Parsing Natural Language using LDS: A Prototype*

Marcelo Finger
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Rodger Kibble
Department of Linguistics
School of Oriental and African Studies
University of London

Dov Gabbay
Department of Computing
Imperial College
University of London

Ruth Kempson
Department of Linguistics
School of Oriental and African Studies
University of London

February 21, 1997

Abstract

This paper describes a prototype implementation of a Labelled Deduction System for natural language interpretation [Gabbay & Kempson 1992], where interpretation is taken to be the process of understanding a natural language utterance. The implementation models the process of understanding *wh*-gap dependencies in questions and relative clauses for a fragment of English. The paper is divided in three main sections. In section 1, we introduce the basic architecture of the system. Section 2 outlines a prototype implementation of *wh*-binding and indicates its potential for explanation of linguistic phenomena, and in Section 3 we briefly set the model within a larger theoretical perspective, comparing it to other type-logical approaches to natural language analysis.

0 A Brief Background on LDS$_{NL}$

In LDS$_{NL}$, natural language understanding is modelled as a syntactic process mapping lexical items onto representations in some logical system. The outcome of the interpretation process is taken to be a well-formed formula (or set of formulas) in the selected logical system. The input provided by lexical items drives the incremental process of establishing the required logical representation. The task of natural language understanding is thus a metalevel task of reasoning which is in part deductive (and determined by the linguistic input), and in part abductive, involving extra choices both at the level of context (anaphora resolution) and also at the level of structure (*wh*-gap binding).

The approach is an extension of the 'parsing as deduction' methodology familiar in both computational linguistics [Pereira & Warren 1980] and categorial grammar [Moortgat 1988, Hepple 1990, Morrill 1994]. Inference is defined over *declarative units* each of which consists

*This research was in part supported by the UK Engineering and Physical Sciences Research Council under grant reference GR/K67397, "A Labelled Deduction System for Natural Language Understanding" and by the Economic and Social Research Council award no. R000-23-2069. M. Finger is partly supported by Brazilian CNPq, grant 300597/95-9.

of a pair: a type-logical *formula* and an algebraic *label*. The label has two constituents, namely the formula's semantic interpretation and a list containing any extra control specifications on the step of inference to be carried out. The formal system is a Labelled Deductive System in the sense of [Gabbay 1996]. Interpretation is defined as a left-to-right process with lexical input projecting both declarative and procedural content, namely type-logical formulas, quantifier-free predicate-logic expressions and control information. The output consists of a database recording the proof steps with linked databases representing relative clauses and nested databases for complement clauses [Gabbay, Kempson & Pitt 1994]. An innovative feature is that certain items such as *wh*-operators are not incorporated into the interpretation at the point where they are encountered but contribute a license to create an assumption at a later stage when syntactic material 'expected' by the preceding context is absent (a 'gap'). This procedural account blocks illicit dependencies without a need to stipulate configurational constraints. The implications of this approach for issues in theoretical linguistics are discussed in detail in setion 2.1.

Some examples of the end product of sentence interpretation (omitting intermediate deductions):

(1) John upset Mary

```
>> john upset mary

   Succeeds for: upset(mary)(john):t
```

(2) John upset Mary who fainted

```
>> john upset mary who fainted

              Linked DB
              Succeeds for: faint(gap(mary)):t

   Succeeds for: upset(mary)(john):t
```

(3) Who upset the man who upset John?

```
>> who upset the man who upset john

              Linked DB
              Succeeds for: upset(john)(gap(the:man)):t

   Succeeds for: upset(the:man)(gap(#unknown#)):t
```

(4) * Who did John upset the man who saw?[1]

```
>> who did john upset the man who saw

no
```

The formal system has been further developed to handle crossover phenomena and ellipsis [Kempson 1995, Kempson & Gabbay forthcoming]. The prototype program does not cover the full range of crossover data but has been extended with a treatment of pronouns which is at least consistent with the following key assumptions of the LDS_{NL} account:

[1] An asterisk is placed in front of incorrect sentences/interpretations.

(i) Pronouns are handled by a free-choice mechanism: they are treated as introducing a premise of type `e` labelled with a 'meta-variable'. This meta-variable must be resolved by identifying an appropriate labelled item in the database, with the sole restriction that the resolvent may not also be a label associated with a premise of type `e` in the current local database. [2]

(ii) Pronoun resolution is handled 'on-line' at the lexical analysis stage and so does not have access to information projected by items in the input string following the pronoun.

(iii) In view of the account of *wh*-gap dependencies mentioned above, a *wh*-operator is not 'visible' for purposes of anaphoric dependency until the 'gap' licensed by *wh* has been encountered.

These last two factors account for the contrast between (5) and (6) below. In (5) we have the sequence *wh*-gap-pronoun, and so the pronoun may be interpreted as if 'bound' by the *wh*-operator. In (6) the sequence is *wh*-pronoun-gap and so the pronoun *he* can't be interpreted as dependent on *who*. Instead it is given an arbitrary identifier indicating that its reference is unresolved, pending an extension of the implementation to a more comprehensive account of pronominal anaphora.

(5) Who$_i$ does John think e_i thinks he$_i$ upset Mary?

```
>> who does john think thinks he upset mary
            Nested DB
                      Nested DB
                      Succeeds for : upset(mary)(pro:wh):t
            Succeeds for: think(upset(mary)(pro:wh))(gap(#unknown#)):t

   Succeeds for: think( think(upset(mary)(pro:wh))(gap(#unknown#)) )
                      ( john ):t
```

(6) Who$_i$ does John think he$_j$ thinks e_i upset Mary?

```
>> who does john think he thinks upset mary
               Nested DB
                        Nested DB
                        Succeeds for: upset(mary)(gap(#unknown#)):t
               Succeeds for: think(upset(mary)(gap(#unknown#)))
                                   (pro:#137):t
   Succeeds for: think( think(upset(mary)(gap(#unknown#)))(pro:#137) )
                      ( john ):t
```

Further details of pronoun resolution and *wh*-binding in the prototype program will be found in section 2.

Deduction is forward-chaining, 'goal-directed' application of Modus Ponens over labelled formulas, also called *declarative units*. As in linear logic, each premise has to be used exactly

[2] If the lexicon is extended to include pronominal and *wh* determiners such as *his, whose*, the restriction on antecedent choice has to be stated as a conditional restriction that if the pronoun projects a formula of type `e`, its resolvent may not also be of type `e` within that current local database.

once. Processing terminates with the derivation of the labelled goal formula `?:t`, provided that all premises licensed by lexical input and all intermediate deductions have been used. The labelling algebra ensures that in the process a grammatical sentence has been recognised, without explicitly constructing a syntactic parse tree. For example the derivation of (1) proceeds as follows:

```
(john, [use_last]):e
(upset,[ ]):e -> (e -> t)
(mary,[]):e
(upset(mary),[ ]):e -> t
(upset(mary)(john),[ ]):t
```

Note that the first declarative unit contains the control information `[use_last]`, which precludes the construction of `(upset(john),[]):e -> t`. How this control information is generated and used in the deduction process will be detailed in the next section. Also note that both the lexical item `john` and the control information `[use_last]` are formally the two constituents of the label. However, for clarity purposes, in the samples of program output shown below control information such as `[use_last]` will be displayed separately from the labelled formulas.

A Prolog implementation of the LDS_{NL} system has been built and ported for both LPA Prolog for Windows and Sicstus Prolog. The system performs rather well, apparently giving the answers "immediately" after request. It is expected that, if we had a lexicon larger than what we had at our disposal, the response of the system would be affected. But that would just mean that we did not have an efficient lexicon access procedure, not that the LDS_{NL} ideas do not allow an efficient implementation. On the other hand, the efficiency we achieved might be credited to the left-right parsing strategy of LDS_{NL}, and on the fact that the parsing tree being built never has to be re-explored. All the decisions made during the parse are based on the context information available at the current point of the parsing tree.

