gefördert vom
Bundesministerium
für Forschung
und Technologie

Verbmobil
Verbundvorhaben

Generalized DATR for flexible lexical access: PROLOG specification

Dafydd Gibbon

UBI, Postfach 10 01 31, 33501 Bielefeld

Vm **Report 2**
October 1993

October 1993

Dafydd Gibbon

Universität Bielefeld (UBI)
Fakultät für Linguistik und Literaturwissenschaft
Universitätsstr. 25
Postfach 10 01 31
33501 Bielefeld

Tel.: (0521) 106 - 3510
e-mail: gibbon@asl.uni-bielefeld.de

Gehört zum Antragsabschnitt: 5.3 Morphologie

Das diesem Bericht zugrundeliegende Forschungsvorhaben wurde mit Mitteln des Bundesministers für Forschung und Technologie unter dem Förderkennzeichen 01 IV 101 B 2 gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei dem Autor.

Generalised DATR for flexible lexical access: Prolog specification

VERBMOBIL-Teilprojekt 5.3, deliverable D1

Dafydd Gibbon

October 1993

Prof. Dr. Dafydd Gibbon

Fakultät für Linguistik
und Literaturwissenschaft
Universität Bielefeld

P 10 01 31
D-33501 Bielefeld

Tel.: **+49.521.106.3510**
Fax: **+49.521.106.6008**
Email: **gibbon@asl.uni-bielefeld.de**

Contents:

Abstract 3

1 Goals 4

2 Integrated lexical knowledge representation 6
2.1 Types of lexical knowledge 6
2.2 Basic principles and architecture of integrated lexicon theory 8
2.3 A microlexicon illustration of integrated lexical knowledge representation 12

3 DATR syntax and semantics 16
3.1 The original DATR BNF definition 16
3.2 An updated BNF syntax for DATR 17
3.3 Pure Prolog data structures as a model for DATR theories 18
3.4 Aspects of DATR semantics: other models for DATR theories 21

4 Query languages for integrated lexicon theories in DATR 23
4.1 The basic DATR Query Language: DQL 23
4.2 Pure Prolog semantics for DQL: "MINIDATR" 25
4.3 Augmented DQL: ADQL 29
4.4 A new Extended DATR Query Language: EDQL 30

5 The Open DATR Engine: ODE 34
5.1 The ODEC compiler and the ODEQ inference engine 34
5.2 Application to an integrated lexicon 35

6 Conclusion 39

Acknowledgements 40

References 41

Appendix 44
MINIDATR: A minimal core DATR engine in Prolog

Abstract

This report discusses criteria for a computational lexicon with flexible access strategies, and specifies a new generalised, constraint-oriented query language for DATR lexica (EDQL) which meets these requirements. A specification of EDQL semantics is given in Pure Prolog, and an implementation based on this specification, the Open DATR Engine (ODE), is described. The implementation is in Sicstus Prolog. The approach to integrated lexica is illustrated with a 'microlexicon' containing elementary semantic and surface lexical information. The ODE concept is intended as a general specification for spoken language lexicon acquisition and application research in the VERBMOBIL project (Work Packages 5.1 and 5.9); it is not a specification for the VERBMOBIL lexicon system.

Dieser Bericht behandelt Kriterien für Computerlexika mit flexiblen Zugriffsstrategien und stellt eine neue, generalisierte, constraint-orientierte Anfragesprache für DATR-Lexika vor (EDQL). Die Semantik für EDQL wird in Kern-Prolog spezifiziert und eine auf dieser Spezifikation basierende Implementierung, die 'Open DATR Engine', ODE, wird beschrieben. Die Implementierungssprache ist Sicstus-Prolog. Der Ansatz im Bereich der integrierten Lexika wird anhand eines Mikrolexikons mit elementarer semantischer und Oberflächen-Information veranschaulicht. Das ODE-Modell soll eine allgemeine Spezifikation für die Akquisition und die Anwendung von Lexika im VERBMOBIL-Projekt (Arbeitspakete 5.1, 5.9) darstellen; es handelt sich nicht um eine Spezifikation des VERBMOBIL-Lexikonsystems.

Ce rapport traite des critères pour les lexiques computationelles avec les stratégies d'accès flexibles et spécifie un langage de contraints généralisé pour l'accès aux lexiques en DATR. La sémantique de ce langage est spécifié en Prolog pure et une implantation qui utilise cette spécification ('Open DATR Engine', ODE), est décrit. Le langage d'implantation est Sicstus Prolog. Cette approche aux lexiques intégrés est illustrée avec un microlexique avec les informations sémantiques et superficielles. Le concept ODE est intendé comme spécification générale pour la recherche sur l'acquisition et l'application des lexiques pour la langue parlée dans le projet VERBMOBIL (Paquets de Travail 5.1, 5.9); il ne s'agit pas d'une spécification du système lexical de VERBMOBIL.

1 Goals

In their study of the adequacy of lexical models, Daelemans & van der Linden (1992) have formulated a set of evaluation criteria for lexical representation formalisms. They distinguish between general formalisms and formalisms based on a specific task analysis, for instance with respect to the lexicon. In their task analysis of the lexicon, they distinguish in particular between three basic functions of a computational lexicon (1992:2):

(1) *Representation*. Representation of morphological, syntactic, semantic and pragmatic information in such a format that it can be easily integrated, and used with grammar, parser, and generator.

(2) *Access*. Given underspecified information, lexical signs compatible with it must be retrieved. Given a lexical sign, information associated with this sign must be retrieved. Given a lexical sign, (information associated with) related signs must be retrieved. These are all instances of a generic *classification* task.

(3) *Acquisition*. Addition of new lexical signs to the lexicon must be possible while keeping consistency of the lexicon.

These general goals overlap with traditional linguistic adequacy criteria (Chomsky 1965) as applied to lexica (Gibbon & Langer 1992): in addition to observational adequacy (a minimal requirement for recording facts) and descriptive adequacy (roughly: the representation criterion), the acquisition criterion covers part of the explanatory adequacy requirement. However, in addition to the more declarative, competence oriented traditional criteria, the access and acquisition criteria of Daelemans & van der Linden also touch on the procedural domain, which might be reconstructed in more traditional terms as observational, descriptive and explanatory adequacy for performance models.

Daelemans and van der Linden make the point that "Classification as a generic task for lexicon access suggests a combination of abstraction, matching and refinement as a problem solving method, which in its turn suggests the use of *taxonomies* for the representation of lexical knowledge." They also list *formalism-internal* criteria of evaluation, such as provision of a formal declarative semantics, formal inference rules, sound and complete inference and computationally tractable inference. They then concentrate on *formalism-external* criteria of notational adequacy and expressivity (all and only the relevant linguistic facts), and discuss five dimensions in hierarchical lexicon design, as follows.

(1) Basic relation (e.g. subsumption or inheritance)
Conclusion: Orthogonal multiple default inheritance is currently the best solution for

conflicts; unification should play a secondary role

(2) Recursive structure (e.g. reference to other lexical signs; path formation in complex feature structures)
Conclusion: Recursive structuring (path formation) is required

(3) Multiple (orthogonal and prioritised) inheritance and single inheritance
Conclusion: Integration of knowledge from multiple sources is required (multiple inheritance or subsumption)

(4) Monotonic and Non-monotonic inheritance
Conclusion: Default reasoning is required (with non-monotonic inheritance or subsumption)

(5) Explicitness in coding non-default information
Conclusion: Implicit blocking (as opposed to explicit encoding of default information) is required.

These conclusions harmonise well with experience in designing lexica for both written and spoken language using DATR as a lexicon representation language (Evans & Gazdar 1989, 1990; Andry, Fraser, McGlashan, Thornton, Youd 1992; Gibbon & Langer 1992; Langer & Gibbon 1992); whether DATR is the optimal language remains to be seen, but currently a variety of applications and general criteria of adequacy suggest that DATR provides a promising foundation for further work.

However, a major weakness of DATR lies in its lack of a general query language. Existing DATR applications either rely on standard DATR procedural semantics, which is a deterministic mapping from single (or sets of) lexical entry nodes to their semantic and phonological or orthographic representations, or they embed DATR within a larger, less well-defined programming context. This state of affairs is not entirely satisfactory with regard to the fulfilment of the goals formulated by Daelemans & van der Linden.

The aim of this report is to develop a solution to this problem by clearly distinguishing between DATR and EDQL, an extended query language for DATR, and by defining the latter as a compositional constraint language. Linguistic motivation for integrated lexica is discussed in Section 2, and a 'microlexicon' for illustration of the formal sections is introduced there (this section can be skipped by readers familiar with integrated lexicon theory and DATR applications in computational lexicology and lexicography). In Section 3, DATR syntax is defined, and an overview is given of models for DATR to be found in the literature. Section 4 treats the existing DATR Query Language DQL (standard DATR queries, Evans & Gazdar 1989, will be referred to in this way) and provides a procedural semantics for DATR in Pure Prolog, as well as an Augmented DATR Query Language ADQL (referring to Evans & Gazdar 1990; Cahill & Evans 1990); finally, EDQL (Extended DATR Query language) is introduced as a constraint-oriented generalisation of DQL and ADQL, building on an approach initiated by Langer (1992). In Section 5, an implementation ODE (Open DATR Engine), is described, with a simple application to the microlexicon introduced in Section 2.

2 Integrated lexical knowledge representation

This Section covers types of lexical knowledge to be considered in developing lexicon models for computational lexica for spoken language, a linguistically motivated lexicon architecture, and a toy 'microlexicon' for illustrating the integration of surface and semantic knowledge in a hierarchical lexicon.

2.1 Types of lexical knowledge

This is not the appropriate context for a detailed survey of the different kinds of lexical knowledge which need techniques of representation and access; this is a topic which has received detailed attention for centuries, and innumerable suggestions have been made during the past three decades of formal and computational linguistics, for instance from Katz & Postal (1963) to Kamp & Rossdeutscher (1992), to take two examples at the chronological extremes.

A number of basic kinds of lexical information need to be represented in an integrated lexicon within the context of a spoken language system, however. These are outlined below.

i. *Surface (orthography, pronunciation - segmental, syllabic, prosodic features)*. In this area, recent work on computational lexicology has included, in addition to the traditional field of written text processing (morphological analysis, including lemmatisation, orthographic lexical representation, orthographic variants), work on phonology, in particular on integrating finite state representation and processing models into the lexicon. The problems treated include underspecification theory; feature geometry hierarchies; dependency hierarchies; autosegmental association; tone sandhi; vowel harmony; tone, vowel and consonant spreading; prosody (stress and tone); this work has been closely connected to work on integrating morphology and the lexicon (cf. esp. Cahill 1993).

ii. *Morphology, as an intermediate level of paradigm variants, between lexical entries and their phonological representations*. This area has been most extensively and successfully dealt with for written language in the Two Level Morphology paradigm (Koskenniemi 1983 and many later studies). However, spoken language problems are not so easily dealt with in this paradigm. In addition to the evidently tractable areas of prefixation and suffixation, there are more difficult problems in the area of morphophonology such as infixation, stem modification; stress assignment, tone assignment in word derivation and inflexion; word-and-paradigm

variation, including conjugation and declension in various Indo-European languages; intercalation in Semitic languages, and tone in African languages (cf. Kay 1987, Gibbon 1992a; Bleiching 1991, 1992; Pampel 1992; Reinhard & Gibbon 1991). The integration of finite state representation techniques into a hierarchical lexicon framework, as 'prosodic inheritance', is demonstrated in the automata for Arabic and Kikuyu in Gibbon (1990).

iii. *Lexical semantics (atomic elements, relations and fields defined in terms of compositional frames)*. Frame-oriented models of lexical semantics (Fillmore 1971) are being developed further within logical and computational frameworks (Kamp & Rossdeutscher 1992; Pustejovsky 1991). Models of lexical fields for kinship terms, cooking recipes, collocations in economics texts, colour terms, institutional domains (e.g. personnel database); cf. Evans & Gazdar collections of DATR theories (1989, 1990) and Bleiching (1990).

iv. *Lemma type (simplex, derived and compound words, idioms)*. Phrasal idioms: Model of various degrees of 'frozenness' in idiomaticity, first modelled in detail by Weinreich (1969).

