Contents

1 DII: Dyana's Integrated Implementation

The DII grammar is the product of collaborative work on two of DYANA-2's implementation deliverables: P3.4, Constraint-Based Semantics in CUF (Stuttgart) and P3.7, Grammar Integration (Edinburgh). DII was developed by Peter Krause in Stuttgart and by David Beaver, Claire Grover, Janet Hitzeman and Ingrid van de Bovenkamp in Edinburgh.

DII is a grammar implementation in CUF which derives initially from König's LexGram system and grammar (König 1994a; König 1995). It was decided to use the LexGram system as a starting point for the implementation in order that issues of semantic implementation could be explored from an early stage.

The syntactic component of König's LexGram grammar is a type of unification-based categorial grammar. König points to similarities between her framework and Head-driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994) and she refers to her framework as "lexicalised HPSG". However, the framework is more strongly lexicalist than HPSG and bears a much closer resemblance to categorial grammar. There are no phrase-structure rules or syntactic schemata as in HPSG and, instead, information about phrase structure is located in the lexicon. The LexGram system makes use of CUF's foreign language interface which allows calls to Prolog goals from CUF sorts and parsing with LexGram grammars is achieved by means of a special Prolog head-driven parser written by Esther König and Peter Krause. The DII grammar takes over LexGram's syntactic framework almost unaltered but extensions have been added to allow for parsing of discourses as well as of sentences. Additionally, the grammar includes a partial implementation of Engdahl and Vallduví's (1994) proposals for representing information structure and this has involved modifying the parser so that it will accept either strings of words or strings of word-accent pairs.

The syntactic coverage of the sentence-level DII grammar includes the following:

- A basic treatment of verbs, nouns, adjectives and prepositions.
- A number of subcategorization possibilities for verbs including intransitives, transitives, ditransitives and verbs taking factive sentential complements.
- A treatment of the auxiliaries *be* and *have*.
- A subset of unbounded dependency constructions, viz. topicalisations and subject and non-subject relative clauses.
- Singular and plural noun phrases occurring with definite and indefinite determiners or with quantifiers.
- Partitives.
- Possessives.
- Nominal adjuncts including attributive adjectives, prepositional phrases and relative clauses.
- Verbal adjuncts including prepositional phrases, temporal modifiers and focus-sensitive adverbs.

Lexical items in the LexGram framework are rather complex objects and therefore a method is provided to allow the lexicon to be specified in a simple way and then mapped into its more complex form. In addition, inflected forms of verbs and nouns are optionally generated by means of the morphophonological software addition to CUF known as CUF-RA (Manandhar 1995).

The semantic component of König's LexGram grammar is an implementation of Frank and Reyle's (1995) Underspecified Discourse Representation Theory (UDRT), and, although the DII grammar reuses many of the sorts which perform the compositional mapping, it was decided to use a simplified version of UDRSs which is close in spirit to the quasi-logical forms of the Core Language Engine (Alshawi 1992). An initial underspecified representation is non-deterministically converted into a scoped representation which makes distributive and collective readings of plural noun phrases explicit. The scoping process is one main area where the DII implementation could be refined in the future.

In DII, anaphoric and presuppositional terms are treated uniformly. In fact, the grammar maps both pronouns and presupposition triggers such as definite descriptions or factives to so-called anaphoric terms. We refer to elements of both classes as "PA terms", and the module that treats them as the "PA component". It is crucial for the PA component that the semantic representation language is dynamically interpreted, with formulae mapping input states to output states. PA terms correspond to a special subclass of formulae which place constraints on their input states. If these constraints are satisfied, we say that a PA term is *well-formed* in its local context. This context is determined both by the interpretation of preceding discourse and by the interpretation of certain parts (the *accessible* parts) of the sentence containing the relevant PA expression. In general, information provided by the surface form of an utterance will not automatically guarantee well-formedness of all PA terms, but to fully understand a speaker's meaning, a hearer must be able to calculate what extra information available to the speaker is needed in order to guarantee well-formedness. We adopt a very general approach to this problem, whereby interpretation involves *abducing* the extra information required for well-formedness. We take it that mechanisms of *presupposition accommodation* described by such scholars as Lewis, Heim and van der Sandt are naturally seen as special instances of an abductive scheme.

Although the DII grammar has limited coverage, the phenomena that are covered have been selected in order to explore issues that have been the subject of theoretical work both in DYANA and DYANA-2. Moreover, the architecture is general enough to allow for straightforward extensions, in particular in the area of anaphoric phenomena. The Stuttgart part of the implementation has concerned itself primarily with plural anaphora, presupposition and general issues of semantic representation and disambiguation:

- The LexGram semantics construction concept has been applied to the construction of basic underspecified representations.
- The noun phrase representations are organized in a typology distinguishing different kinds of anaphoric terms and classifying generalized quantifiers roughly as in Kamp and Reyle 1993.
- Plural anaphora are treated using a mechanism which is descriptively equivalent to the treatment of Kamp and Reyle 1993. It is compatible with recent approaches to plural

anaphora using richer semantic values for quantifiers or an extended external dynamics such as Elworthy 1995 and Fernando 1993. The general presupposition module based on abduction, which was developed together with Edinburgh, has made it possible to treat plural anaphora and plural definites in a uniform way. In the representation of quantifiers, context sets (Westerstahl 1985) are used to be able to determine dynamically the domain of quantification. They are resolved like plural anaphors. Partitive prepositional phrases are mapped to context sets as well.

