Making DATR Work for Speech: Lexicon Compilation in SUNDIAL

François Andry*
Cap Gemini Innovation

Norman M. Fraser†
University of Surrey

Scott McGlashan†
University of Surrey

Simon Thornton‡
Logica Cambridge Ltd.

Nick J. Youd‡
Logica Cambridge Ltd.

We present DIALEX, an inheritance-based tool that facilitates the rapid construction of linguistic knowledge bases. Simple lexical entries are added to an application-specific DATR lexicon *that inherits morphosyntactic, syntactic, and lexico-semantic constraints from an application-independent set of structured base definitions. A lexicon generator expands the* DATR *lexicon out into a* disjunctive normal form lexicon. *This is then encoded either as an* acceptance lexicon *(in which the constraining features are bit-encoded for use in pruning word lattices), or as a* full lexicon *(which is used for assigning interpretations or for generating messages).*

1. Introduction

In this paper we describe DIALEX, a modular inheritance-based tool for the construction of lexicalized grammar knowledge bases. DIALEX has been developed as part of the SUNDIAL (Speech UNderstanding and DIALogue) project—currently one of Europe's largest collaborative research projects in speech and language technology.[1] SUNDIAL's main project goal is to produce four prototype systems that support relatively unconstrained telephone dialogs for limited domains in each of English, French, German, and Italian (Peckham 1991). This paper reports work carried out in the development of the English and French systems. These share a common application domain, namely flight enquiries and reservations.

* Cap Gemini Innovation, 118 Rue de Tocqueville, 75017 Paris, France (andry@capsogeti.fr).
† Social and Computer Sciences Research Group, University of Surrey, Guildford, Surrey, GU2 5XH, U.K. (norman@soc.surrey.ac.uk; scott@soc.surrey.ac.uk).
‡ Logica Cambridge Ltd, Betjeman House, 104 Hills Road, Cambridge, CB2 1LQ, U.K. (simont@logcam.co.uk; nick@logcam.co.uk).
1 The work reported here was supported in part by the Commission of the European Communities as part of ESPRIT project P2218, SUNDIAL. Partners in the project are Cap Gemini Innovation, CNET, IRISA (France); Daimler-Benz, Siemens, University of Erlangen (Germany); CSELT, Saritel (Italy); Logica Cambridge Ltd, University of Surrey (U.K). We acknowledge with gratitude the helpful comments and suggestions of the editors and reviewers of this special issue of *Computational Linguistics* and of Lionel Moser.

The process of writing linguistic knowledge bases has been guided by a number of design requirements on the SUNDIAL project as a whole.

1. First of all, prototype systems must be capable of *understanding* speech. Therefore grammars must be appropriate for the purposes of speech processing. For example, they must reflect the fact that input to the parser is a word lattice or graph from which some of the spoken words (typically short words such as function words) may be missing.

2. Each prototype system must be capable of *producing* speech. Speech generation takes place in two stages. In the first stage, text is generated. In the second stage, a text-to-speech system outputs the message. Therefore the linguistic knowledge must also be structured appropriately for the purposes of *text* generation.

3. Each system must run in real time or near real time. Therefore the linguistic knowledge must be structured so as to allow rapid access and manipulation.

4. Portability to new applications should be simple; work required to write new linguistic knowledge bases should therefore be kept to a minimum.

5. Duplication of effort must be avoided. This must be true in respect of the components of each separate prototype system. For example, the same dialog manager software module has been used in each prototype with minor customizations for each language (Bilange 1991; McGlashan et al. 1992). The same principle should apply to the design of tools for the construction of knowledge bases, including lexical knowledge bases. Thus, the task of adding a new lexical item should only require the addition of knowledge that is idiosyncratic to that lexical item and not predictable from what is already present in the knowledge base.

Section 2 of this paper presents an overview of the SUNDIAL DIALEX tool. Section 3 describes the way in which linguistic knowledge is initially expressed in terms of declarative DATR theories. Section 4 explains how a compact DATR knowledge base is expanded out into a fully specified lexicon. Section 5 relates how the lexicon can be customized for the purposes of real-time speech parsing. Practical experiences of constructing and using DIALEX are recounted in Section 6. Concluding observations are drawn in Section 7.

2. Overview of the System

In common with contemporary generative theories that are unification based and for which information is concentrated in the lexicon (Pollard and Sag 1987; Calder et al. 1988), we adopt the *sign* as our basic unit of linguistic representation. For a given lexical entry, a sign describes the constraints—morphological, syntactic, and semantic—it introduces. The sign for intransitive *arrives,* for example, is:

$$\begin{bmatrix} \mathit{mor}: & \begin{bmatrix} \mathit{root}: \mathit{arrive} \\ \mathit{form}: \mathit{arrives} \end{bmatrix} \\ \mathit{syn}: & \begin{bmatrix} \mathit{head}: & \begin{bmatrix} \mathit{major}: v \\ \mathit{vform}: \mathit{fin} \\ \mathit{tense}: \mathit{pres} \\ \mathit{person}: \mathit{third} \\ \mathit{number}: \mathit{sg} \end{bmatrix} \\ \mathit{args}: & \left\langle \begin{array}{l} \begin{bmatrix} \mathit{syn}: & \begin{bmatrix} \mathit{head}: \begin{bmatrix} \mathit{major}: \mathit{prep} \\ \mathit{case}: \mathit{at} \end{bmatrix} \end{bmatrix} \\ \mathit{sem}: & \begin{bmatrix} \mathit{type}: \mathit{time} \mid \mathit{RestT} \end{bmatrix} \\ \mathit{order}: & \begin{bmatrix} \mathit{opt}: \mathit{opt} \\ \mathit{dir}: \mathit{post} \\ \mathit{adj}: \mathit{any} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \mathit{syn}: & \begin{bmatrix} \mathit{head}: \begin{bmatrix} \mathit{major}: n \\ \mathit{case}: \mathit{nom} \\ \mathit{person}: \mathit{third} \\ \mathit{number}: \mathit{sg} \end{bmatrix} \end{bmatrix} \\ \mathit{sem}: & \begin{bmatrix} \mathit{type}: \mathit{object} \mid \mathit{RestA} \end{bmatrix} \\ \mathit{order}: & \begin{bmatrix} \mathit{opt}: \mathit{oblig} \\ \mathit{dir}: \mathit{pre} \\ \mathit{adj}: \mathit{any} \end{bmatrix} \end{bmatrix} \end{array} \right\rangle \end{bmatrix} \\ \mathit{sem}: & \begin{bmatrix} \mathit{type}: \mathit{arrive} \\ \mathit{thetime}: \begin{bmatrix} \mathit{type}: \mathit{time} \mid \mathit{RestT} \end{bmatrix} \\ \mathit{theagent}: \begin{bmatrix} \mathit{type}: \mathit{object} \mid \mathit{RestA} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The lexical sign for *arrives* combines syntactic head features that help to determine the inflected form, with an **args** list that constrains its environment within the phrase of which it is the head; the **sem** feature represents the semantic structure that will be assigned to that phrase. The sign shows that the verb may optionally be followed by a prepositional phrase whose semantics will fill the semantic role **thetime**.[2] The argument preceding the verb is constrained to be third person singular nominative (i.e. not object-marked), and supplies the filler for the semantic role **theagent**.

