DATR AS A LEXICAL COMPONENT FOR PATR

James Kilbury, Petra Naerger, Ingrid Renz
Seminar für Allgemeine Sprachwissenschaft
Heinrich-Heine-Universität Düsseldorf
Universitätsstraße 1
D-4000 Düsseldorf 1
Federal Republic of Germany
e-mail: kilbury@dd0rud81.bitnet
naerger@dd0rud81.bitnet
renz@dd0rud81.bitnet

ABSTRACT

The representation of lexical entries requires special means which basic PATR systems do not include. The language DATR, however, can be used to define an inheritance network serving as the lexical component. The integration of such a module into an existing PATR system leads to various problems which are discussed together with possible solutions in this paper.

1 MOTIVATION

In the project "Simulation of Lexical Acquisition" (SIMLEX) unification is used to create new lexical entries through the monotonic accumulation of contextual grammatical information during parsing. The system which we implemented for this purpose is a variant of PATR as described in (Shieber, 1986).

Besides collecting the appropriate information for an unknown word, i.e. a lexeme not already specified in the given lexicon, the creation of its new lexical entry is a major goal. In this context questions about the nature of lexical information, the structuring, and the representation of this information must be answered. The present paper is mainly concerned with the structuring and representation of information in lexical entries.

2 REPRESENTATION OF LEXICAL INFORMATION

We assume that certain conditions must be met by an adequate representation of lexical information. The most important of these is that it captures linguistic generalizations, which means that associated information is represented together or bundled. One advantage of this bundled information is its reusability, which allows redundancy to be reduced. The representation of lexical information should enable us to express a further kind of generalization, namely the relations between regularity, subregularity, and irregularity. Furthermore, the representation has to be computationally tractable and -- possibly with the addition of "syntactic sugar" -- more or less readable for human users.

The formalism of PATR offers two possible means of representing lexical information. First of all, the information can be encoded in feature structures directly. Except for computational tractability, none of the other criteria are met. The second facility consists of macros or templates which assemble the linguistic information so that it can be reused in various places in the lexicon. This meets the most important of the above-mentioned conditions and reduces redundancy. But the encoded information is inherited monotonically, i.e. only regularities can be expressed. In order to structure lexical information adequately, other relations like subregularities and exceptions should also be expressible.

Macros fail to achieve this, whereas default inheritance networks are well-suited for the purpose. In the following section we give an overview of one such network formalism which was primarily designed for representing lexical information.

3 OVERVIEW OF DATR

DATR (described in detail by Evans/ Gazdar, 1989a; 1989b; 1990) is a declarative language for the definition of semantic networks which allows for defaults as well as multiple inheritance. Its general properties are non-monotonicity, functionality, and deterministic search.

A DATR *theory* (or network description) is a set of *axioms* (or expressions) which are related to each other by references. Together they define a hierarchical structure, a net. Both regularities and exceptions can be expressed, regularities using default inheritance, and exceptions, overriding.

DATR axioms consist of *node-path pairs* associated with a right-hand side. This can be a *value* (atomic or list), or an *evaluable* DATR expression if the value is to be inherited from another node, path, or node-path pair. The following DATR theory comprising three *node definitions*[1] encodes familiar linguistic information to illustrate some relevant DATR features:

(1)
```
LEXICAL:     <syn major bar> == zero.

NOUN:        <> == LEXICAL
             <syn major nv n> == yes
             <syn major nv v> == no.

ADJ:         <> == LEXICAL
             <syn major nv n> == NOUN
             <syn major nv v> ==
                      <syn major nv n>.
```

The represented information can be retrieved with special DATR queries. These also consist of a node-path pair, whose evaluation returns the value sought. With the above DATR description the following examples show sensible DATR queries and their corresponding values:

(2)
```
NOUN:<syn major nv n> ?
yes      (atomic value)
NOUN:<syn major nv v> ?
no       (atomic value)
NOUN:<syn major bar> ?
zero     (inherited from node LEXICAL)
ADJ:<syn major nv n> ?
yes      (inherited from node NOUN)
ADJ:<syn major nv v> ?
yes      (inherited from node NOUN via path
         <syn major nv n> in node ADJ)
ADJ:<syn major bar> ?
zero     (inherited from node LEXICAL)
```

Seven inference rules and a default mechanism are given for the evaluation of DATR queries. Their precise semantics and properties are described in (Evans/Gazdar, 1989b; 1990).

A major feature of DATR is its distinction between *global* and *local* inheritance. In the above example only local inheritance is involved, but global inheritance plays a crucial role in one of the later examples. Variables constitute an additional device available in DATR but are assumed to have the status of abbreviations.