In more general terms, a lexicon model, representation formalism and architecture are required to express the following central properties of a lexicon which are applicable to each of these areas:

(1) The distinction between "actual" and "potential" words.

(2) Notions such as exception, markedness, lexical blocking, elsewhere conditions in morphology and phonology.

(3) Integration of morphology into the lexicon as a set of lexical generalisations.

(4) Integration of lexical semantics into the lexicon as a set of lexical generalisations.

(5) Integration of the notion of phrasal idiom into a general notion of lexical entries of different ranks (sizes, granularities).

(6) Represention of an integrated lexicon with the properties described in (1) to (5) as a structured object in its own right (such as a default inheritance network, or a type subsumption hierarchy).

(7) Definition of lexical access with flexible combinations of lexical objects, their attributes and their values as keys, and with unification-like complex constraint operations over the information generated.

These areas have been extensively discussed in the literature already cited.

2.2 Basic principles and architecture of integrated lexicon theory

In this Subsection, basic considerations for the architecture of an integrated lexicon are presented in the form of a set of postulates for Integrated Lexicon Theory (ILT), with an interpretation in terms of DATR constructs. The architecture of an ILT lexicon is stratified, with strata ranked in terms of the size (granularity) of lexical entries, with each stratum characterised by three properties. The stratum ordering does not imply any procedural claims.

(1) A distinct semantic and phonetic interpretation at each stratum.

(2) A function of paradigm variation at each level (e.g. inflexional morphology for simplex words, more complex variants at higher strata), mediating between lexical entries (abstract lemmata) and their semantic and phonetic interpretations.

(3) A hierarchy of generalisations at each level, expressed (depending on modelling conventions) either as a subsumption lattice or as a default inheritance hierarchy. This notion replaces older linguistic notions of sets of unconnected redundancy rules. The generalisation hierarchy and the explicitly structured lexicon distinguish this approach from previous notions of stratification or ranking (Halliday 1966-68; Lamb 1966), and relate it to Word Grammar (Hudson 1984) and Mel'cuk & Zholkovsky (1988).

Integrated Lexicon Theory is defined by the following main postulates about lexical entries, their properties and relations; it is not possible to argue for or illustrate all of these in detail here, though some aspects are further illustrated below.

(1) An Integrated Lexicon IL is a tuple of strata (see also Figure 1 below), each defined as a tuple <L, P, S, M>. L is a set of abstract lexical signs or lemmata, P is a set of paradigm variants of L, S is a set of surface interpretations of members of P and a set M of meaning interpretations of members of P (see also Figure 2). The functions from L into P, and from P into S and M, are expressed by attribute structures.

(2) A relation of composition C is defined over members of L such that if l_i and l_j are in L, then $l_k = C(l_i,l_j)$ is in L; the relation $\{<l_k,l_i>,...,<l_k,l_j>\}$ is Immediate Dominance (ID). Additional non-binary relations may be defined; in general, the co-constituency relation C is interpreted as Dependency (e.g. Head-of). The lexical strata are special features of the relation of composition; the relation is non simply stratal in terms of stems, word, etc., but is also recursively defined.

(3) A set of equivalence relations F is defined over L, representing lexical fields of similar abstract lexical signs. In general, the sets are defined by mutually exclusive properties of abstract lexical signs, organised into attribute-value structures. Lexical markedness relations are defined as a (partial) ordering over mutually exclusive values (a default scale). The relations induce a hierarchy of sets of abstract lexical items, expressed as

a tree or lattice (e.g. as a type hierarchy, object hierarchy, class hierarchy, inheritance hierarchy etc.).

(4) The form representations S (with subsets S_{phon}, S_{orth}) are modality dependent (written, spoken, gestural, tactile etc.); phonological surface representations are defined as prosodic or Quasi-Linear Precedence (QLP) structures by the Time Map theory of segmental and prosodic structure (Carson-Berndsen 1993).

(5) The meaning representations are derived from lexical frame semantics (Fischer 1993).

(6) The three functions mentioned in (1) define model-theoretic interpretations on to the semantic and phonetic domains (which are regarded as ontologically of the same type); the first function is realised by morphology, surface syntax, etc., depending on the stratum, the others by theories of phonetic/prosodic and semantic interpretation.

(6.1) Paradigm variation $V(L,P_{det})$ -> P_{var}. The paradigm variants are productive inflected forms, transparent productive derived forms, and the restricted variants of idiom entries.

(6.2) Phonetic interpretation I_{phon}: P_{var} -> F_{phon}. This may of course be expressed as I_{phon}: $V(L,P_{det})$ -> M.

(6.3) Semantic interpretation I_{sem}: P_{var} -> M. This may of course be expressed as I_{sem}: $V(L,P_{det})$ -> M.

In the present context, as illustrated in the microlexicon in the following Subsection, the objects described under (1) are represented by DATR nodes, attribute paths, and the construction of value sequences.

The relation (2) is represented by inheritance of the properties of constituents via constituent attributes; it is generally represented by DATR global inheritance.

The lexical fields described under (3) are represented by attribute hierarchies which define contexts of relevance for sets of properties; in typed attribute systems, these relations would be represented by appropriateness functions.

The representations (4) are generalisations and specifications of multi-tier finite state morphophonological structures. In a fully developed IL, they are constructed by DATR sequence construction and inheritance as temporally parallel or serial structures (QLP relations). The QLP relations are partially determined by the ID relations specified under (2).

The meaning representations (5), like the temporal QLP relations in phonology, are also defined by the ID attribute frames, and in a fully developed system they constitute a mapping on to logical form with quantification (Kamp & Rossdeutscher 1992; Pustejovsky 1991).

Finally, the basic operation (cf. the discussion of Daelemans & van der Linden in Section 1), of inheritance is used to model the functions which map lexical entries on to their morphological paradigm variants and phonetic interpretations, and on to their logical form and denotations.

A specific integrated lexicon theory, as a special case of the applications of the general principles described above, is further characterised by additional constraints which, after empirical testing, may be incorporated into general ILT.

For example, currently the composition relation C is defined such that a partition of L into *morphemes*, (*roots* and *affixes*), *derivates* of a number of kinds, *composites* of a number of kinds, and *idioms* is induced, with specific consequences for the other functions and relations.

There are meaningless lexical units, i.e. units with no semantic interpretation such as "cranberry morphs" (or indeed members of the phonematic segment and prosody inventories) which are outside the domain of I_{sem}. The general theory allows the option of phonetically zero units in specific theories; these would be outside the domain of I_{phon}.

The paradigm determinants include grammatical inflectional categories and productive derivational types; in addition, idioms are defined by a "frozenness" property which allows restricted syntactic variation; this is treated by analogy to paradigm variation in morphology.

Lexical fields will be very specific to given languages and language families, and vary strongly along the lines defined in universal linguistic typology, but general properties can be defined for general IL theory; similarities in I_{phon} at the relevant levels of granularity defined by C, for instance, lead to morphological types such as inflectional, agglutinative, or tonal inflexion.

For the definition of *ad hoc* formations and neologisms it is important that IL theory should define the notion of a lexical entry as an *actual* word, stem, etc., i.e. an inventarised, familiar lexical item, as against the notion of a general category containing potential words, i.e. regular formations which are not inventarised but which are induced by generalisations over lexical fields. It is also important that the notion of *transparent, compositional interpretation* (both phonological and semantic) at some granularity level defined by C is defined in terms of meaning and form as straightforward functions of the meaning and form of constituents, as opposed to a direct, partially idiosyncratic semantic or phonetic interpretation of the item concerned.

Constraints on possible IL theories in DATR include the partition of the node set into lemma and generalisation nodes, the requirement that the inheritance graph be a connected graph such as a subsumption lattice (if defaults are not used), and the explicit definition of a restricted set of "sensible queries" to an IL (currently excluding default overrides, additional "query default garbage", queries to generalisation nodes). Constraints such as these are embedded in the microlexicon described in the following Subsection.

The lexical strata, and their interpretations and generalisations, will not be argued here (cf. Gibbon 1981, 1985 on lexical strata, 1992a on lexical ID structure; Selkirk 1984, Bleiching 1992 for relevant aspects of the prosodic hierarchy). Figure 1 and Figure 2 describe the strata in an ILT lexicon and the detailed structure of a given stratum. For present purposes, the details are to be taken as general indications of the structure, not as a precise empirical hypothesis. Each horizontal stratum and each vertical hierarchy has its own constraints, which complicate the mapping from lexical items to their interpretations. In the left hand column of the figure, only lexical items are listed; more general compositional items (words, phrases, sentences, etc.), are also to be envisaged; from the lexical perspective, they are generalisations over complex lexical items.

Lexical ID structure: compositional hierarchy	Temporal QLP structure: prosodic hierarchy	Logical form: semantic hierarchy
ritual dialogue	discourse prosody	Situation semantics
canned text, proverbs	paratone structure	DRT, propositional
phrasal idioms	intonation phrases	semantics
compound words	foot structure	lexical semantics
derived stems	phonological words	
simplex stems (roots)	(morpho-)phonotactics	
affixes	morphological prosody	
morphophonemes	feature tiers	

Figure 1: Lexical strata and their interpretations

The internal architecture of an ILT lexicon stratum is shown in Figure 2; the notions are motivated further in Gibbon (1992b). The horizontal dimension represents the integration of representations into the 'real world' of model-theoretic semantic and phonetic interpretations in 'absolute' time and space; the intermediate representations in 'relative' time (for phonetics) represent the abstract functions (logical and phonological form) which map entries on to the real world. The notion of lexical category (interpretable as a knowledge base, or as long-term storage) is atemporal.

The vertical dimension of Figure 2 represents the rank or granularity of lexical entries, from simplex lemmata through derived and compound lemmata, to idioms (phrasal lemmata). The paradigm variants of lemmata are an intermediate mapping on to morphological variants (morphophonology).

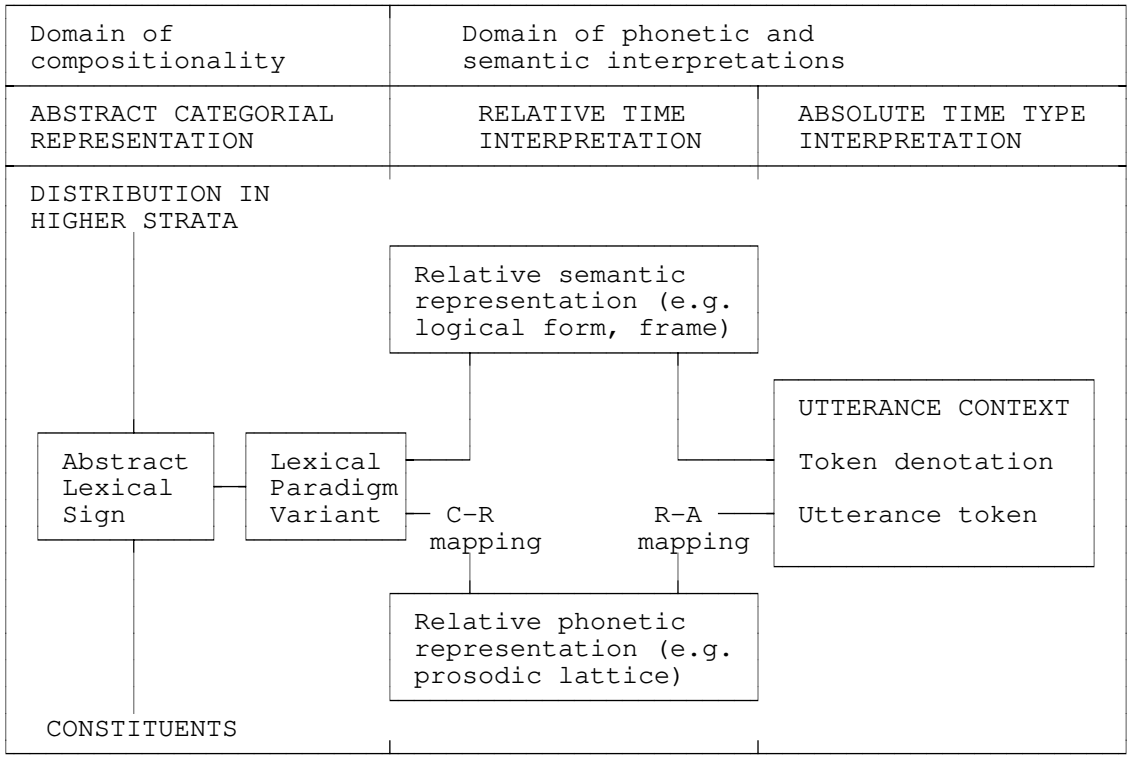

Figure 2: Internal architecture of a lexical stratum.