In the interests of linguistic parsimony and sensible knowledge engineering, it is necessary for lexicalist approaches to factor away at the lexicon-encoding interface as many as possible of the commonalities between lexical items. To this end, we adopt the principles of default inheritance (Gazdar 1987), as embodied in the DATR language (Evans and Gazdar 1989). Areas where abstractions may be made over the lexicon are morphosyntax (Gazdar 1990), transitivity (Charniak and McDermott 1985; Flickinger et al. 1985; Hudson 1990), and combinations of these leading to lexical rules such as passive. To this we have added the area of lexico-semantic relations. In order to generalize over semantic roles, it is necessary to tie these to functional-syntactic roles, such as subject, direct object, etc. These in turn are related to order marked arguments in the **args** frame. Only the latter appear in the final version of the lexicon.

2 In our representation of feature structures we follow Prolog conventions, whereby variables are identified by initial capitals, and a vertical bar introduces the tail of a list.

A major issue for approaches such as ours is whether or not regularities in the lexicon should be expanded out off-line, or remain for lazy evaluation during parsing. We are sympathetic with the latter approach, for reasons of the economies that can be achieved in lexicon size. However, we believe that a precompiled lexicon is more appropriate to current speech recognition technology. Parsing typically involves extremely large lattices of lexical hypotheses with imprecise boundaries, and is thus computationally expensive. Our experience suggests that the trade-off between lexicon size and the cost of online inference is such as to favor lexicon size, in the case of application-specific lexicons of the size required in the SUNDIAL systems (around 2000 words). For inflection-impoverished English and (somewhat richer) French, which form the basis of our work, limited morphological decomposition during parsing is avoided; instead the parser lexicon consists of fully inflected forms.

The parser lexicon we have developed has the following two properties.

1. It is indexed by surface forms, i.e. fully inflected words that are unique at the phonological level. Efficiency of access is achieved by allowing some internal disjunctions within entries in cases where the surface form can be derived from a number of morphosyntactic feature combinations.
2. It consists of two separate knowledge bases: an *acceptance lexicon* and a *full lexicon*. The former is designed for efficient parsing. Only those features that constrain the ability of a sign to combine are represented. These include syntactic head features and semantic types. The encoding technique uses bit-vectors to achieve economy of representation and fast unification. The full lexicon contains signs with no information missing; the information in a full lexical entry is therefore a superset of the corresponding acceptance lexicon entry.

Parsing takes place in two phases: *lattice parsing* using the acceptance lexicon, involving heuristic search with intensive computation; and *structure building*, which operates on the analysis tree produced by the first phase, using term unification to combine the entries from the full lexicon corresponding to the lexical entries found by the first phase.

The lexicon compilation architecture that we present in this paper is outlined in Figure 1.

At the lexical encoding interface, a human lexicon builder builds an application- and sublanguage-specific lexicon, using a set of structured *base definitions*, which generalize over commonalities and provide macros with which to structure entries (Section 3). Both of these are written in DATR; we refer to the output of this as the DATR *lexicon*. The *lexicon generator* then compiles this into a lexicon for which the entries are directed acyclic graphs (DAGs) indexed by surface forms. For this a set of *closure definitions* is used. These constitute a knowledge base forming a set of meta-definitions to complement the DATR lexicon, as well as rendering explicit what may be implicit in the latter (Section 4). The resulting entries are encoded in two ways: for the full lexicon via Prolog term encoding and for the acceptance lexicon via bit coding (Section 5).

3. Encoding Linguistic Knowledge

3.1 DATR

DATR is a declarative language for representing inheritance networks that support multiple default inheritance. Knowledge is expressed in DATR in terms of path equa-

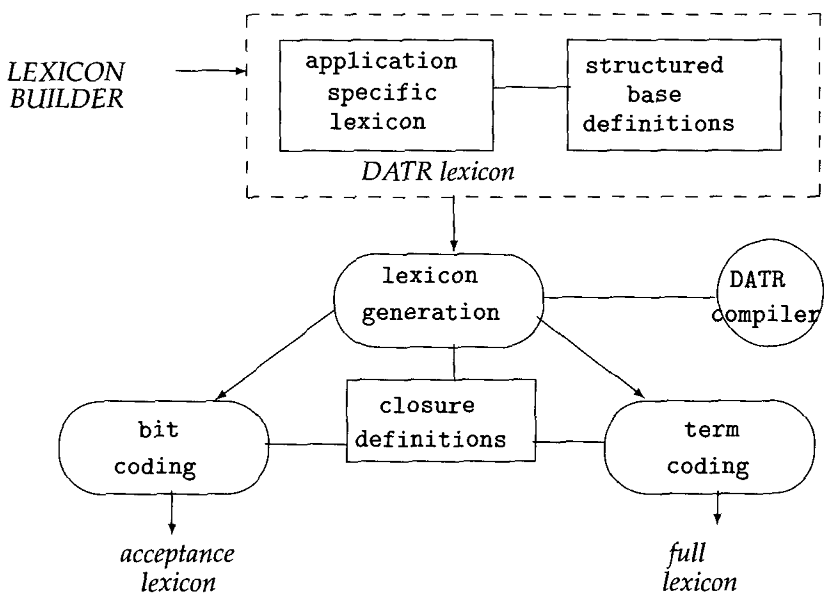


Figure 1
DIALEX lexicon compilation architecture.

tions. The syntax of paths is a superset of that found in the PATR-II language (Shieber 1986). For example, (1) identifies two different paths in the DAG rooted at `Node1` in an inheritance network.

(1) `Node1: <syn head case>`

`Node1: <syn head number>`

The path equations we present in this paper take the following forms:

(2) a. `Node1: <> == Node2`

b. `Node1: Path1 == Value1`

c. `Node1: Path1 == "Path2"`

d. `Node1: Path1 == Node2:Path2`

e. `Node1: Path1 == Node2:<>`

The form shown in (2a) is the special case in which the path at `Node1` is empty. This allows `Node1` to inherit all equations available at `Node2`, except those incompatible with equations at `Node1`. Two equations are incompatible if they both make assignments to the same path. The form shown in (2b) is used to assign values to paths, e.g. `<syn head number> == sg`. Alternatively, a value may be copied from elsewhere in the DAG. (2c) is used to assign to `Path1` whatever value is found for `Path2` at the original query node. The double quotes are significant here because they indicate that `Path2` must be evaluated globally. If the quotes were not present, `Path1` would be evaluated locally and assigned the value of Path2 at Node1 if such a value existed. The form shown in (2d) assigns to Node1:Path1 whatever value is found at Node2:Path2. A special case of this is (2e), which allows extensions of Path1 to be specified at Node2. For example, evaluating the DATR theory in (3) yields the theorems for node Ex2 shown in (4).

```
(3) Ex1:   <head major> == n
          <head case> == nom.


    Ex2:   <syn> == Ex1:<>.
(4) Ex2:   <syn head major> = n.
    Ex2:   <syn head case> = nom.
```

For a more detailed description of DATR see Evans and Gazdar (1990).