Despite their syntactic similarities, DATR and PATR differ completely in their semantics, so that there is no obvious way of relating the two formalisms to each other. Some approaches are discussed in the next section.

4 RELATING DATR AND PATR

A PATR system needs to have the lexical information it uses encoded in feature structures consisting of attribute-value pairs. The lexical information represented in the DATR theory above (1) would appear as follows when stated in feature structures:

(3)
information specific to N0:

$$\begin{bmatrix} \textit{syn}: & \begin{bmatrix} \textit{major}: & \begin{bmatrix} \textit{bar}: & \textit{zero} \\ \textit{nv}: & \begin{bmatrix} \textit{n}: & \textit{yes} \\ \textit{v}: & \textit{no} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

information specific to ADJ0:

$$\begin{bmatrix} \textit{syn}: & \begin{bmatrix} \textit{major}: & \begin{bmatrix} \textit{bar}: & \textit{zero} \\ \textit{nv}: & \begin{bmatrix} \textit{n}: & \textit{yes} \\ \textit{v}: & \textit{yes} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The question that arises is how to relate DATR and PATR so that the hierarchically structured lexical information in DATR can be made available in PATR-usable feature structures.

4.1 A DATR-PATR INTERFACE

The first idea that one might have is to exploit the syntactic similarities between the two formalisms and encode the lexical information in a DATR description like (1). In this way a DATR axiom like *NOUN: <syn major nv n> == yes* would be directly equivalent to the path equation *<NOUN syn major nv n> = yes* in PATR, where the node name in DATR corresponds to the variable name for a feature structure in PATR. Although this looks reasonable, one major problem arises: You must know exactly the path you want to query, i.e. all its attributes and their precise order. If such a query is posed, the answer will be the atomic value yielded by the DATR evaluation.

Such an approach requires an interface with the following functions: Queries that the grammar writer has stated explicitly have to be passed on to DATR. Every query together with the resulting value has to be transformed into a PATR path equation (that partially describes a feature structure) and passed on to the PATR system. What is most disturbing about this strategy is the fact that for every distinct PATR path you have to know the corresponding DATR query. It is tempting to think one could simply check which paths are defined for a given node, but this doesn't work because of inheritance: the entire network is potentially relevant. So in effect all the PATR structures except the atomic values have to be defined twice: once in the DATR statements and once in the queries. This redundancy cannot be eliminated unless types for the feature structure are declared which are consulted in formulating the queries.

4.2 USING DATR OUTPUT DIRECTLY

A completely different approach is to formulate a DATR theory which gives the lexical information in a PATR-usable format (i.e. a feature structure) as the result of the evaluation of a DATR query. Thus, the DATR description reflects the *hierarchical* structure of the lexical information and consequently meets one of the main requirements for an adequate representation that cannot be met by a simple PATR formalism. The resulting feature structures include all the information necessary for PATR but neglect the *inheritance* structure, although the latter is involved in their construction (i.e. the evaluation of queries). There are various DATR-programming techniques that realize these ideas. Three examples will be presented here which cover the lexical information encoded in (1).

The first technique, which is illustrated in (4)[2], uses *global inheritance* (represented with double quotation marks) to store the node at which the query originates. This also allows other information in the global node to be accessed.

```
(4)
SYNTAX:          <> == ( [ syn ':' [ "<synpaths>" ] ] ).
MAJOR:           <> == SYNTAX
                 <synpaths> ==
       ( maj ':' [ "<majpaths>" ] ).

NV:              <> == MAJOR
                 <majpaths> ==
       ( nv ':' [ n ':' "<n>" ,  v  ':' "<v>" ]).
NOUN:            <> == NV
                 <n> == yes
                 <v> == no.
ADJ:             <> == NV
                 <n> == yes
                 <v> == yes.

BAR:             <> == MAJOR
                 <majpaths> == (  bar ':' "<bar>" ).
BAR0:            <> == BAR
                 <bar> == zero.
```

This DATR theory makes it possible to get the feature structure associated with the node *NOUN*, i.e. the evaluation of the DATR query *NOUN:<>*.

To evaluate this DATR query the nodes *NV*, *MAJOR*, and *SYNTAX* are visited. In the node *SYNTAX* part of the corresponding feature specification is constructed and the evaluable path *<synpaths>* refers back to the original node *NOUN*. Then the query *NOUN: <synpaths>* is evaluated in the same way up to the node *MAJOR*, where the next part of the feature structure is built and the evaluable path *<majpaths>* refers again to the global node *NOUN*. At the end of the evaluation the feature structure *[syn:[maj:[nv: [n:yes,v:no]]]]* emerges.