2.3 A microlexicon illustration of integrated lexical knowledge representation

This elementary microlexicon illustrates a minimal application of Integrated Lexicon Theory, covering a very rudimentary compositional lexical semantics and a very rudimentary compositional orthography and phonology. Linguistically adequate applications have been referred to in Section 1 above; a simple theory is a adequate for present purposes. In addition to their descriptive lexical properties, nodes are specified for metatheoretical formal properties within Integrated Lexicon Theory, in particular as lemmata or generalisations (in a dependency model they can be both at the same, for which reason these properties are specified as distinct Boolean attributes rather than alternative values of the same attribute). The microlexion is described in the following.

DATR microlexicon following ILT conventions

(1) *Abstract lemma nodes*. An abstract lemma is specified only for its distinctive properties; all transparent or predictable (shared) properties are defined at generalisation nodes. No priority is accorded to surface or semantic features: the abstract lemma is simply an object with properties of both types.

(1.1) *Compound abstract lemma*. A compound lexical entry is specified for its constituents and other distinctive properties; to the extent that it is transparent (i.e. compositional), it inherits its surface and semantic properties as a function (defined at the node Compound) of its constituents (whose properties are defined individually). A transparent compound lemma is thus 'underspecified' in the extreme: effectively, it consists of a small set of pointers to other objects in the lexicon. The constituents are represented as DATR global inheritance descriptors, because their properties are in the general case totally independent of their context, and inheritance of their properties can thus be represented as a totally modular embedded query.

Tablecloth:

```
<>                  == Compound
<ilex lemma>        == yes
<relation sem>      == for covering
<modifier>          == "Table:<>"
<head>              == "Cloth:<>".
```

(1.2) *Simplex abstract lemmata*. The simplex lemmata inherit all but their distinctive properties from the generalisation hierarchy, in this case the node Simplex. Their meaning is represented in rudimentary 'genus proximum & differentia specifica' dictionary form. Surface properties are either orthographic, represented in the obvious way, or phonological, represented in SAMPA computer readable phonemic notation (Wells 1989). The prosodic feature of regular, recursively defined stress value (i.e. normal, default accent position weighted by depth of embedding) is represented compositionally at the node Compound by an asterisk; the number of asterisks directly represent the stress value (the more asterisks, the greater the stress weighting).

Table:

```
<>                  == Simplex
<ilex lemma>        == yes
<root sem>          == horizontal surface to put things on
<root surf orth>    == t a b l e
<root surf phon>    == t 'eI' b l.
```

Cloth:

```
<>                          == Simplex
<ilex lemma>                == yes
<root sem>                  == variety of textile
<root surf orth>            == c l o t h
<root surf phon>            == k l 'O' 'T'.
```

(2) *Generalisation hierarchy*. This elementary hierarchy defines the properties inherited by the lemma objects in various lexical fields (or contexts). At the node Simplex, the semantic and surface properties of simplex lemmata are defined as being specified as the semantic and surface properties of their roots (inflexional morphology is not treated in this microlexicon). The semantic and surface properties of complex lemmata are defined compositionally (and, for multiple compounds, recursively) as a function of the semantic and surface properties of their modifier and head constituents. The top node in the hierarchy, the node Word, is only specified for formal properties of ILT, with the empty value sequence as a general default value (an elegant solution, but normally avoided for procedural reasons in practical lexica); otherwise the node has no descriptive function in this microlexicon.

Compound:

```
<>                          == Word
<ilex generalisation>       == yes
<ilex type>                 == compound
<surf>                      == "<modifier surf>" "<head surf>"
<surf phon>                 == * "<modifier surf phon>" "<head surf phon>"
<sem>                       ==
                    "<head sem>" "<relation sem>" "<modifier sem>".
```

Simplex:

```
<>                          == Word
<ilex generalisation>       == yes
<ilex type>                 == simplex
<surf>                      == "<root surf>"
<sem>                       == "<root sem>".
```

Word:

```
<>                          ==
<ilex generalisation>       == yes
<ilex type>                 == word.
```

Some theorems (DATR extensional sentences) which may be derived from this theory by standard DATR inference are shown below.

Tablecloth:<relation sem> = for covering.

Tablecloth:<sem> = variety of textile for covering horizontal surface to put things on.

Tablecloth:<surf orth> = t a b l e c l o t h.

Tablecloth:<surf phon> = * t eI b l k l O T.

Table:<surf orth> = t a b l e.

Table:<relation sem> = .

3 DATR syntax and semantics

The syntax for DATR was originally defined by Evans & Gazdar (1989); it is commented on in the following subsection. Then a generalised definition will be given, followed by a Pure Prolog data structure for DATR. Finally, an overview of models for DATR theories is given, which may be seen as starting points for declarative semantics for DATR.

3.1 The original DATR BNF definition

The original BNF definition for DATR contains the following rules (Evans & Gazdar 1989:1f.).

```
<sentence>   ::= <node> : <path> == <lvalue> .
                   | <node> : <path> = <value> .

<lvalue>     ::= <latom> | ( <lseq> )
<gvalue>     ::= <gatom> | ( <gseq> )
<value>      ::= <atom> | ( [seq] )

<latom>      ::= <desc> | <gatom>
<gatom>      ::= " <desc> " | <atom>
<desc>       ::= <node> | <path> | <node> : <path>

<lseq>       ::= <gseq> | <lseq> <desc> <lseq>
<gseq>       ::= <seq> | <gseq> " <desc> " <gseq>
<seq>        ::= epsilon | <value> <seq>

<lpath>      ::= '<' <laseq> '>'

<path>       ::= '<' <aseq> '>'

<laseq>      ::= epsilon | <latom> <laseq>
<aseq>       ::= epsilon | <atom> <aseq>
```

This formulation is overly complex. The variable <gvalue>, which does not figure on a rhs, is present for semantic reasons, and can be dispensed with in the syntax. A modified BNF will be dicussed below.

3.2 An updated BNF syntax for DATR

A number of generalisations have been introduced into DATR syntax on the basis of considerable experience within a fairly large community of DATR users, both individual researchers and European and nationally funded research projects.

The following definition builds on these experiences. It introduces a simple definition of DATR theory, and replaces the original distinction between definitional and extensional sentences with a distinction between the definitional sentences in DATR theories and the extensional sentences in a query language for DATR. Three generalisations pertaining to the right hand sides of DATR equations are introduced:

(1) All DATR rhs are sequences (the old distinction between atomic rhs and sequence rhs is dropped). The modification is due to Gibbon (1989); it leads to a simpler syntax.

(2) All DATR rhs are without parentheses; however, parentheses may optionally be used. This innovation is due to Gazdar (cf. Moser 199a,b). The optionality preserves downward compatibility with older DATR theories.

(3) Evaluable paths in DATR rhs are evaluated in exactly the same way as DATR rhs. This is a recursive principle which permits rhs paths to inherit atom sequences as subpaths, and enables function application to be represented in DATR. This modification is due to Gibbon 1989 in the theory "register.dtr" (in Evans & Gazdar 1990). The device was extended in Moser's formal work on DATR (1992a,b).

The notation given below includes character definitions in an extended BNF which includes set notation, for brevity.

Theories, sentences, sentence constituents:

```
<theory>        ::= <sentence> [<sentence>]*
<sentence>      ::= <node> : [<equation>]*
<equation>      ::= <lhs> == <rhs>
<lhs>           ::= "<" [<atom>]* ">"
<rhs>           ::= [<val_exp>]* | ( [<val_exp>]* )
```

Inheritance expressions:

```
<val_exp>       ::= atom | <descriptor> | " <descriptor> "
<descriptor>    ::= <node> : <path> | <node> | <path>
<path>          ::= "<" [<val_exp>]* ">"
<atom>          ::= <atom_char> [<symbol_char>]* | ' [<char>]* ' | <variable>
<variable>      ::= $ [<symbol_char>]*
<node>          ::= <node_char> [<symbol_char>]*
```

Characters:

<res_char> ::= : | < | > | = | (|) | " | . | ’ | \
<char> ::= char(0) | ... | char(127)
<print_char> ::= char(33) | ... | char(127) | \ <char>
<symbol_char> ::= {x | x in <print_char>} - {y | y in <res_char>}
<node_char> ::= A | ... | Z
<atom_char> ::= {x | x in <symbol_char>} - {y | y in <n_char>}

The BNF definition is primarily intended to serve as a basis for a DATR compiler. An efficient compiler will use (among other things) a modified BNF which maximises right recursion and uses Greibach Normal Form optimisation. However, a structurally transparent, if not speed optimised Prolog compiler using this BNF can easily be implemented with a Definite Clause Grammar; this technique was used in the Sussex DATR implementations by Evans (Evans & Gazdar 1989, 1990) and in the ODE implementation (Section 5 below).

3.3 Pure Prolog data structures as a model for DATR theories

The Prolog data structures for DATR theories are straightforward. Each equation is translated separately, together with its node and the name of the theory in which it is contained, as a four-argument Prolog fact, under the predicate ’datr_sentence’. The argument positions are instantiated as the name of the theory (a Prolog atom), the name of the node (also a Prolog atom), the lhs path, represented as a Prolog list, and the rhs sequence, also represented as a Prolog list.

The rhs elements are translated as follows: atoms are represented as Prolog atoms, and inheritance descriptors are represented as Prolog lists, with an atomic tag for the descriptor type in first position, and nodes and paths at subsequent positions, depending on the descriptor concerned. Internally, the recursively evaluable rhs paths have exactly the same syntax and translation function as the rhs itself. The translation function from DATR to Prolog may be sketched as follows.

<sentence> ==> datr_node(<theory>,<node>,<lhs>,<rhs>).

<lhs> ==> Prolog list of atoms, perhaps empty,
e.g. [], [a], [attribute,list]

<rhs> ==> Prolog list of <descriptor> expressions, perhaps empty,
e.g. [], [a], [aa,bb], etc.

<descriptor> ==> expression of one of the following types,
corresponding to each of the 7 DATR inference rules as a
Prolog atom or a tagged list:

(1) <atom>
(2) [lnp,<node>,<path>] for local node-path
(3) [ln,<node>] for local node
(4) [lp,<path>] for local path
(5) [gnp,<node>,<path>] for global node-path
(6) [gn,<node>] for global node
(7) [gp,<path>] for global path

<node> == <atom>

<atom> ==> Prolog atomic symbol

<path> == <rhs>

Detailed illustrations of the DATR to Prolog translation function, using the microlexicon of Section 2, are shown below.