Efficiency was not our main goal in the development of the prototype. The main goal was to test that the LDS_{NL} ideas could be practically implemented. However, we were not displeased with efficiency matters. It remains to be analysed exactly how the system performs in the parsing of sentences of various sizes, but we can anticipate that the number of items stored in the parsing tree for each sentence increases linearly with the size of the input sentence.

LDS has been used as a basis for other systems' implementation, mainly for the construction of theorem provers. For example, [D'Agostino & Gabbay 1994] presents a general LDS-tableaux theorem prover for substructural logics whose implementation specialised for Linear Logic was described in [Broda & Finger 1995]; LDS has been shown to gracefully apply to Natural Deduction theorem proving [Broda, Finger & Russo 1996]; and there are other works under way in the application of LDS to temporal/modal logics [Russo 1996]. All these implementations share with ours the basic LDS principles: (a) both labels and formulas are jointly handled at each deduction step as a unique declarative unit; (b) formula manipulation is kept as simple as possible, putting on the label the burden of taking care of control and semantic information.

We proceed with a description of the system implemented.

1 System Description

The basic principles of the LDSNL algorithm are the following:

- Parsing is a goal directed deduction process; the goal of parsing a sentence is to obtain a declarative unit whose formula part is `t`.
- The string is processed left to right; each word of the input may project *data* and a set of *expectations*. Data is a declarative unit whose formula part represents the word's type, and whose label may contain syntactical features, control and semantic information. Expectations are (descriptions of) declarative units that are expected to occur in the data generated by preceding a nd subsequent words.
- The set of data projected by words forms a *database*. Some words, such as wh-expressions, project no data but only some control information forcing the creation of a *linked database* in which parsing continues; a linked database is closed when a wh-clause has been recognized, and parsing resumes at the original database level.
- Parsing finishes when a declarative unit matching the goal formula has been deduced, all input words have been consumed, all data have been used for deduction, all expectations have been met and no control information (such as `use_gap`) is unfulfilled.
- Deduction is done between the data accumulated by the process of consuming words and LDSNL relies on a single deduction rule called *Labelled Modus Ponens*, an extension of Modus ponens that copes with control information (`use_last`) and goal directedness; it also combines the label part of declarative units. Labelled MP takes two formulas, removes them from the database and inserts a single deduced formula instead; this guarantees that a formula is never used twice in a deduction.

To exemplify those principles at work, consider again the parsing of example (2)

```
>> john upset mary who fainted
```

The complete output trace generated by the program is as follows, showing the sequence of deductions annotated with control information from the label.

```
Goal : ? : t
    john : e     use_last
    upset : e -> e -> t
    mary : e

            ---------------
            Linked DB
              Goal : ? : t          use_gap
                 gap(mary) : e          use_last
                 faint : e -> t
                 faint(gap(mary)) : t    gap_used

              Goal succeeds for :  faint(gap(mary)) : t
            ---------------
```

```
        upset(mary) : e -> t
        upset(mary)(john) : t

    Goal succeeds for :  upset(mary)(john) : t
```

We now detail and explain all those terms and concepts that have been mentioned in the description and example above. Some of the definitions and data structures below correspond directly to structures defined in the formal system [Gabbay & Kempson 1992] and others result from implementation decisions.

1.1 Basic definitions

The basic building blocks of the system are *declarative units* (DUs) and *databases* (DBs). The *declarative units* (DUs) of our system are Labelled Formulas (LFs), a pair $l : A$ where l is a *label* and A a well formed *type-formula* built according to the rules:

(i) `e, t` are types.

(ii) If `a, b` are types then `a -> b` is a type; `->` is right-associative, so `e -> e -> t` $\equiv$ `e -> (e -> t)`

Databases generalise the classical notion of a theory. Whereas a theory is a set of sentences closed under consequence, a database is a set of labelled objects closed under labelled deduction. Formally, we have:

(i) A declarative unit is a database.

(ii) A set of declarative units closed under Labelled Modus Ponens (see below) is a database.

(iii) A pair of *linked databases* is a database. In the formal system a pair of databases are linked by unification or 'sharing a variable as the result of some substitution process' [Gabbay & Kempson 1992]. We implement such a notion in the prototype system by means of a feature `link(M)` in the control label of the shared element in the primary database (eg the head NP of a relative clause), where `M` is the linked DB.

Labels are applied both to type-formulas, generating labelled formulas, and to databases, generating labelled databases, as described below.

Formula Labels

A label on a formula (pure label) consists of a *content label* (Clabel) and a *resource label* (Rlabel).

(i) The *Content label* consists of quantifier-free predicate-logic expressions such as `upset(mary)`, `john`, etc.

(ii) The *resource label* controls application of the deduction rules. It is represented as a Prolog list of items such as:

- `use_last` identifies subject NP.
- `use_first` identifies object NP (not implemented).
- `link(M)` is a pointer to linked database `M`.

The following features occur only in the label of a Goal Formula: `use_gap` is projected from *wh*-operators; and `gap_used` is inserted when a gapped formula has been used.

Labelled formulas (LFs) are represented as Prolog terms:

```
l(Content-label, Resource-label):Formula
```

For example,

```
l(upset(mary),[]):e -> t
l(john,[use_last]):e
```

Database Labels

A label on a database includes several sentence properties:

(i) `tense` is projected from verb inflection;

(ii) `gap(l(X,RL):F)` indicates a gap specification projected from *wh*-processing and stored in the database label until the corresponding 'gap' is encountered in the input string. In the case of relative clauses `X` is a predicate logical expression; in the case of *wh*-questions, `X` is `#unknown#`. When the predicted gap is encountered the embedded LF is inserted into the database as a premise of the form `l(gap(X),RL):F`, and `gap-used` is added to the Goal Formula label. In the current implementation `F` is always type `e`.

Database labels are implemented as the `NodeLabel` in the construction tree (see below).

Labelled MP

Provided the deduction step is *licensed*, labelled MP performs the following simple deduction step:

$$\frac{\texttt{l(CL}_1\texttt{, P}_1\texttt{): a -> b} \qquad \texttt{l(CL}_2\texttt{, P}_2\texttt{ : a}}{\texttt{l(CL}_1\texttt{(CL}_2\texttt{), P): b}}$$

The deduction step above is licensed whenever:

- `use_last` is not in $\texttt{P}_1$ or $\texttt{P}_2$;
- otherwise the deduction is licensed only if the goal formula under pursuit is `b`; this guarantees that the formula marked with *use_last* can be only used in the last deduction step, i.e. the one that achieves the deduction goal.

If the deduction is licensed then $\texttt{P} = (\texttt{P}_1 \cup \texttt{P}_2) - \{\texttt{use\_last}\}$.

Linked Databases

We distinguish between three types of databases that may be constructed during a parse, namely *main databases* associated to the main clauses, *linked databases* associated to *wh*-clauses, and *nested databases* associated to complement clauses.

In example (2), 'John upset Mary who fainted', the main clause 'John upset Mary' generates the main database. Initially, this main database contains the data {`l(john, [use_last]): e`, `l(upset, []): e->(e->t)`, `l(mary,[]): e`}, which is projected by the data. Then, by closing this data under labelled MP, {`l(upset(mary),[]): e->t`, `l(upset(mary)(john): t`} is added to the main database.

A linked database, on the other hand, is an independent database that is the result of parsing a *wh*-clause. As a consequence, besides the declarative unites that are obtained from the data pertaining to the *wh*-clause, a labelled formula of the form `l(gap(`x`), []):e)` is added to the linked database to represent a gap; furthermore, the control information `use_gap` is added to the linked database label to enforce that the gap has to be used in the deduction of a `t`-typed formula representing the *wh*-clause.

The variable x is instantiated to an `e`-typed formula from the main database. In this respect, the database projected by a *wh*-clause is *linked* to that `e`-typed formula. Apart from that, the linked database is totally independent from the main database. As a result of this independence, only one gap will ever be allowed in a linked database, even in the case where there are nested *wh*-clauses.

