Appeared in *Dependency and Valeny. An International Handbook of Contemporary Research.* Edited by V. Agel, L.M. Eichinger, H.-W. Eroms, P. Hellwig, H.-J. Heringer, H. Lobin. Volume I. S. 593-635. Mouton: 2003.

CORRECTED AND REVISED VERSION, LAST UPDATE 20/08/2004
(Changes as compared to the printed version are marked in blue.)

Dependency Unification Grammar

1. Introduction

A grammar is a theory about a language. Theories may be formulated in a natural language, describing the properties of the object in question. Another way of providing information about an object is designing a model. A model is an artificial construct resembling the object in certain respects. In linguistics, formal models play a major role. A formal model (or formalism) consists of a set of symbols and a set of instructions for symbol manipulation. A popular type of formal model in linguistics is graphs consisting of nodes and edges, especially a particular type of graph called a "tree".

The object of study and the formal model must be related by isomorphism. Therefore, the debate among grammarians centers around two questions: Firstly, what are the empirical facts of language? Secondly, what should the model be like in order to be isomorphic with the facts? The answer is a matter of choice rather than a matter of truth. On the one hand, the choice of facts depends on our interest and the problems we want to solve. On the other hand, the same fact can be modeled by a great variety of formal means. What is important is to be aware that the model is not the reality. There are no trees in natural languages! At best there are some properties in a language that are analogous to properties of tree graphs. Nevertheless, the choice of a formalism does matter. Imagine mathematics with Roman numerals; Arabic numerals are far more practical and provide more insights. The same might be true of some grammar formalisms as compared to others.

Dependency Unification Grammar (DUG) has been developed at the University of Heidelberg in the course of attempts to process natural languages by computer. The first version, called Valency Grammar, was published in Hellwig (1978). The DUG formalism is, as it were, the linguistic programming language of the natural language processing system PLAIN (Programs for Language Analysis and Inference, cf. Hellwig

1980). Meanwhile, the application of PLAIN covers deductive question answering, machine translation, grammar checking, and automatic text summarizing.

Broad coverage is a prerequisite of real-world applications. As a consequence, the formal grammar in a language processing system must be able to mirror the most intricate phenomena of real texts. On the other hand, the formalism has to be perspicuous and simple so that grammarians and lexicographers can easily draw up large data resources. DUG is based on the assumption that the solution to this problem is a formalism that is as close as possible to the informal descriptions of languages which fill the libraries of linguistic departments. If such a formalism is at hand then opening up the wealth of existing knowledge to the formal treatment by computers is just a matter of translation.

Dependency grammars are more natural, in a number of respects, than constituency grammars. Therefore, dependency has been chosen as the basic relationship in our formal model. However, pure dependency trees, like any simple graph structure, cannot represent all the phenomena observed and described in the linguistic literature. DUG is characterized by a number of enrichments of the dependency approach which will be outlined below. Future enrichments are likely to occur, since there will always be new phenomena to be dealt with. The idea is to pair each predicate occurring in traditional linguistic statements with an equivalent in the formal model. Whereas traditional descriptions refer to a certain behavior of the linguistic material, the constructs of the DUG formalism induce a certain operation of the computer so that its input and output are similar to the originally observed material.

The goal of this article is to show how DUG copes with various linguistic phenomena. The article also discusses which properties might be ranked amongst the essentials of dependency grammars in general.

2. The formalism

Data in the PLAIN system, including grammatical resources, are drawn up in Standard Generalized Markup Language (SGML). The format is precisely delineated in a special Document Type Definition (DTD). Such a format is not suitable for human readers, though. Therefore, a semi-formal BNF-definition of the DUG formalism is chosen in this article. Constituents are expanded only as far as necessary to identify them conceptually. The shape of constructs is often illustrated by an example rather than being defined in detail. The hierarchy of rules is relaxed in favor of the perspicuity of the introduction. Some of the definitions are temporary and will be refined later. The numbers of such temporary rules are marked with an apostrophe.

The task of any analysis is to delineate elementary units, to determine the relationship between the elements, and to describe their properties. In linguistics, segmentation, commutation, and classification are the fundamental operations used to solve this task. A grammar formalism must be suitable for representing the resulting units, structure, and properties. Dependency grammars as well as phrase structure grammars use trees (directed acyclic graphs) in order to depict the units and the structure of a given phrase or sentence. However, the isomorphism between the formal model and the original phrase is quite different in the two approaches. The edges in a phrase structure tree represent the composition of larger phrases from elementary ones. Each level in a phrase structure tree denotes another segmentation of the original sentence or phrase. The topmost node covers the whole expression, subordinated nodes cover smaller parts of the same expression and so on. In contrast, each node in a dependency tree refers to an elementary segment (e.g. a word) of the corresponding phrase or sentence. All nodes belong to the same unique and complete segmentation. Composed segments are present only as a combination of elementary segments which are associated with the nodes of a partial or complete dependency tree.

The general shape of the DUG-formalism is defined as follows:

```
(R-01)    <DUG formalism>      :=   <list>+
(R-02)    <list>               :=   <term>
(R-03)    <term>               :=   "(" <projection>?  <attribute>*   <term>* ") "
(R-04)    <projection>         :=   <left dependent> | <right dependent>
(R-05)    <left dependent>     :=   {"<"}+
(R-06)    <right dependent>    :=   {">"}+
(R-07)'   <attribute>          :=   <identifier> "[" <value> "]"
(R-08)    <value>              :=   <feature> {"," <feature>}*
```

Dependency trees are the conceptual backbone of the DUG formalism. The graphic display of trees is unwieldy for input and output by computer, though. Therefore a linear representation called "list" has been introduced. A list starts with a (heading) term. Each term is surrounded by brackets and may contain one or more (dependent) terms each of which is surrounded by brackets as well. The resulting bracketed expression is isomorphic to a directed acyclic graph. The terms correspond to the nodes; the hierarchy of brackets corresponds to the edges between the nodes of the tree.

Dependency trees as well as the corresponding bracketed expression fail to display the concatenation of the strings that are associated with the nodes or terms. Especially the location of the head among the dependents is unclear in a simple dependency tree. The unequivocal projection of the terms of a list onto the linear plane is achieved in DUG by a special projection marker included in the dependent terms. If a dependent unit is situated to the left of its head then its term is marked by "<"; if it is situated to the right of the head unit then its term is marked by ">". If two dependents are situated on the same side of a head then their projection with respect to each other is simply represented by the sequence of their terms in the list. The left dependent and the right dependent marker are also used to describe extra-positioned (non-projective) dependencies. A left dependent symbol or a right dependent symbol occurring *n* times means that the unit corresponding to the term in question is located to the left or the right of the unit that is *n* levels higher in the hierarchy of terms. Examples will be given when we turn to discontinuous word order below.

Any grammar formalism should provide flexible means for the description and classification of linguistic entities. This is achieved by an arbitrary number of attributes that can be listed within each term. A list with attributes is equivalent to a dependency tree with labeled nodes. As a rule, an attribute consists of two parts: an identifier and a value. The identifier reveals the type of the attribute. The value is enclosed in square brackets and consists of a single feature or a disjunction of features separated from each other by commas. Identifiers and features are denoted by arbitrary names. The same identifier cannot occur more than once in a term.

The semantics of the attributes is discussed in detail below. As pointed out earlier, the DUG formalism is supposed to provide attributes that correspond to statements in traditional linguistic descriptions. The interpretation of attributes is realized by particular operations of the computer. With respect to its formal properties, DUG ranks amongst attribute grammars in the sense of Knuth (1968). Attribute grammars are an extension of context free grammars. They give a meaning to every derivation tree of the underlying context free grammar by assigning values to various attributes associated with the nodes of the tree. It depends on the type of attributes whether the resulting grammar is still context-free. With its standard attributes DUG is mildly context-sensitive.

(1) *The robot picks up a big red block.*

```
(string[.] role[illocution] lexeme[statement'] category[sign] utterance[+]

        (< string[picks ] role[predication] lexeme[pick] category[verb] form[finite] tense[present]
        voice[active] person[it,U] s_type[statement] adjacent[+] margin[left] s_position[4,6,14,17]
        first_letter[capital]

                (< string[robot ] role[subject] lexeme[robot] category[noun] number[singular]
                person[it,C] determined[+,U] reference[definite,U] s_position[4] vowel[-,U]
                n_position[2,10]

                        (< string[The ] role[determination] lexeme[definite'] category[determiner]
                        number[singular,C] determined[+,C] reference[definite,C] n_position[2]))

                (> string[up ] role[phrasal_part] lexeme[up] category[particle] s_position[14])

                (> string[block ] role[dir_object] lexeme[block] category[noun] number[singular]
                person[it] determined[+,U] reference[indefinite,U] s_position[17] vowel[-,U]
                n_position[2,6,10]

                        (< string[a ] role[determination] lexeme[indefinite'] category[determiner]
                        number[singular,C] determined[+,C] reference[indefinite,C] n_position[2])

                        (< string[big ] role[attribute] lexeme[big] category[adjective]
                        use[attributive] vowel[-,C] n_position[6])

                        (< string[red ] role[attribute] lexeme[red] category[adjective]
                        use[attributive] vowel[-,C] n_position[6]))))
```

Figure 1: A list in DUG format.

Figure 1 is the parser output for sentence (1). One can visualize a dependency tree in the representation if rotating the list clockwise by 90 degrees and drawing edges between each heading term and all directly indented terms. Some attributes in Figure 1 should be evident, others may be puzzling. At the moment, the only purpose of the example is to illustrate the syntax of the formalism.

Apart from its syntax, a formalism is defined by the operations that are performed with its symbols. Two types of operations are characteristic for the DUG-formalism: feature unification and tree transformation.

Feature unification provides the means to construct a complete well-formed formal representation of a given sentence. Tree transformation is used for simulating all actions that are performed on given sentences in order to form new ones, e.g. paraphrasing, translating, summarizing etc. Formation and transformation are kept strictly separate in DUG. For any sentence or phrase there is a definition of its formation which does not rely on any transformation.

A representation like Figure 1 is a model of a sentence. Lists like Figure 1 substitute for the original sentences in the knowledge base which is drawn up in the PLAIN system for the purpose of information retrieval or question answering. A grammar, on the other hand, is the collection of all the resources needed in order to automatically assign a formal representation to a linguistic fragment. This assignment of a formal representation to instances of natural language is distributed among the following components:

- the morpho-syntactic system,
- the syntactic system,
- the lexical-semantic system,
- the logical-semantic system,
- the system of text cohesion and coherence.

Each of these systems is relatively independent and has its own resources. What the systems share (at least in part) is the dependency list structure of the representation. Words, sentences and texts are passed from one system to the other in the form of lists as defined above. The present article concentrates on the morpho-syntactic and the syntactic system of DUG. The other components will only be outlined briefly.

3. The morpho-syntactic system

3.1. A list of forms

The nodes in a dependency tree as well as the terms in a list correspond to those linguistic units that are elementary within the syntactic system. Except for a few questionable cases, these elementary units coincide with words. The elementary status of words is supported by spaces or punctuation marks which separate them in written language. Words occur in phrases and sentences in inflected forms. Depending on its form a word has a particular syntagmatic capacity which must be accounted for by attributes in the corresponding term. Since morphology and syntax intertwine on the elementary level, the term "morho-syntax" is used for the DUG component that deals with the form of words.

(R-09)'	<DUG>	:=	<morpho-syntax>
(R-10)'	<morpho-syntax>	:=	<form>+
(R-11)'	<form>	:=	<string> <term>+

At the interface to syntax, the morpho-syntactic system consists of strings associated with a term. In exceptional cases more than one term is possible. The string corresponds to an inflected word, and the terms provide the morpho-syntactic features characterizing the word form. The mapping of strings and terms is utilized in the PLAIN system both for the analysis and for the generation of utterances. Figure 2 shows a few examples.

string[call]	(lexeme[call] category[verb] form[infinitive, imperative])
string[call]	(lexeme[call] category[verb] form[finite] tense[present] person[I,you,we, they,they_personal])
string[calls]	(lexeme[call] category[verb] form[finite] tense[present] person[he,she,it])
string[called]	(lexeme[call] category[verb] form[finite] tense[past] person[I,you,he,she,it, we,they,they_personal])
string[sobbed]	(lexeme[sob] category[verb] form[finite] tense[past] person[I,you,he,she,it, we,they,they_personal])
string[fell]	(lexeme[fall] category[verb] form[finite] tense[past] person[I,you,he,she,it, we,they,they_personal])
string[was]	(lexeme[be] category[verb] form[finite] tense[past] person[I, he, she, it])

Figure 2: Morpho-syntactic classification of word forms

The listing in Figure 2 is the simplest type of a morpho-syntactic lexicon. Note that the attributes provide much freedom in grammar writing; lexical meaning, word class , and grammatical features are all included in the description. Figure 2 also illustrates the use of disjunctive features as attribute values. This concise representation of ambiguous forms reduces the number of elements the parser has to check for compatibility. The person attribute in Figure 2 may seem odd. Of course, the traditional attributes 'number[singular,plural]' and 'person[first,second,third]' could have been used as well. This would result in more lexical entries, though.

The simple model of associating a string (a morph) with attributes (morphemes) faces difficulties if a lexeme is associated with a sequence of more than one word or if one word represents more than one lexical morpheme. The first case is dealt with by admitting spaces within elementary strings. The second case (portmanteau morphs) lead to the stipulation that more than one term may be assigned to one string. See Figure 3 for examples. Of course, the decision to treat a given phenomenon one way or the other is controversial.

string[in spite of]	(lexeme[in_spite_of] category[preposition])
string[each other]	(lexeme[each_other] category[pronoun] pro_form[reciprocal] person[we,you,they, they_personal])
string[vice versa]	(lexeme[vice_versa] category[adverb])
string[aren't]	(lexeme[be] category[verb]form[finite] tense[present] person[you,we,they, they_personal]) (lexeme[not] category[adverb])
string[I've]	(lexeme[I]category[pronoun] pro_form[nominal] case[subjective] person[I]) (lexeme[have] category[verb] form[finite] tense[present] person[I])

Figure 3: Multi-word morphs and portmanteau-morphs

3.2. The item-and-arrangement model

At the bottom of the DUG morphology is the item-and-arrangement model (cf. Spencer 1991, ch. 2.3.1) . Words are broken down into smaller segments, especially stems and endings. The segments are rearranged in such a way that the inflection and derivation of words is displayed correctly. Inflection tables in traditional grammars serve as an example. A set of forms in a particular paradigmatic relationship is called a "paradigm". One paradigm may contain stems, other paradigms may contain endings of particular inflection classes, and still other paradigms may contain derivational affixes. Each segment may consist of a character string of any length, including the empty string. For each segment, separate lexical or grammatical information may be specified. Accordingly, we augment the definition of the morpho-syntactic system of a DUG as follows.

(R-10)'	<morpho-syntax>	:=	<form>+ | <paradigm>+
(R-11)	<form>	:=	<string> <term>? <continuation>?
(R-12)	<paradigm>	:=	<paradigm_id> <comment>? <form>+
(R-13)	<continuation>	:=	<paradigm_id>+

Technically, the concatenation between the segments of an inflected or derived word is realized by a finite state transition network (FTN). Each paradigm is implemented as a sub-net with a particular ID. Each sub-net has an initial state; each entry in the paradigm corresponds with an edge leaving the initial state. The target state of each edge depends on the correct linking between the given segment and the segments to follow, if any. The successor segments, e.g. a particular set of endings, must be gathered in a paradigm, i.e. another sub-net. Therefore the target of a continuation is specified in terms of a paradigm ID. The paradigm ID always leads to the initial state of the identified sub-net.

```
paradigm_id=stems  comment="word stems"
string[call]     (lexeme [call])    continuation=vinf,vprs-0,vprs-s, vpast-ed, vprp-ing , vpap-ed
string[talk]     (lexeme [talk])    continuation=vinf,vprs-0,vprs-s, vpast-ed, vprp-ing , vpap-ed
string[try]      (lexeme [try])     continuation=vinf, vprs-0, vprp-ing
string[trie]     (lexeme [try])     continuation=vprs-s, vpast-d, vpap-d
string[sob]      (lexeme [sob])     continuation=vinf,vprs-0,vprs-s
string[sobb]     (lexeme [sob])     continuation=vpast-ed, vpap-ed, vprp-ing
string[fall]     (lexeme[fall])     continuation=vinf,vprs-0,vprs-s, vpap-en, vprp-ing
string[fell]     (lexeme[fall])     continuation=vpast-0
```

Figure 4a: A fragment of the paradigm of English verb stems

```
paradigm_id=vinf comment="verbs, infinitive and imperative"
        string[  ] (category[verb] form[infinitive, imperative])
paradigm_id=vprs-0 comment="verbs, present tense, subset with no ending"
        string[  ] (category[verb] form[finite,subjunctive] tense[present] person[I,you,we,they)
        string[  ] (category[verb] form[subjunctive] tense[present] person[he,she,it])
paradigm_id=vprs-s comment="verbs, present tense, subset with ending -s"
        string[s ]  (category[verb] form[finite] tense[present] person[he,she,it])
paradigm_id=vpast-ed comment="verbs, regular past tense, with ending -ed"
        string[ed ]  (category[verb] form[finite] tense[past] person[I,you,he,she,it,we,they,
        they_personal])
paradigm_id=vpast-0 comment="verbs, irregular past  tense, with no ending"
        string[  ]  (category[verb] form[finite] tense[past] person[I,you,he,she,it,we,they,
        they_personal])
paradigm_id=vprp-ing comment="verbs, present participle"
        string[ing ] (category[verb] form[ing_participle])
paradigm_id=vpap-ed comment="verbs, regular past participle, ending -ed"
        string[ed ]  (category[verb] form[past_participle])
paradigm_id=vpap-en comment="verbs, irregular past participle, ending -en"
        string[en ]  (category[verb] form[past_participle])
paradigm_id=nS comment="derivation of a noun from a verb "
        string[ing ]  (category[noun] number[singular])
```

Figure 4b: A few paradigms of English verb endings

Each stem in Figure 4a is marked for continuation in a number of ending paradigms displayed in Figure 4b. Allomorphs among endings simply result in alternative ending paradigms. Variation within stems is a bit more complicated. If one does not want to give up the item-and-arrangement principle, each variant of a stem must be inserted separately in the stem-paradigm and linked with the appropriate subset of ending paradigms. All word forms can then be generated by finite state transition . All the attributes that are encountered during one path through the paradigms are collected in one term. The result of this mechanism is a mapping of strings and terms identical to the one in Figure 2.