Tablecloth:

```
<>                  == Compound
<ilex lemma>        == yes
<relation sem>      == for covering
<modifier>          == "Table:<>"
<head>              == "Cloth:<>".
```

```
datr_sentence(microlex,'Tablecloth',[ilex,lemma],[yes]).
datr_sentence(microlex,'Tablecloth',[relation,sem],[for,covering]).
datr_sentence(microlex,'Tablecloth',[modifier],[[gnp,'Table',[]]]).
datr_sentence(microlex,'Tablecloth',[head],[[gnp,'Cloth',[]]]).
datr_sentence(microlex,'Tablecloth',[],[[ln,'Compound']]).
```

Table:

```
<>                  == Simplex
<ilex lemma>        == yes
<root sem>          == horizontal surface to put things on
<root surf orth>    == t a b l e
<root surf phon>    == t 'eI' b l.
```

```
datr_sentence(microlex,'Table',[ilex,lemma],[yes]).
datr_sentence(microlex,'Table',[root,sem],[horizontal,surface,to,put,things,on]).
datr_sentence(microlex,'Table',[root,surf,orth],[t,a,b,l,e]).
datr_sentence(microlex,'Table',[root,surf,phon],[t,'eI',b,l]).
datr_sentence(microlex,'Table',[],[[ln,'Simplex']]).
```

```
Cloth:
        <>                          == Simplex
        <ilex lemma>                == yes
        <root sem>                  == variety of textile
        <root surf orth>            == c l o t h
        <root surf phon>            == k l 'O' 'T'.

datr_sentence(microlex,'Cloth',[ilex,lemma],[yes]).
datr_sentence(microlex,'Cloth',[root,sem],[variety,of,textile]).
datr_sentence(microlex,'Cloth',[root,surf,orth],[c,l,o,t,h]).
datr_sentence(microlex,'Cloth',[root,surf,phon],[k,l,'O','T']).
datr_sentence(microlex,'Cloth',[],[[ln,'Simplex']]).


Compound:
        <>                          == Word
        <ilex generalisation>       == yes
        <ilex type>                 == compound
        <surf>                      == "<modifier surf>" "<head surf>"
        <surf phon>                 == * "<modifier surf phon>" "<head surf phon>"
        <sem>                       ==
                            "<head sem>" "<relation sem>" "<modifier sem>".

datr_sentence(microlex,'Compound',[ilex,generalisation],[yes]).
datr_sentence(microlex,'Compound',[ilex,type],[compound]).
datr_sentence(microlex,'Compound',[sem],
        [[gp,[head,sem]],[gp,[relation,sem]],[gp,[modifier,sem]]]).
datr_sentence(microlex,'Compound',[surf,phon],
        ['*',[gp,[modifier,surf,phon]],[gp,[head,surf,phon]]]).
datr_sentence(microlex,'Compound',[surf],[[gp,[modifier,surf]],[gp,[head,surf]]]).
datr_sentence(microlex,'Compound',[],[[ln,'Word']]).


Simplex:
        <>                          == Word
        <ilex generalisation>       == yes
        <ilex type>                 == simplex
        <surf>                      == "<root surf>"
        <sem>                       == "<root sem>".

datr_sentence(microlex,'Simplex',[ilex,generalisation],[yes]).
datr_sentence(microlex,'Simplex',[ilex,type],[simplex]).
datr_sentence(microlex,'Simplex',[surf],[[gp,[root,surf]]]).
datr_sentence(microlex,'Simplex',[sem],[[gp,[root,sem]]]).
datr_sentence(microlex,'Simplex',[],[[ln,'Word']]).
```

```
Word:
        <>                          ==
        <ilex generalisation>       == yes
        <ilex type>                 == word.

datr_sentence(microlex,'Word',[ilex,generalisation],[yes]).
datr_sentence(microlex,'Word',[ilex,type],[word]).
datr_sentence(microlex,'Word',[],[]).
```

A theory-specific query function for DATR queries may be defined:

```
microlex(Node,Path,Value) :- datr(microlex,Node,Path,Value).
```

3.4 Aspects of DATR semantics: other models for DATR theories

This section is taken from Langer & Gibbon (1992). It summarises different formal models, perspectives and metaphors which have been used as the basis of modelling conventions for DATR, each of which has its own specific heuristic value for a given domain-specific modelling task.

(1) In Evans & Gazdar (1989) DATR is given a procedural semantics in terms of *inference systems*, involving concepts such as 'theorem', 'rule of inference'.

(2) Evans & Gazdar (1989) define an explicit *denotational semantics* for DATR. The denotational semantics of a subset of DATR is defined with respect to *automata theory*. On this basis, Evans and Gazdar construct a finite state transducer model that mimics the structure and function of a DATR theory; the transducer interpretation of DATR has been used to model linear phenomena in phonology, morphology-phonology mappings, and access relations for bottom-up access to ILEX lexica.

(3) The *operational semantics* for the DATR-to-PROLOG compiler code published in Evans & Gazdar (1989, 1990) is based on their procedural semantics, and the object code is based on a generalisation of the automata theoretic model applied to Prolog.

(4) The use of DATR to express *function composition* was introduced in Gibbon (1989) and has been elaborated by Moser (1992a).

(5) Langer (1992) provides a formal reconstruction of a subset of DATR as an instance of a more general constraint-based framework. Under this interpretation DATR is regarded as a system of *partial functions* mapping (typed) feature structures onto (typed) feature structures using a version of default unification as the only operation.

(6) DATR can be used as a *frame language* with defaults.

(7) The most frequent metaphor applied to DATR views it as a special form of traditional *inheritance network*, with or without defaults.

(8) DATR can be regarded as a *graph description language* under a specific interpretation for directed cyclic and acyclic graphs.

(9) *Reconstructions of known formal languages and systems* in DATR include: Boolean and many-valued logics, automata and transducers, formal languages up to deterministic indexed languages, shift-register operations, simulation of shift-reduce and 'non-deterministic' top-down search with backtracking, multiple inheritance systems (see the collection of formal DATR theories in Evans & Gazdar 1989, 1990; also Moser 1992a).

(10) DATR has been used to provide a representation and interpretation for autosegmental phonological and morphological graphs (Gibbon 1990; Cahill 1993)

(11) DATR has been used as a compact *lexical metalanguage* for defining PATR-II syntactic feature structures as lexical information (Kilbury et al. 1991)

(12) DATR has been most widely applied to the lexical representation of *feature structures* in path notation, using inheritance hierarchies as implicit type definitions (Reinhard & Gibbon 1991; Gibbon 1992a), or with explicit definition of *lexical type constraints* (Moser 1992b).

These results suggest that existing linguistic and logical theories of the lexicon, whether in phonology, morphology, syntax, semantics or pragmatics, can *a fortiori* be adequately modelled in the DATR formalism. This claim will not be pursued further at this point, however.

4 Query languages for integrated lexion theories in DATR

There is no formal definition for the standard DATR query language and standard DATR queries are somewhat implementation dependent. There are two main types of query, the basic query and the "theorem dump" query; the original Evans & Gazdar basic query language has a special status as the core of all DATR inference, and will be referred to simply as DQL, DATR Query Language.

4.1 The basic DATR Query Language: DQL

The basic DATR query language, which will be called DQL, defines an elementary query as a node-path pair (called a "root" by Evans & Gazdar 1989). For instance, for the specimen theorems given for the microlexicon in the preceding section, the queries are the following:

Tablecloth:<relation sem>
Tablecloth:<sem>
Tablecloth:<surf orth>
Tablecloth:<surf phon>
Table:<surf orth>
Table:<relation sem>

The following is an extended BNF syntax for DQL.

<query> ::= <node> : "<" [<atom>]* ">"

<node> ::= <uc_char> [<non_res_char>]*

<atom> ::= <a_char> [<non_res_char>]* | ' [<nq_char>]* '

<uc_char> ::= A | ... | Z

The definitions of <a_char> and <non_res_char> are somewhat implementation specific; <nq_char> is any character except the single quote. Some implementations use a backslash escape character to permit atoms to start with capitals or reserved characters to occur in nodes and atoms. Atom and node delimiters are white space characters (which are not represented here), or reserved symbols.

Standard DATR inference semantics specifies the following operations:

i. A theory, a query, and two evaluation environment variables. The variables range over roots, i.e. pairs of a node and an atom path, and constitute the global or initial environment, and the local environment.

ii. Initialisation of both environments to the same root.

iii. Requirement that the local query root *connect* with a sentence (a node-equation pair) in the DATR theory such that:

- the sentence node and the local root node are identical;

- an lhs path at this node is a prefix of the local root path, and there is no such lhs path which is longer.

iv. Identification of the *extension suffix* of the local root query with which the connecting lhs concatenates to yield the local root query path.

v. Evaluation of the sequence of rhs expressions (atoms or inheritance descriptors) in the connected equation.

vi. Recursive application of the *extension* rule and the seven *inference* rules to each rhs expression:

- the extension rule extends paths at all depths of embedding in the rhs by the same extension suffix, derived from the local-root-to-lhs connect relation;

- the seven inference rules apply: atoms evaluate to themselves, nodes substitute for nodes in the corresponding environments (local nodes for local nodes, global nodes for both global and local nodes - see ii above), and (extended) paths evaluate recursively to atom paths which substitute for root paths in the corresponding environments (local paths for local root paths, global paths for both global and local root paths - as with nodes).

Some implementations provide a structured trace of inference rule application (PCS-DATR, Gibbon 1989; DDATR, Gibbon, described in Gibbon & Ahoua 1991; QDATR, Kilbury, ca. 1991; the Open DATR Engine, ODE, described in this paper).

Langer (1992) has generalised the notion of a DATR query by reconstructing DATR within a general constraint formalism. The query is defined as an axiom and its consistency with the axioms of the theory, also formalised as attribute-value structures, is checked in terms of an appropriate domain selector and a default unification operator. This approach provided the starting point for the definition of the constraint-based Extended DATR Query Language EDQL (Section 4.4 below).

4.2 Pure Prolog semantics for DQL: "MINIDATR"

The formal procedural semantics for DATR is generally formulated in terms of general substitution rules for inheritance descriptors in the two (global and local) DATR evaluation environments. However, previous descriptions have not been fully explicit in this respect; one consequence of this has been that implementations have not always been correct.

Rather than complete existing descriptions of DATR formal semantics, a different approach is taken here: the inference procedure described above is formulated in Pure Prolog, augmented with two additional assumptions:

i. Equations in the DATR knowledge base are pre-sorted into the required 'longest path first' order; this reduces the *connect* operation over the knowledge base to a simple linear search for the first match.

ii. The "cut" operator is used to terminate linear search after the first match is found and, since rules are exhaustively defined for all cases of DATR inheritance descriptors, also to terminate search for further rule applications when one application has been found.

These are not necessary conditions, but they allow attention to be focussed on the more central issue of operations over DATR evaluation environments. The inference procedure is defined in the following way.

(1) *Definition of DATR query structure*. A query is represented by an operation 'datr' on a quadruple: a theory, and three terms representing a DATR extensional sentence with a node, an atomic path, and an unspecified value. Given this structure, the connect operation is performed, with the global and local environments instantiated to the query node and path. The query is interpreted as a constraint on the inference closure of the DATR theory.

```
datr(Theory,Node,Path,Value) :-
 atomic(Theory),
 atomic(Node),
 nonvar(Path),
 atompath(Path),
 datr_connect(Theory,Node,Path,Node,Path,Value).

atompath([]) :- !.
atompath([First|Rest]) :-
 atomic(First),
 atompath(Rest).
```

(2) *Connect operation*. The connect operation on query and theory is represented by the 'datr_connect' predicate. The theory is defined by a four-place predicate 'datr_sentence', the first argument of which is the theory name; the other three arguments represent the node, lhs and rhs of DATR equations. Linear search through the theory for an appropriate node-path pair is performed by the 'append' predicate with Prefix instantiated from the theory lhs and the Result instantiated from the local environment path; a successful append operation not only defines the connect operation, but also yields the extension suffix for DATR inheritance; a cut is used to ensure that this is a unique choice. The rhs of the connected DATR equation is then evaluated.

```
datr_connect(Theory,Gnode,Gpath,Lnode,Lpath,Value):-
 datr_sentence(Theory,Lnode,Prefix,Rhs),
 append(Prefix,Suffix,Lpath),!,
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Rhs,Value).
```

(3) *Equation rhs evaluation*. The rhs of a DATR equation is a flat sequence (possibly empty or singleton) of value expressions, either atomic or inheritance descriptors, which evaluates to a flat sequence of atoms. Standard tail-recursive evaluation is used here for evaluating the value expressions one by one, and the 'append' operation is used as a constructor for concatenating the results of individual evaluations. The first clause evaluates the empty sequence (the base of the recursive definition) to an empty sequence, and the second evaluates the first value expression of a non-empty sequence, and concatenates the result of evaluation to the result of evaluating the rest of the sequence. Evaluation is performed by the seven DATR inference rules.

```
datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[],[]).
datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[First|Rest],Value) :-
 datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,First,First_value),
 append(First_value,Rest_value,Value),!,
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Rest,Rest_value).
```

(4) *Inference rules*. There are seven inference rules in DATR recursive inference: one (the base of the recursive definition) for atoms, which evaluate to themselves; a set of three recursive inference rules for operations on the local environment; and a set of three recursive inference rules for operations on the global environment. The three types of inference rule for operations on local and global environments define substitutions in these environments; they correspond to the three kinds of inheritance descriptor: node-path pairs, nodes, and paths, where the paths are recursively evaluable in exactly the same way as the rhs of an equation. Nodes evaluate to themselves and substitute for the corresponding node, paths evaluate to atomic paths, are extended by the extension suffix given by the connect operation, and substitute for the path in the corresponding environment or environments. Local substitutions affect the local environment only, while global substitutions affect the global and the local environment. Path suffix extension is represented by the append operation.