In example (2), the clause 'who fainted' generates a database linked to `l(mary,[]):e` in the main database whose content, after closure with labelled MP is {`l(gap(`*mary*`), [use_last]):e)`, `l(faint, []): e->t`, `l(faint(gap(mary)),[]):t`}. The introduction of `use_last` is due to the tense information projected by the verb and will be detailed when we discuss the lexicon. Note that the gap has been used in the derivation of the `t`-typed formula.

On the other hand, sentences such as '*John who Bill loves Mary fainted' are blocked even though the database projected from 'who Bill loves Mary' can deduce a `t` formula; blocking is done by enforcing the `use_gap` feature for the parsing to succeed.

Nested Databases

Finally we need to be able to handle complement clauses as in (7)

(7) John knows Mary fainted.

where the representation of the complement 'Mary fainted' in itself includes a database.

This is achieved in the implemented system by generating a 'derived' declarative unit to represent the complement clause. After building the nested DB `N` which uniquely proves `s:A` (has `s:A` as its unique consequence), we add a DU of form `l(s,N):A` to the primary DB where it is available for use as a premise in MP. `N` is the 'root' of the tree representing the nested database[3].

Implementation works as follows.

[3] In the formal system of [Gabbay & Kempson 1992] this is handled by defining a *metadeclarative unit* which becomes a premise to an extended Modus Ponens:

Definition: A metadeclarative unit $\Delta : A$ is constructed from a database Δ which uniquely proves the DU $s : A$. A is the type of Δ and s is its label.

In general, lexical entries provide data entries to the parser or operational directives to the parser; for this second case, see the construction of linked databases below. In example (7), the lexical entry for the verb projects data entries `d(l(know,[]):t), expect(node(RG,N))` where `RG` is the 'Root Goal' of nested database `N`.

Data entries are consumed in order. At the point where `expect(node(RG,N))` is encountered, the construction of the primary DB is suspended and a new database constructed from the input string following the verb (i.e. `[mary, fainted]` in the analysis of (7)). If this completes successfully, the resulting DU (i.e. `l(faint(mary),P1):t` in the example) is added to the `Data` list in the primary DB and subsequently input to Labelled Modus Ponens as shown in (8).

(8)

$$\dfrac{\dfrac{\texttt{l(know,[]):t->(e->t)} \qquad \texttt{l(faint(mary),[N]):t}}{\texttt{l(know(faint(mary)),[N]):e->t}} \qquad \texttt{l(john,[]):e}}{\texttt{l(know(faint(mary))(john),[N]):t}}$$

1.2 Database Construction

The construction process builds up a tree structure with one tree for each database. The main Prolog routine `parse_lds` calls `initiate_main_node` to initialise a tree structure; for each clause of the input, the predicate `ldsNL_parse_sentence` is called and expands the tree under construction. After the content of each lexical item is added to the tree, control is passed to `deductions` which attempts to apply Modus Ponens using the most recently generated LF and an intermediate deduction as premises.

Processing successfully terminates for a node under the following conditions:

(i) The Goal Formula has been deduced and all premises and intermediate deductions have been used exactly once.

(ii) No unused gap in Goal Label: if `use_gap` is present there must be a matching `gap-used`.

(iii) The list of expected data input is empty.

This checking is carried out by predicate `achieved_goals` which is called when the above mentioned predicate `ldsNL_parse_sentence` completes.

A *node* is the data structure representing a labelled database and its Prolog representation is the following:

Thus the nested database Δ is itself treated as a label. An extended version of labelled MP is defined such that premises can include metadeclarative units. A derivation of *John knows Mary fainted* (omitting details such as tense, `use_last` etc) results in the following application of Extended MP:

$$\dfrac{\texttt{know:t -> (e -> t)} \qquad \texttt{\{mary:e, faint:e -> t\}:t}}{\texttt{know(faint(mary)):e -> t}}$$

Here `faint(mary)` is the label of `{mary:e, faint:e -> t}` and `t` is its type. For implementation purposes we chose to simplify the procedure by leaving MP as it is.

```
node(
  TreeGoal,
  NodeLabel,
  Datas,
    % a list of labelled formulas (in reverse order)
  Expecteds,
    % a list of expected nodes (in direct order)
  UnusedFormulas,
    % a list of labelled formulas not yet used in a deduction
  DeducedFormulas
    % a list of all labelled formulas, premises or derived
     )
```

The branches of the tree are described below. In [Gabbay & Kempson 1992] the construction process is modelled via the successor functions $n_1 \ldots n_i$ and $d_1 \ldots d_i$. If M is the primary database (that is, $n = M$) then $n_1(M), n_2(M)$, etc, are databases linked to M, and $d_1(x) \ldots d_j(x)$ is a sequence of declarative units which constitute a particular database at node x. The successor function n_i is implemented as the linking mechanism, and the lists `Datas`, `Expecteds` and `DeducedFormulas` between them do the work of d_i. Formulas in `Datas` are projected directly from lexical content while those in `DeducedFormulas` result from application of inference rules.

The `TreeGoal` is the goal formula of the form `Label:t`, which is set as follows:

(i) In a *primary database* the initialisation predicate `initiate_main_node` leaves `Label` basically empty, to be filled during node construction.

(ii) In a *linked database*, the lexical processing predicate `wh_treatment` attaches to `Label` the `use_gap` control.

(iii) In a *nested database*, lexical processing `process_expectations` creates a subnode (ie a database nested in the current structure) if a complement clause is 'expected' — see `Expecteds` below.

`NodeLabel` is the database label which contains nesting level, tense and gapping information, as well as any other control information.

If the primary database label includes a `gap` specification this is copied to the subnode label in the subordinate DB for nested but not linked databases. Likewise if the gap is used while constructing a nested database the feature `gap_used` is copied to the Goal Formula label of the higher database. The `gap_used` flag is percolated up the tree until a matching `use_gap` is found. This means that *wh*-dependencies are resolved in nested databases but not in linked databases.

`Datas` and `Expecteds` are built up by the lexical processing that consumes the input string. `Datas` is a list of labelled formulas or control items projected from lexical items; the formulas are passed to the deduction engine at the end of each cycle of lexical processing. Lexical items project other information such as `tense(T)`, which is copied to the database label to indicate that the whole sentence is affected by it, and `wh(Who)` which is handled by the specialised predicate `wh_treatment`.

`Expecteds` is a list of expected nodes (verb arguments) projected from lexical items, eg `expect([ 1( _, [] ): e])` as the expected complement of *see*; `expect( [ node( RG, N`

)]) as the expected complement of *know*. (In the formal system [Gabbay & Kempson 1992], expectations are handled by allowing the successor function d to generate incomplete nodes which need to be filled in by subsequent lexical content.) Expectations are processed as follows.

(i) Expected labelled formulas: if the head of `Expecteds` is a Labelled Formula (LF), it is compared against the immediately following LF. For example in the sentence *john upset mary*, processing *upset* causes the LF `l(_,[]):e` to be added to `Expecteds`. When the LF projected by *mary*, `l(mary,[]):e`, is added to `Datas`, a check is made that the formula `e` and the Rlabel `[]` match the expectation.

This mechanism is also used in auxiliary inversion to predict a subject NP. For example, when processing the sentence *who did john see*, the lexical entry for *did* projects an expectation `l( _, [use_last] ):e`. If the following item is e.g. `l(john,[]):e` then the two are combined to form `l(john,[use_last]):e`, which is added to the `Datas` list. More about this in the section on the lexicon below.

(ii) If the head of `Expecteds` is a node specification `node(RG,N)` it is handled by creating a subnode for a nested database. The nested database has the same tree structure as the primary database and inherits gapping information: if `NodeLabel` in the primary tree contains an item `gap(G)` it is copied to the `NodeLabel` of the sub-tree. The nested database traces the parse of the complement clause which is processed by calling the predicate `ldsNL` recursively.