Let us check up on some other morphological phenomena. Zero morphs as in *call-0* versus *call-s* could be represented by the empty string. Another possibility is to consider the space after a word a part of the ending. In this case, a zero morph is represented by the space character, as in Figure 4b.

Derivation by means of affixes is treated in the same way as inflection. For example, the last paradigm in Figure 4b results in the derivation of nouns from verbs by means of the *ing*-suffix. The FTN-model is especially efficient for languages with rich inflection and derivation. For instance, inflecting adjectives can be derived from the present participle of most verbs in German. For this purpose, the stem of the verb *lieben* (to love) is linked to the paradigm of present participles, containing the affix -*end*- (loving), which in turn is linked to the ordinary paradigm of adjective inflection, yielding 24 adjectives like *lieb-end-e, lieb-end-er, lieb-end-es* etc.

Compounds are at the borderline between word formation and syntax. If the parts of a compound represent distinct lexical meanings, each part should be stored separately in the lexicon and the building of the compound should be left to the syntax module. In general, an elementary syntactic unit (a word) is created if a path trough the transition network of paradigms exists, starting at the initial state and ending at a form with no continuation. When a form with no continuation is reached, the finite state automaton jumps back to the initial state of the whole network, assuming that a new word follows. This mechanism is suitable for single words as well as for multi-word units. It works with compounds as well. For example, the string *steamboats* is decomposed automatically into the two words *steam* and *boat-s,* if both words are stored in the network. The relationship between the two parts of the compound is then a matter of syntax rather than morphology.

So far, our model of morphology is equivalent to a right linear grammar. In terms of the Chomsky hierarchy such grammars define regular languages (type-3 languages). Discontinuous morphs, like the past participle with the prefix *ge-* and the suffix *-en* in German, are beyond the capacity of such grammars. If it is necessary to extend the capacity of the grammar in a controlled way, DUG makes use of attributes and the device of unification. Regarding morphology, it is stipulated that the values of attributes that are gathered in the course of the transition through the morphological paradigms must not contradict each other.

```
paradigm_id=root  comment="the starting node of the network "
          string[ge]  (ge_prefix[+] form[past_participle])  continuation=stems
          string[]   (ge_prefix[-] ) continuation=stems
paradigm_id=stems  comment="verb stems classified for presence or absence of the ge-prefix"
          string[geb]  (lexeme [geben] ge_prefix[+])  continuation=vinf, vpast-en
          string[vergess]  (lexeme [vergessen] ge_prefix[-])  continuation=vinf, vpast-en
paradigm_id=vinf  comment="infinitive morph following the verb stem"
          string[en  ] (category[verb] form[infinitive] ge-prefix[-])
paradigm_id=vpast-en comment="past participle morph following the verb stem"
          string[en  ]  (category[verb] form[past_participle]
```

Figure 5: Contextual constraints by means of attributes in German morphology

The transition network realized in Figure 5 accepts the infinitives *geb-en* and *vergess-en* and the past participles *ge-geb-en* and *ver-gess-en* because the attributes 'ge_prefix' and 'form' are in agreement in these cases. It rejects **geb-en* and **ge-vergess-en* as past participles due to the conflict regarding 'ge_prefix' and it prevents the form *ge-geb-en* from being accepted in paradigm 'vinf' because of the conflicting values of 'form'.

3.3. Morphological classes and morphological units

The morphological system described so far is beneficial for computer usage. A few constructs should to be added in order to improve data acquisition and maintenance.

(R-10)'	<morpho-syntax>	:=	<form>+ | <paradigm>+ <morphclass>+ <morphunit>+
(R-14)	<morphclass>	:=	<morphclass_id> {inflection | derivation}+
(R-15)	<inflection>	:=	<stem_no>? <prefix>? <change>? <suffix>? <paradigm_id>
(R-16)	<derivation >	:=	<stem_no>? <prefix>? <change>? <suffix>? <paradigm_id>
(R-17)	<morphunit>	:=	<lexeme> <reading>? <morphclass_id> <stem >
(R-18)	<stem>	:=	(<stem_no> <string>)+

Obviously it is inefficient to list the ending paradigms with each word. Each identical set of continuations represents a particular morphological class (a morphclass). Each of these classes is given a name (a morphclass_id). The cases of inflection and derivation that make up the particular class are described in a separate data set. Each case of inflection is defined by a variant of the word stem (identified by a number), the prefix or suffix added to the stem or the changes it undergoes if any, and the ID of the appertaining ending paradigm. The specimens of derivation are described in the same way. (Possible switches from upper case to lower case or vice versa, like verb to noun or noun to adjective in German, are taken care of by the change-attribute.)

A morphological unit (morphunit) represents one individual instance of morpho-syntactic behavior. A morphunit is not necessarily associated with a single lexical meaning. Since words are often ambiguous, a distinction is made between "lexeme" and "reading". A lexeme is the basic lexical morpheme of a (possibly ambiguous) word, a reading is one of the meanings of the word. A reading must be specified in a morphological unit only if the inflection or derivation depends on it. For example the plural of the German noun *Bank* is *Bank-en* if the reading is bank and it is *Bänk-e* if the reading is bench. Any morphunit specification is completed by the morphological class and a numbered list of stems. Figure 6a and 6b illustrate the interplay of morphological units and morphological classes. Compare the contents of these tables with Figure 4.

```
morphunit  lexeme[try]  morphclass_id=Va13y
           stem_no=1 string[try]
           stem_no=2 string[trie]
morphunit  lexeme[sob] morphclass_id=Va13b
           stem_no=1 string[sob]
morphunit  lexeme[fall] morphclass_id=Ve17
           stem_no=1 string[fall]
           stem_no=2 string[fell]
```

Figure 6a: The specification of morphological units in the lexicon

```
morphclass_id=Va13y
          stem=1          paradigm=vinf
          stem=1          paradigm=vprs-0
          stem=2          paradigm=vprs-s
          stem=2          paradigm=vpast-ed
          stem=2          paradigm=vpap-ed
          stem=1          paragigm=vprp-ing
          stem=1          paradigm=nS
```

```
morphclass_id=Va13b
          stem=1          paradigm=vinf
          stem=1          paradigm=vprs-0
          stem=1          paradigm=vprs-s
          stem=1          suffix="b" paradigm=vpast-ed
          stem=1          suffix="b" paradigm=vpap-ed
          stem=1          suffix="b" paragigm=vprp-ing
          stem=1          suffix="b" paradigm=nS
```

```
morphclass_id=Ve17
          stem=1          paradigm=vinf
          stem=1          paradigm=vprs-0
          stem=1          paradigm=vprs-s
          stem=2          paradigm=vpast-0
          stem=1          paradigm=vpap-en
          stem=1          paradigm=nS
```

Figure 6b: Morphological classes as referred to in morphological units

3.4. Cardinal forms and the item-and-process model

Specifying morphunits manually is cumbersome. An old arrangement for memorizing the inflection of words is known as cardinal forms. A cardinal form must display enough of the peculiarities of a word so that the whole inflection can be deduced. Figure 7 displays cardinal forms of the three verbs that were used for illustration in Figure 6a.

```
          to try tried tried trying
          to sob sobbed sobbed sobbing
          to fall fell  fallen falling
```

Figure 7: Cardinal forms

In order to draw up morphunits from cardinal forms automatically, a computer program must separate stems and endings, extract the lexeme and the various stems and detect the morphclass on the basis of the demonstrated inflection. This task is not difficult if the program is provided with patterns of the cardinal forms.

Including this extension, the final definition of the morpho-syntactic system is the following:

(R-10)	<morpho-syntax>	:=	<form>+ | <paradigm>+ <morphclass>+ <morphunit>+ <cardlform>+ <carldpat>+
(R-19)	<cardlform>	:=	<lexeme>? <reading>? <item>+
(R-20)	<cardlpat>	:=	<morphclass_id> <charset>? <cardlterm>+
(R-21)	<cardlterm>	:=	{<constant> | <stem_no> <citation form>? <prefix>? <change>? <suffix>? }

Cardinal forms (cardlform) consist of a sequence of items as shown in Figure 7. Optionally, there may be an explicit specification of the lexeme and reading. Cardinal forms are supposed to identify a morphological class whose ID is indicated in a cardinal pattern (cardlpat). A cardinal pattern contains a number of cardinal terms (cardlterm), each of which defines one item of the cardinal forms. An item may be a constant like the particle *to* in Figure 7 or it may be composite. One of the components is the word stem which can be automatically identified by removing the specified prefix or suffix, if any. The detected stems are associated with numbers (stem_no). One of the items among the cardinal forms should be flagged as the citation form of the word. This form is usually the one that is used as the value of the lexeme attribute. If the stem undergoes changes in the inflection then these must be described in the pattern. The change-attribute consists of a regular expression discriminating the relevant parts of the stem and a description of the changes in terms of bound variables and additional letters. Figure 8 shows the patterns necessary to interpret the cardinal forms in Figure 7:

```
morphclass_id=Va13y
          constant="to"
          stem_no=1 citation_form=yes
          stem_no=2 change="(.*)y/$1ie" suffix="d"
          stem_no=2 change="(.*)y/$1ie" suffix="d"
          stem_no=1 suffix="ing"
morphclass_id=Va13b
          constant="to"
          stem_no=1 citation_form=yes
          stem_no=1 change= "(.*)b/$1bb" suffix="ed"
          stem_no=1 change= "(.*)b/$1bb" suffix="ed"
          stem_no=1 change= "(.*)b/$1bb" suffix="ing"
```

```
morphclass_id=Ve17 charset="C=[f]"
        constant="to"
        stem_no=1 citation_form=yes
        stem_no=2 change= "(\C)a(.*)/$1e$2"
        stem_no=1 suffix="en"
        stem_no=1 suffix="ing"
```

Figure 8: Cardinal patterns

Cardinal forms and cardinal patterns not only simplify the creation of the morpho-syntactic lexicon, they represent a theoretically important extension of the DUG model. The item-and-arrangement model is ideal for computer implementation. However, it is suited only to agglutinative languages which combine morphological elements without changes of form or loss of meaning. In many languages an item-and-process model (cf. Spencer 1991, ch. 2.3.1) would be more appropriate, because the morphological elements of these languages vary due to phonological, psychological, etymological or other reasons. Formalisms have been invented that model such processes, for example the influential two-level morphology introduced by Koskenniemi (1983). Koskeniemi's morphology works with an underlying lexical level and the surface morpho-syntactic level. So-called transducers derive the surface word forms from the lexical representation, e.g. the past tense *fell* from the canonical representation *fall*. This is done on the basis of a set of intricate replacement rules. Cardinal forms in DUG cope with the same phenomenon. They deal with the changes by just giving an example. The replacement rules themselves are encoded in the cardinal patterns. While two-level morphology applies processes to words at runtime, the DUG implementation simulates the morphological process while the morpho-syntactic lexicon is drawn up. What is eventually stored in the computer lexicon is surface forms only.

4. The syntactic system

4.1. Syntactic structure

Two approaches are currently in competition as means for representing the syntactic structure of languages: sentence orientation and word orientation. The first approach has been prevalent since Chomsky (1957) defined a language as a set of sentences and provided the formal means to (theoretically) generate this set. Since then, formal syntax is usually conceived of as an abstract structure which results from the recursive division of non-terminal units into immediate constituents. Words are just filling material for such an abstract structure. Context restrictions that have to be accounted for are dealt with by so-called "lexical subcategorization".

The second approach is distinctive for dependency grammars. In this view, syntax is almost entirely a matter of the combination capacity of words. Words are not just fillers of a-priori existing structural patterns; they are the very source of such patterns. From the formal point of view, the essential concept of dependency grammar is the complement. A standard syntactic construction consists of a head element and a number of constituents that complete the head element. This division of syntactic constructions into head and complements follows from lexical semantics. Natural language is used to assign attributes to objects and to state relationships between objects in the real world. For this purpose, there are words that denote the relation and expressions that denote the objects. In the absence of complements a qualifying or relating word is unsaturated. However, the number and the kind of syntactic constructs that are suitable for complementing the particular word are predictable. If this prediction of complements plays a major role in the syntax formalism then we are dealing with dependency grammar.

Certainly, the intuitive distinction between head and complements is a prerequisite of any syntactic analysis. A formal description of syntax must reveal the distribution of linguistic units relative to each other. This program of taxonomic linguistics can be carried through only if the discovery procedures, especially permutation and substitution, conform to the levels of the head-complement hierarchy and the borders between complements. Any successful constituency analysis implicitly contains a correct head-and-complement division. The difference is that dependency grammars make this division explicit.

At this point I must mention what is in my opinion a severe misconception of dependency grammars. It is the assumption that dependency is a relationship between words only and that, therefore, no non-terminal constituents exist in a dependency grammar. If the concept of complement is fundamental for the analysis of syntactic structure, as I think it is, then a grammar cannot do without non-terminal constituents. Obviously complements consist of an arbitrary number of words. For example, the verb *persuade* has three complements (and hence dominates three subtrees in the dependency tree).

(2) *The vice-president persuaded the head of the department to attend the conference*

The three complements are *the vice-president*, *the head of the department* and *to attend the conference* and not just *president, head, attend.* While the heads are terminals, complements are non-terminal constituents! Each complement as a whole is characterized by a grammatical function; most morpho-syntactic features (case, agreement, word order) apply to the whole complement and not just to one word in it.

The mentioned misconception is due to the usual visualization of dependency trees. Since internally each complement is structured again according to the head-complement principle, it looks as if its internal head is the one which is in relationship with the external head and dependency is just a word-to-word relationship. In reality, it is not the edge between two nodes that represents the dependency relation but an edge between a node and a complete tree depending from that node. In Figure 9 the dependent trees are included in boxes in order to elucidate this fact. Each box embraces a complement. The dependency relation exists between words and such implicit boxes.

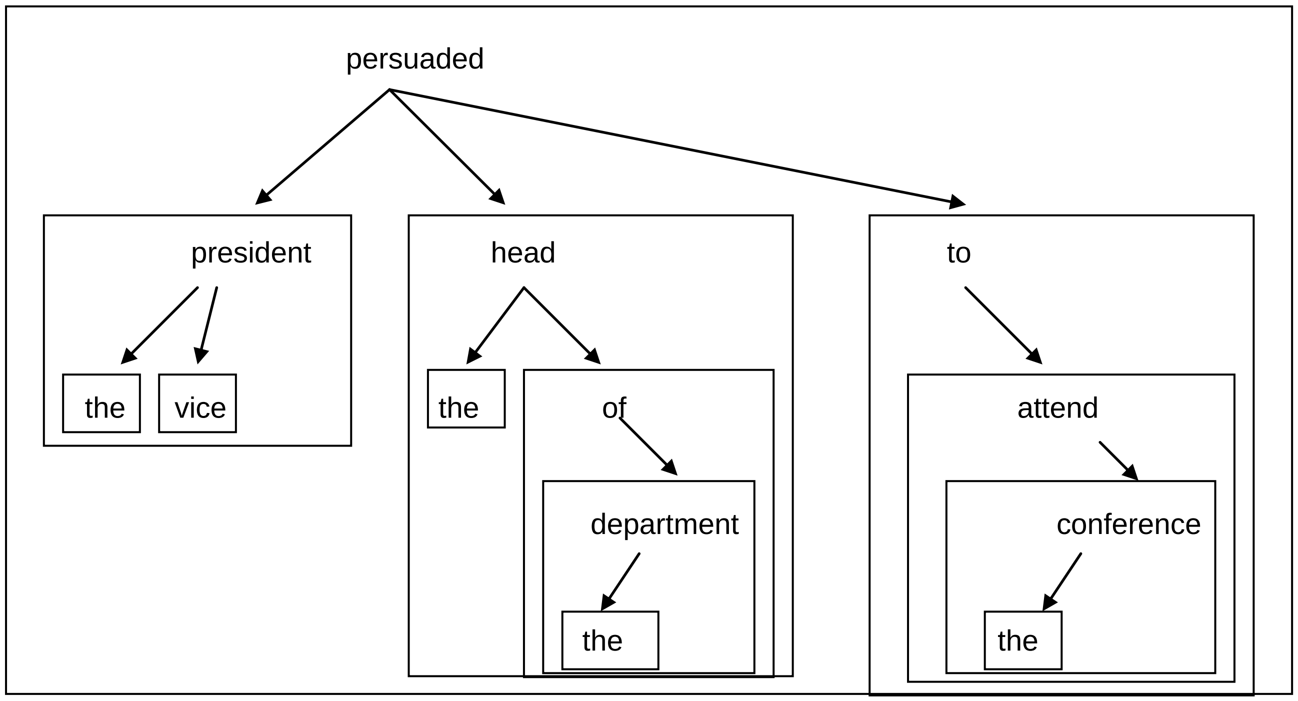

Figure 9: Dependency as a relationship between heads and complements

Contrary to other dependency grammars, the notion of constituent is endorsed in DUG. However, it is a specific constituent structure that results from dependency analysis. Let us define a constituent as the string that corresponds to a node in a dependency tree together with all nodes subordinated to that node (directly or mediated by other nodes). Then, any dependency tree can be dispersed into smaller trees until nodes with no dependents are reached. Each of these trees corresponds to a constituent of the sentence or phrase in question.