(4.1) *DATR Inference Rule 1*. If the expression is atomic, it evaluates to itself.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Expr,[Expr]) :-
 atomic(Expr).
```

(4.2) *DATR Inference Rule 2*. If the expression is a local node-path inheritance descriptor, then evaluate the path in the current environment as if it were the rhs of an equation, extend the result, and substitute node and path in the local environment; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[lnp,Node,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Node,Extension,Value).
```

(4.3) *DATR Inference Rule 3*. If the expression is a local node inheritance descriptor, then substitute the node in the local environment; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[ln,Node],Value) :-
 append(Prefix,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Node,Extension,Value).
```

(4.4) *DATR Inference Rule 4*. If the expression is a local path inheritance descriptor, then evaluate the path in the current environment as if it were the rhs of an equation, extend the result, and substitute the path in the local environment; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[lp,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Lnode,Extension,Value).
```

(4.5) *DATR Inference Rule 5*. If the expression is a global node-path inheritance descriptor, then evaluate the path in the current environment as if it were the rhs of an equation, extend the result, and substitute node and path in the global and local environments; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gnp,Node,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Node,Extension,Node,Extension,Value).
```

(4.6) *DATR Inference Rule 6*. If the expression is a global node inheritance descriptor, then substitute the node in the global and local environments; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gn,Node],Value) :-
 append(Gpath,Suffix,Extension),
 datr_connect(Theory,Node,Extension,Node,Extension,Value).
```

(4.7) *DATR Inference Rule 7*. If the expression is a global node-path inheritance descriptor, then evaluate the path in the current environment as if it were the rhs of an equation, extend the result and substitute in the global and local environment; recursively connect the new local environment to the theory.

```
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gp,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Extension,Gnode,Extension,Value).
```

This inference engine is derived from the PCS-DATR (Gibbon 1989) and DDATR (Gibbon & Ahoua 1991) inference engines originally defined in the LISP dialect Scheme. It differs from previous Prolog implementations in that it abstracts over all DATR objects, including nodes, keeping all components of the data structure as Prolog terms, rather than asserting nodes as Prolog predicates.

This purely relational definition of DATR inference semantics in Pure Prolog permits, in an intuitively evident fashion (given Prolog inference semantics), the generalisation of DATR inference to a 'non-deterministic DATR' with disjunction (by the simple expedient of removing the cuts). This generalisation in principle defines all possible defaults and their overrides, and suggests a number of potentially powerful applications for matching in 'sloppy acquisition' and 'sloppy matching' of lexical knowledge.

The Pure Prolog definition also permits, again in an intuitively evident fashion (given Prolog inference semantics), DATR reverse queries. This claim must be taken *cum grano salis*, however: Prolog semantics will not cope with certain kinds of relatively benevolent circularities, such as left recursions, and empty rhs sequences result in an explosion of the search space. There are also circularities which would perhaps be soluble by defining a pumping lemma for DATR and applying it to a given DATR theory in order to limit the number of inference steps in a circular inference.

More efficient operations, which do not make use of the 'append' operation, can be used in practical applications; in standard DQL deterministic inference, however, this potentially inefficient operation is harmless. In a generalised reverse query search context, this would not be the case.

4.3 Augmented DQL: ADQL

In Evans & Gazdar (1990), a number of augmentations of DQL were suggested; these are somewhat heterogeneous in character, and are not properly described as a query language. They will, however, be referred to for convenience as ADQL. The main techniques are:

- question mark value variables;

- theorem dumps with "hide" and "show" declarations;

- on the fly nodes.

In ADQL, queries are no longer roots, but extensional sentences with unspecified values; unspecified values may be explicitly indicated by question marks:

Tablecloth:<relation sem> = ?
Tablecloth:<sem> = ?
Tablecloth:<surf orth> = ?
Tablecloth:<surf phon> = ?
Table:<surf orth> = ?
Table:<relation sem> = ?

The definition of EDQL given in the next Subsection takes this up, generalising the question mark notation to a notation for variables (an atomic character string prefixed by the question-mark character, as with DATR variables).

A theorem dump query is the iterative application of elementary queries to a finite set of roots defined in the following way:

i. The environment variable for the initial query root node ranges over all nodes in the theory *except* those defined in a "hide" declaration.

ii. The environment variable for the initial query root path ranges over all paths defined in a "show" declaration.

The concept of "on the fly nodes" is mentioned in passing in the 1990 Evans & Gazdar collection; these are queries in the form of a set of equations with the same node, including a standard DATR query equation. The equations constitute a context for initial queries, over and above information contained in the lexicon. In principle, this approach appears to be related to the EDQL concept outlined below.

4.4 A new extended DATR query language: EDQL

Starting from these basic ideas, a more explicit and general query language will be defined here; it will be called Extended DATR Query Language, EDQL. Sentence in the standard language, DQL, are allowed substructures of expressions of EDQL. The query language EDQL in its current specification and in the ODE implementation (see Section 5 below) has the following syntax in EBNF notation.

```
<query>          ::= <constraint> <constraints>

<constraints>    ::= . | , <query>

<constraint>     ::= <operator> ( <theory> , <sentence> )

<operator>       ::= foreach | if | ifnot | then | select | license | ban | map
                       | filemap | tracemap | filetracemap | closure | fileclosure

<theory>         ::= <atom>

<sentence>       ::= <node_exp> : <lhs_exp> = <rhs_exp>

<node_exp>       ::= <node> | <variable>

<lhs_exp>        ::= "<" <atomic_exp> ">" | <variable>

<rhs_exp>        ::= ( <atomic_exp> ) | <variable>

<atomic_exp>     ::= <atom_exp>* <tail_exp>

<atom_exp>       ::= <atom> | <variable>

<tail_exp>       ::= epsilon | <atom_exp> "|" <atom_exp>

<variable>       ::= ? <non_res_char>* | $ <non_res_char>*
```

The salient features of EDQL are as follows:

i. The elementary expressions are binary constraint operators over sets of DATR sentences: there are four main operators, "foreach", "if", "ifnot", "then", with procedurally flavoured synonyms "select", "licence", "ban", "map". The first operand is a DATR theory name (for a set of definitional sentences); the second operand a singleton set consisting of a (possibly generalised) DATR extensional sentence. A

generalised DATR sentence permits variables over node, lhs, rhs, atoms, lhs path tails and rhs sequence tails. The result of the operation is a (possibly empty) set of DATR theorems (extensional sentences).

ii. Conjunctions of elementary constraint expressions are permitted.

An example of an expression in EDQL, given the microlexicon DATR theory introduced in Section 2, called 'microlex', is the following, which defines a mapping from an orthographic representation "t a b l e" to the phonological and semantic representations for this node. The query and its response demonstrate two central tasks in speech technology applications: grapheme-phoneme mapping, and grapheme-meaning mapping. The same inference is immediately available for the phoneme-grapheme and phoneme-meaning mappings. The operations, taken together, have something of the character of a production rule system, but differ from the latter in being most plausibly interpreted in set-theoretic terms.

foreach(microlex, ?node: ?path = ?value),

if(microlex, ?node: <surf orth> = (t|tailvariable)),

if(microlex, ?node: <ilex lemma> = (yes)),

ifnot(microlex, ?node: <ilex type> = (phrasal)),

then(microlex, ?node:<surf phon> = ?value_1),

then(microlex, ?node:<sem> = ?value_2).

The procedurally flavoured version is as follows (with the dollar character borrowed from DATR variable notation).

select(microlex, $node: $path = $value),

license(microlex, $node: <surf orth> = (t|tailvariable)),

license(microlex, $node: <ilex lemma> = (yes)),

ban(microlex, $node: <ilex type> = (phrasal)),

map(microlex, $node:<surf phon> = $value_1),

map(microlex, $node:<sem> = $value_2).

Clearly, the query language can be improved by relating it more closely to first order predicate logic; the procedurally flavoured synonyms are perhaps more appropriate in this version.

The theorem set defined by this complex constraint for the 'microlex' theory is as follows.

Table:<surf phon> = t eI b l.

Table:<sem> = horizontal surface to put things on.

Tablecloth:<surf phon> = * t eI b l k l O T.

Tablecloth:<sem> = variety of textile for covering horizontal surface to put things on.

The query is a constraint on the inference (or inheritance) closure *icl(T)* of the DATR theory. Inference closure is the rule closure of the path closure, *rcl(pcl(T))* (cf. Evans & Gazdar 1989:5) of the 'microlex' theory, and defines a subset of theorems with the following properties.

(1) The node in each of the theorems has a mapping via <surf orth> on to the orthographic representation, which begins with a 't'. In other words, variables are used to represent underspecified input, and the EDQL engine is used to predict the rest of the word on the basis of top-down lexical information.

(2) The node in each of the theorems has the property of being an <ilex lemma>.

(3) The lemma is not phrasal (this condition is vacuous in the 'microlex' theory).

(4) For each such node, its <surf phon> value is generated.

(5) For each such node, its <sem> value is also generated.

The definition of EDQL semantics is straightforward. It has a simple set-theoretic characterisation as operations on DATR theorem sets. An informal description of the operations is as follows.

i. *foreach/select*: definition of a subdomain of the inheritance closure of the DATR theory argument which is consistent with the sentence argument; in general, the sentence will be specified wholly or partially with variables. In the sentence, any node or a variable over all nodes may be specified; similarly, any permissible query path with or without variables may be specified.

ii. *if/license*: within this subdomain, include only items defined by the sentence argument;

in general, the constraint will be specified wholly or partially with instantiations.

iii. *ifnot/ban*: within this subdomain, exclude any items which are compatible with the sentence; in general, the constraint will be specified wholly or partially with instantations.

iv. *then/map*: output theorems compatible with the foreach/select, if/license and then/ban constraints.

It is the conjunction of constraints of this kind which permit the flexible querying of DATR theories. Query macros which abbreviate standard, frequently used constraint combinations are defined as extensions to the query language. At the present stage of development, for practical reasons the node domain is restricted by requiring the <ilex lemma> specification, and the set of possible query contexts (attribute paths) is predefined in the theory as lhs paths of equations under a node "Context". These paths are the "sensible queries" referred to in Sections 2 above and 5 below.

To a limited extent, EDQL allows the emulation of unification and default unification operations. Provided that appropriate default-free modelling conventions are adhered to, and the inheritance hierarchy is used to represent a type subsumption hierarchy, a highly general inference engine for DATR emulations of typed feature structures such as those described by Pollard & Sag (1987), Krieger & Nerbonne (1992), Riehemann (1993), may also be realised on EDQL lines using the Pure Prolog engine. This claim is more than a conjecture, as shown by the examples; however, it will not be pursued further at this point.

5 The Open DATR Engine: ODE

A prototype implementation of the DATR-Prolog mapping as a DATR compiler, and a prototype implementation of the EDQL query language and inference theory as an operational EDQL engine has been made by the author. The implementation language is Sicstus Prolog; applications are run in compiled Sicstus Prolog. The inference engine is open, in the sense that new constraint functions can be freely defined by a user, and linked into the existing engine; for this reason, the complete implementation is named ODE (for "Open DATR Engine"), with the two components ODEC (ODE Compiler) and ODEQ (ODE Query engine).