The Deduction Engine

Deductions are carried out with the help of the data structures `UnusedFormulas` (formulas not yet consumed in a Labelled Modus Ponens step) and `DeducedFormulas` (premises or formulas generated by a Labelled Modus Ponens step). Both consist of a list of labelled formulas, initially empty. Since deductions always occur after the introduction of a labelled formula LF in `Datas`, the `deductions` predicate attempts to use LF plus some member of `UnusedFormulas` in application of Labelled Modus Ponens.

If a deduction is licensed, the deduced labelled formula is added to both `DeducedFormulas` and `UnusedFormulas` to be used as a premise in subsequent deductions; the formula used from `UnusedFormulas` is removed from that list. Otherwise LF is simply added to `UnusedFormulas`. Note that no elements are ever removed from `Datas`. Labelled Modus Ponens differs from standard Modus Ponens in that it enforces the `use_last` protocol: if one of the premises has this feature in its Rlabel then a check is made that the deduced formula is of the same type as `TreeGoal` (normally of type `t`); if this is not the case, the inference fails.

The Lexicon

A lexical entry consists of a pair of an orthographical string and a list containing a labelled formula and possibly control information. Lexical items are processed by the predicate `process_lex` which calls `analyse_lex` to decide how to update the construction tree. The simplest case is the entry for a proper name. e.g. John, which consists of the string 'john' paired with data `l(john,[]):e` where the Resource Label is empty.

```
lexicon( john,
```

```
 [
   d( l( john, [] ) : e )
 ]).
```

Verb inflection is handled by a lexical rule which splits the verb into inflection and stem, having separate lexical entries. For instance the verb-form *saw* is handled by the rule:

```
expand_lex( saw, [ '-ed', 'see '] ).
```

The entry for the suffix *-ed* contains tense information and a Labelled Formula of type `e` with an empty Content Label and the feature `use_last` in the Resource Label.

```
lexicon( '-ed',
 [
   tense( past ),
   d( l( _, [use_last] ) : e )
 ]).
```

The last data entry is to be unified with the LF representing the immediately preceding NP, i.e. the subject. This ensures that for example in processing a sentence *john saw mary*, the LF representing *john* is processed after the LF for *mary*: `upset(mary):e-> t` and `john:e` combine to give `upset(mary)(john):t`. If `john:e` were allowed to combine with the first available formula of type `e -> a` we would have the unwanted derivation

```
 john:e      upset:e-> (e -> t)
                                    
       upset(john):e -> t          mary:e
                                    
           upset(john)(mary):t
```

This can be seen as a proof-theoretic analogue of the 'obliqueness' account of grammatical relations from [Dowty 1991]. When the partial LF, i.e. one containing an anonymous variable as its Content Label, is added to the `Datas` list, predicate `legal_data_insertion` determines that it is to be unified with the current head of the list (immediately preceding NP) rather than added as a new item.

The entry for the base form *see* contains a Labelled Formula of type `(e -> (e -> t))` and the information that an argument of type `e` is 'expected'.

```
lexicon( 'see ',
 [
   d( l( see, [] ) : (e -> ( e -> t ) ) ),
   expect( [ l( _, [] ) : e
 ]).
```

Determiners are represented as labelled formulas of type `e` with an empty slot for a common noun, and nouns are represented as LFs of type `e` with a slot for a determiner. These are unified to form a single declarative unit of type `e` which represents a noun phrase.

```
lexicon( the,
 [
   d( l( the:N, [] ) : e ),
   expect( [ l( the:N, _ ) : e
 ]).
```

```
lexicon( book,
 [
   d( 1( _:book, [] ) : e )
 ]).
```

This is a deviation from a strictly type-theoretical account, with unification rather than type-inference used to process NPs. This results from a decision to treat all NPs as denoting primitive entities of type `e`.

Pronouns appear in the lexicon as names with additional control information. For example, the entry for *he* appears below.

```
lexicon( he,
[
   d( 1( pro, [pro, case:nom] ) : e )
]      ).
```

As was stated in section 1, the formal system [Gabbay & Kempson 1992] treats pronoun resolution as a free-choice mechanism which selects a suitable labelled item from outside the local database. This is only partly implemented in the prototype system, the reason being that lexical processing in a nested or linked database does not have direct access to declarative units in higher-level databases. To demonstrate how pronoun resolution interacts with *wh*-binding, the requisite information is made available in a nested database by means of a feature `wh_bind` which is inserted in the label of a nested database just in case a 'gap' has already been resolved. Pronoun resolution is handled at the lexical analysis stage: if the `wh_bind` feature is present the pronoun translates as $\texttt{pro}_{wh}\texttt{:e}$, otherwise it is given an arbitrary identifier.

The 'case' feature can be used to ensure correct morphology; it would be trivial to implement a constraint such that only `case:nom` may co-occur with `use_last` in a LF of type `e`.

In summary, the different types of lexical information are handled as follows:

(i) Labelled formulas represented as `d( 1(CLabel,Rlabel):Formula )` are copied to the list `Datas`. If `Rlabel` includes the feature `pro` then a check is made for *wh*-binding.

(ii) Tense information `tense(Tense)` is copied to the database label (`NodeLabel`).

(iii) Expectations `expect(LF)` are copied to the `Expecteds` list.

(iv) Wh-operators `wh(Wh)` are handled by the specialised predicate `wh_treatment`. This allows for two special cases:

- If *wh-* is the first word in the sentence (`Datas` is empty) it is assumed to be a *wh*-question. The Goal Formula label is updated with `use_gap` and the item `gap( 1( '#unknown#', [] ) : e)` is added to the Node Label. The embedded LF is retrieved when a 'gap' is encountered in the input string.
- Otherwise it is assumed that *wh-* is a relative pronoun. In that case a subordinate database tree is initialised and linked to the LF representing the head NP in `Datas`; the tree is built up by parsing the embedded clause. The label for the linked database contains the item `gap(LF)` where `LF` represents the head NP.

2 Examples

The following examples are selected to illustrate the processing of *wh*-gap dependencies in both questions and relative clauses, and the corresponding distinction between linked and nested databases. This section concludes with a more general discussion of crossover phenomena and an indication of how they are handled in the LDS_{NL} system.

The examples below exhibit output from the program, which is structured as in the following example of a parse of (9):

(9) Who did John upset?

(i) input string, e.g

```
>> who did john upset
```

(ii) Type of database: Root, Nested or Linked:

```
ROOT
 Node
```

(iii) Database Label and Root Goal:

```
Label : #3     tense(<(utterance)), level(1)
Goal : ? :t     gap_used, use_gap
```

(iv) List of Deduced Formulas

```
john : e     use_last
upset : e->e->t
gap(#unknown#) : e
upset(gap(#unknown#)) : e->t
upset(gap(#unknown#))(john) : t
```

(v) Goal Formula: end result of deductions.

```
Succeeds for :  upset(gap(#unknown#))(john):t
```

Examples (10) and (11) exhibit the use of linked databases in the analysis of relative clauses. (In subsequent examples we will omit details of intermediate deductions for reasons of space.)