In terms of the DUG-notation, this definition can be reformulated as follows: any term in a list together with all terms included by it (directly or mediated by other terms) corresponds to a constituent. In Figure 9 the boxes delineate the constituents, and the recursive inclusion of boxes depicts the hierarchy of constituents. This constituency structure is isomorphic with the recursive inclusion of trees in a dependency tree. Note that heads are not complete constituents; they must be accompanied by complements before they form a constituent. Constituents can be easily identified in the DUG notation: Everything included in the same pair of brackets represents one constituent (compare Figure 1). The outlined constituent structure is more restrictive than ordinary phrase structure, due to the underlying head-complement principle. Therefore, only special phrase structure grammars might prove to be equivalent to dependency grammars of the DUG type. The best candidate is Head Driven Phrase Structure Grammar (HPSG), described in detail in Pollard & Sag (1994).

The emphasis on dependency as a word-to-complement relationship rather than a word-to-word relationship is justified above all from a functional point of view. The boxes in Figure 9 are so-to-speak containers for largely divergent material that possesses the same function.

(3-a) *The vice-president persuaded the head of the department to attend the conference*
(3-b) *The Conservative Party's victory in April's general election persuaded John Brown to run for President again.*

A set of categories must be available to denote the functional identity of complements. In the case of (3-a) and (3-b), the grammatical functions "subject", "object" and "infinitival complement" are needed in order to perform the appropriate substitutions, for example when generating answers to questions:

(4-a) *Who/what persuaded him to do so?* (subject)
The vice president.
The Conservative Party's victory in April's general election
(4-b) *Who was persuaded to do so?* (direct object)
The head of the department.
John Brown.
(4-c) *What was he persuaded to do?* (infinitival complement)
To attend the conference.
To run for President again.

On the one hand, the dependency tree notation implies that each node (or term) corresponds to an elementary segment (a word) of the natural language input. Accordingly, terms embrace attributes of lexical entities provided by the morpho-syntactic system. On the other hand, we insist that non-terminal constituents are necessary in order to cope with complements. As a consequence, the governing term of a tree might incorporate certain attributes that characterize the whole constituent rather than its lexical head. The values of these attributes are often propagated upwards from dependent terms to the head term. For technical as well as theoretical reasons, the category of each constituent and the part of speech of its head coincide. The same category may refer to an unsaturated lexical item or to a completed constituent with complements and specifiers. The difference is made explicit by means of attributes which have an effect similar to that of x-bar levels in x-bar theory (Jackendoff 1977).

```
(> string[block ] role[dir_object] lexeme[block] category[noun] number[singular] person[it]
determined[+,U] reference[indefinite,U] s_position[17] vowel[-,U] n_position[2,6,10]

        (< string[a ] role[determination] lexeme[indefinite'] category[determiner]
        number[singular,C] determined[+,C] reference[indefinite,C] n_position[2])

        (< string[big ] role[attribute] lexeme[big] category[adjective] use[attributive] vowel[-,C]
        n_position[6])

        (< string[red ] role[attribute] lexeme[red] category[adjective] use[attributive] vowel[-,C]
        n_position[6]))
```

Figure 10: Representation of *a big red block*

Figure 10 (a subset of Figure1) comprises attributes of the mentioned types. 'string', 'lexeme', 'number', 'person' are features characterizing individual words. 'role' indicates a grammatical function and clearly refers to the constituent composed of the given term and all terms below it. For example, the whole list is a direct object ('dir_object'). Attribute values marked by 'C' are propagated upwards, values marked by 'U' have been passed through from below. In this way, the attribute 'determined' signals the presence of a determiner and turns the category 'noun' into 'category[noun] determined[+]', i.e. into a noun phrase. The attribute 'determined[+]' is an exclusive feature (cf. below) which prevents the noun from being complemented by another determiner again.

4.2. The generative device of the grammar

A formal grammar is, theoretically, an automaton which generates or accepts all and only the well-formed expressions of a language. When a new formal grammar is established, the way it generates or accepts expressions must be determined. The best known device of this sort is rewriting rules, as introduced in Chomsky's generative grammar. Conceptually, rewriting rules reflect the division of larger strings into immediate constituents (Wells 1947). The question is, how can the dependency approach, based on words and complements, be turned into a full-fledged formal grammar with it own generative mechanism.

A generation device can be quite different from the adopted structural representation, as for example rewriting rules and phrase structure trees. But this is not necessarily so. DUG describes syntactic structures directly as fragments of the target tree representation. The generative mechanism consists of operations on tree graphs which substitute for the language proper. A nice name for this method is "tree adjoining", elaborated in the framework of Tree Adjoining Grammar (Joshi 1975), though Tree Adjoining Grammar works with constituency trees. Given the shape of dependency trees, the idea suggests itself of putting together elementary nodes to form trees and of inserting trees into other trees in order to form larger trees. This process can be controlled in the form of requirements which are stated with a given node and which must be satisfied by other nodes in the environment. A similar model exists in chemistry. The capacity of atoms to combine with each other is defined in terms of their "valency". Tesnière (1959) borrowed this term from chemistry and applied it to words. This has been more than just a nice metaphor, it was the introduction of a formal model which might be as enlightening in linguistics as it is in chemistry. Figure 11 illustrates the basic constellation.

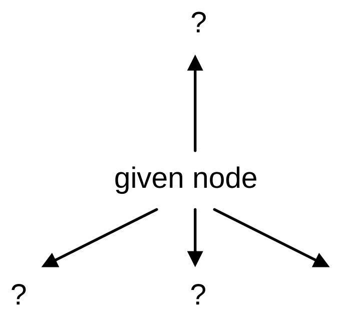

Figure 11: Combinatorial potential of a node in a dependency tree

The possible links of a node in a directed acyclic graph (a tree) reflect the syntagmatic relationships that pertain to a word according to dependency theory. There are top-down and bottom-up relations which we interpret as follows. The number and the attributes of potentially subordinated terms delineate the complements which can be expected if the word in question occurs in a sentence or phrase. Attributes characterizing a potentially superordinated term in the formalism delineate the class of words that would accept the given word (or phrase) as adjunct. The syntactic and semantic features specified for the potential head and dependents must be precise enough to exclude any ungrammatical combination.

The concept of complement goes together with the concept of slot. The boxes in Figure 9 can naturally be seen as slots that must be filled by particular material. In terms of operations on directed acyclic graphs, a complement is a complete tree; a slot is a dependent node labeled with the required attributes of a potential complement; fitting a complement into a slot means replacing the node of the slot by the head node in the complement tree. In order to implement this device, the DUG formalism is extended as follows.

(R-09)'	<DUG>	:=	<morpho-syntax> <syntax>
(R-22)'	<syntax>	:=	<template>+ <synframe>+

The syntactic component of a particular DUG grammar comprises a set of templates and a set of syntactic frames. Templates are general descriptions of all the syntagmatic relationships observed in the language. They are weakly equivalent to rules in a phrase structure grammar. Templates are assigned to individual words in a data structure called a syntactic frame (synframe). Syntactic frames are more or less equivalent to the lexical subcategorization component in a phrase structure grammar. The definition of template is:

(R-23)	<template>	:=	"(" <template head> ") "
(R-24)'	<template head>	:=	<template name> <main category>? <grammatical features>? <slot term>
(R-25)	<slot term>	:=	"(" <projection> <slot type> <role> <main category>? <grammatical features>? <lexical selection>? ") "

A template is a list consisting of two terms: a head and a slot. Each template refers exactly to one edge between two nodes in the dependency tree and describes a single syntagmatic relationship, for example the relation between verb and subject, between verb and object, between noun and determiner, between noun and adjective, between preposition and noun phrase, etc. If a given word has more than one complement then more than one template must be ascribed to it, one for each complement. This modular organization distinguishes DUG templates from phrase structure rules or from rule-based dependency grammars in the style of Gaifman (1965) . Describing a single syntagmatic relationship is much easier than drawing up rules for several immediate constituents at once.

The head term comprises a name which provides the means to refer to the template in a synframe. Furthermore the template head contains a collection of attributes which characterize the dominating element in the syntagmatic relationship and thus constrain the applicability of the template. Among the possible attributes is the main category and other grammatical features which are discussed below.

The head term dominates the slot term (represented by term inclusion). The core of each slot is the grammatical function of the syntagmatic relationship in question. Grammatical functions are distinguished by means of the role attribute within the slot term. The projection feature in the slot term divides dependents on the right of the head from those and on the left. The distinction between slot types plays a role in the case of complicated syntactic constructions. The other attributes in the slot term constrain the appropriate fillers with respect to category, grammatical features or lexical meaning. Figure 12 shows the templates assigned to the verb *pick* which have been filled in the example of Figure 1 by the subject *the robot*, the phrasal particle *up* and the direct object *a big red block.*

```
(template[+subject] category[verb] form[finite,subjunctive] s_type[statement] s_position[6]
          (< slot[regular] role[subject] category[noun] person[C] determined[+] s_position[4]))

(template[+phrasal] category[verb] s_position[6]
          (> slot[regular, select] role[phrasal_part] category[particle] s_position[14,19]))

(template[+dir_object] category[verb] voice[active] s_position[6]
          (> slot[regular] role[dir_object] category[noun] determined[+] s_position[17]))
```

Figure 12: Templates for subject, phrasal verb and direct objects

The head terms in these three templates refer to one and the same word in the sentence, the verb. Three templates are displayed because the verb *pick* has three complements, a fact which results in a threefold branching in the dependency tree. Of course, the attributes in the templates' heads must not contradict each other, for example 'category[verb]' and 's_position[6]'. The contribution of each template to the head term may vary, though. The subject template in Figure 12 requires that the verb be finite and/or subjunctive; the direct object requires that the verb be in active voice. As we mentioned before, head terms are representatives for whole constituents. That is why templates may introduce non-terminal attributes as well. The sentence type ('s_type[statement]') in the subject template is an example. The verb as such is not restricted to a particular sentence type. However, the fact that a subject occurred to the left of the verb conforms to a statement rather than to a question or command. DUG postulates that all of the attributes of non-terminal constituents can be obtained in this way.

Surface variations of complements, due to particular morpho-syntactic conditions, are handled by alternative templates. As long as the grammatical function is the same, the templates carry the same name. This provision is similar to alternative phrase structure rules with the same symbol on the left-hand side of the rule. The applicability of each template is constrained by means of the attributes in the head term. Figure 13 introduces a few more subject templates for the sake of illustration.

a) Subject pronoun in statements:

```
(template[+subject] category[verb] form[finite,subjunctive] s_type[statement] s_position[6]
          (< slot[regular] role[subject] category[pronoun] pro_form[nominal] person[C]
          case[subjective] s_position[4]))
```

b) Subject noun phrase in questions:

```
(template[+subject] category[verb] auxiliary[+] form[finite] s_type[question] s_position[6]
          (> slot[regular] role[subject] category[noun] person[C] determined[+] s_position[7]))
```

c) Interrogative and relative pronoun as subject:

```
(template[+subject] category[verb] form[finite,subjunctive] s_type[question, relative] s_position[6]
          (< slot[regular] role[subject] category[pronoun] pro_form[interrogative,relative,C]
          person[C] gender[C] case[subjective] s_position[2]))
```

Figure 13: Alternative subject templates

The templates in Figure 13 cover sentences like

(5-a) *HE/SOMEONE comes*
I/YOU/THEY/ALL come
(5-b) *Did THE GIRLS sleep?*
Is SUNSHINE (dangerous?)
Whom did JOHN see?
(5-c) *WHO came?*
WHAT gives you that idea?
The girl WHO spoke to him
The table WHICH stands in the corner

Notice the fine-tuning by means of the morpho-syntactic attributes 'pro_form' (with the possible values 'clausal', 'nominal', 'reflexive', 'reciprocal', 'interrogative', 'relative'), the attribute 'case' (with the values 'subjective', 'objective') and the attribute 'gender' (with the values 'personal', 'non_personal').

A particular assortment of syntagmatic relationships appertaining to a word is called a syntactic frame. A syntactic frame (synframe) is defined in DUG as follows.

(R-26)	<synframe>	:=	"(" <frame head> ")"
(R-27)'	<frame head>	:=	<lexeme> <reading>? <main category>? <additional attribute>* {<complement> | <adjunct> | <expected adjunct> }+
(R-28)	<complement>	:=	"(" "complement" "[" { <template name> | "none" } {"," <template name>}* "]" <lexicial selection>? ") "
(R-29)	<lexical selection>	:=	<lexeme> | <hyperonym> | <quote >
(R-30)	<adjunct>	:=	"(" "adjunct" "[" <template name> {"," <template name>}* "]" ")"
(R-31)	<expected adjunct>	:=	"(" "expected_adjunct" "[" <template name> {"," <template name>}* "]" ")"

The frame head identifies the given word. The specification of the lexeme is mandatory, because lexical meaning is assumed to be the source of syntactic dependencies. As mentioned in 3.3., a lexeme represents a word as such, like a keyword in a dictionary. If the word has more than one lexical meaning, disambiguation is achieved by means of the reading attribute. Usually a different reading of a word goes with a different syntactic frame. Lexeme and reading together provide a unique key to the entries in the lexical database. It is possible to add a part of speech category. The purpose of the additional attributes in rule (R-27) is explained below.

One synframe embraces one set of syntagmatic relationships that combine with the given word at the same time. If there are conflicting syntagmatic relationships then several synframes must be drawn up. The syntagmatic counterparts of a given word divide in complements and adjuncts. They are reflected in the frame by an arbitrary number of references to complement or adjunct templates. (So far we have dealt with complements only, adjuncts will be discussed below.)

Each entry in a synframe describes exactly one syntagmatic relationship (i.e. one edge in Figure 11), the core of which is its grammatical function. We explained above that surface variation is described by alternative templates with the same name. It is sufficient to enter this name once in the synframe in order to cover all the morpho-syntactic variations. This method cannot be applied if some of the surface variations of a particular grammatical function occur with a lexical item while others don't. In this case we need distinctive template names, for example for subjects as noun phrase, as *that*-clause and as *to*-infinitive clause.

(6-a) *His reactions surprised me.*
(6-b) *That he should make such mistakes surprises me.*
(6-c) *To see him here surprises me.*

An arbitrary subset of applicable templates with the same grammatical function can be formed by means of a disjunction in the complement and adjunct assignments. The constant "none" is admitted in a complement assignment as well, separately or as part of a disjunction. In isolation this value explicitly introduces a syntactic frame with no complement, usually in contrast to other frames of the same word that do require complements. Occurring in a disjunction this value causes the complement to be interpreted as optional. Finally, it is possible to constrain complements lexically. This can be done by a lexeme ('lexeme'), a generic term ('hyperonym') or a quoted expression ('quote'). Figure 14 displays synframes for some of the words which occurred in previous examples.

```
(lexeme[pick] reading[lift] category[verb]
        (complement[+phrasal] lexeme[up])
        (complement[+subject])
        (complement[+dir_object]))

(lexeme[attend] reading[visit] category[verb]
        (complement[+subject])
        (complement[+dir_object]))

(lexeme[surprise] reading[amaze] category[verb]
        (complement[+subject, +subject_that, +subject_toinf])
        (complement[none, +dir_object]))

(lexeme[head] reading[leader] category[noun]
        (complement[+prep_attribute] lexeme[of]))

(lexeme[of] category[preposition]
        (complement[+phrase]))
```

Figure 14: Lexical assignment of syntagmatic frames

The syntax of a language is now defined by means of the following generative mechanism. For all morpho-syntactic word forms of the language, all synframes are consulted. For each synframe, all the templates are collected that are mentioned in the frame. The attributes of the template heads are accumulated within one term if they agree with the attributes of the word form and with each other. All slot terms in the collected template are subordinated to this term. In each case, the result is a list whose dominating term represents a word and whose dependent terms function as variables for other lists. Any list of this sort is compared with any other list. If the attributes in the head of one list agree with the attributes of one of the slots of the other list then a new list is formed by replacing the slot term in the second list with the head term of the first list, leaving the links to other terms unchanged. The resulting list is compared with any other list, and so forth. Those lists that contain no slot terms (any more) form the particular subset of saturated lists. This set corresponds to the set of well-formed phrases and sentences of the language. Concatenation of the strings in the terms of these lists yields the expressions of the language.