The implementation is provided with a UNIX shell script interface for easy linking of additional Prolog files. Detailed user documentation is in preparation.

5.1 The ODEC compiler and the ODEQ inference engine

The facilities provided by ODEC include the flagging of syntax errors and the production of a set of files in different formats for debugging and development purposes: a normalised character string for tokenisation and parsing; a token representation in standard Prolog notation with syntactic sugar list notation; a canonical "guaranteed read back" representation in dot list notation and quoted atoms; a decompiled version into simple DATR sentences (node-equation pairs) and a decompiled version into complex DATR sentences (node-equation set pairs).

The canonical Prolog notation is sorted into ’longest path first’ order in order to support efficient inference by reducing run-time search. The sort operation is a simplification of the PCS-DATR and DDATR sort operations in Scheme, in which the correct sort order was given by the following function over the Scheme list representation:

(reverse (sort! (copy equations)))

The simplified version is a straightforward alphabetic sort of the canonical Prolog text representation; currently the GNU sort function, which handles lines of arbitrary length, is used for fast and simple sorting. The alphabetic sort gives correct results because of the fortunate coincidence that a longer lhs in a DATR equation has a space (ASCII 32) where the shorter lhs has a right parenthesis (ASCII 42).

The facilities provided by the ODEQ inference engine are currently very simple. The

main feature is the provision of a detailed trace concept modelled on the trace concept of the PCS-DATR and DDATR implementations, with additional visual representation of depth of inheritance and parametrised inheritance, and tracking of evaluation of DATR rhs sequences. There is also an option of output to a theorem dump file. The trace instrumentation which reflects inference flow is also used included to trap some (though not all) cases of circularity in the DATR theory, or inherent in the Prolog semantics of the ODE engine.

Implementation of the constraint system is based on the notion of "sensible query": this is a query over lemma nodes only, with a pre-defined set of query paths. The latter are currently defined at a DATR node called "Context" (see following Subsection); it may be regarded as dominating the entire inheritance hierarchy. Their values are not consulted, and can be left empty. Further types of sensible query are planned.

5.2 Application to an integrated lexicon

A brief illustration of the application of ODE to the microlexicon discussed in the earlier sections of this report is given in the following section.

The following small set of query contexts are defined for "sensible queries" to the "microlex" theory; the Context node replaces the "hide/show" declarations of Sussex DATR.

```
Context:
        <sem>                   ==
        <surf phon>             ==
        <surf orth>             == .
```

A number of different queries are illustrated below, to cover the basic kinds of mapping which are relevant in, for instance, spoken language recognition and generation lexica, and in lexicographic development.

(1) With the query

```
        foreach(microlex,?node:?lhs=?rhs),
        then(microlex,?node:?lhs=?rhs).
```

or

```
        select(microlex,$node:$lhs=$rhs),
        map(microlex,$node:$lhs=$rhs).
```

the ODEQ inference engine defines the following theorem set:

Cloth:<sem> = variety of textile. % microlex theorem

Cloth:<surf orth> = c l o t h. % microlex theorem
Cloth:<surf phon> = k l O T. % microlex theorem
Table:<sem> = horizontal surface to put things on. % microlex theorem
Table:<surf orth> = t a b l e. % microlex theorem
Table:<surf phon> = t eI b l. % microlex theorem
Tablecloth:<sem> = variety of textile for covering horizontal surface to put things on.
% microlex theorem
Tablecloth:<surf orth> = t a b l e c l o t h. % microlex theorem
Tablecloth:<surf phon> = * t eI b l k l O T. % microlex theorem

(2) With the query

```
foreach(microlex, ?node:?lhs=?rhs),
ifnot(microlex, Cloth:?lhs=?rhs),
then(microlex, ?node:?lhs=?rhs).

or      select(microlex, $node:$lhs=$rhs),
        ban(microlex, Cloth:$lhs=$rhs),
        map(microlex, $node:$lhs=$rhs).
```

the ODEQ inference engine generates the following theorem set:

Table:<sem> = horizontal surface to put things on. % microlex theorem
Table:<surf orth> = t a b l e. % microlex theorem
Table:<surf phon> = t eI b l. % microlex theorem
Tablecloth:<sem> = variety of textile for covering horizontal surface to put things on.
% microlex theorem
Tablecloth:<surf orth> = t a b l e c l o t h. % microlex theorem
Tablecloth:<surf phon> = * t eI b l k l O T. % microlex theorem

(3) The query for all theorems beginning with "t" but without phonological representations

```
foreach(microlex, ?node:?lhs=?rhs),
if(microlex, ?node:?lhs=(t|predict)),
ifnot(microlex, ?node:<surf phon>=?rhs),
then(microlex, ?node:?lhs=?rhs).

or      select(microlex, $node:$lhs=$rhs),
        license(microlex, $node:$lhs=(t|predict)),
        ban(microlex, $node:<surf phon>=$rhs),
        map(microlex, $node:$lhs=$rhs).
```

yields the two theorems

Table:<surf orth> = t a b l e. % microlex theorem
Tablecloth:<surf orth> = t a b l e c l o t h. % microlex theorem

(4) With the query

tracemap(microlex,Tablecloth:<surf phon>=$rhs).

the ODE implementation defines the following inference trace:

```
"Tablecloth:<surf orth>"-Tablecloth:<>^<surf orth>=Compound.
 [1,0,1] ->  Compound: ... ^ <surf orth> by III
"Tablecloth:<surf orth>"-Compound:<surf>^<orth>="<modifier surf>" "<head surf>".
  [2,0,1] ->  "<modifier surf>" ^ <orth> by VII
  [2,1,1] ->  modifier by I
  [2,1,2] ->  surf by I
"Tablecloth:<modifier surf orth>"-Tablecloth:<modifier>^<surf orth>="Table:<>".
   [3,0,1] ->  "Table:<>" ^ <surf orth> by V
"Table:<surf orth>"-Table:<> ^ <surf orth>=Simplex.
    [4,0,1] ->  Simplex:<> ^ <surf orth> by III
"Table:<surf orth>"-Simplex:<surf>^<orth>="<root surf>".
     [5,0,1] ->  "<root surf>" ^ <orth> by VII
     [5,1,1] ->  root by I
     [5,1,2] ->  surf by I
"Table:<root surf orth>"-Table:<root surf orth>^<>=t a b l e.
      [6,0,1] ->  t by I
      [6,0,2] ->  a by I
      [6,0,3] ->  b by I
      [6,0,4] ->  l by I
      [6,0,5] ->  e by I
  [2,0,2] ->  "<head surf>" ^ <orth> by VII
  [2,1,1] ->  head by I
  [2,1,2] ->  surf by I
"Tablecloth:<head surf orth>"-Tablecloth:<head>^<surf orth>="Cloth:<>".
   [3,0,1] ->  "Cloth:<>" ^ <surf orth> by V
"Cloth:<surf orth>"-Cloth:<>^<surf orth>=Simplex.
    [4,0,1] ->  Simplex:<> ^ <surf orth> by III
"Cloth:<surf orth>"-Simplex:<surf>^<orth>="<root surf>".
     [5,0,1] ->  "<root surf>" ^ <orth> by VII
     [5,1,1] ->  root by I
     [5,1,2] ->  surf by I
"Cloth:<root surf orth>"-Cloth:<root surf orth>^<>=c l o t h.
      [6,0,1] ->  c by I
      [6,0,2] ->  l by I
      [6,0,3] ->  o by I
      [6,0,4] ->  t by I
      [6,0,5] ->  h by I
```

Tablecloth:<surf orth> = t a b l e c l o t h. % microlex theorem

The trace shows the global environment (in double quotes) and the local environment, with the extension path suffix, and gives the Roman number of the inference rule to be applied.

In addition, the trace instrumentation records the depth of inheritance and parametrised path embedding, and the position in the equation rhs of the descriptor under evaluation. Indentation corresponds to the sum of inheritance and path recursions. Further improvements to EDQL are planned in order to make the logic of the queries more transparent, and to facilitate interfacing with other systems.

6 Conclusion

The basic requirement for a flexible lexicon engine in a spoken language system development environment was examined. Experience with DATR and DQL (DATR query language, standard DATR queries) in formal lexicology with the integration of morphological, prosodic and semantic information, suggested that the extension of DATR and DQL could be profitable in this application.

The DATR language was retained as the basic Lexical Knowledge Representation language, but DQL was replaced by EDQL, an extended DATR query language which permits conjunctions of constraints from different lexical subdomains. An informal set-theoretic semantics and a Pure Prolog inference semantics for EDQL was given; this approach provides the basic specification for the Open DATR Engine, ODE. The ODE concept permits standard DATR inferences, the use of standard DATR inferences as constraints on other DATR inferences, and the definition of subdomains of the lexicon based on constraints with different types of lexical information.

The types of inference permitted also include restricted DATR reverse query search with a modified generate and test strategy, and an open interface to Prolog which allows the development and incorporation of further ODE engine components, including specialised reverse queries, and DATR theory analysis and development engines.

Future work on the Open DATR Engine concept will need to address fundamental questions of complexity and efficient incremental solution of constraint conjunctions, and of domain-specific limitations on the power required for an Open DATR Engine.

The ODE prototype implementation is currently being used for lexicon development in the VERBMOBIL Sub-Project on Lexicon and Morphology (Teilprojekt 5: Lexikon und Morphologie), Work Packages 5.1 (Corpus Analysis), 5.3 (Morphology) and 5.9 (Lexicon Construction), and the ODE specification has been made available for the development of the VERBMOBIL lexical database and lexicographic tools. This report constitutes Deliverable P1 for Work Package 5.3 (U Bielefeld). The system will be made available for further distribution at a later date, after completion of tests and immediate applications.

Acknowledgements

The debt to other scholars in the fields of linguistic and computational lexicology is evident from the cited literature; the prime single debt is owed to Gerald Gazdar and Roger Evans for their creation of DATR. More recently, the ODE concept and implementation has profited from fundamental research and development work in the VERBMOBIL-ASL and VERBMOBIL main phase projects, by Hagen Langer, Martina Pampel, and Doris Bleiching. It has also benefited from discussions with Gerald Gazdar, Roger Evans, James Kilbury, Joe Trubisz, Hans-Jürgen Eikmeyer and Guido Drexel. Doris Bleiching has tested the ODE implementation in lexicon development, and has suggested improvements to syntax and semantics of EDQL, to the ODEQ interpreter interface, and to the trace and filing facilties.

Fundamental contributions to the theoretical linguistic and computational foundations for the ODE concept were made by Hagen Langer (1992; Langer & Gibbon 1992) in the VERBMOBIL-ASL and VERBMOBIL main phase projects. These contributions include the definition and implementation of the DELASOUL constraint engine, the reconstruction of DATR with feature structures and default unification, a highly general solution for DATR reverse query search, and a reduction of DATR to a DATR subset with only two inference rules.

My gratitude is also due to the VERBMOBIL partners in TP 5 (Lexikon und Morphologie) for helping to make possible the development of the ODE concept and its application to integrated lexica for spoken language both intellectually and materially: Humboldt-Universität Berlin (Prof. Jürgen Kunze, Markus Duda, Gunter Gebhardi), IBM-Deutschland (Dr. Tibor Kiss), CAP debis Systemhaus (Ralf Kese) and Daimler-Benz Research Laboratory, Ulm (Dr. Helmut Mangold, and Prof. Stefan Böttcher, now Ulm Polytechnic College).

References

Andry, F., N. M. Fraser, S. McGlashan, S. Thornton, & N.J.Youd (1992). Making DATR Work for Speech: Lexicon Compilation in SUNDIAL. *Computational Linguistics* 18-3, 245-268.

Bleiching, D. (1990). *Das Wortfeld 'family' als semantisches Netz*. SII Thesis, U Bielefeld.

Bleiching, D. (1991). *Default-Hierarchien in der deutschen Wortprosodie*. ASL-TR-19-91/UBI, U Bielefeld.

Bleiching, D. (1992). Prosodisches Wissen im Lexikon. In: Görz, G., ed., *KONVENS '92*. Berlin: Springer-Verlag, 59-68.