The Edinburgh parts of the implementation provide the following coverage:

- The representation of tense and aspect and the calculation of temporal relations. Here we follow the methods of Kamp and Reyle 1993, using features in the syntax to percolate tense and aspect information to the root, where the semantics makes use of it to calculate narrative progression. An additional fine-grained classification of temporal connectives has been implemented using results from the ESPRIT project Dandelion (cf. Oversteegen 1993; Hitzeman 1995a; Verschuur 1995) demonstrating a degree of compatibility between the two projects.

- An implementation of Vallduví's (1990, 1994) theory of information structure which is broadly compatible with the HPSG-based encoding proposed in Engdahl and Vallduví 1994. Information about focus-ground structure is used by the PA component in two ways. Firstly, it helps determine how anaphors are resolved, this being achieved by feeding focus-ground information into a modified *centering* model. Secondly, it determines what the presupposition actually is in the case of focus sensitive presupposition triggers such as “too” and “only”.

- An implementation of the centering approach to anaphora resolution. The standard centering approach (Brennan et al. 1987; Grosz et al. 1995) is adopted but augmented in several ways:
  - Information structure is used to influence whether a center is retained.
  - Because each pronoun is presented to the centering module along with a set of accessible NPs, sentences with more than one pronoun may have different antecedent possibilities. For example, in the sentence *He gave Bill his hat*, *Bill* is accessible to *his* but not to *He*. Rather than combine these antecedent sets in order to create the traditional ordered list of possible antecedents which constitutes the forward center list, potentially losing information, the forward center list is an ordered list of possible moves.
  - Traditional centering deals only with sentences with no embedded clauses, but our data includes more complex constructions. We have augmented our algorithm to handle these constructions based on (unpublished) observations by Massimo Poesio and Rosemary Stevenson in their work on nominal anaphora and centering.

2 Loading and Running DII

In this section we provide information about how to use DII. For users who are unfamiliar either with CUF or with the LexGram system it is recommended that they also consult the CUF User's Manual (Dörre et al. 1994) and the LexGram documentation (König 1994a; König 1994b). In addition, a number of papers in the DYANA-2 report (Beaver 1995) contain detailed descriptions of various aspects of the DII grammar and users may find it valuable to consult these. For those users who wish to use the version of the lexicon generated by the morphophonological component CUF-RA, it is recommended that its user manual be consulted (Manandhar 1995).

DII is made up of a large number of files. The main grammar files are all located in the top level directory `DII`. The subdirectory `lg-sys` contains the parser and other LexGram system files. There are two other subdirectories, `Morph` and `Morph-sol`. These contain DII-customised versions of the morphophonology component CUF-RA. `Morph` contains a version compatible with the SunOS 4.1.3 operating system while `Morph-sol` contains a version compatible with the Solaris (SunOS 5.4) operating system (CUF-RA uses compiled C++ code which is not compatible across operating systems).

The file `dii.cnf` is a standard CUF `.cnf` file which contains grammar initialisation information including the information about which CUF and Prolog files are to be loaded. `dii.cnf` should be loaded using the `load_cnf` command. Once `dii.cnf` has been loaded the grammar should be compiled using the `gload` command as follows:

```
                        CUF - Version 2.30

        Institut fuer maschinelle Sprachverarbeitung (IMS),
                  Universitaet Stuttgart, Germany

            Michael Dorna, Jochen Doerre & Joerg Junger

                              July 1994


%%% global initialization: ..........................
yes
| ?- load_cnf(dii).
.
.
.
yes
| ?- gload(dii).

%%% (re)loading grammar dii:

%%% CUF WARNING: Do not interrupt during file handling!
.
.
```

The DII grammar takes approximately the same form as the German grammar supplied with König's LexGram and, in particular, König's philosophy of modularity is reflected in the fact that the grammar is split across a large number of files, each representing a sub-component of the whole system. Our grammar is simpler than König's in that we have virtually eliminated those parts that would make the grammar reversible for generation: the only remnant of LexGram's generation module occurs in the lexicon where entries still retain an extra 'slot' that was intended for the semantic 'guide' for generation. This was left in place in case the grammar should ever be used in future for generation.

The file `dii.cnf` requires a number of `.pl` and `.ctrl` files to be consulted as follows:

```
:- [
      %%%%% LexGram system

      'lg-sys/auxpreds.pl',   % auxiliary predicates/parser
      'lg-sys/hdparser.pl',   % parser
      'lg-sys/lgi_uif.ctrl',  % LexGram user interface
      'lg-sys/tree.ctrl',     % tree data structure,

      %%%%% grammar specific

      %% lexicon
      'mk_lexif.ctrl',        % lexicon expander
      'is_mk_lexif.ctrl',     % ditto for information structure
      'lex_transpo.ctrl',     % contains gram_t/4
      'is_lex_transpo.ctrl',  % ditto for information structure

      %% Text Processing & Inferencing
      % note: lg-sys/lgi_uif.ctrl contains text handling code
      'resolution.pl',  % resolution module
      'abducer.pl',
      'centering.pl'
                ].
```

As with LexGram, the files in the subdirectory `lg-sys` contain the parser and associated code. The files `lg-sys/auxpreds.pl`, `lg-sys/hdparser.pl`, `lg-sys/lgi_uif.ctrl` and `lg-sys/tree.ctrl` are DII-customised versions of the LexGram files of the same name.