(10) John upset Mary who fainted.

```
>> john upset mary who fainted
ROOT
 Node
   Label : #4     tense(<(utterance)), level(1)
   Goal : ? :t
      john : e     use_last
      upset : e->e->t
      mary : e

              ---------------
              Linked DB
                Label : #5    tense(<(utterance)), level(2), subnode(#4)
                Goal : ? :t   gap_used, use_gap
                   gap(mary) : e     use_last
                   faint : e->t
                   faint(gap(mary)) : t

                   Succeeds for :  faint(gap(mary)):t
              ---------------

      upset(mary) : e->t
      upset(mary)(john) : t

      Succeeds for :  upset(mary)(john):t
```

(11) Who did John upset who saw Mary?

```
>> who did john upset who saw mary
ROOT
 Node
   Label : #49     tense(<(utterance)), level(1)
   Goal : ? :t     gap_used, use_gap
      john : e     use_last
      upset : e-> e-> t
      gap(#unknown#) : e

              ---------------
              Linked DB
                Label : #51  tense(<(utterance)),
                             level(2) subnode(#49)
                Goal : ? :t     gap_used, use_gap
                   gap(gap(#unknown#)) : e     use_last
                   see : e-> e-> t
                   mary : e
                   see(mary) : e-> t
                   see(mary)(gap(gap(#unknown#))) : t

                   Succeeds for :  see(mary)(gap(gap(#unknown#))):t
              ---------------

      upset(gap(#unknown#)) : e-> t
      upset(gap(#unknown#))(john) : t
```

```
Succeeds for :  upset(gap(#unknown#))(john):t
```

Example (11) demonstrates that *wh*-gap dependencies are resolved in the local database where they are introduced, where a linked database does not count as 'local'. The first *who* causes the feature `use_gap` to be stored in the label of the Goal Formula and `gap(#unknown#)` to be stored in the node label with the identifier #49. The gap is resolved by when the second *who* is encountered at the point where the object of *upset* is expected. The subtree at the node labelled #51 is linked to the control label of the formula `gap(#unknown#):e` and the goal specification `use_gap` projected by the second *who* is resolved in the domain of the subordinate (linked) database.

(12) * Who did John upset the man who saw?

```
>> who did john upset the man who saw

no
```

The fact that *wh*-dependencies are resolved within the local database prevents example (11) from being successfully processed. At the point where the second *who* is encountered a linked database is initialised with the Goal Formula `l(_,[use_gap]):t` and with the Node Label `l( #ID, [gap(l(the:man,[]):e), level(2), subnode(#3)] )`. As in example (5) the node label of the primary database includes `gap(#unknown#)` but this is not passed down to the linked database. So the subordinate clause cannot be successfully parsed as it is impossible to derive the goal formula `?:t`.

(13) Who did John think upset Mary?

```
>> who did john think upset mary
ROOT
 Node
   Label : #52     tense(<(utterance)), level(1)
   Goal : ? :t     gap_used, use_gap

               ---------------
               Node
                 Label : #53     level(2), subnode(#52),
                                 tense(<(utterance))
                 Goal : ? :t     gap_used
               Succeeds for :  upset(mary)(gap(#unknown#)):t
               ---------------

          Succeeds for :  think(upset(mary)(gap(#unknown#)))(john):t
```

Example (12) above illustrates that the `use_gap` specification can be resolved in a nested database. Note that successful completion requires `use_gap` to be matched by a 'local' occurrence of `gap_used` and this is accomplished by percolating `gap_used` up to the primary database. The gap specification is threaded down to the nested database by copying the control item `gap(l(#unknown#,[]):e)` into the subordinate node label[4]. It is notable that

[4]Note that this does not appear in the node label in the above printout of the database, since the gap specification is removed from the node label once it has been used in order to prevent over-generation.

the resulting system presents the information needed to project this result without any ancillary nonlogical syntactic restrictions. The significance of this lies in the fact that data such as these are standardly said to require such semantically-blind syntactic restrictions. (Such claims have been familiar since [Ross 1967] but see [Chomsky 1995] for a recent restatement.) This result is achieved by the combination of analysing *wh*-expressions as a look-ahead device, imposing a filter on the output, and the analysis of relative clauses as linked databases, adjuncts to the principal database and not contained within it.

(14) Who$_i$ does John think e_i thinks he$_i$ upset Mary?

```
>> who does john think thinks he upset mary
ROOT
 Node
   Label : #162     tense(now), level(1)
   Goal : ? :t     gap_used, use_gap

               ---------------
               Node
                 Label : #163     tense(now), level(2), subnode(#162)
                 Goal : ? :t     gap_used
                                 ---------------
                            Node
                              Label : #165  tense(<(utterance)), wh_bind
                                            level(3), subnode(#163)
                              Goal : ? :t
                                 Succeeds for: upset(mary)(pro:wh):t
                             ---------------

                 Succeeds for: think(upset(mary)(pro:wh))(gap(#unknown#)):t
               ---------------

Succeeds for: think(think(upset(mary)(pro:wh))(gap(#unknown#)))(john):t
```

This is a case where the *wh*-operator appears to bind a pronoun in a complement clause. In fact the 'binding' is mediated by the process of gap-resolution: the initial *who* licenses the presence of the gap e_i in the example sentence, causing `gap(#unknown#):e` to be added to the database and `gap_used` to be inserted in the Node Label. At the point where the nested database is initialised the feature `wh_bind` is added to the Node Label, and the presence of this feature allows lexical analysis to translate *he* as `(pro:wh):e`.

(15) Who$_i$ does John think he$_{j/*i}$ thinks e_i upset Mary?

```
>> who does john think he thinks upset mary
ROOT
 Node
   Label : #169     tense(now), level(1)
   Goal : ? :t     gap_used, use_gap
                  ---------------
            Node
              Label : #171     tense(now), level(2), subnode(#169)
```

```
Goal : ? :t     gap_used
                              ---------------
          Node
            Label : #173  tense(<(utterance)), level(3),
                          subnode(#171)
            Goal : ? :t     gap_used

            Succeeds for: upset(mary)(gap(#unknown#)):t
          ---------------

Succeeds for: think(upset(mary)(gap(#unknown#)))(pro:#172):t
---------------


Succeeds for: think(think(upset(mary)(gap(#unknown#)))(pro:#172))(john):t
```

This example contrasts with the previous one in that *he* may not be interpreted as dependent on *who* (as indicated by the subscript $_{j/*i}$), even though it is within the c-command domain of the *wh*-operator. In fact this restriction falls out naturally if we assume that *wh*-items do not add content to the database but license the creation of a data item at the point where the gap is resolved. In this case the gap comes after the pronoun and so there is nothing to set up a suitable resolvent. This is indicated by attaching an arbitrary ID to the label of the declarative unit representing the pronoun, in this case the numeral #172. In actual fact the pronoun in this example could be resolved to *John*, but this is not catered for in the current state of the implementation. The reason for this is simply that in the prototype system operations on a nested database do not have direct access to higher-level databases, which is why the feature-passing mechanism has been set up for *wh*-binding.

2.1 Crossover Phenomena

This interaction between *wh* and pronoun construal is part of a larger phenomenon known as crossover, which continues to prove puzzling under orthodox perspectives [Chomsky 1985, Aoun & Li 1993, Pollard & Sag 1994] in which the *wh*-expressions are analysed as variable-binding operators albeit with idiosyncratic binding properties. The principal difficulty is not simply as we have seen that *wh* does not behave like a regular quantifier, binding any pronominal that it c-commands, but that the restrictions associated with these binding properties are partially context-sensitive. In general, the correlation between some initial *wh*-expression and the "gap" where its interpretation is projected, is identical in the two different environments of *wh*-question and *wh*-relative. For example, restrictions associated with *wh*-gap binding displayed in a question such as (14) are equally displayed in relative constructions. (15) is no less illformed than (14), for the same reason:

(16) *The book which John upset the man who criticised e was very poor.

However in crossover constructions this parallelism between relatives and questions breaks down. In questions, the sequence '*wh* ... pronoun ... gap' is invariably precluded, as (14) and (16) show:

(17) *Who$_i$ did his$_i$ mother ignore e_i?

But in relatives, the restriction is sensitive to the type of pronominal. Though the relative construction in (17) also precludes the construal of the pronoun as dependent on the *wh* (the so-called "strong crossover" restriction), the relative in (18), in which the pronoun is in a determiner position freely allows such dependency:

(18) *John$_i$, who$_i$ Sue thinks that he$_i$ believes that Bill liked e_i, fell ill during the exam period.

(19) John$_i$ who$_i$ his$_i$ mother regularly ignored e_i, fell ill during the exam period.

Furthermore, the crossover restriction extends to cases in which the *wh*-expression is contained within some larger expression and it is this and not the pronoun that is coindexed with the gap:

(20) Whose$_i$ exam results$_j$ e_j confirmed that he$_i$ was better than everyone else?