3.2 The Linguistic Framework

Linguistic knowledge is structured in terms of a simple unification categorial grammar (Calder et al. 1988) in which featural constraints at the levels of morphology, syntax, and semantics may all occur in a lexical sign. The basic sign structure of lexical entries is shown in (5).

$$(5)\quad \begin{bmatrix} \mathit{morphology} : [\ldots] \\ \mathit{syntax} : [\ldots] \\ \mathit{semantics} : [\ldots] \end{bmatrix}$$

The basic sign structure of the **syntax** feature value is shown in (6).

$$(6)\quad \mathit{syntax} : \begin{bmatrix} \mathit{head} : [\ldots] \\ \mathit{args} : [\ldots] \end{bmatrix}$$

The **head** feature includes attribute-value structures for such things as tense, person, number, and definiteness. The **args** feature is stack-valued, with stack position determining the order in which arguments may be combined by functional application.

The basic sign structure of the **semantics** feature value is shown in (7).

$$(7)\quad \mathit{semantics} : \begin{bmatrix} \mathit{id} :< \mathit{value} > \\ \mathit{type} :< \mathit{value} > \\ \mathit{modus} : [\ldots] \\ \mathit{role}{*} : [\ldots] \end{bmatrix}$$

Each semantic object has a unique index (**id**). The **type** feature locates the object in a sortal hierarchy. The **modus** feature specifies a number of constraints imposed on the interpretation of semantic objects, such as polarity, aspect, and tense. Semantic roles (such as **theagent, thetime, theinstrument**) are specified within the inheritance-based definitions for semantic types.

The signs are defined in terms of a dual-component DATR lexicon. The *base definitions* represent an application-independent account of morphosyntax, transitivity, and lexico-semantic constraints. They define what can be thought of as most of the higher nodes of an inheritance hierarchy. The base definitions as a whole are, of course, language specific, although significant exchange of definitions has been possible during the parallel development of our English and French DATR theories. The *application-specific lexicon* can be thought of as a collection of lower nodes that hook onto the bottom of the hierarchy defined by the structured base definitions. Whereas the structured base definitions provide a general morphological, syntactic, and lexico-semantic account of a language fragment, the application-specific lexicon provides a vocabulary and a task-related lexical semantics. Ideally, a change of application should only necessitate a change of an application-specific lexicon. Naturally, application-specific lexicons take much less time to construct than the base lexicon. Much of our discussion in the rest of this section will focus on the structured base definitions.

3.3 Morphosyntax

Since the requirements of speech processing in real time rule out online morphological parsing, a full-form lexicon must be produced. However, the task of entering all possible forms of a word into the lexicon by hand would be both time consuming and repetitive. We therefore provide a subtheory of morphology in the DATR base definitions so that the grammar writer need only specify exceptional morphology for each lexeme, leaving the lexicon generator to expand out all of the regular forms.

The surface form of an English verb encodes information relating to finiteness, tense, number, and person. What is required in the DATR theory is a number of condition–action statements that say things like:

```
IF    a verb is finite
THEN  IF   it is present tense AND singular AND third person
      THEN  its form is <root>+s
ELSE  its form is <root>.
```

The desired effect is achieved by means of DATR's *evaluable paths*. The following path equation is included at the VERB node.

```
(8) VERB: <mor form> == VERB_MOR:<>
```

The VERB_MOR node looks like this:

```
(9) VERB_MOR: <bse> == "<mor root>"
              <prp> == ("<mor root>" ing)
              <psp> == ("<mor root>" ed)

              <fin> == "< "<syn head tense>" "<syn head number>"
                         "<syn head person>" >"
              <pres> == "<mor root>"
              <pres sg third> == ("<mor root>" s)
              <past> == "<mor form psp>".
```

The base, present participle, and past participle forms are immediately available. If the verb is finite it is necessary to construct an evaluable path consisting of tense, number, and person values. If the tense is past (last line), the form is copied from the form of the past participle. If the form is present singular third person (second last line), the form is <root> +s. Otherwise, the present tense form is copied from the root form.

Exceptional forms are stated explicitly, thus overriding default forms. For example, the following entry for *hear* specifies that it has exceptional past forms.

```
(10) HEAR:   <> == VERB
             <mor root> == hear
             <mor form psp> == heard.
```

The evaluable path mechanism is also used to set the value of an agreement feature **agr** to tps (third person singular) or not_tps. The path equation shown in (11), augmented by the information at the V_AGREE node (12) then requires subject and verb to share the same **agr** feature value. The subject's **agr** feature is set by the definitions in (13).[3]

```
(11) VERB:   <syn args gr_subject syn head agr> ==
             V_AGREE:< "<syn head tense>" "<syn head number>"
                    "<syn head person>" >.
(12) V_AGREE:   <pres> == not_tps
                <pres sg third> == tps.
(13) NOUN: <syn head agr> ==
           N_AGREE:<agr "<syn head number>" "<syn head person>">.

     N_AGREE:   <agr> == not_tps
                <agr sg third> == tps.
```

English verb morphology presents no real problems; noun morphology is even simpler. French morphology is rather more complicated. However, it can be handled by means of the same general technique of allowing evaluable paths to act as case statements that select the appropriate morphological form. Instead of a unified account of French verb morphology there are a number of distinct inflectional paradigms from which different verbs inherit. A more sophisticated account of subject–verb agreement is also required.

3.4 Transitivity

Consider the relationship between verbs of different transitivity. An intransitive verb takes a subject only. A transitive verb takes a subject and an object. A ditransitive verb takes a subject and two objects, one direct and the other indirect. This information

3 In a few exceptional cases (e.g. *am/are/is* in the singular of *BE*) more complex constraints on agreement are stated in the relevant lexical entries.

is easily expressible in terms of an inheritance hierarchy. Facts about subjects are associated with a top node, for example a node called VERB. Facts about direct objects are associated with another node, for example, a node called TRANS_V. By saying that TRANS_V is a VERB, the general information about subjects is inherited at the TRANS_V node. This relationship can be expressed simply in DATR. A similar treatment can be adopted for ditransitive verbs (DTRANS_V):

```
(14) VERB:   <syn head major> == v
             <syn args gr_subject> == GR_SUBJECT:<>.

     TRANS_V:   <> == VERB
                <syn args gr_direct> == GR_DIRECT:<>.

     DTRANS_V:   <> == TRANS_V
                 <syn args gr_indirect> == GR_INDIRECT:<>.
```

Entries of the form <syn args gr_subject> == GR_SUBJECT:<> represent a convenient way of packaging up all information relating to an argument type at a single node (part of the information stored at this node can be found in (18) below; notice that different arguments are identified by unique labels such as gr_subject and gr_direct). We have already noted that in our sign representation, arguments are distinguished by their position in a stack. This ought to render unique argument labels superfluous. In fact, there are a number of reasons why it is desirable to use unique labels in the DATR theory. Firstly, they allow individual arguments of a word to be picked out (see Section 3.4.1 below). Secondly, they allow classes of argument to be identified and generalizations to be made where appropriate. For example, we show in Section 3.4.2 how order and optionality generalizations can be made over argument types, and how a system organized around named arguments can be mapped within DATR into an order-marked system. Finally, grammatical relation labels are much easier for grammar writers to remember and manipulate than positionally encoded argument structures.