The mathematical properties of this mechanism deserve to be studied. Probably its computational complexity is lower than the one of systems with phrase structure rules. As Barton, Berwick & Ristal (1987) point out in detail, the prerequisite of a computationally tractable problem is its modular, constrained and local structure. In a concrete application of DUG, the number of lists to be tested is far more restricted than the definition above may suggest. A parser, for example, is faced only with the lists appertaining to the words in the input sentence. These lists are stored in a well-formed substring table (also called a "chart"). For example, when parsing sentence (1), the synframe for the verb *pick* in Figure 14, in combination with the morpho-syntactic paradigm 'vprs-s' in Figure 4b and the templates displayed in Figure 12, leads to the following entry:

```
(string[picks] lexeme[pick] reading[lift] category[verb] form[finite] tense[present] person[he,she,it]
voice[active] s_type[statement] s_position[6]
        (< slot[regular] role[subject] category[noun] person[C] determined[+] s_position[4])
        (> slot[regular] lexeme[up] role[phrasal_part] category[particle] s_position[14,19])
        (> slot[regular] role[dir_object] category[noun] determined[+] s_position[17]))
```

Figure 15: Initial list stored in a well-formed substring table

In the same way, lists are assigned to the other words of sentence (1). After determiners and adjectives have been combined with their nouns (which will be discussed below), the constituents *the robot, up, the big red block* are part of the well-formed substring table. The attributes in the head terms of these constituents agree with the slots of Figure 15. The insertion of the respective lists into the slots yields the list in Figure 1. The testing of attributes is completely local. Normally, only two terms are compared: the slot term in the main list and the head term in the filler list. In the case of congruence (symbol "C" in the attribute), the attribute values of the filler and the head in the main list (e.g. the term for *pick* in Figure 15) are compared as well. As a result, attribute values might be propagated from one term to the next higher one. This is a very short distance as opposed to the long paths necessary in phrase structures. The search for compatible lists can be further refined. For a comparison of the slot-filler device with other prototypes of parsers see Hellwig (1989).

DUG makes a technical and theoretical distinction between complements and adjuncts. Technically, each dependency relationship can be defined in two ways (cf. Figure 11): Either the dominating term is provided with a description of an appropriate dependent term, or the dependent term is provided with a description of an appropriate dominating term. So far we have concentrated on the first alternative, the assignment of complement slots to the head term. Valency Grammar, the first version of DUG, had no other option in order to construct dependency trees. Subsequently the second alternative was developed as well, the description of the dependency potential of a word by means of adjunct templates.

Formally, there is no distinction between complement and adjunct templates. A template always consist of a head term and a slot term, describing an asymmetric syntagmatic relationship between a word and a constituent. What makes the difference is the place where the template is introduced in the lexicon. Complement templates are specified in the synframe of the dominating word; adjunct templates are specified in the synframe of the depending word. The mechanism of attribute testing is the same again; just known and unknown are converted. While in a complement frame the head term represents the known and the slot represents the searched for, in an adjunct frame the dependent term represents the known and the head term is searched for. The advantage of adjuncts is a gain in efficiency when parsing optional dependents. While complement slots must be processed although the dependent might not be existing in the input at all, adjuncts are processed only if the optional dependent really exists in the input.

Our present DUG of English treats determiners and adjectives as adjuncts, because they are optional components of noun phrases. A small subset of determiner templates is shown in Figure 16.

a) Determiner compatible with singular count noun:

```
(template[%dete_count_singular] category[noun] noun_type[count] number[singular]
n_position[10]
        (< slot[regular] role[determination] category[determiner] determined[C] reference[C]
        n_position[2]))
```

b) Determiner compatible with plural count noun:

```
(template[%dete_count_plural] category[noun] noun_type[count] number[plural] n_position[10]
        (< slot[regular] role[determination] category[determiner] determined[C] reference[C]
        n_position[2]))
```

c) Determiner compatible with non-count noun:

```
(template[%dete_non_count] category[noun] noun_type[non-count] n_position[10]
        (< slot[regular] role[determination] category[determiner] determined[C] reference[C]
        n_position[2]))
```

Figure 16: A subset of templates for determiners

Note that these templates are still constructed from the point of view of the dominating noun expecting a determiner, although they work as a device for a determiner to select a noun. The reason for uniform templates is the fact that both complements and adjunct are dependents. A complement specified in a synframe is an instruction to attach a constituent below the given word in the dependency tree, an adjunct in a synframe is an instruction to attach the given word to something above it in the tree. A template is in both cases an instruction stating how a dominating and a dependent term are to be connected. The following figure displays a few assignments of adjunct templates to determiners.

```
(lexeme[the] category[determiner]
        (adjunct[%dete_count_singular, %dete_count_plural, %dete_non_count]))
(lexeme[this] category[determiner]
        (adjunct[%dete_count_singular, %dete_non_count]))
(lexeme[every] category[determiner]
        (adjunct[%dete_count_singular))
(lexeme[those] category[determiner]
        (adjunct[%dete_count_plural))
(lexeme[enough] category[determiner]
        (adjunct[%dete_non_count))
```

Figure 17: Lexical assignment of determiners as adjuncts

The theoretical status of adjuncts is under debate among dependency grammarians. Usually this distinction between complements and adjuncts is in accordance with the division of phrases into objects and adverbials. The main criterion is grammatical function in connection with lexical semantics. Complements are dependents of a lexical item that are required by the word's inherent semantics; adjuncts augment the dependency structure by their own virtue.

(7-a) *The cooking time depends on the size of the potato.*
Drugs act on the central nervous system.
The soldiers asked for food.

(7-b) *He slept on the deck.*
She kissed him on the cheek.
They kissed for half a minute.

The prepositional phrases in the sentences (7-a) are presented as complements, while the prepositional phrases in (7-b) are adjuncts. In the case of complements, the prepositional phrase refers to a genuine participant in the depend-event. The meaning of *on* can be understood only in relation to the verb *depend*. As opposed to complements, adjuncts like time, place, direction, and manner can be understood in isolation, because they are inherent in many events. The preposition *on* denotes a local or directional relationship between two entities, one of which is the verb of which *on* is an adjunct , the other one is the complement of *on*.

The absence of a complement leaves the dominating word unsaturated while an adjunct may or may not be present from a grammatical point of view. That is why mandatory or optional occurrence is often suggested as a criterion. The situation is not always as clear-cut, though. On the one hand, complements can often be omitted if they are known in the situation. On the other hand, time, place, direction, manner and other adverbials must be interpreted as genuine and compulsory participants of some verbs.

(7-c) *The book is lying on the table.*
She puts the book on the table.
The event lasts for a particular length of time.

In the framework of *lie* and *put,* the place seems constitutive and in the framework of *last* a time period is crucial. According to the aforementioned criterion, the prepositional phrases in (7-c) would be complements. On the other hand, the examples in (7-c) share their semantic function and their syntactic behavior with the adjuncts in (7-b) rather than with the complements in (7-a).

(8) *Where is the book lying?*
Where does he sleep?
** Where does the cooking time depend?*

The solution of this problem are so-called expected adjuncts. Expected adjuncts are real adjuncts and they are covered by the same templates as optional adjuncts. The same grammatical function is assigned to both kinds. However, since expected adjuncts must be present in the syntactic framework of a particular governing word they are added to the synframe of that word. The DUG parser recognizes an adjunct that is expected; if the expected adjunct does not appear then the phrase or sentence is rejected. Here are the frames of the verbs and prepositions occurring in (7-c):

```
(lexeme[lie] reading[occupy_a_place] category[verb]
        (complement[+subject]
        (expected_adjunct[%adverb_place]))
(lexeme[put] reading[place] category[verb]
        (complement[+subject]
        (complement[+dir_object])
        (expected_adjunct[%adverb_place]))
(lexeme[last] reading[continue] category[verb]
        (complement[+subject]
        (expected_adjunct[%adverb_duration]))
(lexeme[on] category[preposition]
        (complement[+phrase] hyperonym[object])
        (adjunct[%adverb_place]))
(lexeme[for] category[preposition]
        (complement[+phrase] hyperonym[period])
        (adjunct[%adverb_duration]))
```

Figure 18: Syntagmatic frames including complement, adjuncts and expected adjuncts

Synframes are the repository for one more kind of information. It is possible to assign arbitrary additional attributes to the given word in the top term of a synframe. This possibility is motivated as follows. There are two complementary uses of attributes. On the one hand, attributes are the means to state conditions under which certain items are acceptable, for example, as fillers of a slot. On the other hand, attributes must be ascribed to the individual items that are supposed to meet these conditions. So far, the only way of classifying items has been the morpho-syntactic lexicon. There are features, however, that are syntactic or semantic in nature and are not reflected by morphology. Synframes can be used to ascribe such attributes to the words in question. The parser includes these attributes in the head term of the initial list which represents the given word in the well-formed substring table. Here are examples regarding the attributes 'noun_type' and 'hyperonym' occurring in Figure 16 and 18:

```
(lexeme[block] reading[cube] category[noun] noun_type[count] hyperonym[object])

(lexeme[water] category[noun] noun_type[non-count] hyperonym[liquid])

(lexeme[minute] category[noun] noun_type[count] hyperonym[period])
```

Figure 19: Attribute assignment in syntagmatic frames

5. The unification of attributes

5.1. The types of categories

At the present state of linguistics, it is possible to discern a family of unification grammars, independent of the distinction between constituency and dependency. One essential ingredient of all these grammar formalisms is the special form of categories. Words, phrases and sentences are classified with a set of features rather than with a single category. Each feature is divided into the indication of the feature type (attribute) and the feature itself (value). Instead of 'accusative' or 'singular', one writes 'case[accusative]' and 'number[singular]'. There are many different conventions about the details of such categories in different formalisms. Some theories allow attribute-value pairs in which the values may again be attribute-value pairs. The advantage of all kinds of feature structures is the possibility to state general relationships in terms of feature types (attributes). For example, the well-known rule for the expansion of S into NP and VP can be refined as follows:

(9) S -> NP number[x] person[y] + VP number[x] person[y]

If only the concrete features 'singular', 'plural', '1st', '2nd', '3rd' were available, one would need six rules to cover all the cases of agreement between NP and VP. Complex features have been used in computational linguistics for a long time. The original formalism with attribute-value pairs in Hellwig (1978) relied on Brockhaus (1971) and Kratzer, Pause, von Stechow (1974). Nowadays, the computation of feature structures is called unification, a term which originates from theorem proving. What is the essence of the unification mechanism? Technically, it is a kind of bi-directional pattern matching. Conceptually, it means that a grammar is conceived of as a set of equations which are to be resolved, rather than a set of rules which are to be applied consecutively. The instantiations of the variables in the grammatical equations are supplied by the lexicon. These instantiations are then propagated through the equations. In my opinion, this view is an important change in the concept of grammar writing, so that the label "unification grammar" is more than just a fashionable accessory.

The mechanism of unification is not uniform in DUG. The semantics of the attributes varies and therefore the method of their unification varies as well. Remember that we aim at simulating descriptive statements of traditional linguistics in the DUG formalism. Therefore, the grammar writer should be provided with the kind of attributes that suit his descriptive demands. The following types of attributes are currently available in the syntactic component of DUG. Together with choosing an identifier, the linguist has to specify what type the attribute in question is supposed to be.

<LX>	lexeme	single, multiple, unique
<RD>	reading	single, unique
<HY>	hyperonym	single, multiple, unique, C
<RL>	role	single, unique
<MC>	main category	single, multiple, unique
<DF>	disjunctive feature	zero, single, multiple, C
<CF>	conjunctive feature	zero, single, multiple, C
<EF>	exclusive feature	zero, single, multiple, C
<OF>	overwriting feature	single, C
<UT>	utterance feature	zero, fixed, unique
<CH>	character string	single, unique
<QU>	quotation	single, unique
<LP>	left punctuation mark	zero, single, multiple, unique
<RP>	right punctuation mark	zero, single, multiple, unique
<CS>	upper or lower case	fixed, unique, C
<LT>	left dependent	fixed, unique
<RT>	right dependent	fixed, unique
<SC>	succession feature	single, multiple, C
<AJ>	adjacent feature	fixed, unique
<MG>	margin feature	fixed, unique
<SL>	slot indicator and flags	fixed, multiple, unique
<TP>	template	single, unique
<CP>	complement in synframe	single, multiple
<AD>	adjunct in synframe	single, multiple
<EA>	expected adjunct in synframe	single, multiple
<RS>	raising	single, multiple

In the SGML representation of DUG data, the type of each attribute is made explicit by a special tag, indicated in the first column of the above list. In semi-formal representation, the type of each attribute is implicit in the identifier. The following groups of attributes can be distinguished:

- semantic features (LX, RD, HY),
- grammatical function (RL),
- part of speech (MC),
- grammatical features (DF, CF, EF, OF, UT),
- word order features (LT, RT, SC, AJ, MG),
- surface form (CH, QU, LP, RP) and
- a set of system tags (SL, TP, CP, AD, EA).

The third column in the list is to be interpreted as follows:

- single: the attribute consists of a single arbitrary value;
- fixed: the attribute consists of one ore more values with a fixed meaning;
- multiple: the attribute may consist of several fixed or arbitrary values;
- zero: the absence of the attribute can be queried by means of a special value;
- unique: the same type of attribute must occur at most once in a term;
- C: the attribute may be propagated upwards and checked for congruence with the head term.

Before we are able to define the semantics of the attributes, the relationship between the terms that are to be unified must be taken into account. Figure 20 depicts the typical constellation :

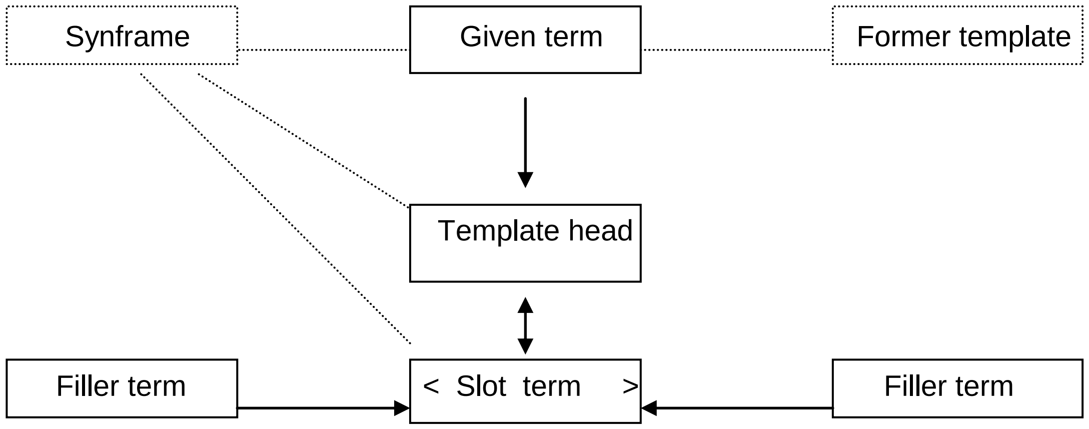

Figure 20: Relationships between terms

The given term represents one classification of a given word. The given term may be constrained due to previous unification with other templates. One of the synframes of the given term is selected which leads to a template with a head term and a slot term. A potential filler is available. In this constellation, three pairs of terms are subject to unification:

i. The unification of the given term with the template results in a possibly constrained head term, called "unihead", or in the rejection of the template. This test is necessary because there may be templates with the same name that are designed for other morpho-syntactic circumstances.
ii. The unification of the slot term with the head term of a filler candidate (to the left or to the right of the given word) results in a term with unified attributes called "unislot" or in the rejection of the filler.
iii. The unification of the term unislot with the term unihead reflects the requirements of agreement between head and dependent and results in a possibly constrained unislot term and a possibly constrained unihead term or in the rejection of the filler.

Due to its nature as bi-directional pattern matching, unification does not preserve the original terms. When attaching the complement to the original term, the original term is replaced by unihead and the head term of the inserted filler list is replaced by unislot.

5.2. The unification of given term and template

We turn now to the semantics of attributes in the three relationships. The first case of unification relates the given term and the head term of the template.

given term:	(string[block] lexeme[block] category[noun] number[singular] person[it] reading[cube] noun_type[count] hyperonym[object])
template head:	(template[%dete_count_singular] category[noun] noun_type[count] number[singular] n_position[10])
result (unihead):	(string[block] lexeme[block] category[noun] number[singular] person[it] reading[cube] noun_type[count] hyperonym[object] n_position[10])

Figure 21: Unification of a given term with the head term of an adjunct template

The given term comprises the morpho-syntactic categories of the given word as well as the additional attributes introduced by the head term of the synframe. There might also be attributes that were passed to the given term in previous slot-filling operations. A template is chosen if one of the values of 'complement', 'adjunct' or 'expected adjunct' in the corresponding synframe is identical with the value of the attribute 'template' in the template. It is the only function of the system tags TP, CP, AD, EA to provide this link. Note that adjunct templates, as in Figure 21, emanate from the synframe of the dependent term rather than from the given term. The template '%dete_count_singular' characterizes determiners like *every, each, either, neither, another.*

The goal of the grammar writer is to discard those templates that do not agree with the attributes of the given term. For example, a template designed for count nouns must not be associated with non-count nouns. Conditions of this sort involve the part of speech attribute (MC) and an arbitrary number of grammatical features (DF, CF, EF, OF, UT). The values of the part-of-speech attribute must be identical in both terms. The grammatical feature attributes must contain at least one value that is present in both terms, i.e. there must be a non-empty intersection of values for each tested attribute. If the test is successful then the attribute is copied into the resulting term provided with only those values that are part of the intersection.

Only those attributes are tested that are present in both terms. Attributes that occur in only one of the terms are interpreted as additional features. They are copied right away into the resulting term. That is the case because attributes present in the given term but missing in the template are just irrelevant on the level of abstraction which characterizes the template. Attributes present in the template but missing in the given term play a specific role. As mentioned before, attributes are needed that characterize a whole constituent rather than the topmost word. Such attributes are introduced in the head term of templates. They often denote properties which adhere to the constituent after the slot is filled. That is why they are ignored during the applicability test of the template.

