Implementing LDS$_{NL}$: Strategies for Pronoun and *Wh*-Gap Resolution

RODGER KIBBLE

Abstract

This paper reports on a computational implementation of the LDS$_{NL}$ system, focussing on application to the phenomenon of 'crossover' in English wh-questions and relative clauses. Following a brief outline of the underlying formalisms, we discuss control strategy, transition rules and lexical structure, and demonstrate that different runtime options involving interaction between pronouns and *wh*-gaps lead to varying reults with marginally acceptable sentences while producing consistent results for clear cases.

1 Introduction

This paper reports on a computational implementation of the LDS$_{NL}$ system as set out in recent theoretical papers by Ruth Kempson and associates (Kempson MS; Meyer Viol & Kempson 1996, 1997; Meyer Viol et al 1997). Discussion will focus on the particular application to the phenomenon of 'crossover' in English as exemplified in the following contrast:

1. (a) *John$_i$, who$_i$ Sue thinks he$_i$ knows Mary likes e_i, ignores Mary.

   (b) John$_i$, who$_i$ Sue thinks e_i knows Mary likes him$_i$, ignores Mary.

   (See Appendix A for an inventory of varieties of 'crossover')

This is not the place for detailed linguistic analysis or a comprehensive exposition of the formal system, though the general architecture is outlined in Section 1.2, with references to relevant papers. The underlying philosophy of the system is to model knowledge of language *dynamically* in the form of a deductive parser which operates left-to-right, incrementally constructing a binary function-argument tree representing the logical structure of a sentence. Decisions which determine whether a string is acceptable, such as the interpretability of semantically underspecified items such as pronouns and 'gaps', are made on the basis of processing information which is available at the point where the item is encountered rather than in terms of configurational facts such as 'c-command' etc. This approach has been applied in some detail to the problem of crossover phenomena (Kempson MS) as well as quantifier scope and term dependencies (Meyer Viol et al 1997) and ellipsis.

While the theoretical papers cited above provide an abstract specification for a family of deductive parsers, the process of implementation has necessitated decisions on control strategy, lexical processing and various linguistic issues, which are of independent interest. Furthermore the implementation not only allows for testing of the particular claims made for the theory but also allows experimentation with different strategies for resolution of pronouns and other partially-specified items, which turn out to have interesting empirical consequences. To anticipate the discussion in section 3.2: the argument in Kempson (MS) is that a relative pronoun as in (1a/b) induces a tree node with an initially unfixed position in the tree and semantic content determined by the head NP. In a grammatical sentence like (1b) the unfixed node fills the 'missing' argument position corresponding to the 'gap' e_i. Example (1a) fails because the requirement for a tree node with a fixed position and identical content to the head NP is satisfied by resolution of the pronoun he_i and so the content of this unfixed node is 'merged' with the interpretation of the pronoun and is no longer available at the point where the 'gap' has to be processed. Kempson argues further that this principle leads to the prediction that *resumptive pronouns* are freely available in English and that this is correct, subject to a pragmatic '*Avoid Pronoun*' principle:

2. ?The student$_i$ who Sue says he$_i$ should have failed has turned up to class.

I will show that the implementation allows for various ways of 'cashing out' the intuitive requirement to merge representations of a pronoun and an unfixed node, and that the crossover data can also be handled without the consequence of admitting resumptive pronouns. This may provide a way of modelling dialectal and cross-linguistic variation.

1.1 Coverage

It should be made clear that the system is not claimed to be a general-purpose, wide-coverage parser but is intended as a demonstration system to exhibit the distinguishing features of the underlying formalism and particularly the way they interact to provide novel solutions to long-standing linguistic problems. So we have not focussed on core issues in parsing technology such as efficient handling of ambiguous input, for which various techniques are already well-established (see e.g. Tomita 1986), or agreement phenomena, which have received a natural treatment in unification-based frameworks. In fact we adopt the typed-feature formalism ProFIT (Erbach 1995) to handle pronominal and other forms of agreement.

The implementation has been tested with a selection of the example sentences in Appendix A and various runtime options, and consistently makes the desired predictions for examples 2, 10, 12, 13, 14, 16, 17 and 20 and accepts or rejects the 'borderline' example 18 and the 'resumptive' examples 21 and 27 depending on runtime options; see section 3.2 for detailed discussion.