Lexical entries defined with the DATR network above have the form *FROG: <> == ("NOUN" "BAR0")*, which means intuitively that the lexeme *frog* is an *n0*. Given the network in (4), the value of the query *FROG:<>* will inherit the information of the global nodes *NOUN* and *BAR0*. Thus, the global environment is changed in the course of the evaluation.

As a declarative language, DATR is independent of the procedural evaluation strategies embodied in particular DATR-implementations. Nevertheless, DATR theories like (4) may themselves reflect different evaluation strategies (just as different search strategies may be implemented in pure PROLOG, independently of the particular PROLOG implementation).

The evaluation strategy in (4) can be described as *top-down depth-first* and is rather costly because of the cyclic returns to the global nodes. A more efficient strategy is illustrated in (5). This DATR description embodies a *breadth-first* search and uses variables (designated by the prefix *$*) instead of changing the global environment.

```
(5)
SYNTAX:          <$NV $BAR> ==
       ( [ syn ':' [ MAJOR:<$NV $BAR> ] ] ).

MAJOR:           <$NV $BAR> ==
       ( maj ':' [ NV:<$NV> , BAR:<$BAR> ] ).

NV:              <$NV> ==
       ( nv ':' [ N:<$NV> , V:<$NV> ] ).
N:               <$NV> == ( n ':' N_VAL:<$NV> ).
V:               <$NV> == ( v ':' V_VAL:<$NV> ).
N_VAL:           <noun> == yes
                 <adj> == yes
                 <> == no.
V_VAL:           <verb> == yes
                 <adj> == yes
                 <> == no.

BAR:             <$BAR> ==
       ( bar ':' BAR_VAL:<$BAR> ).
BAR_VAL:         <bar0> == zero
                 <bar1> == one
                 <bar2> == two.
```

Here an appropriate query would be *SYNTAX: <noun bar0>*. At the origin of the query the outer layer of the feature structure is already constructed. The rest of the feature structure results from evaluating *MAJOR:<$NV $BAR>*, where *$NV* is instantiated with *noun* and *$BAR* with *bar0* as in the original query.

We then obtain the feature structure *[syn:[maj:[nv:[n:yes,v:no],bar:zero]]]* as the result of the evaluation. Unlike the network in (4), it is not possible to ask for just a part of this feature structure: Neither the information about the N/V-scheme nor the information about the bar level can be queried separately.

An entry for the lexeme *frog* given the network (5) would have the form *FROG:<> == SYNTAX:<noun bar0>*, which of course also means that the lexeme *frog* is an *n0*. But this time the information is inherited from the node *SYNTAX*, where the value provides the frame for the resulting PATR feature structure.

Apart from the differing DATR techniques employed, the resulting feature structures for a lexical entry also differ slightly. While the first is nearer to a set of PATR paths which has to be collapsed into a single feature structure, the second has exactly the form required by the PATR system we use.

The third technique is illustrated in (6).

```
(6)
SYNTAX:       <> == ( syn ':' [ MAJOR ] ).
MAJOR:        <> == ( maj ':' [ NV , BAR ] ).
NV:           <> == ( nv ':' [ N , V ] ).
BAR:          <> == ( bar ':' "<bar>" ).
N:            <> == ( n ':' <value "<cat>"> )
              <value n0> == yes
              <value adj0> == yes
              <value> == no.
V:            <> == ( v ':' <value "<cat>"> )
              <value v0> == yes
              <value adj0> == yes
              <value> == no.

LEXICAL:      <> == ( [ SYNTAX ] )
              <bar> == zero.
NOUN:         <> == LEXICAL
              <cat> == n0.
ADJ:          <> == LEXICAL
              <cat> == adj0.
```

An appropriate query for this DATR theory would be *NOUN:<>*, the value of which is *[syn:[maj:[nv:[n:yes,v:no],bar:zero]]]*. The evaluation of this query is similar to the one in (5) in that the value of *SYNTAX:<>* constitutes the frame of the resulting PATR-usable feature structure. Unlike (5), no variables are used; instead, information from the global node is used via global *path* inheritance to specify the values. Notice that whereas with (4) the global node is changed, it remains unchanged during the evaluations with (6).

The advantages of (6) are obvious. Since neither variables nor global nodes are used, fewer DATR facilities are involved. Nevertheless, the required PATR feature structures can be defined. For example, the lexical entry for *frog* would be *FROG:<>==NOUN*, where the noun-specific information is inherited from *NOUN*.