(21) *Whose$_i$ exam results$_j$ was he$_i$ certain e_j would be better than everyone else's ?

Again the pronoun must follow the gap if it is to be construed as "bound" by the *wh*-expression even though the pronoun and gap are not construed as bound by the same operator (the *wh*-operator is contained within the expression which binds the gap). These cases too are context sensitive, with the restriction that precludes (20) disappearing in the very same type of relative as the so-called "weak crossover" cases:

(22) John, whose$_i$ exam results$_j$ he$_i$ had been certain e_j would be better than everyone else's, failed dismally.

These data have proved recalcitrant for orthodox binding analyses, some of which have postulated up to 4 discrete phenomena to account for such interaction [Lasnik & Stowell 1991, Postal 1993].

The present analysis allows a straightforward account of the data. The starting point is the data in questions already displayed in (13) and (14), where there is no variation according to context. Because the initial *wh*-expression projects a goal specification, it is a mere target on the outcome, a license for some assumption to be constructed later, and so cannot be visible for anaphoric dependency. Once the assumption necessary to meet that target has however been constructed, it is visible for purposes of anaphoric dependency within the domain within which that assumption has been constructed. Hence the pattern (22)-(25):

(23) *Who$_i$ does Joan think that he$_i$ worries e_i is sick ?

(24) Who$_i$ does Joan think e_i worries he$_i$ is sick ?

(25) *Who$_i$ does Joan think that his$_i$ mother worries e_i is sick ?

(26) Who$_i$ does John think e_i worries his$_i$ mother is sick ?

In relative clauses, it is the linking of information in the two databases that makes a critical difference, though again, the analysis of *wh* as a goal specification is central. When a second database is linked to a first through an initiating *wh*-expression, it is the *wh*-variable identified with the corresponding term in the first database, and its status as a goal specification in the second database, which provides a means of carrying down the information from the

first database through the construction of the second database. As before, the information projected by the instruction `wh_bind` projected as a goal specification is copied into each database nested as a subpart of some initial containing database. The effect is that a fully identified declarative unit will be treated as projecting a putative entry in each subordinate database it gets carried through in the building up of this linked database. And with this information being locally available in each of these subordinate databases, we can explain the context-sensitivity of the crossover phenomenon as a locality clash between a pronoun within that database and the information in the presented goal specification. The locality condition imposed by the pronoun is the restriction that a pronoun, if functioning as a minor premise in a given database, may not select as antecedent the label of any other minor premise of type `e` in that database. Because *who*, identified as 'John$_i$' in the interpretation of (17) and carried down from one subordinate database to the next, projects a putative entry of the form `John`$_i$`:e` in each succeeding database, a pronoun projecting an incomplete premise of type `e` in any such database will not be able to use the information projected by the *wh*. Correctly in these *wh*-structures a clash is predicted between the *wh*-expression and a pronoun in the database being built if the pronoun precedes the gap. Hence (17) is precluded. And, also correctly, no clash is predicted in structures such as (18) and (21) in which either the pronoun or the *wh*-element is internal to a determiner.

The success of these predictions turns on the incremental nature of the interpretation process. The data are predicted because the analysis not only defines a relation between the initial *wh*-expression and the position from which its semantic contribution is defined, but also defines the information available at each step in between the projection of that initial position and the position of the gap. So from the twin assumptions that *wh*-initial expressions project a goal specification and that the linking of the concept of linked databases provide a means of transferring of information from one database to another, the LDS$_{NL}$ system succeeds in predicting data which remain puzzling from more orthodox nonprocedural perspectives.

3 Comparison with Other Type-Deduction Systems

3.1 The Categorial Landscape

Incorporating as it does the concept of type deduction, the LDS$_{NL}$ system falls within the family of categorial grammar systems all related in various ways to the Lambek calculus.

According to the use made of the Lambek calculus in categorial grammar, the family of syntactic types is defined in terms of the two primitive types, n - denoting entities, s - denoting truth-values, with the two order-sensitive connectives left- and right- application '\', '/' and any combination of 'a/b', 'a\b' for 'a' a type 'b' a type. As formulas in a well-founded logical system, syntactic and semantic properties are defined strictly in tandem. A string of category 'a/b' combines with a string of category 'b' to its right to yield a string of category 'a', a move which is paralleled by functional application in the semantics. The result is a strong compositionality requirement that denotations of all complex expressions are based solely on denotations of the constituent parts and their mode of combination, a restriction which constitutes the heart of a categorial system. Modifications of this Lambek system have involved departures which loosen the tightness of the syntax-semantics match. We give first an overview of the types of modification that have been proposed, and then an indication of how LDS$_{NL}$ fits within this family of systems.

The first problem facing a Lambek-based system is how to characterise the discontinuity

phenomena prevalent in natural language. The rigidity of the connectives '/', '\', allows discontinuity effects only when the "gap" position is left- or right- peripheral; and a number of alternative ways of characterising non-peripheral gaps have been explored:

(i) The directionality of the connectives '/', '\', can be removed, replacing them with a semi-directional or direction-neutral $\rightarrow$, and characterising linear order restrictions either within the prosodic algebra directly [Oehrle 1994], or as part of the construction algorithm [Gabbay & Kempson 1992, Doerre et al 1996].

(ii) The directional operators are retained but additional operators are defined such as permutation, associativity or more recently infixing operations which locally remove their effect [Moortgat 1988, Morrill 1994].

(iii) Additional inference postulates are added to the axioms that define a basic set of combinatorial operators, which have the effect of mapping one string sequence into another [Steedman 1993].

(iv) Combinatorial processes over units larger than individual typed expressions can be defined, giving a mixed-level system [Joshi & Kulick 1995, Gabbay & Kempson 1992, Lecomte & Retore 1995]. This last alternative allows operations which may interpolate one such macro-unit within another in ways redolent of Tree-Adjoining Grammars (TAG).

The syntactic problem of discontinuity is part of the more general problem of how to preserve a concept of compositionality of meaning given the uncontroversial assumption that natural languages have denotational semantics. On the one hand, the strict syntax-semantics correspondence is not fully preserved in natural language phenomena. Not all syntactic operations induce a corresponding denotational operation; and some syntactic restrictions appear to have no denotational reflex. On the other hand, natural language expressions display a context-dependence in their interpretation which is not expressible without some extension of the denotational semantics carried over from orthodox formal logics (which by definition display no such variation for a single string). Furthermore language interpretation takes place in real time, and interpretation is built up on a left-right basis.

At least two forms of modification have been envisaged within the family of categorial systems to address these problems.

(i) Modes of syntactic combination can be licensed which do not correspond to semantic operations. These may carry out syntactic operations which have no semantic reflex, eg permutation, or they may define domains within which semantic operations take place, enabling syntactic restrictions without semantic consequence to be defined [Hepple 1990, Morrill 1994].

(ii) Some categorial systems impose an additional requirement of left-right compositionality over the string as a whole [Steedman 1993, Milward 1994, Milward 1995], making use of type lifting and composition of function operators to define types from which a denotational semantics can be incrementally projected. The infinity of types for a linguistic string allowed by all pure categorial formalisms, the so-called phenomenon of "spurious ambiguity", is taken to be problematic, but not prohibitive [Hepple 1990, Moortgat 1995, Hendriks 1993].

Further, more radical, liberalisation is required if the system is to incorporate the widely-observed partiality of information presented by natural language expressions. This phenomenon poses two distinct types of problem. On the one hand, at any sub-point in the interpretation process, the intrinsic content of the expression may not be sufficient to determine its structural role within the configuration in which it occurs. On the other hand, the encoded content of any individual expression may not fully determine the denotation to be assigned to it under an interpretation, but it may contribute, rather, to the way in which interpretation is built up (as in the data considered in section 2 above). Phenomena such as these conflict with the compositionality requirement strictly construed. Two solutions might be envisaged within a broadly categorial system: [5]

(i) A denotational semantics can be preserved, while using the syntax nontrivially to narrow down the class of interpretations made available by an over-broad semantic characterisation. This approach [Milward 1994] allows the partiality of information projected at any subpa rt of the interpretation process to be captured as an appropriate semantic abstraction (unspecified as to possible syntactic forms of completion).