5.3. The unification of slot and filler

The second case of unification relates a slot term and the head term of a potential filler list.

slot term:	(< slot[regular] role[subject] category[pronoun] pro_form[interrogative, relative,C] person[C] gender[C] case[subjective] s_position[2])
filler head:	(string[what] category[pronoun] pro_form[interrogative] case[subjective,objective] person[it] gender[non_personal]
result (unislot):	(string[what] role[subject] category[pronoun] pro_form[interrogative,C] case[subjective] person[it,C] gender[non_personal,C] s_position[2])

Figure 22: Unification of a slot term with the head term of a filler candidate

The slot term in Figure 22 belongs to a subject template (compare Figure 13). The filler term belongs to the pronoun *what*. A slot term is recognized by the system tag 'slot' (SL). Each slot term contains a role attribute (RL) which is taken over into the resulting term without testing. The part-of-speech attribute (MC) and grammatical features (DF, CF, EF, OF, UT) provide the means to constrain the potential fillers just as much as to avoid any ungrammatical combination and, at the same time, obtain a maximal level of generality. The slot term in Figure 22 covers pronouns in interrogative as well as relative sentences; however the word *what* can be used only in an interrogative context as is specified in its term. The intersection between both terms, the attribute 'pro_form[interrogative]', is the result of the unification. The attribute 'case' is unified the other way around. Here the word *what* is able to satisfy two values, 'subjective' and 'objective' (as opposed to *he*, *who* and *him*, *whom*); the slot requires the value 'subjective'. The latter value is the intersection of values and therefore stored in unislot. The value "C" will be explained later; it is always copied from the slot term to the resulting term.

As opposed to the relation between a given term and the template head, all attributes that are included in the slot term must be present in the filler term, except for the role attribute and the word order attributes. That is, most attributes in a slot are conditions. It is assumed that a condition is not met if the attribute is missing in the filler term. An exception is the zero value. If the zero value is the only value or is among the disjunction of values in an attribute then a filler is accepted exactly if the attribute is absent in the filler.

It is desirable to possess means for the individual selection of words or larger entities. Fixed selectional restrictions may be specified in a slot. Usually, lexical selection is encoded in the syntactic frames. All attributes that enrich a complement in a synframe are copied into the slot term of the corresponding template. In this way, lexical collocations can be precisely described. If a template requires selectional information from the synframe, e.g. the template for phrasal parts in Figure 12, then the slot tag must contain the value 'select'.

A lexical selection may consist of a lexeme (LX) with a single or a disjunctive value. The condition is met if one of the values matches the lexeme in the filler term. A lexical restriction can also be achieved by means of a generic attribute (HY) with a single or a disjunctive value. This condition is met if one of the required generic values has been previously assigned to the filler. In the framework of the syntactic system, such an assignment may take place as an additional attribute in the head term of a synframe. On a larger scale, generic attributes are provided by the lexical-semantic system via a specific interface.

The reading attribute (RD) is not used for selection in a slot, because the filler term usually lacks this information. However, a reading can often be determined in the course of slot filling. While morpho-syntactic features are relatively independent of particular meanings, the opposite is true about the syntagmatic properties. They usually differ with each reading of a word. That is why readings should be specified in synframes. If this is done, there is a good chance to disambiguate a given word. At first, a great variety of syntactic frames are retrieved on the basis of just the lexeme. Eventually, few frames or even one syntactic frame is left with all the slots filled. The reading that has been specified in the successful frame is the appropriate reading of the word.

Selectional constraints can also be imposed on the surface form of the filler. The character string and the quotation attributes (CH,QU) are used for this purpose. The character string attribute can be used in regular slots; its values must be identical in the slot term and in the head term of the filler. The quotation attribute is part of a special type of slot, which is recognized by the tag 'quote'. The value of the quotation attribute in the slot term must be identical to the string associated with the whole filler list. Filling such a slot results in a single term comprising no other filler information than the quotation attribute and its value. This device is convenient in the case of fixed phrases and idiomatic expressions. The literal citation of the fixed or idiomatic part of a constituent prevents these expressions from being exposed to an unnatural dependency analysis.

(10-a) *I finally came to terms with myself.*
(10-b) *But how could he kick the bucket if he were dead?*

The examples in (10-a) and (10-b) can be handled by means of a general template and syntactic frames with quoted complements:

```
(template[+supplement] category[verb]
          (> slot[quote] role[supplement] adjacent[+]))
(lexeme[come] reading[accept] category[verb]
          (complement[+subject]
          (complement[+supplement] quote[to terms])
          (complement[+prep_object] lexeme[with]))

(lexeme[kick] reading[die] category[verb]
          (complement[+subject]
          (complement[+supplement] quote[the bucket]))
```

Figure 23: Treatment of fixed or idiomatic expressions

Punctuation is another surface phenomenon which seems to resist dependency representation. This phenomenon can be integrated into the formalism, however, if the concept of constituents is accepted as advocated by DUG. Punctuation marks delineate clauses and phrases. It is reasonable to treat them as attributes of the constituents they mark. We introduce two such attributes, punctuation to the left (LP) and punctuation to the right (RP) of a constituent. The values of these attributes are the various punctuation marks such as comma, colon, period etc. It is assumed that punctuation is usually in accordance with the functional structure formed by complements and adjuncts. Therefore, punctuation attributes must occur in slots if a particular punctuation is required for the constituent in question. The attribute is satisfied if one of its values is the same as the real punctuation mark attached to the constituent which is filling the slot; otherwise the filler is rejected. If a punctuation mark is optional then it must be specified in disjunction with the zero value. The zero value is satisfied if no punctuation mark is attached to the filler. The absence of a punctuation attribute in a slot must always match with the absence of a punctuation mark around the filling constituent. It must be mentioned that some punctuation marks carry meaning, like the full stop or the question mark at the end of the sentence. These signs are treated as word-like elements which take a sentence as complement (cf. the first term in Figure 1).

Capitalization is another attribute that often applies to constituents rather than to terminals. For example, one cannot avoid referring to the sentence as a constituent when specifying the regularities of capitalization at the beginning of a sentence. This holds for other constituents as well. A special upper and lower case attribute (CS) is available which may be included in a slot term. The attribute is satisfied if its value is in accordance with the first letter of the slot filling constituent; otherwise the filler is rejected. Note that the constituent is not the same as the string corresponding to its topmost term.

A fundamental syntactic relationship is word order. As mentioned in section 2, the dependency trees of DUG are made projective - and thus encounter the same problems of coping with discontinuity and free word order as phrase structure grammars. The key for a satisfactory explication of word order phenomena lies in the attributes. Formalisms like DUG belong to the class of attribute grammars. Attributes may be introduced that go beyond context free grammars. For example, a language of the type $a^n b^n c^n$ is recognized by a grammar that includes a count attribute. In the same way an attribute grammar can be extended in order to cope with all phenomena of word order. It is easy to program a computer so that it examines the linear order of the segments that are associated with the nodes of a dependency tree. DUG offers a variety of position features whose concrete values are computed in this way. The computed values are then subject to unification with the required ones in slot terms.

A general way of indicating the position of a linguistic unit is by means of its offset, i.e. the number of characters that it is away from the beginning of the file it is stored in. With regard to dependency trees, we are interested in the offset of the beginning and the end of the string associated with the head term and the offset of the beginning and the end of the string associated with each dependent term. Obviously the following relations hold:

(11-a) offset-of-begin-of-head $\leq$ offset-of-end-of-head
(11-b) offset-of-begin-of-dependent $\leq$ offset-of-end-of-dependent

The attributes left dependent (LT) and right dependent (RT) provide a relative positioning of each dependent to the left or the right of its head. One of the two attributes must be specified in every slot because the DUG parser relies on this information. (12-a) states the condition under which the attribute left dependent is satisfied , (12-b) states the condition under which the attribute right dependent is satisfied.

(12-a) offset-of-end-of-dependent $<$ offset-of-begin-of-head
(12-b) offset-of-begin-of-dependent $>$ offset-of-end-of-head

Word order is relatively fixed in some languages and even if the word order is free it is possible to distinguish certain absolute positions in a syntagma. In the DUG of English, 30 positions of verb dependents are established. Here is a tentative list of the first seven:

1	text connective adverbs (e.g. *furthermore*)
2	interrogative and relative pronoun (e.g. *who, how, which*)/ adverbial or subject clause (e.g. *if, whether ..., that...*)
3	adverbial of speaker's attitude or certainty (e.g. *hopefully, certainly*)
4	subject in statements
5	adverbial in front of the finite verb
6	clause head (finite verb)
7	subject in questions

Figure 24: Positions at the beginning of English sentences

The positions of the head segment and the immediate dependent constituents are forced into a numeric order by means of the succession attribute (SC). It is possible to define several attributes of this type for different constructions, e.g. positions of constituents in a sentence ('s_position') or in the noun phrase ('n_position'), compare Figure 1. The succession attribute is satisfied if the following is true for any pair of items formed from the head term and an immediately dependent term or from two immediate dependent terms:

(13-a) offset-of-end-of-item-with-smaller-value < offset-of-begin-of-item-with-greater-value
(13-b) for any value: value-of-one-item ≠ value-of-another item (in a regular slot)

Varying word order is coped with in two ways. Firstly, several templates can be drawn up with different succession values according to the word order variation of a given complement or adjunct. For example, the three templates in Figure 13 describe the subject occupying the second, fourth, and seventh position in the sentence. Secondly, the succession attribute in a slot can have disjunctive values describing alternative places where the complement might occur. Compare the template for the phrasal verb *pick* in Figure 15. The phrasal part can occur before and after the direct object: *pick up the block* versus *pick it up*. Therefore its word order attribute is 's_position[14,19]' while the object is in 's_position[17]'. The condition (13-b) determines that usually no position can be occupied twice. (An exception to this rule will be explained below.) Due to this rule, free word order can be constrained elegantly. Different constituents may be possible in a particular position; as soon as the position is occupied by one of the candidates the other constituents must accommodate the rest of the positions. This process is continued until one particular order emerges.

The succession attribute must have just one value in template heads, as opposed to the slots with their disjunctive succession values. It is necessary to have a fixed head position in order to compute the possibilities of multiple disjunctive values within the slots. Any new slot filling must be tested to determine if its succession value is compatible with the previous assignments. In order to organize these checks, the succession value of each filled slot is passed up and included in the succession attribute of the head term. The values of the attribute in the head term reveal which positions are already taken in the emerging constituent.

We mentioned an exception to the rule (13-b). Under certain circumstances one may want to deviate from the restriction that each slot must be occupied just once. An example are the adjectives *big, red* in Figure 1: *a big red block.* It is subject to debate whether the role of these adjectives is the same and whether an arbitrary number of such adjectives may modify the noun. To be on the safe side, DUG provides a special type of slot that can be filled several times. The slot tag must contain the value 'multiple'. Since the same template is re-used in this case, the succession values of the fillers of such a slot are identical.

There are two more position attributes. The attribute adjacent (AJ) has just one value and is satisfied if either (14-a) or (14-b) is the case:

(14-a) offset-of-end-of-dependent = offset-of-begin-of-head minus 1
(14-b) offset-of-begin-of-dependent = offset-of-end-of-head plus 1

The attribute margin (MG) has two fixed values: adjacent to left margin and adjacent to right margin. Its purpose is to define the boundaries of a constituent, especially those of a compound or a sentence. (15-a) states the condition under which the first value is satisfied , (15-b) states the condition for the second value:

(15-a) offset-of-begin-of-filler-string = offset-of-begin-of-whole-constituent
(15-b) offset-of-end-of-filler-string = offset-of-end-of-whole-constituent

5.4. The unification of head and dependent

.
The third case of unification relates the term of a candidate dependent (unislot) with the term of the dominating element (unihead).

head term:	(string[surprises] lexeme[surprise] reading[amaze] category[verb] form[finite] tense[present] person[he,she,it] voice[active] s_type[question,relative] s_position[6]
dependent term (unislot):	(string[what] role[subject] category[pronoun] pro_form[interrogative,C] case[subjective] person[it,C] gender[non_personal,C] s_position[2])
resulting head term (unihead):	(string[surprises] lexeme[surprise] reading[amaze] category[verb] form[finite] tense[present] person[it,U] voice[active] s_type[question,relative] pro_form[interrogative,U] gender[non_personal,U] s_position[2,6]

Figure 25: Unification of a head term with the congruent attributes of a dependent term

The head term in Figure 25 is derived from the lexical attributes of the verb *surprises* and the template for interrogative and relative pronouns as subject, introduced in Figure 13. The dependent term is the result of matching the word *what* with the slot in this template, as demonstrated in Figure 22. The unification of head and dependent is confined to those attributes that are marked with the symbol "C" (congruence) in the unislot term. Any other attributes are left unchanged. If the operation has been successful then the symbol "U" (unified) is added to the values of the corresponding attribute in the unihead term.

The two types of unification we discussed before regulated the appropriate substitution of templates and instances. The unification of head and dependents accounts for all kinds of correlations between entities in syntactic constructions. Morpho-syntactic agreement between particular words belongs to this type, as well as the compatibility of close or distant constituents. Remember that the head term in a list does not only stand for a word, but also stands for the whole constituent. That is why the well-formedness of the whole construction can be tested by means of unifying each dependent separately with the head.

The attributes differ with respect to the tests and the resulting values. Disjunctive features (DF) are the first choice for enforcing many kinds of compatibility between items. The hyperonym attribute (HY) works in the same way. If both the dependent term and the head term comprise a particular attribute of this type, then the intersection of values is formed. If the intersection is empty then the unification fails, otherwise the old values are replaced by the intersecting values both in the head term and in the dependent term. If only the dependent term contains the attribute then this attribute with its values is copied into the head term. This mechanism allows upwards propagation of attributes which are not anticipated in the head term. As soon as the attribute is passed to the head term, any other dependent must form a non-empty intersection in order to unify with this attribute. In this way the agreement between two or more dependents can be computed via the head term. Another goal of passing attributes upwards is to qualify a formerly unspecified head to enter in a particular slot. Of course, the propagation of attributes can be continued by means of other templates all the way up in the hierarchy of the dependency tree.

Conjunctive features (CF) simply record the occurrence of particular phenomena without testing their compatibility. The case feature (CS) works the same way. The values of the attribute in the head term are replaced by the union of values of the dependent term and the head term. If only the dependent contains the attribute then the attribute is copied into the head term. The head-dependent unification of this type of attribute can never fail. Note, however, that these attributes behave in the slot-filler unification as restrictively as the other types.

Exclusive features (EF) are appropriate for adjuncts that must occur only once in a certain construction. The value of the attribute in the dependent term must not be present in the head term already, otherwise the unification fails. For example, the first attachment of a determiner to a noun results in the value 'determined[+]' in the head term. Because this is an exclusive attribute, no more determiners are accepted.

Admittedly, the overwriting feature (OF) has been designed especially for the English determiners *a* and *an*, but perhaps it is usable for other purposes too. The value of this attribute is overwritten in the head term by the value of the dependent term. For example, nouns and adjectives are lexically provided with the attribute 'vowel[+]' or 'vowel[-]' depending on their first letters. By means of the value 'vowel[C]' in the adjective slots, this value is overwritten in the head term each time an adjective is attached to the noun. Slot filling always starts with the closest dependents first, the more distant ones later. So in front of the noun, the attachment of adjectives proceeds from right to left. Eventually, the determiners *a* or *an* try to link up as adjuncts; *a* includes 'vowel[-]', *an* 'vowel[+'] in the template head. The unification of given term with template head yields the desired result.

The values of the succession feature (SC) are computed as described in (13-a) and (13-b). The succession value of a slot is automatically stored in the head term, as soon as the slot is filled. The marker "C" is not necessary for that. If "C" is specified with a succession attribute, the values of the same attribute in the filler are accounted for as well. In this way the numerical order of positions can be extended across several levels of dependency.

6. Extensions of the formalism

We now turn to a number of enrichments of the DUG formalism which are all due to shortcomings of the simple dependency model. Although the solutions may be idiosyncratic, the problems should be of common interest.

6.1. Sentencehood

Sentences and clauses have an additional semantic quality that goes beyond the meanings of the participating words.

(16-a) *The robot picks up a red block.*
(16-b) *Does the robot pick up a red block?*

In (16-a) and (16-b) the difference between statement and question is supplemental to the word dependencies..

(17-a) *THAT WE NEED MORE EQUIPMENT is obvious*
(17-b) *I wonder WHETHER WE NEED MORE EQUIPMENT*

In (17-a) and (17-b) the conjunctions *that* and *whether* indicate the propositional character of the complements.

(18-a) *HOW THE BOOK WILL SELL depends on its author*
(18-b) *I knew HE WAS WRONG*

There is no conjunction in the subclauses of (18-a) and (18-b). This causes problems with the role attribute. Obviously, the clause *how the book will sell* is subject of *depends* and hence the role in the top-most term of the clause should be 'subject'. On the other hand, the top-most term *sell* is the predicate within the clause and therefore the role attribute in the term must be 'predication'. If the top-most term in the clause would simply substitute for the subject slot, there would be two role attributes in one term - a case that must be ruled out.

(19-a) *the girl WHO ARRIVED*
(19-b) *the car HE REPAIRED*

The relative clauses in (19-a) and (19-b) cause similar problems. The relative pronoun cannot function as conjunction because it has to fill a grammatical function within the relative clause (subject, object) or it is even missing as in (19-b).