Consider the following partial DATR entry for English infinitival complement verbs.

```
(15) INF_COMP_V:   <> == VERB
                   <syn args gr_comp> == GR_COMP:<>
                   <syn args gr_comp syn args gr_subject> ==
                     "<syn args gr_subject>".
```

The first line states that an infinitival complement verb inherits from the VERB node, i.e., it is a verb that must have a subject. The second line introduces a number of constraints on the complement. These constraints—collected at the GR_COMP node—include the fact that the complement must be the infinitive form of a verb. The next line enables the complement to share the subject of the matrix verb, i.e., in a sentence like *Amy wants to fly*, *Amy* is the subject of both *want* and *fly*.

3.4.1 Unevaluated Path Equations. Consider the relationship between the semantics of a verb and the semantics of its subject. The semantics of the subject must be coindexed with a semantic role of the verb such as **theagent**, as shown in (16).

$$(16) \quad \begin{bmatrix} syn : \begin{bmatrix} args : \begin{bmatrix} gr\_subject : \begin{bmatrix} sem : A \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ sem : \begin{bmatrix} theagent : A \end{bmatrix} \end{bmatrix}$$

This reentrancy can be expressed in DATR as follows:

```
(17)  <sem theagent> == "<syn args gr_subject sem>".
```

The argument labeled gr_subject is typically underspecified in the lexicon and awaits full specification at parse time. Because of this, the constraint is carried over to the DAG-encoding phase of lexicon compilation, where it becomes a reentrancy, as described in Section 4.

3.4.2 Argument Order and Optionality. While arguments in the structured base definitions are identified by grammatical relation labels, such as gr_subject, the lexicon generation process requires arguments encoding order and optionality constraints that are identified by relative position in an **args** list. Two techniques are used to produce DATR theories with arguments structured in this way.

The first technique is to define featural constraints of order and optionality for each grammatical relation label. Three types of constraint are defined:

dir: indicating whether the argument precedes or follows the functor (pre or post);

adj: indicating whether the argument is adjacent to the functor or not (next or any); and

opt: indicating whether the argument is optional or obligatory (opt or oblig).

Arguments identified as gr_subject and gr_oblique, for example, inherit the following ordering constraints:

```
(18) GR_SUBJECT:  <order dir> == pre
                  <order adj> == next
                  <order opt> == oblig.

     GR_OBLIQUE:  <order dir> == post
                  <order adj> == any
                  <order opt> == opt
```

Whereas the subject is obligatory, and precedes the functor and allows for intervening constituents, the oblique argument is optional and may appear in any position following the functor.

The second technique maps arguments identified by relation labels onto arguments identified by position in a linked list. Relative position is encoded in terms of the features **first** and **rest**: **first** identifies the first argument in a list, and **rest** identifies the linked list of remaining arguments.

Consider part of the base definition for transitive verbs, as shown in (19).

```
(19) TRANS_V:  <> == VERB
               <syn args gr_direct> == GR_DIRECT:<>
               <syn args> == TVARGS:<>.
```

Part of the collection of nodes devoted to mapping named arguments onto order-marked arguments is shown in (20).

```
(20) TVARGS:   <> == DTVARGS
               <rest> == DARGS:<>.

     DTVARGS:  <first> == "<syn args gr_subject>"
               <rest> == ARGS1:<>.

     DARGS:    <first> == "<syn args gr_direct>"
               <rest> == ARGS3:<>.

     ARGS3:    <first> == "<syn args oblique1>"
               <rest> == ARGS4:<>.
```

`TVARGS` inherits the value of `<first>` from `DTVARGS`, which finds it by evaluating the path `"<syn args gr_subject>."` The `<rest>` is then inherited from `DTVARGS` where the `<first>` argument of `<rest>` inherits from `"<syn args gr_direct>."` The `<rest>` of `<rest>` then inherits from `ARGS3`, which specifies the position of oblique arguments within the **args** list of transitive verbs.

3.5 Lexico-Semantic Constraints

A word lattice is likely to include numerous semantically anomalous but syntactically well-formed constructions. In a system that aims toward real time speech understanding it is vital that semantic selectional restrictions be introduced as early as possible in order to eliminate false phrasal hypotheses at an early stage.

Selectional restrictions are typically associated with lexemes. Each content word has a semantic type, and many words specify the semantic types of their arguments. For example, the semantic type of *tell* is **inform** and the type of its role **theexperiencer** (associated with the indirect object) is almost always **human** in our trial domain. This can be expressed as follows.

```
(21) TELL:  <> == DTRANS_V
            <mor root> == tell
            <sem type> == inform
            <sem theexperiencer type> == human.
```

Certain argument types can be assigned default semantic types. For example, by default the semantic type of subjects must be **sentient** (a superclass of **human**). This works for a large majority of verbs. Of course, defaults of this kind can be overridden for individual lexemes (such as the verb *rain*) or word classes (such as copular verbs).

3.6 Example: The French Noun Phrase

By way of example, we show how two entries from the French SUNDIAL lexicon, the determiner *le* ('the.MASC') and the common noun *passager* ('passenger'), are encoded in DATR; to put the following section in context, we also show the DIALEX output.

In a simple French noun phrase, we treat the common noun as the head, with the determiner as an optional argument. Therefore, most of the information is associated with the common noun.

A common noun inherits from the node NOUN:

```
(22) NOUN:   <> == WORD
             <syn head major> == n
             <syn head gender> == masc
             <syn head case> == nom
             <syn args gr_determiner> == GR_DETERMINER:<>
             <syn args gr_determiner syn head gender> ==
                 "<syn head gender>"
             <syn args> == NOUNARGS:<>
             <syn head number> ==
                 "<syn args gr_determiner syn head number>"
             <syn head def> ==
                 "<syn args gr_determiner syn head def>"
             <sem type> == entity.
```

NOUN itself inherits general word features, such as default morphology, from the node WORD:

```
(23) WORD:   <mor form> == "<mor root>".
```

Syntactic and semantic default values such as category (n), gender (masc), case (nom), and semantic type (entity) are given at the NOUN node. Some of these values may be overridden, for example in the definition of *passager*:

```
(24) Passager:   <> == NOUN
                 <mor root> == passager
                 <sem type> == passenger.
```

The number and definiteness of the noun phrase are specified by the determiner when