The pre-compiled lexicon is in the file `lexicon.cuf` in the DII directory and although this file is not loaded into CUF, it is read in the course of lexicon compilation. The file `mk_lexif.ctrl` contains Prolog code for transforming `lexicon.cuf` into the expanded lexicon file `expanded_lexicon.cuf`. The lexicon compilation process requires that mapping rules be defined and these are to be found in the file `lex_transpo.ctrl`. The two files `is_mk_lexif.ctrl` and `is_lex_transpo.ctrl` are variants of the first two which compile the lexicon into a format including accent information for use when computing information structure. Section 4 discusses the information structure component and Section 5 discusses the lexicon in more detail.

The files `resolution.pl`, `centering.pl` and `abducer.pl` are used in the resolution and abduction stages of processing.

The file `dii.cnf` ensures that all the relevant `.cuf` files are loaded as follows:

- LexGram system files. These are based on general files from LexGram but have been altered for use with DII:

```
category,            % category data type
daccess,             % subroutines of grammar interpreter
lgi_general,         % general (public) stuff
drsguide,            % guides for generation
tree,                % tree data structure
```

- General purpose files:

```
general,             % general useful sorts
ptests,              % test sentences
pluraltests,         % tests for plural phenomena
```

- Lexicon files (see sections 5 and 6) :

```
expanded_lexicon,    % lexicon minus verbs and nouns
is_expanded_lexicon, % ditto for information structure
verbs_exp,           % verb lexicon (generated by CUF-RA)
is_verbs_exp,        % ditto for information structure
nouns_exp,           % noun lexicon (generated by CUF-RA)
is_nouns_exp,        % ditto for information structure
```

- Morpho-syntax files (see section 3):

```
morph,               % morphology
subcat,              % subcategorization frames
synT,                % syntactic data type
synschema,           % basic syntactic schemata
lexrules,            % word class definitions
synsem,              % semantic construction rules
inf_struc,           % information structure (IS) and
                     % duplicate sorts for IS
```

- Semantics files (see section 3):

```
sem,                 % lexical semantics
psemT,               % partial semantics data type
usemT,               % underspecified semantics data type
scopedsemT,          % scoped, unresolved semantics
rsemT,               % scoped and resolved semantics
disamb,
            collect_accessibles,
            pi_expansion,
            inference_rules,
            resolution,
            centering_sorts,
            centering
```

There are two parsing predicates defined in DII: `p/1` is used for parsing single sentences and returns a result where stored terms have been scoped but no disambiguation has taken place; `pt/1` is used for parsing discourses and the full range of processing takes place. Examples for parsing must be declared using the sorts `psent/2` (for single sentences) and `ptext/1` for discourses. A wide range of example sentences can be found in the files `ptests.cuf` and `pluraltests.cuf`. Some examples are as follows:

```
psent(565,max_vp) := [every,doctor,cures,her,patient].
psent(571,max_vp) := [the,man,and,the,children,sing].
ptext(208) := [[a,woman,finds,a,soup],[she,eats,it]].
ptext(748) := [[a,man,entered,the_white_hart],
               [he,was,wearing,a,black,jacket]].
```

The second argument of the sort `psent` gives the category that must be asscociated with the parse. Sort definitions for these are included at the end of the file `ptests.cuf`. To parse these examples, one types either `p(Number)` or `pt(Number)` in the CUF window:

```
| ?- p(565).

| ?- pt(208).
```

Input to the two parse predicates may be either a list of words (as above) or, when information structure is to be calculated, a list of word-accent pairs:

```
psent(441,max_vp) := [[rudi,b],[only,u],[gave,u],[the,u],[man,a],
                      [the,u],[book,u]].

ptext(806) := [[[rudi,b],[found,u],[a,u],[doctor,a]],
               [[he,u],[likes,a],[him,u]]].
```

3 The DII Grammar

Many aspects of the DII grammar are described in some detail in the papers collected in Beaver 1995 and it is beyond the scope of this manual to attempt to reproduce those descriptions here. Instead the (necessarily brief) discussion in this section is limited to those aspects of DII that are not described in Beaver 1995. This section therefore deals only with the syntactic component (including the initial syntax-semantics mapping) and the treatment of tense and aspect. See Krause 1995 and Beaver and Krause 1995 for information about the semantics components (underspecified, scoped and resolved) and Beaver and van der Bovenkamp 1995 for discussion of the implementation of information structure.

3.1 Syntax and the Syntax-Semantics Mapping

DII's syntactic component will be familiar to LexGram users and will be easily comprehensible to users familiar with categorial grammar. The basic driving force are the lexical items—each lexical item contains not just information about its syntactic category but also about the constituents it needs to combine with. In this way, each lexical item defines a piece of syntactic tree and there is no need for a set of rewrite rules to state how elements combine. The following is the syntactic part of the feature structure associated with the transitive verb *likes*:

```
(stree &
 root:(synsem_pair &
       syn:(fin_pres &
            agr:(agr &
                 number:sg &
                 person:third))) &
 leaves:[(argument_leaf &
          dir:right &
          cat:(arg_stree &
               root:(synsem_pair &
                     syn:(dp &
                          case:acc)))),
         (argument_leaf &
          dir:left &
          cat:(arg_stree &
               root:(synsem_pair &
                     syn:((dp & ~trace) &
                          case:nom &
                          agr:(agr &
                               number:sg &
                               person:third)))))])
```

The syntactic category of *likes* is encoded in the feature `root` while the feature `leaves` contains information about the two determiner phrases that `likes` must combine with to form a sentence. The verb first combines with the accusative `dp` to its right and then with the nominative `dp` to its left. In this way the entire structure of sentences headed by `likes` is encoded within the lexical entry.