While phrase structure grammars provide constituents that represent sentencehood (like S, S'), there seems to be no room in dependency representations to reflect the semantic force of propositions. The reason is the extreme word-orientation of dependency grammars. The measure taken by DUG is to "lexicalize" the propositional character of clauses and sentences as well. Special terms are introduced on top of clauses and sentences and are provided with appropriate role and lexeme attributes. Where do these terms come from? There are certain surface phenomena that denote sentencehood, for example punctuation marks. Full stop and question mark at the end of the sentence are treated by DUG as words with the lexical meaning "statement" or "question" (cf. the top-most term in Figure 1). These lexical items are associated via synframes with templates like the ones in Figure 26:

```
(template[+statement] role[illocution] category[sign]
        (< slot[regular] role[predication] category[verb] form[finite] s_type[statement] adjacent[+]
        margin[left]))

(template[+wh_question] role[illocution] category[sign]
        (< slot[regular] role[predication] category[verb] form[finite] s_type[question]
        pro_form[interrogative] adjacent[+] margin[left]))
```

Figure 26: Templates assigned to full stop and question mark

In the same way, conjunctions like *that, whether, if* are interpreted as lexical representations of the force of the clauses they dominate. They have complements and can fill slots themselves. In many cases, there are no such lexical anchors for clauses. We have to introduce artificial terms. There is a particular kind of template in which an artificial term intervenes between the head term and the slot term. The definition of templates is revised accordingly:

(R-24)	<template head>	:=	<template name> <main category>? <grammatical features>? {<intermediate term> | <slot term>}
(R-32)	<intermediate term>	:=	"(" <role> <lexeme>? <slot term> ")"

The intermediate term contains a role attribute, the slot term contains another one. In this way a clash of roles is avoided.

```
(template[+subject_quest] category[verb] form[finite,subjunctive] mood[active] person[it]
s_type[statement] s_position[6]
        (role[subject] lexeme[quest']
                (< slot[regular] role[predication] category[verb] form[finite] s_type[relative]
                pro_form[interrogative] s_position[2])))

(template[+dir_object_that] category[verb] form[finite,subjunctive] s_position[6]
        (role[dir_object] lexeme[clause']
                (> slot[regular] role[predication] category[verb] form[finite] s_position[29])))
```

Figure 27: Templates with an intervening term between head and slot

The templates in Figure 27 handle the examples (18-a) and (18-b). The second template describes a "zero" *that*-clause and is an alternative to the template with the same name that describes the real *that*-clause.

While clausal complements can be dealt with as shown, clausal adjuncts create new technical problems. The next figure presents a template for relative clauses, like the ones in (19-a) and (19-b).

```
(template[%relative_clause] category[noun] n_position[10]
        (role[attribute] lexeme[relative_clause']
                (> slot[regular] role[predication] category[verb] form[finite] s_type[relative]
                pro_form[relative, elliptic] gender[C] n_position[18])))
```

Figure 28: Template for relative clauses

Relative clauses are adjuncts rather than complements. There is no lexical item, though, which could serve as a keyword in order to introduce such an adjunct. The artificial lexeme 'relative_clause' is not available in the lexicon. It seems that we have to assign these adjuncts to any verb that may be the head term in a relative clause. The definition of synframes is refined for this purpose:

```
(R-27)'     <frame head>          :=    {<lexeme> <reading>? <main category>? |  {<lexeme_variable>
                                        <main category> }<additional attribute>*
                                        {<complement> | <adjunct> <expected adjunct> }+
```

This definition provides an alternative access to a synframe besides lexemes. Since the lexeme can be substituted by a variable, it is now possible to associate a synframe with a part of speech category.

6.2. Discontinuity

The problem of discontinuity depends to a certain extent on the adopted formalism rather than on the language itself, because the way a constituent is defined and recognized is in part a theoretical decision. Nevertheless, there will always be phenomena which can hardly be covered by a strictly continuous arrangement of constituents.

(20) *What have you been looking at?*

In principle, dependency grammar faces fewer problems with discontinuity than phrase structure grammar. Traditional phrase structure trees cannot ignore the linear concatenation of constituents while dependency trees usually focus on functional relationships and may avoid the question of linear succession all together. Figure 29 shows a dependency tree representation of (20) .

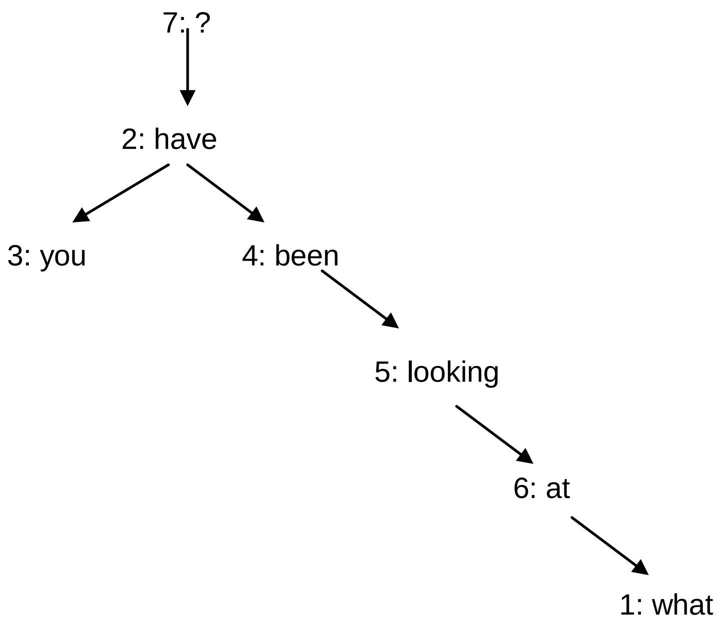

Figure 29: Dependency tree with discontinuous word order

Information about the word order must be added to dependency trees in form of labels. Certainly, a simple numbering of words, as in Figure 29, won't do. But in principle there is no limitation to enriching a dependency tree by explicit attributes which may describe any kind of word order. The positional attributes that have been chosen in the DUG have been described in 5.3. A considerable restriction is imposed upon the possible linear sequence by means of the left-dependent (<) and right-dependent (>) markers. Here is a representation of (20) in this format:

```
(string[?]
        (< string[have]
                 (> string[you])
                 (> string[been]
                          (> string[looking]
                                   (> string[at]
                                             (<<<< string[what] ))))))
```

Figure 30: A dependency representation with projectivity markers

Ascending in the hierarchy, each dependent string is concatenated to the left or the right of the string on the next higher level. If the projection symbol is repeated *n* times then the dependent string is concatenated to the left or the right of the string that has been formed on a level *n* times higher in the hierarchy. In this way the string 'what' ends up to the left of 'have you been looking at'. According to this mechanism, extra-positioned parts of constituents must be adjacent to a dependential mother constituent. It is subject to further study whether this limitation on discontinuity is empirically adequate. A mother constituent is the string that corresponds to a node higher in the hierarchy than the given one together with all nodes subordinated to it. Adjacency to the mother constituent does not necessarily imply adjacency to the head in that constituent.

DUG provides a special slot type for describing complements and adjuncts that are extra-positioned. An example is the following template for the pronoun *what* in (20). This complement is assigned to the preposition *at*.

```
(template[+phrase] category[preposition]
        < (slot[discontinuous] role[prep_complement] category[pronoun] case[objective]
        pro_form[interrogative,C]))
(category[verb] form[finite] s_position[6])
```

Figure 31: A template for a discontinuous complement

The template consists of the usual description of the dominating term and the dependent slot. The slot tag must have the value 'discontinuous'. In a second list the head of the mother constituent is described that allows the adjacent concatenation with the filler of the slot. A template like this is shifted by the parser until a mother constituent has been formed which is in agreement with the second list in the template. At this moment the discontinuous slot is released for adjacent filling.

6.3 Nucleus and valency propagation

So far, the dominating terms in the dependency tree have been simple words. How does DUG account for composite heads of phrases?

(21-a) *Terry is eager to test the robot.*
(21-b) *That he is ill is a fact.*
(21-c) *To whom is John going to speak?*
(21-d) *That I was ill did not concern her.*

Obviously the type of the subject in sentences like (21-a) and (21-b) is determined by the predicative adjective or noun rather than by the copula verb. The complements in (21-c) and (21-d) depend functionally on the main verbs *speak* and *concern* rather than on the auxiliary verbs. On the other hand, there is no doubt that the subjects depend morpho-syntactically on the finite verb. If the unification of agreement features should be carried out as usual, then the subject must be subordinated to the term with the finite verb. Consequently, valency information must be propagated from the adjective to the copula and from the main verb to the top-most auxiliary.

It has been suggested that copula verb and adjective or auxiliary and main verb be treated as one complex entity which, as a whole, dominates complements and adjuncts. Tesnière (1959) introduced the concept of "nucleus" for this purpose. Järvinen and Tapanainen (1997) stress that nucleus is the genuine syntactic primitive of dependency grammars rather than the word. In their theory, the nodes of the dependency tree should represent nuclei. For example, chains of verb like *is going to speak* or *did not concern* should be associated with a single node.

DUG in principle adopts the nucleus as a theoretical concept. Predicates can be modified, leading to composite heads. This does not spare us, however, from having to describe the internal structure of nuclei. In order to do so, the nodes of the dependency tree continue to represent elementary segments of the input language. Nuclei consist of several such segments and are therefore distributed among several terms in the DUG list representation. There is no explicit representation of nuclei except for choosing specific role attributes. The composition of nuclei follows the usual slot-filling lines. Regarding the above examples, there must be a complement template for *is* which can be filled by *eager* or *nice*, there is another template for *is* which can be filled by *going,* and there must be a complement template for *going* which can be filled by any infinitive with *to.*

The difference with nucleus slots is that selected templates from the synframe of the slot filler are raised to the head. This is enforced by special entries in the synframe of the head.

```
(R-27)'    <frame head>        :=   {<lexeme> <reading>? <main category>? } |  {<lexeme_variable>
                                    <main category> } <additional attribute>*
                                    { {<complement> | <adjunct> | <expected adjunct> }+
                                    |  <nucleus complement> <raising > }
```

The following figure illustrates the new feature:

```
(lexeme[be] reading[copula] category[verb]
        (nucleus complement[+pred_adjective]
        raising [+subject, +subject_that, +subject_to_inf, +subject_quest]))
(lexeme[be] reading[progressive] category[verb]
        (nucleus complement[+ing_participle]
        raising [+subject, +subject_that, +subject_to_inf, +subject_quest]))
```

Figure 32: Specification of templates to be raised in a nucleus framework

Unfortunately, the head in a nucleus is not always adjacent to the complement from which a particular template must be raised. This causes technical problems for a parser that combines adjacent constituents from left to right. For example, in (21-c) the subject template of *to speak* must first be passed to *going* and then to *is*. However, the string *John* intervenes between *is* and *going* so that the latter cannot fill the slot of *is* and pass the subject template upwards. If 'nucleus complement' is specified in a synframe the parser must be ready for discontinuous behavior.

The formal machinery can be used for other kinds of complement raising.

(22-a) *Picking up blocks seems to bore the robot.*
(22-b) *Terry is easy to please.*
(22-c) *That Bob refused to play with the robot is hard to believe*.

```
(lexeme[seem] category[verb]
        (nucleus complement[+to_infinitive]
        raising [+subject, +subject_that, +subject_to_inf, +subject_quest]))
(lexeme[easy] category[adjective]
        (nucleus complement[+to_infinitive] raising [+patient]))
(lexeme[be] reading[modal] category[verb]
        (nucleus complement[+pred_adjective] raising [+patient]))
```

Figure 33: Synframes for raising verbs and adjectives

The raising verb *seem* is treated similar to auxiliaries. The synframes of the adjective *easy* incorporates an infinitive phrase from which a '+patient' template is to be raised. This template is obtained from a direct object by means of a synframe extension which will be sketched below. Finally, there is a synframe for *be*, distinct from the copula, which accepts the adjective and receives the patient complement from it. The latter can be a noun phrase or a *that*-clause, depending on the type of the object of the embedded verb. If necessary, raised templates can be distinguished from the original ones by means of special constraint attributes.

Usually a distinction is made between raising verbs and control verbs. While a raising verb takes a complement that is not one of its a genuine arguments, a control verb dominates its complements by virtue of its own inherent properties. The following sentences contain control verbs.

(23-a) *Sue tried to pile up the cubes*
(23-b) *Sue persuaded Terry to pile up the cubes*

The infinitive clauses in these examples are normal complements. Since the subjects of the embedded predicates are identical to the subject or object of the control verb, the subject is missing in the infinitive clause. It is not necessary to reconstruct the omitted subjects, but we need a means to cancel part of the valency requirements of the embedded verbs. Elliptic slots serve this purpose.

```
(template[+subject] category[verb] form[infinitive, imperative, to_infinitive, ing_participle]
        (slot[elliptic, remove]))
(template[+direct_object] category[verb] form[finite] mood[active] s_type[relative]
        (slot[elliptic] role[direct_object] proform[elliptic,C]))
```

Figure 34: Elliptic slots

An elliptic slot must be introduced if a complement in a syntactic frame can or must be omitted in a particular context. The valency requirements of a word are fulfilled by an elliptic slot to the same extent as by a filled slot. The first template in Figure 34 accounts for the omission of the subject, if the verb is not finite. The second template licenses the omission of an object in a relative clause, as in sentence (19-b) *the car he repaired.* The slot tag must contain the value 'elliptic'. If the tag includes the value 'remove' then the dependent term will be removed from the representation, otherwise it will be kept as a trace of the deleted element.

6.4. Coordination

At the first sight, coordination seems at odds with the dependency representation. Dependency trees have been designed for modeling the hierarchy of semantically unsaturated elements and their complements. Coordination involves a linear grouping of surface elements orthogonal to the dependency relations. Therefore the integration of coordinative constructions is a challenge to dependency formalisms.

(24-a) *The robot passed a block to Terry and a pyramid to me*.
(24-b) *Terry goes to New York on Wednesday and to Washington on Friday.*
(24-c) *The robot grabs and crunches pyramids.*

One is inclined to describe coordination as an operation rather than a structure. Intuitively, a certain pattern is copied from the left of the conjunction and filled with new material on the right of the conjunction. The copied portion may involve one or more complements, one or more adjuncts and even the head of the underlying dependency structure. Other complements, adjuncts or the head may stay outside of the scope of this operation, which implies that their grammatical roles must now agree with the functional structure in both conjuncts.

In the formal framework, these operation might be modeled by transformations. Genuine (semantic) coordination might be restricted to fully specified sentences. Actual coordinative constructions might result from the deletion of identical portions of the original sentences. In the other direction, ellipted portions might be reconstructed in the course of parsing, so that the eventual formal representation is a full-fledged sentence coordination. "The best way of dealing with coordination in syntax is not to deal with it at all, but 'process it away' immediately" (Lobin (1993).

DUG does not follow this direction. The design of DUG is based on Carnap's conception of the logical syntax of language (Carnap 1934) which implies a strict separation of formation and transformation. Formation rules determine exhaustively how the sentences of the language can be constructed. Transformation rules determine how from given sentences we may infer others. The semantic component of the grammar may incorporate transformation rules that derive a coordinative construction from coordinated sentences and vice versa (see the next chapter). But first we need an independent representation of any phrase or sentence as the basis for these operations.

The structural representation of coordination within the dependency framework is faced with problems. Coordination is another relation than dependency. The grammatical functions as well as the morpho-syntactic attributes within the conjuncts are clearly determined by the original head in the dependency tree and not by the conjunction. Therefore, the conjunction cannot function as a normal head node. Introducing a new type of edges in the dependency tree (Tesnière's "junction") does not help either. Any graph representation is faced with the problem to hold apart and, at the same time, to relate the pairs or triples of constituents that arise if more than one complement or adjunct is in the scope of the coordination, for example, *a block to Terry* on the one hand and *a pyramid to me* on the other.

The solution elaborated within the DUG framework is shown in Figure 35. It is the representation of sentence (24-a); all attributes except significant roles and lexemes have been removed for perspicuity.

```
(role[illocution] lexeme[statement']
        (< role[predication] lexeme[pass]
                (< role[subject] lexeme[robot] (lexeme[definite']))
                (> role[dir_object] lexeme[block] (lexeme[indefinite'))
                (> role[indir_object] lexeme[to] (lexeme[terry]))
                (> role[conjunct] lexeme[and]
                        (> role[dir_object] lexeme[pyramid] (lexeme[indefinite'))
                        (> role[indir_object] lexeme[to] (lexeme[me])))))
```

Figure 35: *The robot passed a block to Terry and a pyramid to me.*

Conjunctions like *and* dominate exactly one conjunct, usually the one to the right of the conjunction. A conjunct contains one or more constituents in the DUG sense. A constituent is, according to our definition, a string that corresponds to a node in a dependency tree and to all nodes subordinated to that node. Each constituent has a grammatical function which is represented by the role attribute in the top-most term. The main idea now is to conceive the conjunction as a special kind of term which is "transparent" with respect to the ordinary dependency relationships. The conjunction dominates the elements in the conjunct directly, but regarding dependency it is a "pseudo-head". The complements in the scope of the conjunction continue to depend morphologically and functionally on their original head. This head must exist exactly one level higher in the dependency tree than the conjunction. There is a "view" on the representation which ignores the intertwining conjunction and interprets the constituents in the conjuncts as direct dependents of the term one level above the conjunction. This term is the one that gives rise to the existence and the roles of the dependents within the conjunct.