1.2 Background: outline of the formalism

This document is intended to be self-contained but may be read in conjunction with theoretical papers by Kempson and others already cited. A prototype implementation of the earlier formulation of (Gabbay & Kempson 1992, Kempson 1996) is described in (Finger et al forthcoming), while the program described in this paper is an extension of a partial implementation reported in (Meyer Viol et al 1997).

Rather than analysing (surface) syntactic structure, we generate 'incrementally' a logical representation consisting of a (possibly singleton) set of linked binary trees whose nodes are decorated with composite expressions (*declarative units*) consisting of type-logical formulas, λ-expressions including epsilon terms (Meyer Viol 1995, Meyer Viol et al 1997) as arguments in quantifier-free formulas and other features to do with tree node addressing, agreement etc. Terms from the ϵ-calculus have the following semantics:

> The term $\epsilon x\phi$ denotes some arbitrary d in the domain which has the property ϕ, if there are any such objects, and an arbitrary d *tout court* if there are no such objects. Given this choice of d, it is evident that $\phi[\epsilon x\phi/x]$ is true precisely if $\exists x\phi$ is... We interpret the term $\tau x\phi$ as shorthand for $\epsilon x\neg\phi$, i.e. it denotes an arbitrary object d s.t. d does *not* have the property ϕ if there is such an object, otherwise an arbitrary d. The following equivalence holds: $\forall x\phi \leftrightarrow \phi[\tau x\phi/x]$
>
> (Meyer Viol et al 1997)

The parser is an instance of a Labelled Deductive System in the sense of (Gabbay 1996) in that a proof proceeds applying various inference regimes in parallel to the component items in a composite expression. The final logical form is derived by successive function application between sister nodes (and linked nodes if present).

The parse process essentially consists of the progressive 'completion' of a partial tree structure. This involves use of the modal tree description language LOFT (Blackburn & Meyer Viol 1994) extended with the LINK relation of (Gabbay & Kempson 1992): relations between nodes include $\langle d\rangle$ 'daughter' ($\langle d_0\rangle$, $\langle d_1\rangle$ are first and second daughters respectively), $\langle u\rangle$ 'mother', $\langle L\rangle$ 'link', with their reflexive transitive closures $\langle d\rangle*$, $\langle u\rangle*$. So, $\langle d\rangle * \phi$ is read as "ϕ holds at the current node or somewhere below it".

Tree nodes are addressed as follows:

a. $Tn(0)$ is a root node address

b. If $Tn(n)$ is an address, $Tn(n0)$, $TN(n1)$ are daughter nodes of $Tn(n)$

c. If $Tn(n)$ is an address, $Tn(nL0)$ is the root node of a subtree 'linked' to n, where *link* is a relation which is not reducible to standard configurational relations.

d. If $Tn(n)$ is an address, $Tn(n@)$ is the address of a node 'dominated' by $Tn(n)$, where @ is replaced by a (possibly empty) sequence of binary integers.

So the following equivalence holds:

$$Tn(n), \langle d \rangle * \phi \leftrightarrow Tn(n@)\phi$$

Each tree node has an associated *task* which is represented by a data structure which keeps track of the current tree node address, the logical type required at that address and semantic content as represented by various logical formulas, as will shortly be described. A set of task states makes up a *parse state*, and the parser specification includes a series of transition rules which license sequences of parse states. The implemented transition rules are described in section 2.3 For a detailed specification see e.g. (Meyer Viol & Kempson 1997).

2 Implementing the formalism

The program consists of three modules:

LDSNL contains the transition rules, flow of control and interfaces to **Apply** and **Lexicon**

Apply implements function composition and construction of ϵ-terms

Lexicon : lexical entries define a relation between pairs of string positions s and partial trees T such that $\langle s,\ T \rangle \rightarrow \langle s',\ T' \rangle,\ s \leq s'$ if the word at s matches the lexical item.

The system is implemented in Sicstus Prolog 3.5 and ProFIT (Erbach 1995), and runs with MS Windows 95 and Unix (Solaris). It is intended that the Prolog code will be made publicly available via the WWW.