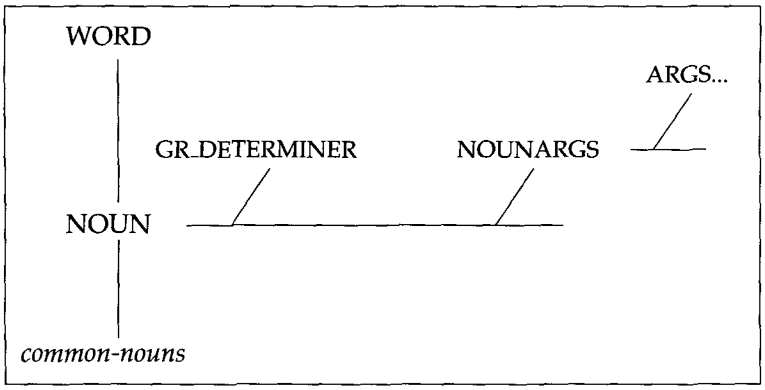


Figure 2
Inheritance graph for French common nouns.

present, whereas the gender of the determiner is copied from the common noun. Where a feature value is already specified for both noun and determiner at parse time, the values must be the same if combination is to take place. The definitions for GR_DETERMINER and NOUNARGS are shown in (25):

```
(25) GR_DETERMINER:  <syn head major> == det
                     <order adj> == any
                     <order opt> == opt
                     <order dir> == pre.

     NOUNARGS:       <first> == "<syn args gr_determiner>"
                     <rest> == ARGS:<>.
```

The definition of GR_DETERMINER specifies order and optionality information as well as the syntactic category (det). NOUNARGS defines the mapping of case-marked to order-marked arguments, for simple determiner–noun NPs.

The inheritance graph for this set of DATR sentences is illustrated in Figure 2.

In fact, common nouns may be more complex than our example suggests; they may have several obliques, for example. Fortunately, DATR allows the creation of intermediate nodes between the NOUN node and the common nouns, and these nodes specify distinctive properties of each distinct class of nouns. For example, a RELDAY node has been created for French in order to describe common grammatical properties for relative day references such as *lendemain* ('tomorrow') and *veille* ('the day before'). In the same spirit, NPs with genitive postmodifiers such as *le numero du vol* ('the number of the flight/the flight number'), where two nouns are combined, use the node GNOUN, which specifies general features of the arguments of the head noun.

The definition of the determiner node, DET, is simple in comparison with the NOUN node, inheriting only from the WORD node. Example (26) shows the definition of DET,

$$\begin{bmatrix} \mathit{mor}: & \begin{bmatrix} \mathit{root}: \mathit{le} \\ \mathit{form}: \mathit{le} \end{bmatrix} \\ \mathit{syn}: & \begin{bmatrix} \mathit{head}: & \begin{bmatrix} \mathit{major}: \mathit{det} \\ \mathit{gender}: \mathit{masc} \\ \mathit{def}: \mathit{the} \\ \mathit{number}: \mathit{sg} \end{bmatrix} \end{bmatrix} \\ \mathit{sem}: & \begin{bmatrix} \mathit{modus}: \begin{bmatrix} \mathit{def}: \mathit{the} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \mathit{mor}: & \begin{bmatrix} \mathit{root}: \mathit{passager} \\ \mathit{form}: \mathit{passager} \end{bmatrix} \\ \mathit{syn}: & \begin{bmatrix} \mathit{head}: & \begin{bmatrix} \mathit{major}: \mathit{n} \\ \mathit{case}: \mathit{nom} \\ \mathit{gender}: \mathit{masc} \\ \mathit{def}: A \\ \mathit{number}: B \end{bmatrix} \\ \mathit{args}: & \begin{bmatrix} \mathit{first}: & \begin{bmatrix} \mathit{syn}: & \begin{bmatrix} \mathit{head}: & \begin{bmatrix} \mathit{major}: \mathit{det} \\ \mathit{gender}: \mathit{masc} \\ \mathit{def}: A \\ \mathit{number}: B \end{bmatrix} \end{bmatrix} \\ \mathit{order}: & \begin{bmatrix} \mathit{opt}: \mathit{opt} \\ \mathit{dir}: \mathit{pre} \\ \mathit{adj}: \mathit{any} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \mathit{sem}: & \begin{bmatrix} \mathit{type}: \mathit{passenger} \end{bmatrix} \end{bmatrix}$$

Figure 3
DAG lexicon entries for *le* and *passager*.

together with entries for *le* and *la* ('the.FEM').

```
(26) DET:  <> == WORD
           <syn head major> == det
           <syn head def> == the
           <syn head number> == sg
           <syn head gender> == masc
           <sem modus def> == the.

     le:   <> == DET
           <mor root> == le.

     la:   <> == DET
           <mor root> == la
           <syn head gender> == fem.
```

The lexical entries for *le* and *passager* produced by the DAG-encoding phase of compilation (see Section 4) are shown in Figure 3.

4. Lexicon Generation

4.1 Obtaining the DNF Lexicon

In order to generalize across morphological instantiations, a DATR theory makes use of nodes at the level of the lexeme. In general, the constraints in a lexeme cannot be simply represented as a union of paths. This is due to the fact that the sentences making up the definition of a lexeme for which morphosyntactic variations exist implicitly contain disjunctions. Because we require the lexicon to be disjoint, our strategy is to cash out all embedded disjunctions that reference each surface form. The lexicon thus obtained can be described as being in disjunctive normal form (DNF). This *DNF-lexicon* will contain all lexical signs, where a lexical sign incorporates both the surface form and the corresponding lexeme.

In order to govern the expansion of information in the DATR lexicon, it is necessary to make a closed world of the feature space defined there. The values that features may take may be implicit in the DATR lexicon; however such implicit knowledge is not necessarily complete. Nothing prevents arbitrary extension of the lexicon by additional features and values, and this may lead to unwanted interactions with existing rules. We therefore enumerate the possible values of features in a knowledge base known as the *closure definitions*. This enumeration is designed to be recursive, to take into account category-valued features such as the **args** list. Figure 4 gives an example of closure definitions, for a sign with only **syn** and **mor** attributes. These state the features that make up a sign; the definition is recursively introduced at the level of <`syn args`>. A closure definition takes the form:

$$cdef(Feature, Fields, FieldVals, FCRs).$$

A complex feature is composed of fields—either these are atomic valued, and enumerated or declared as open class in *FieldVals;* or they are complex and their definitions are to be found elsewhere.

```
cdef(sign, [syn,mor],_, [mor:form=>syn:vform]).
cdef(syn, [head,args],_,_).
cdef(head, [major,type,vform,tense,number,person],
        [
        major==[n,v,det,prep],
        vform==[fin,bse,prp,psp],
        tense==[pres,past],
        number==[sg,pl],
        person==[first,second,third]
        ],
        [vform:fin => [tense,person,number]]
        ).
cdef(args,setof(sign),_,_).
cdef(mor, [form,root], [open(form),open(root)],_).
```

Figure 4
Example closure definitions.