This third approach forms the base for our current lexicon. Some of the related issues are raised in the next section.

5 THE DATR LEXICON

It has been shown above that DATR theories can serve as a lexicon for a PATR system where the lexemes are represented as DATR nodes and the returned values of queries are the corresponding feature structures. In a lexicon which is formulated as in (6), apart from the lexical nodes (i.e. nodes like *FROG* which define lexemes) two other kinds of nodes can be distinguished: nodes like *SYNTAX* or *NV*, which correspond to PATR attributes, and nodes like *NOUN* or *LEXICAL*, which represent a kind of type information (see Pollard/Sag, 1987). The lexemes inherit this information through reference to the type nodes, while the lexeme-specific information is associated directly with the lexical nodes.

There are several differences between these three kinds of nodes. Whereas it is appropriate to pose a query like *FROG:<>* or *NOUN:<>*, such queries make no sense for nodes like *SYNTAX*. In this respect lexemes and types are related.

Another property distinguishes lexical nodes from type nodes. The latter are hierarchically structured, while the former are unstructured in the sense that they refer to types but not to other lexemes. The structuring of the type nodes reflects the above mentioned regularities as well as irregularities.

The following DATR theory is a lexicon fragment for a possible classification of intransitive verbs in German. Regular verbs (e.g. *schlafen* 'sleep') take a nominative subject and inherit all type-specific information from the node *INTRANS_VERB*. One exception are verbs with expletive subject (e.g. *regnen* 'rain'), another those with nonnominative (accusative or dative) subject (e.g. *dürsten* 'suffer from thirst' with accusative). These verbs refer to the types nodes *INTRANS_VERB_EXPL* and *INTRANS_VERB_ACC*, respectively. The latter types inherit from the node *INTRANS_VERB* but override some of its information.

(7)

```
INTRANS_VERB:          <> == VERB
                       <cat subject> == n2
                       <case subject> == nominative
                       <status subject> == norm.
INTRANS_VERB_EXPL:     <> == INTRANS_VERB
                       <status subject> == expletive.
INTRANS_VERB_ACC:      <> == INTRANS_VERB
                       <case subject> == accusative.
```

6 CONCLUDING REMARKS

We have seen that it is possible to formulate the lexicon of a PATR system as a DATR theory. That is, given a lexical entry in DATR, a corresponding feature structure can be derived. A system postulating new entries for unknown words on the basis of contextual information during parsing (Kilbury, 1990) must be able to convert a given feature structure into a corresponding lexical entry in DATR so that the new lexeme is located and integrated in the lexical network. To solve this problem the concept of type nodes can be exploited.

A final difficulty involves certain PATR-specific devices like disjunctions and reentrancies for which no obvious DATR facilities are available. At present we still have only *ad hoc* solutions to these problems.

FOOTNOTES

```
1. NOUN:      <> == LEXICAL
              <syn major nv n> == yes.
abbreviates
  NOUN:       <> == LEXICAL
  NOUN:       <syn major nv n> == yes.
```

2. The colons in single quotes, the commas, and the square brackets are DATR atoms, not part of the language itself.In contrast, the parentheses of DATR enclose a *list value*.

ACKNOWLEDGEMENTS

The research project SIMLEX is supported by the DFG under grant number Ki 374/1. The authors are indebted to the participants of the Workshop on Inheritance, Tilburg 1990.

REFERENCES

Daelemans, Walter / Gazdar, Gerald (eds.) (1990) *Proc. of the Workshop on Inheritance in Natural Language Processing.* ITK Tilburg, The Netherlands.

Evans, Roger / Gazdar, Gerald (1989a) Inference in DATR. In *Proc. of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, 66-71.

Evans, Roger / Gazdar, Gerald (1989b) The Semantics of DATR. In A. Cohn (ed.) *AISB89, Proc. of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, 79-87. London: Pitman.

Evans, Roger / Gazdar, Gerald (eds.) (1990) *The DATR Papers: February 1990* (= *Cognitive Science Research Paper* 139). School of Cognitive and Computing Sciences, University of Sussex, Brighton, England.

Gazdar, Gerald (1987) Linguistic application of default inheritance mechanisms. In Peter J. Whitelock et al. (eds.) *Linguistic Theory and Computer Applications*, 37-67. London: Academic Press.

Kilbury, James (1990) Simulation of Lexical Acquisition. In *Proc. of ALLC-ACH 90: The New Medium*, 129-130. University of Siegen, FRG.

Pollard, Carl / Sag, Ivan (1987) *Information-Based Syntax and Semantics*, I: *Fundamentals*. Stanford, Calif.: CSLI.

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