The actual lexical entry for *likes* is stated much more economically using the sort `verb/5`:

```
l(likes,_342):=verb(weak,v(sg3),stative,v2,like_prime).
```

All of the sorts used in lexical entry definitions can be found in the file `lexrules.cuf`. These generally call other sorts which can be found in other files depending on what their function is. The subcategorisation properties of categories are defined using the sorts in the file `subcat.cuf`. In the case of *likes*, the fourth argument of `verb/5`, `v2`, encodes the fact that the verb is transitive. This argument is passed on to the sort `subcat/1` in `subcat.cuf` and the relevant sort definition is as follows:

```
subcat(v2) := [ (dp(acc) & dir:right), (subj_dp(nom) & dir:left) ].
```

This expands out to the list that is the value of `leaves` in feature structure above.

It is not possible to describe the syntactic component in any great detail here and users are referred to König 1994b for more information. The syntactic types and sorts in DII are all based on the original German grammar described by König although many simplifications have been made because the fixed nature of English word-order is simpler to describe than the more flexible word-order of German.

The basic lexical sorts in `lexrules` also call sorts that perform the initial mapping from syntax to underspecified semantics. These mapping sorts have "synsem" in their names and can be found in the file `synsem.cuf`. These sorts in turn make reference to sorts which can be found in `sem.cuf` (basic semantics of different category types), `psemT.cuf` (types for partial semantics) and `usemT.cuf` (types for underspecified semantics). For the entry for the verb `likes` above, the calls to semantic sorts result in a syntactic-semantic feature structure as follows (less relevant features suppressed):

```
(stree &
 root:(synsem_pair &
       syn:(fin_pres &
            agr:(agr &
                 number:sg &
                 person:third)) &
       sem:(clause_sem &
            store:[(A & store_term),
                   (B & store_term),
                   (event_term &
                    res:(basic_relation &
                         rel:state &
                         args:(argument_frame &
                               arg1:(C & at))) &
                    index:C &
                    lexinfo:eventintro &
                    mods:[])] &
            matrix:(basic_relation &
                    rel:like_prime &
                    args:(argument_frame &
                          emarker:C &
                          arg1:(G & term) &
                          arg2:(H & term))))) &
 leaves:[(argument_leaf &
          dir:right &
          cat:(arg_stree &
               root:(synsem_pair &
                     syn:(dp & case:acc) &
                     sem:(I &
                          partial_sem &
                          lambda_index:H &
                          body:B)))),
         (argument_leaf &
          dir:left &
          cat:(arg_stree &
               root:(synsem_pair &
                     syn:((dp & ~trace) &
                          case:nom &
                          agr:(agr &
                               number:sg &
                               person:third)) &
                     sem:(J &
                          partial_sem &
                          lambda_index:G &
                          body:A))))])
```

Here the semantics of the verb is realised by the two features `store` and `matrix`. The `matrix` contains the basic argument structure of the verb while the `store` contains terms whose semantics derive from the semantics of the two `dp` arguments of *likes*. In addition the store contains a Davidsonian-style `event_term` in whose `mods` slot information about adjuncts is encoded.

3.2 Temporal Relations

The temporal portion of the implementation currently covers these main areas:

- Constants
- Syntactic features holding information crucial to processing of temporal information
- Events/states and the times they occur
- Narrative progression
- Temporal connectives

Except for the information concerning temporal connectives (which is taken from Hitzeman 1995a), the temporal information is implemented in a manner consistent with Kamp & Reyle (1993) Chapter 5.

3.2.1 Syntactic features

In order to handle temporal information, several features have been added to the syntactic tree:[1]

```
verb ::
  stat : bool,
  tp   : bool,
  tppt : referent,
  rpt  : referent,
  prev_context : list,
  durative : bool.

  bool = {plus, minus}.

verb  = finite | nonfinite.

          finite ::
             tense : tense,
             perf  : bool.
          tense = {past, pres, fut}.
```

The features can be described as follows:

- **stat:** If **stat** has a *plus* value, the aspect of the verb is *stative*; a *minus* values indicates an *event*. The value of this feature is determined by the **verb** sort (lexrules.cuf) at a time when the tense of the VP is known, because the present and progessive tenses make a sentence stative.

[1]These features are discussed in Kamp & Reyle p. 598.

Table 1: The values for *TP* and *TENSE*.

	TP	**TENSE**
Simple Present	*-PAST*	*pres*
Simple Past	*+PAST*	*pres*
Simple Future	*-PAST*	*fut*
Present Perfect	*-PAST*	*pres*
Past Perfect	*+PAST*	*past*
Future Perfect	*-PAST*	*fut*

- **tppt:** The Temporal Perspective Point (TPpt) is like Reichenbach's time point R and will be discussed further in the section on narrative progession, below. (See also Kamp & Reyle p. 610.)
- **tp** and **tense:** Their values are set according to Table 1.
- **rpt:** The Reference Point (Rpt) aids in tracking narrative progression. Its value is determined according to the following algorithm, where S is the current sentence (K&R, p. 545):
  - The part of the discourse preceding S contains an earlier event-sentence in the past tense. For this case we stipulate that the reference point be the discourse referent representing the event described by the most recent past tense event-sentence before S.
  - The antecedent part of the discourse contains no past tense event-sentence. In this case we let the reference point be the location time of the most recent past tense state-sentence. Otherwise we set the reference point equal to some new arbitrary time (represented by a new discourse referent).