Besides providing closure for DNF lexicon expansion, these definitions have a number of uses:

1. they are used to determine which possible paths the compiler should try to evaluate in order to build a DAG representation of a lexical sign. The search proceeds depth-first through the closure definitions, ignoring those fringes of the search space for which no evaluation is possible. Values of paths, constraints representing unevaluable reentrancies, and consistent combinations of these are returned;
2. they provide a filter on the output of the DATR lexicon. Only those features present in the closure definitions are output. Constraints incorporating functional labels such as `gr_subject` are no longer needed;
3. they include a complete set of knowledge definitions for our semantic representation language (SIL), which is inheritance based. The inheritance hierarchy for semantic types, for example, is used in bit coding (Section 5), so that semantic selectional restrictions can be tested during parsing;
4. they furnish a set of declarative templates against which bit coding and DAG-term conversion may be carried out.

In addition to an enumeration of feature values, the closure definitions contain Feature Cooccurrence Restrictions (FCRs) (Gazdar et al. 1985). In principle these could be encoded in the DATR lexicon, for example, using the feature-value *unspec* to represent negative occurrence. Their presence here is not only to restrict the possible feature combinations that can appear at the head of a sign, but also to detect dependencies that govern DNF expansion.

The DNF lexicon is obtained as follows. Those features on which the surface form of a full lexical sign depend, which we shall refer to as its *surface form dependency features*, may be derived from the FCRs contained in the closure definitions. Then for each pair consisting of a DATR node Λ and a possible assignment to its unassigned surface form dependency features Φ, generate a new DATR node Λ^{Φ}, which inherits from Λ and contains the feature assignments in Φ. The DATR theory for Λ^{Φ} is then used to produce the set of evaluated and unevaluated constraint sentences that describe it. For example, the base lexical entry for *arrive* is defined at the DATR node `Arrive1`, which is underspecified for the paths `<syn head tense>`, `<syn head person>`, and `<syn head number>`. For the assignment of values `pres`, `third`, `sg` (respectively) to these paths, the node `Arrive1_presthirdsg` is created.

4.2 Producing Unevaluated Paths

As we have shown, reentrancies can be specified in DATR using global inheritance; see, for example, (15) in Section 3.4.1. However, such sentences may not appear directly in the DAG representation, either because they include paths not derivable within the closure definitions, or because interaction with higher-ranking exceptions may lead to weaker equivalences being derived. Any DATR sentence that does not explicitly introduce a value is treated as a candidate reentrancy constraint; at the stage where constraint sentences are being derived from a DATR theory, all unevaluated constraint sentences are retained. In the case of `Arrive1_presthirdsg`, the following constraint sentences are derived by inheritance from `Verb`:

```
(27) <syn args first sem> = <syn args gr_subject sem>.
     <sem theagent> = <syn args gr_subject sem>.
```

DATR inference takes the form of successive reduction of right-hand sides; in (27), neither sentence is further reducible—both would be ignored by a standard DATR theorem-prover. By passing both constraints to the DAG-building procedure however, where equality is reflexive as well as transitive (Section 4.3), the two constraints may be combined to derive the reentrancy between <sem theagent> and <syn args first sem>.

4.3 DAG Building and Disjunction Optimization

The constraint sentences derived for a DATR node Λ or for an extension of it Λ^{Φ} are of the form Path = Value or Path1 = Path2. If consistent, they can be used to build a DAG corresponding to Λ^{Φ}. Our DAG-building procedure is based on one described in Gazdar and Mellish (1989). It builds DAGs by unification of constraints, so that directionality is irrelevant. For this to succeed, the input constraints must not contain inconsistencies. This property of correctness is only partially guaranteed by the constraint-derivation stage, which will tolerate an unevaluated constraint whose left-hand side is a proper prefix of an evaluated one (but not vice versa), as in (28).

```
(28)  <sem theagent type> = object.

      <sem theagent> = <syn args gr_subject sem>.
```

This will work so long as a contradictory type is not derivable elsewhere. The form of encoded DAGs is known as *normal form* (Bouma 1990); that is, if two DAGs share a common sub-DAG, this is explicitly represented in both, with the exception of unevaluated sharing sub-DAGs that are represented as Prolog variables. Once the DAG is built, any remaining unwanted paths are filtered out. In the case of Arrive1_presthirdsg, this amounts to removing those sub-DAGs introduced at paths containing gr_subject and gr_oblique1.

Although the closure definitions ensure that the number of surface form dependency feature assignments for each lexeme is finite, in practice for languages like English where a number of morphosyntactic feature combinations map onto a smaller set of surface forms, the DNF lexicon will have more entries than there are distinct surface forms. In cases where a number of entries differ only in a single feature, a phase of *disjunction optimization* serves to reduce these, according to the simple equivalence:

$$(\phi_1 \wedge \phi_2 \wedge \ldots \phi_n) \vee (\phi_1' \wedge \phi_2 \wedge \ldots \phi_n) \equiv (\phi_1 \vee \phi_1') \wedge \phi_2 \wedge \ldots \phi_n.$$

Apart from this optimization, the lexicon produced is in DNF form.

5. Bit Coding

5.1 Motivation and Requirements

The last step toward the production of data structures for efficient parsing and generation is the construction of two separate lexicons: a Prolog term encoding of the DAGs and a compact bit-encoded lexicon. The motivation for two separate lexicons is the decision to split the task of parsing into its two aspects: determining grammaticality and assigning an interpretation. Since in speech recognition there is also the added complication of identifying the best-scoring sequence of words from a lattice of hypotheses, and since an interpretation is only needed for the best sequence, not for every acceptable one, it is more efficient to separate these tasks. This involves separating lexical entries into those features that are constraining (i.e. which affect a sign's capacity to combine with others) and those that simply contribute to its eventual interpretation. The former set is used to produce the bit-coded 'acceptance' lexicon, the latter to form a term-encoded 'full' lexicon.

As well as being used in sentence interpretation, the full lexicon is also used in sentence generation. However, we shall concentrate here on the bit-encoded acceptance lexicon.

Since the search space when parsing a typical word hypothesis lattice is potentially great, the acceptance lexicon must be both compact and suitable for processing by efficient low-level operations. Bit encoding allows unification of feature structures to be performed by Boolean operations on bit strings, which enables a parser to be implemented in an efficient programming language such as C; it also provides a convenient representation of disjunctions and negations of feature values.

Two distinct kinds of bit coding are used to represent semantic types and syntactic head features: both produce vectors of bits that can be stored as integers or lists of integers.

