun article:def number:pl gender:masc ref:r1 ]
(
  ATTR  attack [ class:verb mood:past-part ]
  (
    ATTR  BY1 [ ]
    (
      II  Mary [ class:proper_noun article:no-art ]
    )
```

```
    )
)
END:
```

20.4 Shortcomings

- A serious bug in the current version is that the lower node cannot be labeled 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 man whose tiara was stolen* or *a situation up with which I will not put.*

21 Capitalization

21.1 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.

21.2 Example

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

```
DSYNTS:

BE1 [ caps:none ]
( I  THIS2 [ number:sg ]
  II TEST2 [article : indef]
)

END:
```

21.3 Notes

None.

21.4 Shortcomings

There are no known shortcomings.

22 Punctuation

22.1 Description

By default, the system generates a sentence with a final period, unless feature `question:+` is used (Section 14, page 30), 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 ('(', ')') around the sentence.
- `between:punct:square-bracket` puts square brackets ('[', ']') around the sentence.
- `between:punct:double-quote` puts double quotes (' " ', ' " ') around the sentence.
- `between:punct:single-quotes` puts single quotes (' ` ', ' ' ') 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.

22.2 Example

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

DSYNTS:

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

END:
```

22.3 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.

22.4 Shortcomings

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

23 HTML Annotations

23.1 Description

To add an HTML annotation (tag with or without attributes), proceed as follows:

- To add an HTML tag α to the string corresponding to just the node in question, add `sgml=`α, where α can be any HTML tag (eg. A for anchor, B for bold, etc.).
- To add an HTML tag α with attributes β to the string corresponding to just the node in question, add `sgml=`α`:`β.
- To add an HTML tag α (with attributes β) to the string generated from the subtree rooted in the node in question, add `between:sgml=`α (`between:sgml=`α`:`β).

23.2 Example

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

DSYNTS:

BE1 [ ]
( I  THIS2 [ number:sg ]
  II CoGenTex [class:proper_noun
               article:no-art sgml=A:"HREF=http://www.cogentex.com" ]
)

END:
```

23.3 Notes

- To output the results of these features, the formatting must be set to HTML.
- Surround the string representing the attributes of an HTML tag with double quotes if it contains a colon (eg. *sgml=A:"HREF=http://www.cogentex.com") or if it contains spaces.*

23.4 Shortcomings

There are no known shortcomings.

24 Points to Consider When Modifying the Grammar

There are some dependencies regarding the knowledge encoded in the lexicon and the various grammars: these linguistic resources share similar labels for the features, lexemes, syntactic relations, etc. In the current version of RealPro, it is the task of the developer to ensure that any modification s/he does to one linguistic resource is consistent with the information found in the other linguistic resources.

Here is a list of some typical items to verify when the grammar is modified:

- Verify that the deep-syntactic relations used in a DSyntS are covered in the DSynt-Grammar.
- Verify that the deep-syntactic relations introduced in the government pattern of a lexical entry are covered in the DSynt-Grammar.
- Verify that the surface-syntactic relations introduced in a DSynt-Grammar rule are covered in the SSynt-Grammars.

25 Other Documents

Here is a list of documents currently available or that will be available soon:

- *RealPro Resource Specification Reference Manual* (Jan. 1997)
- *RealPro C++ API Reference Manual* (Jan. 1997)
- *Dependency Syntax: Theory and Practice.* I. Mel'čuk. State University of New York Press, 1988.

26 Index