  (See also Kamp & Reyle p. 610.)
- **prev_context:** Initially the previous context is set to a list containing only the **now point** (described below in the section on Constants). After each parse, the semantics of the sentence parsed is added to the *prev_context* slot, which represents the accumulated semantics of the discourse.
- **durative:** Whether an event occurs at a moment or has some greater duration is important for processing temporal connectives. An event such as **build a house** will have the value *plus* and an event such as bf notice a problem will have the value *minus*.
- **perf:** When a sentence is in the perfect tense, the value of **perf** at the root of the tree will be set to *plus*. This value is passed up from the auxiliary.

3.2.2 Constants

The only constant currently in use is Kamp & Reyle's **now point**, which they refer to as **n**. The now point is currently the total initial contents of the store, and is used in expressing temporal relations, e.g., a past tense event will always precede the now point. The **now point** is added to the **prev_context** slot in the file lg-sys/hdparser.pl.

3.2.3 Events/states and times

When an event or state is added to the discourse, a store term of type *eventintro* is added to the store. Accompanying this, there is a store term that indicates whether this is an event or a state (important for narrative progression). These store terms are added to the store in the file psemT.cuf by means of the **verb_sem** sort.

3.2.4 Narrative progression

Narrative progression is dealt with by means of Kamp & Reyle's *Rpt*. It is necessary to distinguish between two types of reference times: Reichenbach's *R* and an additional reference time used in DRT. Kamp & Reyle call Reichenbach's *R* the *Temporal Perspective point (TPpt)* to reflect Reichenbach's idea that this is the time from which the event is viewed, and they call the other reference time, which is used to track narrative progression, the *Rpt*. The distinction is motivated by the following example:

(1) Fred arrived at 10. He had got up at 5; he had taken a long shower, had got dressed and had eaten a leisurely breakfast. He had left the house at 6:30. (Kamp&Reyle 5.161)

In (1) the *TPpt* of each of the past perfect sentences is the time of Fred's arrival, but the *Rpt* changes as the narrative progresses. Kamp & Reyle describe how the *Rpt* may be determined, giving the following two cases, where S is the current sentence (p. 545):

(2) The part of the discourse preceding S contains an earlier event-sentence in the past tense. For this case we stipulate that the reference point be the discourse referent representing the event described by the most recent past tense event-sentence before S.
(3) The antecedent part of the discourse contains no past tense event-sentence. In this case we let the reference point be the location time of the most recent past tense state-sentence. Otherwise we set the reference point equal to some new arbitrary time (represented by a new discourse referent).

The reference times for future tense are chosen similarly.

I will follow Kamp (1979), Hinrichs (1981), and Partee (1984) in assuming that a new event is interpreted as following the current *Rpt*, while a new state overlaps it. For example, the event of the fox ducking into the foxhole is interpreted as occurring after the fox hears the noise in (4), but the state of the fox being in the foxhole is interpreted as overlapping the event of the fox hearing the noise in (5):

(4) The fox heard a noise. He ducked into the foxhole.
(5) The fox heard a noise. He was in the foxhole.

The choice of the *TPpt* depends on the tense of the sentence. I will follow Kamp & Reyle in their use of the feature *TENSE* to express the relationship between the eventuality and the *TPpt*. When *TENSE* has the value *pres* the time of the eventuality coincides with the *TPpt*, when it has the value *past* the time of the eventuality precedes the *TPpt*, and when it has the value *fut* the time of the *TPpt* precedes the eventuality. Similarly, I will use the feature *TP* to express the relationship between the *TPpt* and the speech time. The values for *TP* and *TENSE* for each tense are shown in Table 1.

When *TP* has the value *pres*, the *TPpt* is the same as speech time, which Kamp & Reyle refer to as **n**. When *TP* does not have the value *pres*, the eventuality whose time is the time of the *TPpt* must be found. For example, in (1) the *TPpt* for each of the past perfect sentences is the time of Fred's arrival. How to decide what this eventuality is is a difficult problem, and beyond the scope of this work. I will assume that some algorithm exists to accomplish this, and when *TP* has a value other than *pres* I will set the *TPpt* to some referent intended to represent the appropriate *TPpt*.

3.2.5 Temporal connectives

For the Dandelion work on temporal connectives, each tensed clause requires a series of values that describe in TTT terms (A "Two-Track Theory of Tense") the tense and aspect of the clause and the temporal relationship(s) of the eventuality it describes and other eventualities in the discourse. These have all been gathered in one store term of type `ttt` for modularity. The relevant code can be found in the file usemT.cuf. Temporal connectives use this information to determine their acceptability with various eventualities, and to calculate the temporal relations between the two eventualities they relate.

A detailed discussion of the temporal connectives can be found in Hitzeman 1995a; Verschuur 1995 and in Hitzeman 1995b.