5.2 Semantic Type Coding

The principal semantic type of a lexical entry is a node in a tree-structured (single-inheritance) sortal hierarchy. Coding for types in the hierarchy is straightforward:

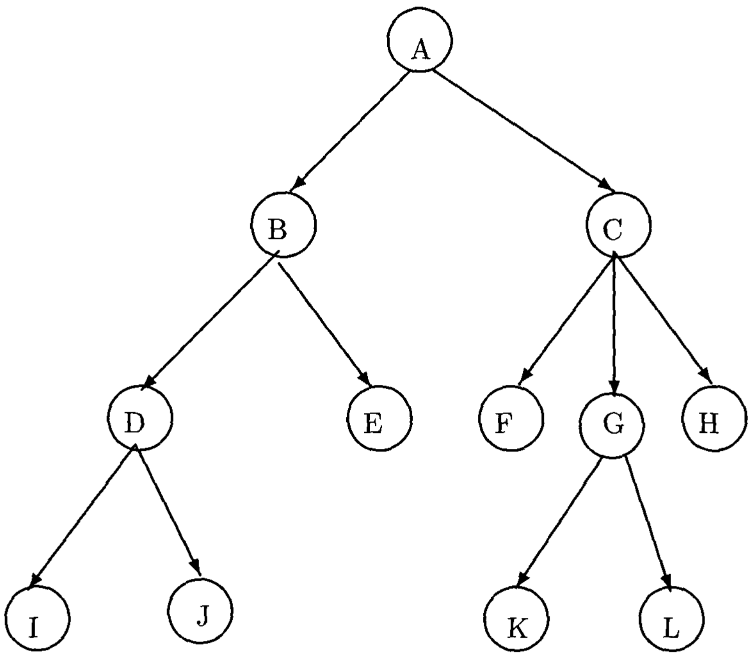
- a terminal node has one unique bit set;
- a nonterminal node is represented by the bitwise Boolean OR of the codings for the nodes it dominates.

This scheme requires as many bits as there are terminal nodes in the tree and, assuming that every nonterminal node dominates at least two subnodes, assigns a unique bit vector to every node. (A simple example is given in Figure 5). The most specific types are represented by a bit vector containing a single '1,' and the most general by a vector with all its bits set. Unification of two types is performed by bitwise AND; since the hierarchy is tree structured the result of this will be the coding of the more specific type, or 0 indicating failure if the types are incompatible. The same coding scheme would also serve if the hierarchy were extended to a multiple-inheritance graph, the only difference being that bitwise AND could then result in a type distinct from either of its arguments.

5.3 Syntactic Feature-Value Coding

Our approach to the encoding of the feature structures used to represent syntactic categories is very similar to that proposed in Nakazawa et al. (1988) for implementing GPSG-style grammars.

A set of features is represented by a bit vector in which for every n-valued feature, $n + 1$ bits are assigned, one associated with each value and one bit indicating that the feature is not present. A value of '0' for a bit means that the feature does not have the corresponding value; a '1' indicates that the value is a possible one. If the value of a feature can be specified precisely, the corresponding bit is set, and all the others for that feature are cleared. Hence the negation of a feature-value pair can be represented by clearing a bit, and a disjunction of values by setting more than one bit in the representation of a feature. This fact can be utilized to pack lexical entries together: if two entries differ only in one atomic-valued feature, they can be combined into a single entry by this method. Unification is again performed by bitwise AND; failure is indicated if all the bits for some feature are turned off, meaning that the structures



Type	Bit Vector
A	1111111
B	1110000
C	0001111
D	1100000
E	0010000
F	0001000

Type	Bit Vector
G	0000110
H	0000001
I	1000000
J	0100000
K	0000100
L	0000010

Figure 5
Bit coding of the semantic type hierarchy.

being unified have no common value for this feature. Since this operation only turns bits off, unification of bit vectors is order-independent (commutative and associative).

The bit vector representation is straightforward for encoding flat feature-value structures, but presents difficulties when features have categories as values, given the requirement that the possible values for all features can be enumerated in order to produce bit vectors of finite size. Although a general solution can be proposed that uses some pattern of bits to indicate a recursive feature and associates with this feature another bit vector of the same length (the solution adopted by Nakazawa et al. 1988), we have chosen a more *ad hoc* encoding, specifying in advance which features can be recursive and representing them by pointers to similarly coded structures. The features that are recursive are the list of arguments of a functor sign and the **slash** feature used to handle long-distance dependencies.[4] (This approach enables the parser to process

4 We follow GPSG in the use of the category-valued feature **slash** as a propagating device to handle extraction phenomena. For example in the question 'what did you say?', the phrase 'did you say?' can

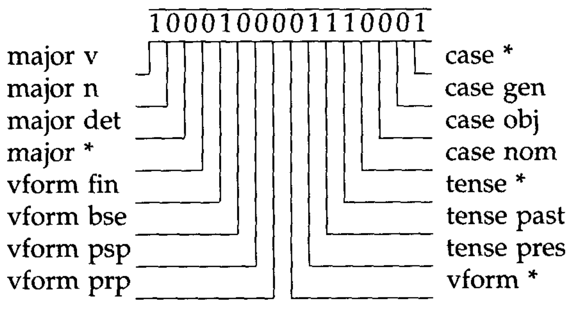


Figure 6
Sample bit vector for head features *major*, *vform*, *tense*, and *case*.

signs more efficiently, but unfortunately makes it partly dependent on their structure).

The bit encoding procedure takes as input a DAG representation of individual lexical entries and is guided in its translation by the closure definitions. A set of declarations is used to indicate which features are to be included in the acceptance lexicon, and how they are to be encoded: using either of the bit representations discussed above, or simply as a list of encodings of their constituents. If no coding type is specified for a feature, then it is simply ignored.

As a simple example, consider the following partially specified feature structure:

$$(29) \quad \begin{bmatrix} head : \begin{bmatrix} major : v \\ vform : fin \end{bmatrix} \end{bmatrix}$$

Assume that the closure definitions specify values for the head features **major**, **vform**, **tense** and **case**, and the FCR:

$$case \Rightarrow major : n$$

Then if the node **head** is declared for bit coding, the vector shown in Figure 6 will be produced. (The symbol '*' stands for 'not present'). Note that bits have been set for all values of the unspecified feature **tense**, indicating that nothing is known about its value, but that only the '*' bit is set for the feature **case**, since the FCR blocks its presence for entries whose **major** feature is not *n*.

5.4 Variable Sharing

Although the representation of variables and their instantiation to (more or fully) specified values is straightforward, the implementation of variable sharing or reentrancy presents a serious problem for bit coding schemes, since there is no means of representing identifiable variables. We have adopted a two-fold solution, depending on the type of the variable. For sign-valued variables, and other large scale structures, sharing is achieved by means of pointers to common data objects.

This approach cannot be extended down to the level of bit-coded features, since these involve data below the level of the machine word. Instead a solution based on

be partially characterized, in our notation, as

$$\begin{bmatrix} syn : \begin{bmatrix} head : \begin{bmatrix} major : v \end{bmatrix} \\ args : [] \end{bmatrix} \\ slash : \begin{bmatrix} first : \begin{bmatrix} syn : \begin{bmatrix} head : \begin{bmatrix} major : n \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

indicating that it is a sentence from which a noun phrase has been extracted.

the use of bit masks has been adopted. The key to this is the recognition that variable sharing between structures is a limited form of unification, carried out between a restricted set of their features. If two feature structures represented by bit vectors β_1 and β_2 share a variable for the feature ϕ, a mask μ is constructed in which all the bits representing ϕ are cleared, and all the rest are set. The values for ϕ in the two bit vectors are unified in the result of the expression:

$$\beta_1 \wedge (\beta_2 \vee \mu)$$

Note that a single mask may represent more than one variable shared between two structures.