Logically, there is always an antecedent conjunct and a postcedent conjunct in a coordination. Both conjuncts can be introduced by a coordinator, as in the case of *either - or, both - and, neither - nor.* Infix coordinators like *and* and *or* are treated as pseudoheads of the postcedent conjunct only. The scope of the antecedent conjunct remains unmarked. Implicitly the antecedent comprises exactly those complements or adjuncts under the same (real) head that bear the same role labels as the complements and adjuncts in the postcedent. It is left to the semantic component of DUG to extract the appropriate portions of the dependency tree if necessary. Leaving the antecedent conjunct unmarked allows so-called incremental parsing (and understanding). The beginning of (24-a) *the robot passed a block to Terry* is processed like a sentence without coordination. Within a conjunct, no role must occur more than once. That is why the role labels suffice for correlating the complements or adjuncts in the antecedent and postcedent conjuncts, for example *a block* and *a pyramid* (dir_object) and *to Terry* and *to me* (indir_object).

A conjunct headed by a conjunction is a constituent according to our definition of constituents. There is no constituent, though, corresponding to the whole coordinated portion of the sentence (compare Gazdar et al. 1985, ch. 8, for a similar treatment in GPSG). This is due to the dependential basis of coordination. Each conjunct embraces one or more dependents; in this way it conforms to the dependency structure. No such correspondence would exist if the conjunction were to be interpreted as the head of the whole coordination.

(25) *The robot will stack up the red blocks and put a green cube or a pyramid on top.*

Constituents can be coordinated on any level of the structural hierarchy and coordinations can be nested. Figure 36 is a rough outline of the way sentence (25) is represented.

```
(illocution: statement'
        (predication: will
                (subject: the robot)
                (part_of_predicate: stack up
                        (direct_ object: the red blocks))
                (conjunct: and
                        (part_of_predicate: put
                                (direct_object: a green cube)
                                (conjunct: or
                                        (direct_object: a pyramid))
                                (place_adjunct: on top))))))
```

Figure 36: Multiple coordination and a "view" on one of the included dependency trees

If the conjunctions are viewed as mere relays of dependency relations then the contours of several simple dependency trees emerge from this representation. The one that comprises the shaded elements corresponds to sentence (26):

(26) *The robot will put a pyramid on top.*

Note that coordination can occur within a nucleus; the nuclei *will stack up* and *will put* are split into a common part and a coordinated part. This is why a nucleus cannot be represented by a single term.

So far, we have avoided introducing terms that had no counterpart on the surface. The formal representation has been as concise as the described sentences. However, there are cases of gapped coordination, especially if head terms are coordinated, where the reconstruction of ellipsis can no longer be circumvented.

(27) *Mary saw a butterfly and laughed.*

```
(illocution: statement'
        (predication: saw
                (subject: Mary)
                (direct_object: a butterfly))
        (conjunct: and
                (predication: laughed
                        (subject: * elliptic * ))))
```

Figure 37: Gapped coordination

In reality, *Mary* in (27) is outside of the linear scope of the coordination and hence can function as a joint subject of the verbs in both conjuncts. The dependency representation cannot reflect this finding. According to the principle of incremental parsing, the subject must be made dependent of the first verb. Since the joint subject is now included in the antecedent conjunct, the postcedent conjunct is lacking a subject. The general regulation is the following: If a conjunct is missing part of the predicate-argument structure and if the missing part is not available in the dependency tree that results from viewing the conjunction as a mere relay, then a trace of the missing element must be created. This trace consists of a term with a role attribute and an ellipsis marker. Due to the slot-and-filling approach, it is not difficult to create such a term in a DUG. What is expected (and then missing) is listed in the synframe of the respective head. The only additional thing left to do is to turn the expected slot into an elided slot.

Having more or less determined the representation of coordinative constructions in DUG, we must define a specification format. The specification of coordinated constituents is divided into three tasks: (i) describing the internal structure of each conjunct, (ii) describing the correct attachment of conjuncts within the rest of the sentence, (iii) classifying conjunctions with regard to both their conjuncts and their attachment.

(28-a) *The robot and Terry passed a block and a pyramid to me*.
(28-b) *The robot picked up a block and Terry a pyramid*.

Antecedent conjuncts with no conjunction, like *the robot* and *a block* in both examples, need no separate description because they are part of the basic sentence structure. For any constituent that occurs in a headed conjunct, there must be a template for the coordinated version of that constituent in addition to the uncoordinated case. This looks like much effort for the description of phenomena that might be accounted for by a general and uniform operation. However, DUG is committed to declarative rather than procedural specification, and the coordinative constructions in different languages are by no means the same. As compared to the original complements and adjuncts, the templates within conjuncts are restricted and easily drawn up.

```
(template[+co_subject] category[conjunction] dependent[subject] c_position[1]
        (> slot[regular] role[subject] category[noun]  c_position[2]))
(template[+co_dir_object] category[conjunction] dependent[dir_object] c_position[1]
        (> slot[regular] role[dir_object] category[noun] c_position[2,3]))
```

Figure 38: Templates for conjuncts with subject and object

The complement templates in Figure 38 are specified in the synframes of conjunctions like *and* and *or*. Of course the synframes of conjunctions include many other templates. 'c_position' regulates the internal word order within the conjunct, usually reflecting the word order of the uncoordinated case. The 'dependent' attribute is necessary to propagate information about the occupied roles in the conjunct when these are checked for conformity with the antecedent conjunct.

Each conjunction, together with the conjunct it dominates, must be inserted at an appropriate position into the matrix tree. Special templates describe this connection. Similar to adjuncts, these templates are listed in the synframe of the dependent conjunction rather than in the synframe of the functional head. A conjunct searches for a head rather than having a head marked to expect a conjunct. The syntax of the synframes is augmented for this purpose by the entry '<conjunct>':

```
(R-27)      <frame head>          :=   {<lexeme> <reading>? <main category>? } |  {<lexeme_variable>
                                       <main category> } <additional attribute>*
                                       { {<complement> | <adjunct> | <expected adjunct> | <conjunct> }+
                                       |  <nucleus complement> <raising > }
```

The following templates are needed for attaching the conjuncts to the verb in sentences (26-a) and (26-b).

```
(template[&subject] category[verb] number[plural] s_position[6]
        (> slot[conjunct] role[conjunct] category[conjunction] s_position[4] dependent[subject,C]))
(template[&dir_object] category[verb] s_position[6]
        (> slot[conjunct] role[conjunct] category[conjunction] s_position[17]
        dependent[dir_object,C]))
(template[&subject_dir_object] category[verb] s_position[6]
        (> slot[conjunct] role[conjunct] category[conjunction] s_position[17] dependent[subject,
        dir_object,C]))
```

Figure 39: Templates for conjunct attachment

The 'dependent' feature enforces the presence of the respective role or roles in the matrix structure. Any information from other conjuncts must be propagated via the head term, because the parser does not allow a direct inspection of sister nodes. As a consequence, all templates that can form part of the (unmarked) antecedent conjunct must contain the 'dependent' attribute, too.

A conjunct is attached adjacently to the right of the corresponding constituent in the matrix structure. The same succession value is used for both the original constituent and the conjunct. The rules for the succession feature (cf. 13-a and 13-b above) must augmented by

(13-c) offset-of-end-of-matrix-item-with-same-value = offset-of-begin-of-conjunct-with-same-value minus 1 (in a conjunct slot)

The next figure exemplifies the combination of all information needed for (28-a) and (28-b) in the synframe of the conjunction *and.*

```
(lexeme[and] category[conjunction]
        (complement[+co_subject, none]
        (complement+co_dir_object, none]
        (conjunct [&subject, &dir_object, &subject_dir_object]))
```

Figure 40: Partial description of the conjunction *and*

7. Transformation

7.1. General mechanism

There are phenomena of language which involve a transition from one instance of representation to another one, for example paraphrases, translation equivalents, inferences. The formal means to relate different representations in the DUG formalism are rules.

```
(R-33)    <rule>             :=   "(" <rule head> ")"
(R-34)    <rule head>        :=   <rule attribute> <pattern> <pattern>+
(R-35)    <pattern>          :=   <term> |  <pattern conjunction> |  <pattern disjunction>
(R-03)    <term>             :=   "(" {  { <projection>?  <attribute>+   <attribute_variable>?}  |
                                  <term_variable>  }  <term>* ") "
(R-07)    <attribute>        :=   <identifier> "[" <value> |  <value_variable> "]"
```

Each rule consists of a head term and a variety of patterns which mirror dependency trees. The value of the rule attribute denotes the kind of operation the rule applies to, e.g. translation, paraphrasing, deduction. The rule attributes guarantee that the right set of rules is chosen for the given purpose. In the simplest case, a rule consists of two patterns, a condition pattern and a construction pattern. If the condition pattern matches an instance of a dependency tree then a new tree is constructed according to the construction pattern.

Each pattern is a list starting with a term. Pattern conjunction and pattern disjunction are special terms which are used to combine several patterns within one rule. A pattern is made more general by means of variables. All variables are recognized by their names. The first character of each name must be the asterisk (*).Variables are not distinguished by content but on a graph-theoretical basis. Each variable stands for a particular part of a dependency tree, e.g. an attribute value, a number of attributes, a single term or a number of terms. If the variable occurs in a condition pattern than the matching part of the given tree is bound to the variable; if the same variable occurs in a construction pattern then the bound part is incorporated in the emerging tree.

A value variable binds the value of the specified attribute in an instance term. An attribute variable occurs in a term together with other attributes. The other attributes restrict the applicability of the pattern. Applied to an instance, the attribute variable binds all attributes in the instance term except for the attributes explicitly mentioned. A term variable must occur alone in a term of the pattern. Applied to an instance, this variable binds a whole term on the given level of dependency. In addition, the variable binds the terms of sister nodes as well, if these are not covered by other term variables. If the term is a leaf in the pattern then the term variable also covers all subordinated terms, i.e. the variable stands for a subtree rather than a single term. With these instruments, practically any labeled dependency tree can be transformed into any other labeled tree.

7.2. Syntactic frame extension

Active and passive sentences can be derived systematically from one another. That is why only one of the two syntactic structures, usually the active version, is encoded in the synframes of the respective verbs. The templates for the derived structure are associated with the lexical items by means of syntactic frame extensions. The definition of the DUG formalism is augmented accordingly:

```
(R-22)     <syntax>                 :=   <template>+  <synframe>+  <synframe_extension>*
(R-36)     <synframe_extension>     :=   "(" <rule_head> <synframe_pattern> <synframe_pattern > ")"
```

Frame extensions are rules, consisting of the rule head and two patterns. The first describes the lexically encoded synframe, the second one describes the synframe to be created.

(29-a) *The robot picks up a block.*
(29-b) *A block is picked up by the robot.*

```
(rule[frame_extension]
        (*lex  category[verb]
               (complement[+subject] *1)
               (complement[+dir_object] *2)
               (*3))
        (*lex  category[verb]
               (complement[+patient] *2)
               (complement[+agent] *1)
               (*3)))
```

Figure 41: Synframe extension rule for passive constructions

The first part of the rule defines the applicability of the rule. The attribute variable *lex refers to the lexeme and possibly other attributes of the original synframe; the attribute variables *1 and *2 stand for any other information in addition to the required complements; the term variable *3 substitutes for any other complements or expected adjuncts which may be present in the synframe. The second part of the rule is the blueprint for the synframe to be created. Applied to the synframe of *pick up* in Figure 14, the following new synframe is formed:

```
(lexeme[pick] reading[lift] category[verb]
      (complement[+patient])
      (complement[+agent])
      (complement[+phrasal] lexeme[up]))
```

Figure 42: Derived synframe for the verb *pick up* in the passive

The target templates '+patient' and '+agent' must be drawn up like any other template in the overall set of templates describing all kinds of syntactic constructions. '+patient' is the subject and '+agent' is the prepositional phrase in passive sentences. The '+phrasal'-complement has been taken over from the old into the new synframe via the variable '*3' .

Many other templates for complicated constructions can be drawn up and then linked to the lexicon by means of frame extensions.

(30-a) *That Bob won is hard to believe*.
(30-b) *It is hard to believe that Bob won.*
(30-c) *What have you been looking at?*

For (30-a) we need a *that*-clause as patient which is obtained by extension from the direct object clause of *believe* and then raised via the adjective to the copula. A template for the anticipating *it* is used in (30-b) which is extended from all synframes with *that*-clauses. Instead of treating (30-c) as an instance of discontinuity (see section 6.2.), one could draw up two templates, one for the extrapositioned pronoun and another one for the preposition with no complement, and then write an extension rule which derives an extra synframe with these two templates from all synframes with a prepositional object.

Note, that all these synframe derivations are not transformations of the corresponding sentences into one another. The principle of separating formation and transformation stands. Synframe extension is just an economical way of providing all the templates needed for a purely declarative description of any syntactic construction.

8. The semantic system

8.1. The domain of semantics

A grammar formalism is relatively futile if it is not paired with a concept of semantics. In principle, we share Wittgenstein's view that the meaning of a language is its use. In a formal system, the use of a language can be manifest only in form of symbol manipulations allegedly reflecting operations that humans perform with real expressions. This fact leads to a definition of the domain of semantics in the framework of computational linguistics: Semantics is anything (sensible) the system does with the input after syntactic analysis. If one adopts this view then any successful application of a natural language processing system contributes to the elucidation of semantics.

DUG distinguishes neatly between three kinds of meaning: lexical meaning, referential meaning, sentence meaning. Lexical meaning deals with the differences between the elements of the vocabulary. We assume that the use of words reflects the understanding of the object domain shared by the current speech community. Lexical meaning intertwines with world knowledge; the relationships within the vocabulary mirror the prototypes of objects, properties and relations which are recognized (or imagined) in real life. Since the collective understanding of the world is a process, lexical semantics has a dynamic character. The next type, referential meaning, is concerned with the concrete denotation of a word in an utterance. Leaving out the contribution of lexical semantics to reference, determination and anaphora remain as the genuine domains of referential semantics. The functioning of articles and pronouns and the cohesion of texts are the central issues. The third kind of semantics is sentence meaning. The essence of sentence semantics becomes clear if one subtracts from it the contribution of lexical meaning and referential meaning. What is left is the logical form of sentences.

Dependency structure is a very convenient basis for the formulation of rules which reflect semantic relationships. Usually the semantic rules in a DUG are confined to the role and lexeme attributes of syntactically well-formed dependency trees. This format provides a smooth interface between the syntactic and the semantic system.

```
(R-09)    <DUG>            :=   <morpho-syntax> <syntax> <semantics>
(R-37)    <semantics>      :=   <semantic rule>+
```

8.2. Canonical representation

A first set of rules may be drawn up to turn the plain representation of surface structure into a form which is better suited for semantic operations. Anything in syntax you did not like from a semantic point of view can be changed now. For example, the separate lexemes *pick* and *up* in the phrasal verb construction may be replaced by a single lexeme *pick up* :

```
(rule[replace]
          (lexeme[pick]  *attr1
                    (role[phrasal_part] lexem[up] *attr2)
                    (*term1))
          (lexeme[pick up]  *attr1
                    (*term1)))
```

Figure 43: The rule for changing the representation of the phrasal verb *pick up*

Rules for substituting phrasal coordination by sentence coordination are another example. They would turn the original analysis of (31-a) into the same representation the parser associates with (31-b):

(31-a) *The robot passes the blocks to Terry and the pyramids to me after checking them.*
(31-b) *The robot passes the blocks to Terry after checking them and the robot passes the pyramids to me after checking them.*

Canonical rules may also undo the separation of the extrapositioned pronoun and the preposition with no complement we discussed when dealing with sentence (30-c). The pronoun can now be subordinated to the preposition in order to arrive at a uniform representation of prepositional objects. If one follows this strategy in all similar cases, the parser could be spared completely from discontinuity.

8.3. Inferences

The capacity to draw inferences is a practical goal of natural language processing systems. The grammar must provide the entities which are relevant to specifying the logical behavior of sentences.

(32) *All borogoves were mimsy.*
The mome raths are borogoves.
Hence: The mome raths were mimsy.

The domain of logical relationships embraces all cases where the possible truth values of two or more sentences are not independent of one another. Logical relationships depend on the structure of sentences and the role of certain words like *all, the, be (were, are)* rather than on the lexical meaning of the descriptive words and the associated world knowledge. Obviously, the syllogism (32) is valid whatever the words *borogoves, mimsy, mome, raths* mean. If the two premises in the syllogism are true (i.e. if they are acknowledged by the speaker and the hearer) then the conclusion is true (i.e. it must be acknowledged as well). Just as grammatical constituents can be identified by permutation within well-formed constructions, logical constituents can be detected by permutation within the sentences of a syllogism without destroying the validity of the inference.

```
(role[illocution] lexeme[statement']
        (role[predication] lexeme[*lex1]
                (role[*role1] lexeme[*lex2]
                        (role[determination] lexeme[all])
                        (*terms1))
                (*terms2)))
(role[illocution] lexeme[statement']
        (role[predication] lexeme[be]
                (role[subject] lexeme[*lex3]
                        (*terms3))
                (role[pred_subject] lexeme[*lex2])
                        (*terms1))))
(role[illocution] lexeme[statement']
        (role[predication] lexeme[*lex1]
                (role[*role1] lexeme[*lex3]
                        (*terms3))
                (*terms2)))
```

Figure 44: The classical syllogism in DUG representation

The patterns of English sentences in Figure 44 match the Aristotelian syllogism in (33) as well as the premises and the conclusion in (34). Variables are inserted after the words they match.

(33) *All men* (*role1, *lex2) *are mortal* (*lex1).
Socrates (*lex3) *is a man* (*lex2).
Socrates (*role1, *lex3) *is mortal* (*lex1).