Boguraev, B. K. (1991). Building a lexicon: The contribution of computers. In Branimir K. Boguraev, ed., *Building a Lexicon*. Special issue of *International Journal of Lexicography*.

Cahill, L. (1993). Morphophonology in the lexicon. *Proceedings, EACL 1993, Utrecht*, 87-96.

Cahill, L. & R. Evans (1990). The TIC Lexicon. In: Evans & Gazdar 1990.

Carson-Berndsen, J. (1993). *Time Map Phonology and the Projection Problem in Spoken Language Recognition*. Ph.D. thesis, U Bielefeld.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge Mass.

Daelemans, W. & E.-J. van der Linden (1992). *Evaluation of Lexical Representation Formalisms*. ITK Research Memo 14, Institute for Language Technology and AI, University of Tilburg, The Netherlands.

Evans, R. & G. Gazdar (1989). *The DATR Papers: May 1989*. Cognitive Science Research Paper, U Sussex.

Evans, R. & G. Gazdar (1990). *The DATR Papers: February 1990*. Cognitive Science Research Paper 139, U Sussex.

Fillmore, C. (1971). Types of lexical information. In Danny D. Steinberg & Leon A. Jakobovits, eds., *Semantics: An Interdisciplinary Reader in Philosophy, Linguistics and Psychology*. CUP, Cambridge.

Fischer, K. (1993). *Kompositionelle Semantik im Lexikon am Beispiel der englischen Nominalkomposita*. M.A. Thesis, U Bielefeld.

Gibbon, D. (1981). Idiomaticity and functional variation. A case study of international amateur radio talk. In *Language and Society* 10:21-42.

Gibbon, D. (1985). Context and variation in two-way radio discourse. In C.Ferguson, ed. Special Issue of *Discourse Processes.*

Gibbon, D. (1989). PCS-DATR: A DATR implementation in PC-Scheme. English/Linguistics Occasional Papers 4, U Bielefeld.

Gibbon, D. (1990). Prosodic Association by Template Inheritance. In W. Daelemans & G. Gazdar, eds., *Inheritance in Natural Language Processing.* Institute for Language Technology and AI, U Tilburg, the Netherlands.

Gibbon, D. (1991). *Lexical Signs and Lexicon Structure: Phonology and Prosody in the ASL-Lexicon*. ASL-MEMO-20-91/UBI, U Bielefeld.

Gibbon, D. (1992a). ILEX: A linguistic approach to computational lexica. In: U. Klenk (ed.), *Computatio linguae. Aufsätze zur algorithmischen und quantitativen Analyse der Sprache*, Beihefte der Zeitschrift für Dialektologie und Linguistik. Franz Steiner Verlag, Stuttgart. S. 32-53.

Gibbon, D. (1992b). Prosody, time types, and linguistic design factors in spoken language system architectures. In: Görz, G., ed., *KONVENS '92*. Berlin: Springer-Verlag, 90-99.

Gibbon, D. & F. Ahoua (1991). DDATR: un logiciel de traitement d'héritage par défaut pour la modélisation lexicale. In: *Cahiers Ivoiriens de Recherche Linguistique (CIRL)* 27, 5-59.

Gibbon, D. & H. Langer (1992). Linguistische Wortmodellierung. In: D. Reimann [ed.]: *Beiträge des ASL-Lexikonworkshops*. Wandlitz, 26.-27. November 1991. ASL-TR-40-92/ZSB. pp. 50-56.

Halliday, M.A.K. (1961). Categories of the theory of grammar. *Word* 17:241-292.

Halliday, M.A.K. (1966-8). Notes on transitivity and theme in English. *Journal of Linguistics* 2:37-81, 3:199-244, 4:179-215.

Hudson, R. (1984). *Word Grammar*. Blackwell, Oxford.

Kamp, H. & A. Rossdeutscher (1992). *Remarks on Lexical Structure, DRS-Construction and Lexically Driven Inferences*. Arbeitspapiere des SFB 340, Nr. 21.

Katz, J. J. & P. M. Postal (1963). *An Integrated Theory of Linguistic Descriptions*. MIT Press, Cambridge Mass. Ch. 1 & 2.

Kay, M. (1987). Nonconcatenative Finite-State Morphology. In *Proceedings EACL 3, Copenhagen*, 2-20.

Kilbury, J., P. Naerger & I. Renz 1991: DATR as a lexical component for PATR: EACL 1991. pp.137-142.

Koskenniemi, K. (1983). *Two-level Morphology: A General Computational Model for Word Form Recognition and Production*. Ph.D. thesis. University of Helsinki.

Krieger, H.-U. & J. Nerbonne (1992). Feature-based inheritance networks for computational lexicons. In: T. Briscoe, A. Copestake & V. de Paiva [eds.]: *Default Inheritance within Unification-Based Approaches to the Lexicon*. Also DFKI Research Report RR-91-31.

Lamb, S. (1966). *Outline of Stratificational Grammar*. Georgetown UP, Washington DC.

Langer, H. (1992). *DELASOUL: Eine constraintbasierte Beschreibungssprache für lexikalische Repräsentationen*. ASL-TR-26-92/UBI. U. Bielefeld.

Langer, H. & D. Gibbon (1992). DATR as a graph representation language for ILEX speech oriented lexica. ASL-TR-43-92/UBI, U Bielefeld.

Mel'cuk, I. & B. Zholkovsky (1988). The explanatory combinatorial dictionary. In: Evens, M. W., ed.: *Relational Models of the Lexicon*. Cambridge UP, Cambridge.

Mertins, I. (1992). Lexical semantics: theories and their application to the analysis of verbs belonging to the register of cooking. Ms., U Bielefeld.

Moser, L. (1992a). *DATR paths as arguments*. Technical Report CSRP 215, School of Cognitive & Computing Sciences, University of Sussex, Brighton UK.

Moser, L. (1992b). *Lexical constraints in DATR*. Technical Report CSRP 216, School of Cognitive & Computing Sciences, University of Sussex.

Pampel, M. (1992). *Die Repräsentation lexikalischen phonologischen Wissens am Beispiel der Wortbetonung*. ASL Technical Report ASL-TR-58-92/UBI, U Bielefeld.

Pollard, C. & I.A. Sag (1987). *Information-Based Syntax and Semantics. Vol. I: Fundamentals*. CSLI Lecture Notes 13. Stanford: CSLI.

Pustejovsky, J. (1991). The generative lexicon. *Computational Linguistics* 17.

Reinhard, S. & D. Gibbon (1991). Prosodic inheritance and morphological generalisations. In: Proceedings of EACL 1991, Berlin, pp. 131-136.

Riehemann, S. (1993). *Word Formation in Lexical Type Hierarchies. A Case Study of **bar**-Adjectives in German*. SfS-Report-02-93, Seminar für Sprachwissenschaft, Computerlinguistik, U Tübingen.

Selkirk, E. O. (1984). *Phonology and Syntax*: The Relation between Sound and Structure. MIT press, Cambridge, Mass.

Weinreich, U. (1969). Problems in the analysis of idioms. In Jaan Puhvel, ed., *Substance and Structure of Language*. U California Press, Berkeley and Los Angeles.

Wells, J.C. (1989). Computer-coded phonemic notation of individual languages of the European Community. *Journal of the International Phonetic Association* 19/1:31-54.

Appendix

```
/*
---- This is an executable Prolog file --------


                MINIDATR: A minimal core DATR engine in Prolog

                                Dafydd Gibbon
                                 U Bielefeld
                            gibbon@asl.uni-bielefeld.de
                                 23 September 1994


This document contains a simple version of the core DATR inference engine in
Prolog in order to illustrate the principles of DATR inference to Prolog
programmers. Note that in minor details it departs slightly from DATR
conventions:
- nonstandard nodenames are permitted;
- the knowledge base must be pre-sorted to permit 'longest path first'
      inference;
- queries include the theory name.

Note also that this is not a directly usable implementation: there is no user
interface, no DATR-Prolog interpreter, no DATR-specific trace or debugging,
no attention paide to efficiency, etc. The aim is to provide a minimal 'core
DATR standard inference' interpreter in logical style.


1 Illustration of a DATR theory: a 'microlexicon'

MINILEX.DTR

Tablecloth: <>            == Compound
            <ilex>        == lemma
            <relation>    == (for covering)
            <modifier>    == "Table:<>"
            <head>        == "Cloth:<>".

Table:      <>            == Simplex
            <ilex>        == lemma
            <meaning>     == (horizontal surface to put things on)
            <orthography> == (t a b l e).

Cloth:      <>            == Simplex
            <ilex>        == lemma
            <meaning>     == (variety of textile)
            <orthography> == (c l o t h).
```

```
Compound:   <>             == Word
            <ilex>         == generalisation
            <type>         == compound
            <meaning>      == ("<head meaning>" "<relation>"
                               "<modifier meaning>")
            <orthography> == ("<modifier orthography>" "<head orthography>").

Simplex:    <>             == Word
            <ilex>         == generalisation
            <type>         == simplex.

Word:       <ilex>         == generalisation
            <type>         == word.


Theorems:

Tablecloth:<relation>=(for covering).
Tablecloth:<meaning>=(variety of textile for covering horizontal surface to
                    put things on).
Tablecloth:<orthography>=(t a b l e c l o t h).
Table:<orthography>=(t a b l e).
Table:<relation>=undefined.


2 A Prolog translation of MINILEX.DTR: MINILEX.PRO knowledge base.

Note the 'longest path first' reordering of the theory in Prolog.

Tablecloth: <>             == Compound
            <ilex>         == lemma
            <relation>     == (for covering)
            <modifier>     == "Table:<>"
            <head>         == "Cloth:<>".
*/
datr_sentence(minilex,'Tablecloth',[ilex],[lemma]).
datr_sentence(minilex,'Tablecloth',[relation],[for,covering]).
datr_sentence(minilex,'Tablecloth',[modifier],[[gnp,'Table',[]]]).
datr_sentence(minilex,'Tablecloth',[head],[[gnp,'Cloth',[]]]).
datr_sentence(minilex,'Tablecloth',[],[[ln,'Compound']]).
/*
Table:      <>             == Simplex
            <ilex>         == lemma
            <meaning>      == (horizontal surface to put things on)
            <orthography> == (t a b l e).
*/
datr_sentence(minilex,'Table',[ilex],[lemma]).
datr_sentence(minilex,'Table',[meaning],[horizontal,surface,to,put,things,o
n]).
datr_sentence(minilex,'Table',[orthography],[t,a,b,l,e]).
datr_sentence(minilex,'Table',[],[[ln,'Simplex']]).
/*
Cloth:      <>             == Simplex
            <ilex>         == lemma
            <meaning>      == (variety of textile)
            <orthography> == (c l o t h).
*/
datr_sentence(minilex,'Cloth',[ilex],[lemma]).
datr_sentence(minilex,'Cloth',[meaning],[variety,of,textile]).
datr_sentence(minilex,'Cloth',[orthography],[c,l,o,t,h]).
datr_sentence(minilex,'Cloth',[],[[ln,'Simplex']]).
/*
Compound:   <>             == Word
            <ilex>         == generalisation
```

```
            <type>         == compound
            <meaning>      == ("<head meaning>" "<relation>"
                               "<modifier meaning>")
            <orthography> == ("<modifier orthography>" "<head orthography>").
*/
datr_sentence(minilex,'Compound',[ilex],[generalisation]).
datr_sentence(minilex,'Compound',[type],[complex]).
datr_sentence(minilex,'Compound',[meaning],
   [[gp,[head,meaning]],[gp,[relation]],[gp,[modifier,meaning]]]).
datr_sentence(minilex,'Compound',[orthography],
   [[gp,[modifier,orthography]],[gp,[head,orthography]]]).
datr_sentence(minilex,'Compound',[],[[ln,'Word']]).
/*
Simplex:    <>             == Word
            <ilex>         == generalisation
            <type>         == simplex.
*/
datr_sentence(minilex,'Simplex',[ilex],[generalisation]).
datr_sentence(minilex,'Simplex',[type],[simplex]).
datr_sentence(minilex,'Simplex',[],[[ln,'Word']]).
/*
Word:       <ilex>         == generalisation
            <type>         == word.
*/
datr_sentence(minilex,'Word',[ilex],[generalisation]).
datr_sentence(minilex,'Word',[type],[word]).

minilex(Node,Path,Value) :- datr(minilex,Node,Path,Value).
/*
```