4 Using Information Structure

The DII component which computes information structure has a more much more restricted coverage than the general DII grammar and the process of using accent information to calculate information structure often results in ambiguities which are not strictly relevant to other aspects of DII. For these reasons it was felt that the use of this component should be optional and efforts have been made to allow users to ignore or even remove the information structure component. In general the addition of the information structure component has involved writing counterparts to existing CUF sorts or Prolog predicates which take an extra accent argument. For the parser the extra predicates are to be found in the file `lg-sys/auxpreds.pl`: they are `leaf_constr/3`, `assert_positions_aux/3` and `cuflist_prologlist/2`. Lexical look-up for normal parsing involves the sort `l/2` but for the information structure component it uses `l/3` where the extra argument encodes accent information. In the grammar itself, certain features, types and sorts specific to information structure can be found in the file `inf_struc.cuf`. This file also includes alternate versions of certain sorts which affect the expansion of lexical entries (i.e. some sorts whose usual definitions can be found in `lexrules.cuf`, `sem.cuf` and `synsem.cuf`). In order for the parser to receive word-accent pair input as an alternative to its usual input of simple words, an alternate word-accent form for each lexical entry must be created. The file `lexicon.cuf` contains entries in a form which can be expanded either with or without the accent. Some examples are as follows:

```
word(book) := [noun,neut_sg,count_noun,book,(A & accent_info)].
word(laugh) := [verb,weak,v(0),process,v1,laugh,(A & accent_info)].
word(every) := [det,d(sg),every,u].
```

As explained in the next section, these entries are expanded in two different ways to create unaccented and accented expanded lexical entries (in `expanded_lexicon.cuf` and `is_expanded_lexicon.cuf` respectively):

Unaccented:

```
l(book,_406):=noun(neut_sg,count_noun,noun_sem(book_prime)).
l(laugh,_357):=verb(weak,v(0),process,v1,laugh_prime).
l(every,_380):=det(d(sg),determiner_sem(every)).
```

Accented:

```
l(book,_414,_290&accent_info):=
     noun(neut_sg,count_noun,noun_sem(book_prime,_290&accent_info)).
l(laugh,_453,_318&accent_info):=
     verb(weak,v(0),process,v1,laugh_prime,_318&accent_info).
l(every,_380,u):=det(d(sg),determiner_sem(every,u)).
```

The unaccented lexical entries are defined using the sort `l/2` while the accented ones are defined with `l/3` where the third argument of `l/3` is the accent argument. The unaccented entries are defined in terms of the sorts `noun/3`, `verb/5` and `det/2` while the sorts for the accented entries have an extra accent argument: `noun/4`, `verb/6` and `det/3`. The shared variable in the final arguments of `l/3` and the sorts `noun/4` and `verb/6` allow the noun and verb entries to be underspecified for accent.

During parsing, the information structure component uses accent information to compute values for the feature `info_struct`. This feature occurs in the semantic representation of noun phrases and verbs and indicates whether an element is part of the focus, the link or the tail. Information about the `info_struct` status of elements is used in the later stages of processing for a variety of purposes: for example it interacts with the centering component for anaphora resolution and with the presupposition component for examples involving focus-sensitive adverbs.

For more information about the theory of information structure see Vallduví 1990; Vallduví 1994 and Engdahl and Vallduví 1994. For further details concerning the DII implementation of the information structure component see Beaver and van der Bovenkamp 1995.

5 Creating the Expanded Lexicon

As briefly outlined in the previous section, DII has an initial lexicon, `lexicon.cuf`, in which lexical information is encoded in a format which is relatively theory- and grammar-neutral.

This initial lexicon is input to a lexicon expansion process which results in the lexicon `expanded_lexicon.cuf` and this expanded lexicon is the one that is actually used by the grammar.

The mapping process is performed by a call to the command `gram_mk_lexif` (defined in the file `mk_lexif.ctrl`). This command reads entries from the file `lexicon.cuf` and, using a set of mapping rules defined by the user in the file `lex_transpo.ctrl`, it transforms them into the output format and writes them to the file `expanded_lexicon.cuf`. Users wishing to add or ammend lexical entries should therefore edit the file `lexicon.cuf` and also check in case changes are needed to the mapping rules in `lex_transpo.ctrl`. The lexicon is then expanded by a call to `gram_mk_lexif` and the resulting new version of `expanded_lexicon.cuf` will be utilised after a call to the `gload` command:

```
| ?- gram_mk_lexif.

%%%* LEXICON EXPANSION:   lexicon.cuf  ==>  expanded_lexicon.cuf

| ?- gload.
```

If the information structure component is being used then any changes to `lex_transpo.ctrl` should be mirrored in the file `is_lex_transpo` (which contains the mapping rules for accented lexical entries). A new version of the file `is_expanded_lexicon.cuf` should then be created by a call to the command `is_gram_mk_lexif`:

```
| ?- is_gram_mk_lexif.

%%%* LEXICON EXPANSION:   lexicon.cuf  ==>  is_expanded_lexicon.cuf
```

The morphophonological extension to CUF, CUF-RA, can be used to generate a full set of entries for inflectional paradigms and in DII we have provided the option of creating verb and noun entries using CUF-RA. In the files `lex_transpo.ctrl` and `is_lex_transpo.ctrl` the mapping rules for verbs and nouns have been commented out because these entries occur in the CUF-RA output files `verbs_exp.cuf` and `nouns_exp.cuf` (and `is_verbs_exp.cuf` and `is_nouns_exp.cuf` for the information structure component) . Users not wishing to use CUF-RA should edit the file `dii.cnf` to prevent the files `verbs_exp.cuf`, `nouns_exp.cuf`, `is_verbs_exp.cuf` and `is_nouns_exp.cuf` from being loaded. The verb and noun mapping rules in `lex_transpo.ctrl` and `is_lex_transpo.ctrl` should be uncommented, the lexicon re-expanded and the system reloaded.

6 Using CUF-RA

CUF-RA is a system for generating morphological paradigms for words. It represents lexical items as finite-state automata and it effectively implements Bird and Ellison's (1994) "one-level" approach to phonology and extends it to deal with morphophonological phenomena. For details about CUF-RA and how to use it, see Manandhar 1995.