(ii) The possibility of a new definition of semantics might be explored, enriching the denotational semantics with forms of interpretation which reflect the incremental procedure whereby the interpretation is built up. (cf. the work by Lecomte and Retore [Lecomte & Retore 1995]). It is this latter route which is being explored in ongoing LDS_{NL} work [Gabbay, Kempson & Meyer Viol (in prep)].

3.2 LDS_{NL} and Other Current Models Compared

LDS_{NL} departs from Lambek assumptions in allowing operations other than type deduction, in adopting an explicitly procedural perspective, and also in assuming a structural approach to interpretation. It licenses two modes of combinatorial operation, that defined as steps of type deduction and that defined as operations on sets of premises taken as a unit (a database). These structural operations are explicitly defined to be part of the interpretation process, and an ineliminable level of structure is assumed, the projection of which requires a process-oriented form of semantics. With the shift of the semantics to a procedural perspective, a stricter form of the compositionality principle can however be retained, with the assumption of systematic word by word incrementation of structure as interpretation. A relatively orthodox model-theoretic evaluation can then be defined in a familiar bottom-up way upon the output of this structural process, preserving assumptions about the form of such a semantics but defined n ot for the natural language string directly, but rather for the wellformed formula onto which the parsing process defines a projection.

Several consequences follow from this shift. Because the compositionality requirement is now met at the level of building an appropriate logical structure, there is no need to impose a left-right bottom-up projection of model-theoretic content on the string (as guaranteed by the '/', '\' connectives). To the contrary, this requirement is abandoned in favour of an assumption of strictly incremental accumulation of structure from lexical input. Furthermore

[5] The problem of compositionality has been a primary focus of attention since the work of Barwise, Perry and Kamp in the early 1980's [Kamp 1981, Barwise & Perry 1983]. However work in Situation Theory and DRT does not in the main seek to sustain a matching of syntactic and semantic generalisations.

type deduction no longer needs to be defined to be sensitive to linear order as in the Lambek system. Nor is there any call to assign higher type specifications to enforce left-right compositionality as in the Steedman or Milward systems [Steedman 1993, Milward 1995].

The essence of the LDS$_{NL}$ account is that it procedurally builds up structure as interpretation in a top-down (and structural) way in which information about the Wh-phrase can be made available at intermediate points in virtue of the way information is accumulated and transmitted between the initial position and the position of the "gap". Indeed this is what critically distinguishes our account of "crossover" from others available (including the categorial account of crossover). The categorial account of Wh is a classical binding account, which can manipulate information about the Wh-expression only at the position of the gap and the position of the binding operator.

Of the number of categorial type-logical formalisms available, the LDS$_{NL}$ formalism is most closely related to Milward's implementation of incremental parsing algorithms for categorial grammar and dependency grammar, in its conceptualisation of the relationship between grammar and parser. [Milward 1994] presents a parsing algorithm for Lexicalised Dependency Grammar in terms of a dynamic specification of the state transitions licensed by the grammar. It turns out that the dynamic specification is sufficiently general to form the basis of a grammar in its own right, Dynamic Dependency Grammar, where the resulting grammar generates transition sequences rather than parse trees. LDS$_{NL}$ is also related to the term-labelled categorial type system of [Oehrle 1994]. Both systems make explicit use of the Gabbay LDS methodology, using labels not merely as a semantic reflex of steps of type-deduction, but as an additional control mechanism. There are also similarities with the TAG-influenced system of Joshi and Kulick [Joshi & Kulick 1995]. Like LDS$_{NL}$, Joshi and Kulick restrict the problem of spurious ambiguity faced by

systems which allow associativity, permutation, and infixing operations by replacing the simple mono-level type-deduction system with a richer inference system, allowing inference to be defined over sets of premises, taken as proof objects, in addition. In particular adjunct structures, and raising constructs are defined in this way. There are also interesting correlations with Lecomte and Retore who define words as projecting partial proof objects, as Joshi and Kulick but in an explicitly linear logic (POMSET) framework [Lecomte & Retore 1995].

There is also at least a superficial parallelism between the LDS$_{NL}$ formalism and the more orthodox Flexible Categorial Grammar (FCG) formalisms such as [Morrill 1994]. Like the LDS$_{NL}$ formalism, Morrill uses a natural-deduction format for displaying the projection of prosodic string plus semantic content, (moreover he uses a system of labelling to present this ordered pair of information at each stage of the projection). Furthermore, the projection of *wh*-containing strings is characterised in terms of the licensing of constructed assumptions, with required discharge of all such assumptions by appropriate steps of conditional introduction immediately prior to their combination with some suitable operator. The LDS$_{NL}$ system might therefore be construed as a procedural analogue of the natural deduction variant of the FCG formalism. However FCG is committed to the strongest form of syntax-semantics correspondence, with the syntax characterised exclusively in the model-theoretic terms presented by the two dimensions of interpretation - prosodic and denotational. Against this background, the interpretation of *wh* expressions within FCG has to explained exclusively in terms of the interpretation of the position filled by the assumption as determined by the *wh* operator which binds it, for this is the only dimension of interpretation which the formalism makes available. Because the syntax is epiphenomenal, no reference can be made to structural properties of intermediate steps in the process of compiling that interpretation. The claim

that such reference is essential to characterising the context-dependency effects of crossover therefore remains a puzzle yet to be addressed within the FCG formalism. More generally, FCG has not addressed the issue of the context-dependency intrinsic to natural language interpretation, which is the principal focus of LDS_{NL}.

4 Issues to be addressed

In comparison with FCG formalisms, the system outlined here faces a major problem — the definition of an appropriate semantics. The language needed to define the type-logical formulas is unproblematic. It comprises a set with just two primitives combined within a logical system whose only connective is $\rightarrow$ (and possibly :). Such a system is familiar — an impoverished relative of much richer current categorial systems [Morrill 1994, Moortgat 1995, Hepple 1990, Hepple 1995]. The language of the labelling algebra is however extremely heterogeneous. A quantifier-free predicate logic expresses concepts associated with natural language expressions. But there are a number of additional features of this labelling language, none of which is unproblematic.

Firstly, there are metavariables whose status within the predicate logic language remains to be defined. Secondly, the labelling algebra also contains control specifications such as 'use me last' and a range of restrictions on how to instantiate individual metavariables, whose semantics will have to be in terms of the process of building the tree structure. Thirdly, entire databases can act as labels; and this involves a blurring of the label-formula distinction, since type-logical formulas have a role not only within the formula system but also (within a database) in the labelling algebra. Without an associated semantics for this complex system, its formal properties remain unclear. Yet the heterodox nature of the various labelling devices makes the possibility of a revealing semantics seem remote.

Confronted with this problem, more recent work of Gabbay, Kempson & Meyer-Viol has led to a revised system, in which predicate algebra expressions are built up as formulas, and all controls on that operation, including type specifications, are defined in the labelling algebra, whose sole purpose is to drive the incremental construction of a logical formula as interpretation for the natural language string. The semantics of the revised label-formula pairs is exclusively procedural. It is only its output, a form of epsilon calculus, which has an orthodox model-theoretic semantics. A partial implementation of this system has been developed; see [Meyer Viol et al 1997].

Despite the shift of balance as between label and formula, the underlying motivation of the earlier system is carried over unchanged in this revised system. The goal remains that of modelling the process of natural language understanding, with the accompanying commitment to modelling the asymmetry between some relatively weakly specified linguistic input and the representation corresponding to some assigned interpretation. The interpretation of natural language expressions qua expressions is, as before, strictly syntactic — a set of meta-level procedures on the building of logical configurations. In so far as it is successful in predicting natural language data, this LDS_{NL} system is of interest in being a parser without any independent grammar. The concept of structure defined for the natural language system is by definition the structure of the parse process through which a configuration corresponding to a familiar model-theoretic concept of interpretation is projected. Syntax here is no more and no less than the internal structure associated with the deductively driven parsing mechanism. The significance of the model will lie in whether this claim remains sustained

across a broader array of "syntactic" data than so far addressed.