3 A DATR test theory, MINITEST.DTR

This theory illustrates the seven cases of DATR standard inference. The relevant theorems are the following:

```
A:<> = (via node A via node B via node C undefined).
A:<1> = (via node A Rule 1).
A:<2> = (via node A Rule 2).
A:<3> = (via node A Rule 3).
A:<4> = (via node A Rule 4).
A:<5> = (via node A via node C Rule 5).
A:<6> = (via node A Rule 6).
A:<7> = (via node A Rule 7).
A:<1 2> = (path <1 2> extends path <1>).
A:<nest a> = (via node A nested global path with a).
A:<nest b> = (via node A nested global path with rubbish).
```

The MINITEST.DTR theory:

```
A:<>              == (via node 'A' B)
  <1>             == <one>
  <2>             == <two>
  <3>             == <three>
  <4>             == <four>
  <5>             == <five>
  <6>             == <six>
  <7>             == <seven>
  <seventh>       == 'Rule 7'
  <1 2>           == (path '<1 2>' extends path '<1>')
  <param>         == alpha.

B:<>              == (via node 'B' C)
  <one>           == 'Rule 1'
```

```
  <two>            == C:<second>
  <three>          == C
  <four>           == <fourth>
  <five>           == "C:<fifth>"
  <six>            == "C"
  <seven>          == "<seventh>"
  <fourth>         == 'Rule 4'
  <nest>           == <elsif "<param>">
  <elsif alpha a> == 'nested global path with a'
  <elsif>          == 'nested global path with rubbish'.

C:<>               == (via node 'C' D)
  <6>              == 'Rule 6'
  <second>         == 'Rule 2'
  <three>          == 'Rule 3'
  <fuenf>          == 'Rule 5'.

D:<>               == undefined
  <fifth>          == "<fuenf>".
```

4 A BNF description of core DATR syntax

```
<theory>     ::= <sentence> | <sentence> <theory>
<sentence>   ::= <node> : <equations>
<equations>  ::= . | <equation>
<equation>   ::= <lhs> == <rhs>

<lhs>        ::= "<" <atomseq> ">"
<atomseq>    ::= nullseq | <atom> <lhseq>

<rhs>        ::= <valseq> | ( <valseq> )
<valseq>     ::= nullseq | <val> <valseq>
<val>        ::= atom | <descriptor> | " <descriptor> "
<descriptor> ::= <node> : <path> | <node> | <path>
<path>       ::= "<" <valseq> ">"

<atom>       ::= <a_char> <s_seq> | ' <charseq> '
<node>       ::= <n_char> <s_seq>

<charseq>    ::= nullseq | <char> <charseq>
<s_seq>      ::= nullseq | <s_char> <s_seq>

<res_char>   ::= {:, <, >, =, (, ), ", .}
<char>       ::= {char(0),...,char(127)}
<p_char>     ::= {char(33),...,char(127)} | \ <char>
<s_char>     ::= <p_char> - <res_char>
<n_char>     ::= {A,...,Z}
<a_char>     ::= <s_char> - <n_char>
```

5 MINITEST.DTR to MINITEST.PRO translation (outline)

```
<sentence>   ::= datr_node(<theory>,<node>,<lhs>,<rhs>).
<lhs>        ::= Prolog list of atoms, perhaps empty,
                  e.g. [], [a], [attribute,list]
<rhs>        ::= Prolog list of <descriptor> expressions, perhaps empty,
                  e.g. [], [a], [aa,bb], etc.
<descriptor> ::= expression of one of the following types,
                corresponding to each of the 7 DATR inference rules as a
                Prolog atom or a tagged list:
                (1) <atom>
                (2) [lnp,<node>,<path>] for local node-path
```

```
                (3) [ln,<node>] for local node
                (4) [lp,<path>] for local path
                (5) [gnp,<node>,<path>] for global node-path
                (6) [gn,<node>] for global node
                (7) [gp,<path>] for global path
<node>        ::= <atom>
<atom>        ::= Prolog atomic symbol
<path>        ::= <rhs>
```

6 Illustration of DATR-Prolog translation for MINITEST.DTR

This line-by-line illustration includes "longest path first" pre-sorting:

```
/* A:<1 2>            == (' path <1 2> extends path <1>').        */
datr_sentence(minitest,'A',[1,2],[' path <1 2> extends path <1>']).
/* A:<1>              == <one>.                                   */
datr_sentence(minitest,'A',[1],[[lp,[one]]]).
/* A:<2>              == <two>.                                   */
datr_sentence(minitest,'A',[2],[[lp,[two]]]).
/* A:<3>              == <three>.                                 */
datr_sentence(minitest,'A',[3],[[lp,[three]]]).
/* A:<4>              == <four>.                                  */
datr_sentence(minitest,'A',[4],[[lp,[four]]]).
/* A:<5>              == <five>.                                  */
datr_sentence(minitest,'A',[5],[[lp,[five]]]).
/* A:<6>              == <six>.                                   */
datr_sentence(minitest,'A',[6],[[lp,[six]]]).
/* A:<7>              == <seven>.                                 */
datr_sentence(minitest,'A',[7],[[lp,[seven]]]).
/* A:<seventh>        == 'Rule 7'.                                */
datr_sentence(minitest,'A',[seventh],[' Rule 7']).
/* A:<param>          == alpha.                                   */
datr_sentence(minitest,'A',[param],[alpha]).
/* A:<>               == (via node 'A' B).                        */
datr_sentence(minitest,'A',[],[' via node A',[ln,'B']]).

/* B:<one>            == 'Rule 1'.                                */
datr_sentence(minitest,'B',[one],[' Rule 1']).
/* B:<two>            == C:<second>.                              */
datr_sentence(minitest,'B',[two],[[lnp,'C',[second]]]).
/* B:<three>          == C.                                       */
datr_sentence(minitest,'B',[three],[[ln,'C']]).
/* B:<four>           == <fourth>.                                */
datr_sentence(minitest,'B',[four],[[lp,[fourth]]]).
/* B:<five>           == "C:<fifth>".                             */
datr_sentence(minitest,'B',[five],[[gnp,'C',[fifth]]]).
/* B:<six>            == "C".                                     */
datr_sentence(minitest,'B',[six],[[gn,'C']]).
/* B:<seven>          == "<seventh>".                             */
datr_sentence(minitest,'B',[seven],[[gp,[seventh]]]).
/* B:<fourth>         == 'Rule 4'.                                */
datr_sentence(minitest,'B',[fourth],[' Rule 4']).
/* B:<nest>           == <elsif "<param>">.                       */
datr_sentence(minitest,'B',[nest],[[lp,[elsif,[gp,[param]]]]]).
/* B:<elsif alpha a> == 'nested global path with a'.              */
datr_sentence(minitest,'B',[elsif,alpha,a],[' nested global path with a']).
/* B:<elsif>          == 'nested global path with rubbish'.       */
datr_sentence(minitest,'B',[elsif],[' nested global path with rubbish']).
/* B:<>               == (via node 'B' C).                        */
datr_sentence(minitest,'B',[],[' via node B',[ln,'C']]).

/* C:<6>              == 'Rule 6'.                                */
```

```
datr_sentence(minitest,'C',[6],[' Rule 6']).
/* C:<second>          == 'Rule 2'.                                */
datr_sentence(minitest,'C',[second],[' Rule 2']).
/* C:<three>           == 'Rule 3'.                                */
datr_sentence(minitest,'C',[three],[' Rule 3']).
/* C:<fuenf>           == 'Rule 5'.                                */
datr_sentence(minitest,'C',[fuenf],[' Rule 5']).
/* C:<>                == (via node 'C' D).                        */
datr_sentence(minitest,'C',[],[' via node C',[ln,'D']]).

/* D:<fifth>           == "<fuenf>".                               */
datr_sentence(minitest,'D',[fifth],[[gp,[fuenf]]]).
/* D:<>                == undefined.                               */
datr_sentence(minitest,'D',[],[' undefined']).

minitest(Node,Path,Value) :- datr(minitest,Node,Path,Value).
/*


Outline of the MINIDATR.PRO inference engine

datr(Theory,Node,Path,Value).
       - defines initial DATR global and local query environments
       - 1 clause
    datr_connect(Theory,Gnode,Gpath,Lnode,Lpath,Value).
       - accesses DATR theory with query environments
       - 1 clause
    datr_rhs(Gnode,Gpath,Lnode,Prefix,Suffix,Rhs,Value).
       - evaluates RHS of DATR equations as lists
       - 2 clauses
    datr_rule(Gnode,Gpath,Lnode,Prefix,Suffix,Descriptor,Value).
       - 7 DATR inference rules
       - 7 clauses, for evaluation of atoms and of local and
         global node-path, node, and path descriptors.
    append(Prefix,Suffix,Whole).
       - expresses RHS sequence evaluation and path extension
       - 2 clauses (standard definition).

The following queries illustrate standard DATR inference.

   Query                      Response

minitest('A',[],X).       X = [ via node A, via node B, via node C,undefined]
minitest('A',[1],X).       X = [ via node A, Rule 1]
minitest('A',[2],X).       X = [ via node A, Rule 2]
minitest('A',[3],X).       X = [ via node A, Rule 3]
minitest('A',[4],X).       X = [ via node A, Rule 4]
minitest('A',[5],X).       X = [ via node A, via node C, Rule 5]
minitest('A',[6],X).       X = [ via node A, Rule 6]
minitest('A',[7],X).       X = [ via node A, Rule 7]
minitest('A',[1,2],X).     X = [ path <1 2> extends path <1>]
minitest('A',[nest,a],X).  X = [ via node A, nested global path with a]
minitest('A',[nest,b],X).  X = [ via node A, nested global path with rubbish]


7 Code for the MINIDATR inference engine

Note that this minimalistic core DATR inference engine allows
nonstandard node-names, has no DATR variables, and does not
include the 'longest path first' default connection condition.
This means that the 'longest path first' ordering must be
pre-defined in the Prolog knowledge base.
*/
```

```
datr(Theory,Node,Path,Value) :-
 atomic(Theory),
 atomic(Node),
 nonvar(Path),
 atompath(Path),
 var(Value),
 datr_connect(Theory,Node,Path,Node,Path,Value),!.

atompath([]) :- !.
atompath([First|Rest]) :-
 atomic(First),
 atompath(Rest).

datr_connect(Theory,Gnode,Gpath,Lnode,Lpath,Value):-
 datr_sentence(Theory,Lnode,Prefix,Rhs),
 append(Prefix,Suffix,Lpath),!,
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Rhs,Value).


datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[],[]).

datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[First|Rest],Value) :-
 datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,First,First_value),
 append(First_value,Rest_value,Value),!,
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Rest,Rest_value).


/* DATR Inference Rule 1                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Expr,[Expr]) :-
 atomic(Expr).

/* DATR Inference Rule 2                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[lnp,Node,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Node,Extension,Value).

/* DATR Inference Rule 3                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[ln,Node],Value) :-
 append(Prefix,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Node,Extension,Value).

/* DATR Inference Rule 4                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[lp,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Gpath,Lnode,Extension,Value).

/* DATR Inference Rule 5                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gnp,Node,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Node,Extension,Node,Extension,Value).

/* DATR Inference Rule 6                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gn,Node],Value) :-
 append(Gpath,Suffix,Extension),
 datr_connect(Theory,Node,Extension,Node,Extension,Value).

/* DATR Inference Rule 7                                        */
datr_rule(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,[gp,Path],Value) :-
 datr_rhs(Theory,Gnode,Gpath,Lnode,Prefix,Suffix,Path,Path_value),
 append(Path_value,Suffix,Extension),
 datr_connect(Theory,Gnode,Extension,Gnode,Extension,Value).
```