The example morphological descriptions distributed with CUF-RA are small illustrative descriptions which do not reflect the potential complexity of using CUF-RA to creat lexical entries for large scale CUF grammars like DII. For this reason it has been instructive to use CUF-RA with a large and heavily lexicalist grammar like DII and indeed the demands of DII have provoked certain additions to CUF-RA. The major problem that was encountered involved lexical definitions containing delayed goals. CUF-RA takes a stem form and generates inflected lexical entries using a set of word-formation rules. In these word-formation rules the output feature description is expressed using standard CUF feature terms but in the course of processing the feature terms are fully evaluated and the feature structure that results is written to the output file. With the DII grammar, several sorts which occur in the feature term description of lexical entries are ones which have delayed goals and these could not be written to an output file and subsequently loaded into CUF. The solution was to isolate the morphological information relevant to the word-formation rules from other information associated with lexical items and to let CUF-RA manipulate the former while simply writing out the latter without evaluating it. The morphological description for verbs in the file `verbs.pl` also exists in a simplified form as the example description english_verbs.pl distributed with the CUF-RA system. A stem entry in the simpler version looks like this:

```
enter -*-> vweak.
```

while the equivalent stem in `verbs.pl` look like this:

```
(enter,verb(weak,_,achievement,v2,enter_prime))
             -*-> verb(weak,_).
```

In the simple case there is no information apart from morphological information associated with the verb while, in the complex case, the DII grammar clearly needs a variety of syntactic and semantic information in verb lexical entries. In order to accommodate the more complex cases, CUF-RA was extended to allow two kinds of stem definitions. The complex kind has the stem word paired with extra non-morphological information. When CUF-RA generates inflected forms from a complex stem it ignores the extra information in the stem definition but it does write it to the output file conjoined with the result of morphological processing. For example, the following is the third person singular entry deriving from the complex stem definition for *enter*.

```
l(enters,[]):=
   (stree &
    root:(synsem_pair &
           syn:(fin_pres &
                 agr:(agr &
                       number:sg &
                       person:third) &
                 morph:weak)))
& verb(weak,_12814,achievement,v2,enter_prime).
```

Notice that the sort `verb/5` used to describe the extra non-morphological information is the same sort as is used in the definitions of verb lexical entries in the expanded lexicon described in the previous section. The second argument of `verb/5` is where morphological information is usually encoded and in this case it is unspecified precisely because the feature structure resulting from the CUF-RA generation process supplies this information. The version of CUF-RA distributed with the DII system has been customised for use with DII to ensure that output lexical entries are defined using the sort `l/2`: this means that there is an exact equivalence between the CUF-RA entry for *enters* above and the expanded lexicon entry for *enters* shown below:

```
l(enters,_357):=verb(weak,v(sg3),achievement,v2,enter_prime).
```

It is beyond the scope of this manual to describe the word-formation rules and paradigms involved in CUF-RA descriptions and users are referred to Manandhar 1995 to help them understand the contents of the files `verbs.pl` and `nouns.pl`. In general the process of adding a new verb will involve making decisions about whether it is weak or strong (regular or irregular) and how its orthographic form is affected by affixation. For example, a regular verb like *arrange* will need the following stem definition:

```
(arrange-[e],verb(weak,_,process,v2,arrange_prime))
            -*-> verb(weak,_).
```

The final 'e' is enclosed in square brackets to indicate that it will be elided when occurring with a suffix beginning with a vowel, e.g. the past tense form is *arranged* not *arrangeed*.

To generate the inflected forms for verbs and nouns, the DII grammar must be loaded. Then the file containing the DII-customized version of CUF-RA must be consulted (`Morph/cuf_ra` or `Morph-sol/cuf_ra` when using the Solaris operating system) and the commands `lex_compile(verbs)` or `lex_compile(nouns)` must be given:

```
| ?- ['Morph/cuf_ra'].
{consulting /projects/DYANA/DII/Morph/cuf_ra.pl...}

Loading foreign files...
...loaded

CUF-RA Version 1
{/projects/DYANA/DII/Morph/cuf_ra.pl consulted, 1170 msec 73552 bytes}

yes
| ?- lex_compile(verbs).
```

The results of `lex_compile` are written to the output files `verbs_exp.cuf` and `nouns_exp.cuf` respectively and are available following a call to `gload`.

As with the previous method of lexicon creation, a separate accented lexicon for use with the information structure component can be generated. The DII-customized file `Morph/cuf_ra` (or `Morph-sol/cuf_ra`) contains extra top-level calls to define the predicate `is_lex_compile`. In addition separate input files are required namely `is_verbs.pl` and `is_nouns.pl`. These are based on the files `verbs.pl` and `nouns.p` but extra information about accent has been added. The stem definition for *enter* in `is_verbs.pl` is as follows:

```
(enter,verb(weak,_,achievement,v2,enter_prime,A))
            -*-> (verb(weak,_),A).
```

and the third person singular output is this:

```
l(enters,[],_12823):=
   (stree &
    root:(synsem_pair &
          syn:(fin_pres &
               agr:(agr &
                    number:sg &
                    person:third) &
               morph:weak)))
& verb(weak,_12829,achievement,v2,enter_prime,_12823).
```

As before, these entries make use of the sorts `l/3` and `verb/6` which contain extra accent arguments not in the standard sorts `l/2` and `verb/5`. Notice the shared variable in the final argument positions of `l/3` and `verb/6` which permit the verb to be underspecified for accent. The information structure variant of CUF-RA is used in the same way as the regular version:

```
| ?- ['Morph/cuf_ra'].
{consulting /projects/DYANA/DII/Morph/cuf_ra.pl...}

Loading foreign files...
...loaded

CUF-RA Version 1
{/projects/DYANA/DII/Morph/cuf_ra.pl consulted, 1170 msec 73552 bytes}

yes
| ?- is_lex_compile(is_verbs).
```

The output files are `is_verbs_exp.cuf` and `is_noun_exp.cuf`.