References

[Aoun & Li 1993] Aoun, J & Li, A. 1993. *Wh*-elements in situ: syntax or LF? Linguistic Inquiry 24.2, pp.199-238.

[Barwise & Perry 1983] Barwise, J & J Perry, 1983. Situations and Attitudes.

[Broda & Finger 1995] Broda, K & M Finger, 1995, KE-Tableaux for a Fragment of Linear Logic, in Proc 4th International Workshop on Analythic Tableaux and Related Methods. Koblenz.

[Broda, Finger & Russo 1996] Broda, K, M Finger & A Russo, LDS: Natural Deducton for Substructural Logics, Submitted to the Journal of the IGPL, Special Issue Workshop "Logic, Language, Information and Computation", 1996.

[Chomsky 1985] Chomsky, N. 1985. Knowledge of Language. Praeger.

[Chomsky 1995] Chomsky, N. 1995. On minimalism. MIT Press, Cambridge MA.

[D'Agostino & Gabbay 1994] D'Agostino, M & D Gabbay, 1994, A generalisation of analytic deduction via labelled deductive systems. Part I: Basic substructrual logics. *Journal of Automated Reasoning 13.*

[Doerre et al 1996] J.Doerre, E.Koenig, D.Gabbay, 1996. Fibred semantics for feature-based grammar logic. Journal of Logic, Language and Information vol.5 nos.3-4. pp.387-422.

[Dowty 1991] Dowty, D. 1991. Thematic proto-roles and argument selection. Language

[Gabbay 1996] Gabbay, D. 1996. *Labelled Deductive Systems, vol. 1*. Oxford University Press.

[Gabbay & Kempson 1992] Gabbay, D. & Kempson, R. 1992. 'Natural-language content: a proof-theoretic perspective' Proceedings of 8th Amsterdam Semantics Colloquium. Amsterdam.

[Gabbay, Kempson & Meyer Viol (in prep)] Gabbay, D., Kempson, R. & Meyer-Viol, W. 'Labelled deduction for natural language understanding'.

[Gabbay, Kempson & Pitt 1994] Gabbay, D., Kempson, R. & Pitt, J.V. (1994) 'Labelled abduction and relevance reasoning' in Demolombe & Imielinski, T. (eds.) Nonstandard queries and nonstandard answers, 155-96. Clarendon Press, Oxford.

[Hendriks 1993] Hendriks H. 1993. Studied Flexibility. Ph.D ILLC Amsterdam.

[Hepple 1995] Hepple, M. Hybrid Categorial Logics. IGPL Vo.3 Nos.2-3, pp.343-55.

[Hepple 1990] Hepple, M. 1990. The Grammar and Processing of Order and Dependency: A Categorial Approach. Ph.D Edinburgh.

[Joshi & Kulick 1995] Joshi, A and S Kulick 1995. Partial proof trees as building blocks for a categorial grammar. (Revised version Jan. 1996 skulick@linc.cis.upenn.edu, joshi@linc.cis.upenn.edu).

[Kamp 1981] Kamp, H, 1981. A Theory of truth and semantic representation. In J. Groenendijk et al (eds), Formal Methods in the Study of Language, Mathematisch Centrum, Amsterdam.

[Kempson 1995] Kempson, R. 1995. 'Ellipsis: a natural deduction perspective' in ed. Kempson, R.Bulletin of The Interest Group of Pure and Applied Logics: special edition on Language and Deduction. Vol.3, nos.2-3.

[Kempson 1996] Kempson, R. 1996. 'Semantics, pragmatics, and natural-language interpretation' in Lappin, S. (ed.) Handbook of Contemporary Semantic Theory. Blackwell.

[Kempson & Gabbay forthcoming] Kempson, R. & D.Gabbay (forthcoming) Crossover: a unified account. Journal of Linguistics.

[Lasnik & Stowell 1991] Lasnik, H. & Stowell, T. 1991. 'Weakest crossover'. Linguistic Inquiry 22:687-720

[Lecomte & Retore 1995] Lecomte and Retore 1995. Pomset Logic as an alternative categorial Grammar. Formal Grammar. Proceedings of the Conference of the European Summer School in Logic, Language and Information, Barcelona. email lecomte@shm.grenet.fr, retore@loria.fr

[Meyer Viol 1995] Meyer-Viol, W. 1995. Instantial Logic: An Investigation into Reasoning with Instances. ILLC Dissertation Series 1995 -11. ILLC. Amsterdam.

[Meyer Viol et al 1997] Meyer Viol W, R Kibble, R Kempson & D Gabbay, 1997, Indefinites as Epsilon Terms: A Labelled Deduction Account, in Bunt, Kievit, Muskens & Verlinden (eds), *Proceedings of the Second International Workshop on Computational Semantics*, Tilburg University.

[Milward 1994] Milward D, 1994. Dynamic Dependency Grammar. Linguistics and Philosophy 17.6:561–606.

[Milward 1995] Milward D, 1995. Incremental interpretation of categorial grammar. EACL proceedings.

[Montague 1974] Montague 1974 Formal Philosophy. Yale University Press.

[Moortgat 1995] Moortgat 1995. Multimodal Linguistic Inference IGPL. Vol.3, Nos.2-3, 371-401

[Moortgat 1988] Moortgat, M. 1988. Categorial Investigations. Foris. Dordrecht.

[Moortgat 1992] Moortgat, M. 1992. 'Labeled deductive systems for categorial theorem proving' in Dekker, P. & Stokhof, M. (eds.) Proceedings of Eighth Amsterdam Colloquium.

[Moortgat 1995] Moortgat, M. 1995. 'Residuation in mixed Lambek systems' in Kempson, R. (ed.) Language and Deduction. Special Issue of Bulletin of the Interest Group in Pure and applied Logics. Vol. 3, Nos. 2-3. London.

[Morrill 1990] Morrill, G. et al 1990.'Categorial deductions and structural operations' in Barry, G. & Morrill, G.(eds.) Studies in Categorial Grammar. Edinburgh Working Papers in Cognitive Science. Edinburgh.

[Morrill 1994] Morrill, G. 1994. Type-logical Grammar. Kluwer.

[Morrill 1995] Morrill G. 1995. Clausal proofs and discontinuity. IGPL. Vo. 3. Nos2-3, pp.403-27

[Oehrle 1994] Oehrle, R. 1994. Term-labelled categorial type systems. Linguistics and Philosophy 17.

[Oehrle 1995] Oehrle, R. 1995. 'Some three-dimensional systems of labelled deduction. IGPL. Vol.3, Nos.2-3, 429- 48.

[Pereira & Warren 1980] Pereira, F & D Warren, 1980. 'Definite clause grammars for language analysis', Artificial Intelligence 13: 231-278.

[Pollard & Sag 1994] Pollard, C. & Sag, I. 1994. Head-Driven Phrase-Structure Grammar. University of Chicago, Chicago & London.

[Postal 1972] Postal, P. 1972. Crossover Phenomena. New York: Holt, Rinehart & Winston.

[Postal 1993] Postal, P. 1993. 'Remarks on weak crossover effects' Linguistic Inquiry 24, 539-56.

[Ross 1967] Ross, J.R. 1967. 'Constraints on Variables in Syntax' PhD dissertation, Massachusetts Institute of Technology.

[Russo 1996] Russo, A.M. 1996, 'Modal Logics as Labelled Deductive Systems', PhD dissertation, Imperial College, University of London.

[Steedman 1993] Steedman, M. 1993. Categorial grammar. Tutorial overview'. Lingua 90. 221-258.