(34) *This rule* (*terms2) *covers* (*lex1) *all cases* (*role1, *lex2) *of categorical propositions* (*terms1).
Most (*terms3) *sentences* (*lex3) *with an all-quantifier* (*terms3) *are cases* (*lex2) *of categorical propositions* (*terms1).

This rule (*terms2) *covers* (*lex1) *most* (*terms3) *sentences* (*role1 *lex3) *with an all-quantifier* (*terms3).

There are two possible strategies to cope with the logical form of sentences. If one assumes that there is a principled difference between syntactic and logical constituents, the solution is a translation of syntactic structures into some "deep structure" or into predicate calculus. If one believes that grammatical form and logical form coincide, then the syntactic formalism must directly reveal the logical relationships between syntactic units. The examples above suggest that a dependency representation with functional categories meets this requirement.

Syllogisms are based on the syntax of natural language. They provide a natural way to demonstrate how the possible truth values of sentences depend on each other. The following examples relate quantifiers and propositional adverbs. Inferences of this sort go beyond common predicate calculus, because there is no translation equivalent for quantifiers like *many*, *some* and *few* and no truth value similar to *probably* and *might.* Insert whatever you like for *borogove*, *rath* and *mimsy*.

(35-a) *Every borogove is mimsy.*
This rath is a borogove.
Hence: This rath is mimsy.

(35-b) *Many borogoves are mimsy.*
This rath is a borogove
Hence: This rath is probably mimsy.

(35-c) *Some borogoves are mimsy.*
This rath is a borogove.
Hence: This rath might be mimsy.

(35-d) *Few borogoves are mimsy.*
This rath is a borogove.
Hence: This rath is probably not mimsy.

The formal means exemplified so far suffice for implementing an algorithm for automatic inference. Figure 44 can be paraphrased as follows: If a sentence matches the first pattern and if another sentence matches the second pattern then a new sentence can be constructed according to the third pattern. If we have a closer look, we notice that the three sentences share certain constituents and that the status of the two premises is different. The Greeks distinguished between major and minor premise. The major premise (the first sentence in the syllogisms above) is a statement about a general relationship which makes it possible to proceed from the minor premise (the second sentence) to the conclusion. As Toulmin (1958, 98) puts it, the major premise is the "warrant" or "inference license" for drawing a conclusion from a given fact. What we need to do is to turn the general sentence into a real inference rule.

In the DUG approach, the three-part scheme of the classical syllogism is divided into two transformation rules of different types: a meta-linguistic rule drawn up by the linguist for the transition from a general statement to an inference rule, and this inference rule itself which controls the transition from the second premise to the conclusion in an actual case. Instead of illustrating this method in detail with the example in Figure 44, let us consider the classical modus ponens:

(36-a) if α then β.
α.

β.

(36-b) (if α then β.) $\Rightarrow$ (α. |- β.)

"$\Rightarrow$" is a rule attribute denoting the transition from a sentence to an inference rule, "|-" is a rule attribute denoting the transition from a minor premise to a conclusion. Meta-linguistic rules like (36-b) are drawn up for any general sentence. If a sentence matches the pattern ' if α then β.' then a new representation is created according to the pattern 'α. |- β.' and stored in a knowledge base. If a factual sentence occurs that matches the pattern 'α.' in the inference rule then a new sentence can be created according to pattern 'β.' The advantage of this approach to logic is that it is purely syntactic. Any syntactic construction expressing regularities in an object domain (e.g. the major premises in 35-a through 35-d) can be the input to a meta-rule and thus turned into an instruction for automatic inferring. As opposed to an artificial expert system, a natural language inference system gains its actual deductive capacity by being fed with knowledge drawn up in natural language.

The task of inferring is formally defined in the PLAIN program as the expansion of a premise p by the conjunction of all conclusions k_1 to k_n, which can be generated from p on the basis of the given inference rules. The task of proving a sentence (or answering a question) is defined as the expansion of a conclusion k by a disjunction of all the sets of premises p_1 to p_m from which k results on the basis of the given inference rules. Applied in question answering, the system affirms a sentence, if one of the derived sets of premises is present in the knowledege base. If not, the system may output the derived premises in the form of requests for more information.

In practice, the rules for the two kinds of expansion differ slightly from (36-b). Firstly, if 'α. |- β.' is valid then 'not β. |- not α.' is valid as well (so-called contraposition). One would prefer one expansion rule for both inferences. Furthermore, expansion rules are not always binary. In the case of a general sentence with complex coordination (i.e. with several nested conjunctions) the transformations eventually result in a representation which is known in logic as a disjunctive normal form. This canonical form consists of an arbitrary number of components which are either simple (asserted or negated) statements (so-called literals) or simple statements coordinated by 'and'. All the components of the formula are coordinated by 'or'. In order to match this representation, expansion rules in DUG are adapted to tautological exclusion rather than to a tautological implication:

(37) (or (α.) (β.) ... (ω.)) $\Rightarrow$ (¦ (α.) (β.) ... (ω.))

"¦" is the rule attribute denoting a non-binary expansion rule. Each disjunct in the canonical form of the input sentence is reflected by a separate pattern in the expansion rule. In order to carry out an inference, the rule is applicable to any sentence that matches the negation of one of the patterns in the rule. The result of the application is a disjunction of the sentences which are formed according to the rest of the patterns in the rule. The syllogisms in (38-a) justify this usage of the rule:

(38-a)	or (α.) (β.) ... (ω.)	or (α.) (β.) ... (ω.)	or (α.) (β.) ... (ω.)
	not α.	not β.	not ω.
	----------------------	----------------------	----------------------
	or (β.) ... (ω.)	or (α.)... (ω.)	or (α.) (β.) ...

In order to carry out a proof, the rule is applicable to any sentence that matches one of the pattern in the rule. The result of the application is a conjunction of the negations of the sentences which are formed according to the rest of the patterns in the rule. The sylogisms in (38-b) justify this usage of the rule:

(38-b)	or (α.) (β.) ... (ω.)	or (α.) (β.) ... (ω.)	or (α.) (β.) ... (ω.)
	and (not β.) ... (not ω.)	and (not α.)... (not ω.)	and (not α.) (not β.) ...
	----------------------------	---------------------------	----------------------------
	α.	β.	ω.

Here is a concrete example:

(39-a) *If a dissertation is accepted it is of considerable length or it contains a convincing argument*

(39-b) (NOT *the dissertation is accepted*) ¦ (*the dissertation is of considerable length*) ¦ (*the dissertation contains a convincing argument*)

(39-c) NOT *the dissertation contains a convincing argument*
Inferred: (NOT *the dissertation is accepted*) OR (*the dissertation is of considerable length*)

(39-d) *The dissertation contains a convincing argument*
Proven by: (*The dissertation is accepted*) AND (NOT *the dissertation is of considerable length*)

(39-a) is the original sentence in natural language; (39-b) displays the principal structure of the expansion rule. The rule is used for deriving conclusions from a given premise in (39-c) and for generating premises that prove a given conclusion in (39-d).

The list of attributes introduced in 5.1. is augmented by the following items in order to implement the described concept of inference and proof:

<NO>	logical not	single, fixed, unique
<AN>	logical and	single, fixed, unique
<OR>	logical or	single, fixed, unique
<TR>	logical true	single, fixed, unique
<FL>	logical false	single, fixed, unique

8.4. Meaning postulates

Carnap (1952) suggests defining the lexical meaning of predicates in predicate calculus by means of meaning postulates. His example is

(40) $(x)(Bx \supset Mx)$ (i.e. for all x: if B applies to x then M applies to x)

B stands for *being bachelor* and M for *being married.* The essence of Carnap's proposal is the formulation of meaning rules in the same language to which they apply. This is in accordance with the way meaning is explained in daily life or in dictionaries. Meaning postulates are major premises which can be turned into inference rules like any other general sentence:

(41-a) *A bachelor is a man who was never married.*

(41-b) (NOT *someone is a bachelor*) ¦ (*someone is a man*)
(NOT *someone is a bachelor*) ¦ (NOT *someone is married*)

(42-a) *When two people marry, they legally become husband and wife in a special ceremony.*

(42-b) (NOT *two people marry*) ¦ (*two people legally become husband and wife in a special ceremony*)

The definitions in (41-a) and (42-a) were taken from Collins COBUILD Learner's Dictionary 1996. (41-b) and (42-b) sketch the structure of expansion rules which are derived from these definitions due to their logic form. Obviously, these rules reflect the inferences which someone is able to draw if he knows the lexical meaning of *bachelor* and *marry.* What are needed, then, are just carefully worked out dictionary definitions in natural language.

Should one view meaning postulates as analytic, i.e. as sentences which are true due to language conventions? It is striking that meaning postulates can be converted into inference rules by means of the same meta-linguistic rules which turn general descriptions of object domains into inference rules. This suggests dropping the distinction between linguistic and encyclopedic knowledge in lexical semantics and considering meaning postulates as part of world knowledge. This is justified, because
world understanding and the acquisition of meaning go hand in hand. If someone does not know the meaning of a word he often does not know the object the word denotes. While he comes to know the object he learns how to use the corresponding word.

One must admit that in practice it is awkward to formulate every meaning relationship in natural language, because the natural means of expression are not precise enough. Therefore, the linguist may draw up meaning postulates in a semi-formal format. Here are a few examples relating the verbs *buy, sell, charge, cost.*

(43-a) *someone buys something from someone for a certain sum ↔ someone sells something to someone for a certain sum*

(relation: equivalence (predication: buy (subject: *1) (dir_object: *2) (prep_object: from *3) (prep_object: for *4)) (predication: sell (subject: *3) (dir_object: *2) (indir_object: *1) (prep_object: for *4)))

(43-b) *someone sells something to someone for a certain sum ↔ someone charges a certain sum for something*

(relation: implication (predication: sell (subject: *3) (dir_object: *2) (indir_object: *1) (prep_object: for *4)) (predication: charge (subject: *3) (dir_object: *4) (prep_object: for *2)))

(43-c) *someone charges a certain sum for something ↔ something costs a certain sum*

(relation: implication (predication: charge (subject: *3) (dir_object: *4) (prep_object: for *2)) (predication: cost (subject: *2) (supplement: *4)))

The relations 'equivalence' and 'implication' are transformed into disjunctive normal form and then yield expansion rules which operate on the full-fledged syntactic patterns of the verbs.

Translation is another case of equivalence. The logical consequences to be drawn from both the source sentence and the target sentence must be the same. Syntactic differences between source and target language can be easily accounted for:

(44-a) *Meine Arbeit macht mir Spaß ↔ I like my job*

(relation: equivalence (predication: machen (subject: *1) (indir_object: *2) (supplement: Spaß)) (predication: like (subject: *2) (dir_object: *1)))

(44-b) *Er wird wohl zu Hause sein ↔ I guess he will be at home*

(relation: equivalence (predication: *1 (prop_adverb: wohl) (*2))
(predication: guess (subject: I) (complement: *1 (*2)))

The examples show that DUG advocates a truth conditional approach to semantics. Concepts are meaningful as far as they are used to infer (see Brandom 2000 for recent philosophical support for this position). As opposed to model theoretic approaches in semantics, deriving from Montague (1970) and others, truth values are treated in DUG as a pragmatic matter. Actual inferences depend on the willingness of the speaker and hearer to assume certain regularities in the object domain. Whenever a regularity is understood it can be expressed. Consequently, actual inferences can be derived from the major premises which are accepted by the parties in an argument. However, such general sentences are negotiable; they change. The meanings of descriptive terms are negotiated as well and change. Meaning postulates are prototypical rather than analytic. There is no independent basis for truth values beyond communication.

What are analytic indeed are the meta-linguistic rules that turn contingent sentences into inference licenses. These rules rely completely on the syntactic structure and on a few non-descriptive terms. Meta-rules assign truth conditions to syntactic structures in the form of patterns for expansion rules. No world knowledge, no model, is involved in drawing up such rules. This approach is viable for two reasons: Dependency representations closely reflect the structure of predicates and arguments; role attributes identify exactly those units which are relevant from the perspective of logical form.

9. References

Barton, G. Edward /Berwick, Robert C./Ristad, Eric Sven (1987): *Computational Complexity and Natural Language.* Cambridge, MA: MIT Press.

Brandorn, Robert B. (2000): *Articulating Reasons: An Introduction to Inferentialism*. Cambridge, MA: Harvard University Press.

Brockhaus, Klaus (1971): *Automatische Übersetzung.* Braunschweig : Viehweg.

Carnap, Rudolf (1934): *Logische Syntax der Sprache*. Wien: Julius Springer.

Carnap, Rudolf (1952): Meanng Postulates. In: *Philosophical Studies* 3, 65-73.

Chomsky, Noam (1957): *Syntactic Structures*. The Hague: Mouton.

Collins COBUILD Learner's Dictionary (1996). London: Harper/Collins.

Gaifman, Haim (1965): Dependency systems and phrase-structure systems. In: *Information and Control*, 8, 304-337

Gazdar, Gerald/Klein, Ewan/Pullum Geoffrey K./ Sag, Ivan A. (1985): *Generalized Phrase Structure Grammar*. Oxford: Blackwell.

Hellwig, Peter (1977a): Ein Computermodell für das Folgern in natürlicher Sprache". In: Eisenberg, Peter (Hrsg.) (1977): *Semantik und künstliche Intelligenz. Beiträge zur automatischen Sprachbearbeitung* II. Berlin, New York: de Gruyter, 59-85.

Hellwig, Peter (1977b): Dependenzanalyse und Bedeutungspostulate - eine Alternative zur generativen Transformationsgrammatik. In: *Linguistische Berichte* 52, 32 - 51.

Hellwig, Peter (1978): *Formal-desambiguierte Repräsentation. Vorüberlegungen zur maschinellen Bedeutungsanalyse auf der Grundlage der Valenzidee.* Stuttgart: HochschulVerlag.

Hellwig, Peter (1980): PLAIN - A Program System for Dependency Analysis and for Simulating Natural Language Inference. In: Bolc, Leonard (ed.) (1980): *Representation and Processing of Natural Language.* München, Wien: Hanser; London: Macmillan, 271 - 376.

Hellwig, Peter (1986): Dependency Unification Grammar (DUG). In: *Proceedings of the 11th International Conference on Computational Linguistics (COLING 86)* , Bonn: Universität Bonn , 195-198.

Hellwig, Peter (1988): Chart Parsing According to the Slot and Filler Principle. In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING 88).* Budapest: John von Neumann Society for Computing Sciences Budapest. Vol. I., 242-244.

Hellwig, Peter (1989): Parsing natürlicher Sprachen. In: Bátori,S./ Lenders, W./ Putschke, W. (Hrsg.) (1989): *Computational Linguistics. Ein internationales Handbuch zur computerunterstützten Sprachforschung und ihrer Anwendung(= Handbücher zur Sprach- und Kommunikationswissenschaft).* Berlin: De Gruyter, 348 - 431.

Hellwig, Peter (1993): Extended Dependency Unification Grammar. In: Hajicova, Eva (ed.) (1993): *Functional Description of Language.* Prague: Faculty of Mathematics and Physics, Charles University, 67-84.

Jackendoff, Ray (1977): *X-Bar Syntax: A Study of Phrase Structure*. Cambridge, MA: MIT Press.

Järvinen, Timo / Tapanainen, Pasi (1998): Towards an implementable dependency grammar. In Kahane, Silvain and Polguere, Alain (eds.): *Processing of Dependency-Based Grammars*. Montreal, Canada: COLING-ACL'98, Association for Computational Linguistics, Université de Montreal, 1-10

Joshi, Aravind K. / Levy, L. S. / Takahashi, M. (1975): Tree Adjunct Grammars. In: *Journal of Comput. Syst. Sci*., Vol. 10-1

Knuth, Donald E. (1968): *The Art of Computer Programming. Fundamental Algorithms*. Reading, Massachusetts: Addison-Wesley.

Koskenniemi, Kimmo (1983): *Two-level morphology: a general computational model for word-form recognition and production.* Publication No. 11. Helsinki: University of Helsinki Department of General Linguistics.

Kratzer, Angelika / Pause, Eberhard / von Stechow, Arnim (1974): *Einführung in die Theorie und Anwendung der generativen Syntax. Zweiter Halbband: Anwendung.* Frankfurt/M : Athenäum

Lobin, Henning (1993): Linguistic Perception and Syntactic Structure. In: Eva Hajicova (ed.) (1993): *Functional Description of Language.* Prague: Faculty of Mathematics and Physics, Charles University, 163-178.

Montague, Richard. (1970): English as a formal language. Reprinted in: Thomason, R.H. (ed.) (1974): *Formal philosophy. Selected papers of Richard Montague.* New Haven: Yale University Press, 188-221.

Montague, Richard. (1973): The proper treatment of quantification in ordinary English. Reprinted in: Thomason, R.H. (ed.) (1974): *Formal philosophy. Selected papers of Richard Montague.* New Haven: Yale University Press, 247-270.

Pollard, Carl/ Sag, Ivan A. (1994): Head-Driven Phrase Structure Grammar. Chicago : University of Chicago Press

Spencer, Andrew (1991): *Morphological Theory. An Introduction to Word Structure in Generative Grammar*. Oxford: Blackwell

Tesnière, Lucien (1959): *Eléments de syntaxe structurale*. Paris.

Toulmin, Stephen E. (1958): *The Uses of Argument*. Cambridge 1958

Wells, Rulon S. (1947): Immediate constituents. In: *Language* 23, 81-117

Peter Hellwig, Heidelberg