Postscript

Any queries or requests for assistance with DII should be directed to one of the following:

Claire Grover	`grover@cogsci.ed.ac.uk`
Janet Hitzeman	`janet@cogsci.ed.ac.uk`
Peter Krause	`peter@ims.uni-stuttgart.de`

References

Alshawi, H. (ed.): 1992, *The Core Language Engine*, The MIT Press

Beaver, D. (ed.): 1995, *The Dyana Integrated Implementation*, DYANA-2 Report R3.7, ILLC/Department of Philosophy, University of Amsterdam

Beaver, D. and Krause, P.: 1995, The Architecture and Semantic Representation Formats of Dyana's Integrated Implementation, in Beaver, D. (ed.), *The Dyana Integrated Implementation*

Beaver, D. and van der Bovenkamp, I.: 1995, Application and Implementation of Information Packaging, in Beaver, D. (ed.), *The Dyana Integrated Implementation*

Bird, S. and Ellison, M.: 1994, One-level phonology: Autosegmental representations and rules as finite automata, *Computational Linguistics* 20(1)

Brennan, S., Friedman, M., and Pollard, C.: 1987, A centering approach to pronouns, in *Proc. ACL-87*, pp 155–162

Dörre, J., Dorna, M., and Junger, J.: 1994, *The CUF User's Manual*, Technical report, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart

Elworthy, D. A.: 1995, A Theory of Anaphoric Information, *Linguistics and Philosophy* 18, 297–332

Engdahl, E. and Vallduví, E.: 1994, Information packaging and grammar architecture: a constraint-based approach, in E. Engdahl (ed.), *Integrating Information Structure into Constraint-based and Categorial Approaches*, pp 39–80, DYANA-2 Deliverable R1.3.B, ILLC/Department of Philosophy, University of Amsterdam

Fernando, T.: 1993, Generalized quantifiers as second-order programs, in Paul Dekker and Martin Stokhof (ed.), *Proceedings of the Ninth Amsterdam Colloquium*, Vol. II, pp 287–300, Universiteit van Amsterdam

Frank, A. and Reyle, U.: 1995, Principle based semantics for HPSG, in *Proceedings of the 6th Meeting of the Association for Computational Linguistics, European Chapter*, Dublin

Grosz, B. J., Joshi, A. K., and Weinstein, S.: 1995, Centering: A framework for modeling the local coherence of discourse, *Computational Linguistics* 21(2), 203–225

Hinrichs, E.: 1981, *Temporale Anaphora in Englischen*, StaatsExamen thesis, Universität Tubingen

Hitzeman, J.: 1995a, *A Constraint-based Grammar of English Temporal Connectives*, ESPRIT Basic Research Project 6665 R2.3.3, DANDELION

Hitzeman, J.: 1995b, Temporal Connectives: A Combined Dyana/Dandelion Implementation, in Beaver, D. (ed.), *The Dyana Integrated Implementation*

Kamp, H.: 1979, Events, instant and temporal reference, in R. Bauerle, U. Egli, and A. von Stechow (eds.), *Semantics from Different Points of View*, pp 376–417, Springer-Verlag

Kamp, H. and Reyle, U.: 1993, *From Discourse to Logic*, D. Reidel, Dordrecht

König, E.: 1994a, *A Study in Grammar Design*, Arbeitspapier des Sonderforschungsbereich 340 54, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart

König, E.: 1994b, *LexGram User's Manual*, Technical report, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart

König, E.: 1995, Lexgram - a practical categorial grammar formalism, in *Proceedings of the Workshop on Computational Logic for Natural Language Processing. A Joint COMPULOGNET/ELSNET/EAGLES Workshop*, Edinburgh, Scotland

Krause, P.: 1995, Plural Phenomena in Dyana's Integrated Implementation, in Beaver, D. (ed.), *The Dyana Integrated Implementation*

Manandhar, S.: 1995, *The CUF-R/CUF-RA Guide*, Dyana report

Oversteegen, L.: 1993, *Dutch Temporal Connectives*, ESPRIT Basic Research Project 6665 R1.3.2a, DANDELION

Partee, B. H.: 1984, Nominal and temporal anaphora, *Linguistics and Philosophy* 7

Pollard, C. and Sag, I. A.: 1994, *Head-Driven Phrase Structure Grammar*, The University of Chicago Press, Chicago

Vallduví, E.: 1990, *The Informational Component*, Ph.D. thesis, University of Pennsylvania

Vallduví, E.: 1994, The dynamics of information packaging, in E. Engdahl (ed.), *Integrating Information Structure into Constraint-based and Categorial Approaches*, pp 1–26, DYANA-2 Deliverable R1.3.B, ILLC/Department of Philosophy, University of Amsterdam

Verschuur, L.: 1995, *Constraint-based Grammar of Dutch Temporal Connectives*, ESPRIT Basic Research Project 6665 R2.3.1a, DANDELION

Westerstahl, D.: 1985, Determiners and context sets, in J. van Benthem and A. ter Meulen (eds.), *Generalized Quantifiers in Natural Language*, pp 45–71, Foris, Dordrecht