At this point the reader may be concerned about the computational implications of incorporating a modal logic into the formalism. In fact it has not turned out to be necessary to implement a complete theorem-prover for the full tree logic, and it is not yet clear whether the full expressivity of LOFT is required for the particular applications we have so far investigated. Only a subset of the definable relations are explicitly made use of in this implementation, namely $\langle d \rangle$, $\langle L \rangle$ and the recursively defined $\langle u \rangle *$, and they have an operational semantics defined by the transition rules where they are explicitly mentioned (see section 3.2 below). The **Introduction**, **Subgoal**, **Elimination** and **Completion** rules employ $\langle d \rangle$, $\langle L \rangle$ features in **Gap Introduction**, **Adjunction**, **Elimination** and

Completion, and $\langle u \rangle *$ is involved in **Gap Resolution** where it is implemented as the following Prolog predicate:

```
upstar(N1, N1).
upstar(N1, N2):-
  up(N1, N3),
  upstar(N3, N2).
```

This appears to be operationally equivalent to the **dominates** relation of REFS.

2.1 Data Structures

D1.: Current task states represented as unit clauses in Prolog database, manipulated using assert/retract:

`task(ID, tn(Node), show(Goal), Todo, Done).`

ID: task identifier (integer)

Node: tree node address as a reverse list of binary integers:

- **[0]** is an address
- **[n|Path]** is an address, n ∈ {0,1} if **Path** is an address.
- **[@|Path]** is an address,
- **[0,link|Path]** is an address.

Goal: type requirement for current task

ToDo: list of requirements

Done: partial or complete Declarative Unit (DU) formula

D2. DUs are represented as (ordered) lists:

`[ty(Type), fo(Formula), Agr]`

Type: type-logical formula over `(e, cn, t, ->)`.

Formula: possible values are $\epsilon|\tau$-term or λ expression.

Agr: feature matrix implemented as ProFIT term, for agreement

D3. λ-Formulas are represented as `fo(l(Vars), Qff, Terms)` where `Vars` is a sequence of abstracted variables (possibly empty, shown as `l([ ])`), `Qff` is a quantifier-free formula constructed from a set of predicates R^n, connectives `and, ->, not`, individual variables x_n and terms $T_n \in$ `Terms`.

D4. Terms: `t(op(ep|tau), v(Var), dep(_), f(Qff))`. NB: the operator `dep(_)` has to do with term dependency and quantifier scope; this feature is not relevant to the concerns of this paper and is temporarily disabled in the current implementation.

2.2 Transition Rules

The rules set out below define a transition between parse states $\mathcal{PS} \rightarrow \mathcal{PS}'$ such that $\mathcal{PS}'$ differs from $\mathcal{PS}$ at most in that:

(i) There is some task state $T_i' \in \mathcal{PS}'$ such that there is no corresponding $T_i \in \mathcal{PS}$; or

(ii) One or more task states $T_i, \ldots T_n \in \mathcal{PS}$ are replaced by new task states $T_i', \ldots T_n' \in \mathcal{PS}'$

(The parse state also includes a 'book-keeping device' which keeps track of the current string position and task state; this is suppressed in what follows.)

The parsing process consists of two conceptually separate phases:

1. constructing a binary function-argument tree (in the process recognising a grammatical sentence)
2. iteratively applying functions to arguments to derive a logical form.

The "structure-building" rules which construct the initial parse tree are:

T1. **Introduction**: if the current task has an unsatisfied requirement $Ty(X) \in ToDo$, rewrite the requirement as $\langle d \rangle Ty(Y), \langle d \rangle Ty(Z)$ where this is licensed by a rule of combination $(Y,\ Z) \Rightarrow X$.

T2. **Subordination**: if the current task T has an unsatisfied requirement $\langle d \rangle \phi \in ToDo$, initiate a daughter task T' with $ToDo = \phi$ and pass control to T'.

T3. **Scanning**: if the current task state has $Ty(X) \in ToDo$, and the current word in the input string matches a lexical item with $Ty(X)$ specified, update the parse state as specified in the lexical entry. This in fact is where the bulk of the 'structure-building' takes place: see section 2.4 below for discussion and examples.