A disadvantage of this technique is that it requires the construction of masks for all possible feature structures within a sign between which variables may be shared. In practice we can assume that this means only the recursively nested signs of the **args** list and **slash**, and so need relatively few masks.

A description of the two-stage parsing procedure can be found in Andry and Thornton (1991).

6. Implementation and Coverage

DIALEX is implemented in Quintus Prolog; benchmark tests indicate that compilation time is linear in the size of the lexicon. Development of very large scale lexicons is somewhat hindered by the current lack of effective debugging tools. We have, however, succeeded in constructing lexicons that cover a broad range of syntactic phenomena in both French and English. For example, the English DATR lexicon covers all distinctive lexical forms in our corpus gathered from simulations of flight enquiry dialogues (Fraser and Gilbert 1991). Furthermore, one of the major advantages of DATR's inheritance-based approach is ease of adding new lexical entries. For example, a large number of entries for cities is required in the flight information domain. With the definition of a CITY_PROP node to specify general properties of proper nouns identifying cities, individual cities such as *Paris* are simple and quick to define:

```
(30) Paris:   <> == CITY_PROP
              <mor root> == paris
              <sem thecity value> == paris.
```

Extending the lexicon to include new verbs, especially verbs with idiosyncratic properties like *try*, takes more time and effort.

This paper has been mainly concerned with the definition and compilation of lexicons for understanding. In fact, SUNDIAL applications are such that a production lexicon shares a considerable portion with its recognition counterpart. To this end, DIALEX has been adapted for compilation of a generation lexicon (Youd and McGlashan 1992). This is derived from the same DATR definitions but differs from the parser lexicons in that indexing is based on semantic type and complexity, rather than the surface string, and inflection is factored away from the lexical entries.

7. Conclusion

In the design of our lexicon compilation tool, we have shown how linguistic knowledge can be arranged in terms of a set of DATR structured base definitions that are portable across applications. Knowledge at the levels of morphology, syntax, and semantics combines in a single reusable DATR database. The fact that this knowledge is expressed in a high-level representation language does not limit its usefulness. On the contrary, it allows the designers of base definitions or application lexicons to think clearly about the structural relations that hold between objects in the representation and to maximize generalizations. Default inheritance allows generalizations to trickle down to specific instances, unless overridden. As a consequence, every good generalization captured during the design of structured base definitions represents labor saved during subsequent application-specific work.

We have also shown how high-level knowledge can be entered by the lexicon builder at the appropriate conceptual level and then compiled into a lower level form appropriate for a chosen application. The system we describe produces two kinds of output: a term-encoded full lexicon for use in sentence interpretation and generation, and a lower level bit-encoded acceptance lexicon for use in lattice pruning during speech processing. The modular design of our system makes it particularly easy to exchange existing coding modules for new ones, thus allowing linguistic knowledge to be customized for a wide variety of speech or language applications.

References

Andry, François, and Thornton, Simon. (1991). "A parser for speech lattices using a UCG grammar." In *Proceedings, 2nd European Conference on Speech Communication and Technology.* Genova, September 1991, 219–222.

Bilange, Eric. (1991). "A task independent oral dialogue model." In *Proceedings, 5th Meeting of the European Chapter of the Association for Computational Linguistics.* Berlin, April 1991, 83–88.

Bouma, Gosse. (1990). "Defaults in unification grammar." In *Proceedings, 28th Annual Meeting of the Association for Computational Linguistics.* Pittsburgh, June 1990, 165–172.

Calder, Jo; Klein, Ewan; and Zeevat, Henk. (1988). "Unification Categorial Grammar: a consise extendable grammar for natural language processing." In *Proceedings, COLING-88.* Budapest, August 1988, 83–86.

Charniak, Eugene, and McDermott, Drew. (1985). *An Introduction to Artificial Intelligence.* Lawrence Erlbaum Associates.

Evans, Roger, and Gazdar, Gerald. (1989). "Inference in DATR." In *Proceedings, 4th Meeting of the European Chapter of the Association for Computational Linguistics.* Manchester, April 1989, 66–71.

Evans, Roger, and Gazdar, Gerald, eds. (1990). *The* DATR *Papers.* Research Report CSRP 139, School of Cognitive and Computing Science, University of Sussex.

Flickinger, Daniel P.; Pollard, Carl J.; and Wasow, Thomas. (1985). "Structure-sharing in lexical representation." In *Proceedings, 23rd Annual Meeting of the Association for Computational Linguistics.* Chicago, 262–267.

Fraser, Norman M., and Gilbert, G. Nigel. (1991). "Effects of system voice quality on user utterances in speech dialogue systems." In *Proceedings, 2nd European Conference on Speech Communication and Technology.* Genova, September 1991, 57–60.

Gazdar, Gerald. (1987). "Linguistic applications of default inheritance mechanisms." In *Linguistic Theory and Computer Applications,* edited by Peter Whitelock; Harold Somers; Rod Johnson; and Mary McGee Wood. Academic Press.

Gazdar, Gerald. (1990). "An introduction to DATR." In *The* DATR *Papers,* edited by Roger Evans and Gerald Gazdar. Research Report CSRP 139, School of Cognitive and Computing Science, University of Sussex: 1–14.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffry; and Sag, Ivan. (1985). *Generalized Phrase Structure Grammar.* Harvard University Press. Cambridge, MA.

Gazdar, Gerald, and Mellish, Chris. (1989). *Natural Language Processing in Prolog.* Addison Wesley.

Hudson, Richard A. (1990). *English Word Grammar.* Basil Blackwell.

McGlashan, Scott; Fraser, Norman M.; Gilbert, G. Nigel; Bilange, Eric; Heisterkamp, Paul; and Youd, Nick J. (1992). Dialogue management for telephone information systems. In *Proceedings of the 3rd Conference on Applied Natural Language Processing.* Trento, April: 245–246.

Nakazawa, Tsuneko; Neher, Laura; and Hinrichs, Erhard W. (1988). "Unification with disjunctive and negative values for GPSG grammars." In *Proceedings, 8th European Conference on Artificial Intelligence.* Munich, August 1990, 467–472.

Peckham, Jeremy. (1991). "Speech understanding and dialogue over the telephone: an overview of the SUNDIAL project." In *Proceedings, 2nd European Conference on Speech Communication and Technology.* Genova, September 1991, 1469–1472.

Pollard, Carl, and Sag, Ivan A. (1987). *Information-Based Syntax and Semantics.* CSLI, Stanford, CA.

Shieber, Stuart M. (1986). *An Introduction to Unification-Based Approaches to Grammar.* CSLI, Stanford, CA.

Youd, Nick J.; and McGlashan, Scott. (1992). Generating utterances in dialogue systems. In *Aspects of Automated Natural Language Generation: Proceedings of the 6th International Workshop on Natural Language Generation,* edited by Robert Dale; Eduard Hovy; Dietmar Rösner; and Olivero Stock. Academic Press, London.