T4. **Gap Resolution**: If there is no lexical entry which matches the current input string and (potentially) satisfies the required type specification, attempt to satisfy the requirement with the content of an 'unfixed' node of the requisite type and whose partial address may be unified with the address of the current tree node. That is, if the current node is at $Tn(n)$ and the unfixed node has the address $Tn(m@)$ then the modality $Tn(n), \langle u \rangle * Tn(m)$ must hold. This handles long-distance dependencies and may be compared with the well-know technique of *gap-threading* (cf Pereira & Shieber 1987). This is currently implemented for items of type e only.

The "interpretive" rules are

T5. **Completion**: if the current task T has no further requirements (empty *ToDo*) return control to the initiating task T', either the mother or a linked node. In the former case add $\langle d\rangle\phi$ to *Done* in T', where ϕ is the content of *Done* in T.

T6. **Elimination**: If either of the following holds:
1. $Done = \langle d\rangle\phi, \langle d\rangle\psi$
2. $Done = \phi$ and *Done* in a linked task is ψ

and some rule of combination licenses $(\phi, \psi) \Rightarrow \chi$; rewrite *Done* as χ.

This interfaces to the specialised λ-reduction and term construction rules in the `apply` module; see (Meyer Viol et al. 1997) for detailed discussion.

The following rules only apply if lexically selected. Both induce a tree node with an unfixed address and are triggered by the presence of *wh*-items in the input string:

T7. **Adjunction** processes *wh*-relativisers: this initiates a type t task LINKed to the current node with address $Tn(\overline{n}L0)$ and an unfixed node with an initial address $Tn(\overline{n}L0@)$ and content copied from the head item. (further details in section 3.1 below on relative clauses)

T8. **Gap Introduction** processes *wh*-questions: triggered by sentence-initial *wh*-items and builds an unfixed node with no initial semantic content.

2.3 Control

The parser operates with a combination of top-down and bottom-up processing. Initially, and in the absence of lexical input either the **Introduction** rule is invoked to decompose the type specification of the current `ToDo` feature using a subset of type-deduction rules equivalent to a context-free grammar, or where the type requirement cannot be matched by a sequence of **Introduction** and **Subgoal** steps, the **Gap Resolution** rule attempts to match it by retrieving an unfixed node. In practice in the current implementation the only place where **Introduction** applies without being lexically selected is at the top level where $Ty(t)$ is decomposed to $\langle d\rangle Ty(e), \langle d\rangle Ty(e \rightarrow t)$. (This reflects the limitations in linguistic coverage referred to in Section 1.1 above.) The lexical format allows for projection of tree structure, invocation of transition rules and other actions.

2.3.1 Top-down processing

The basic flow of control is defined recursively as the `parse` predicate shown in Figure 1. The general rule schema has the form `parse(S1,S2):- rule(S1,S1a), parse(S1a,S2)`: *S1/S2* is a valid transition if there is an intermediate state *S1a* and a rule licensing *S1/S1a*, and *S1a/S2* is a valid transition (by some sequence

```
parse(State,State):-error_state.
parse(State,State):-goalstate.
parse(State,NewState):-scan(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-eliminate(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-complete(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-gap_resolve(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-subgoal(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-intro(State,MidState),
      parse(MidState,NewState).
parse(State,NewState):-backtrack(State,MidState),
      parse(MidState,NewState).
```

Figure 1: Main Parse Loop

of transition rules). The inference regime is 'goal-directed' in that it terminates in either of two cases:

(i) `goalstate` predicate succeeds: type t has been recognised at the top-level task, no task states have non-empty *ToDo* and every tree node structure has a 'complete' address (i.e. no unfixed nodes remain).

(ii) `error_state` predicate succeeds: this indicates that some transition rule has encountered an unrecoverable error.

In addition to the transition rules we have a 'meta-rule' `backtrack` which retries failing paths generated by **Introduction** and **Subgoal**. This predicate has less to do in the current implementation since lexical projection of structure results in a more 'deterministic' algorithm than was the case for the initial prototype reported in (Meyer Viol et al 1997). The rule ordering is dictated in part by efficiency considerations, bearing in mind that we adhere to Prolog's default depth-first search strategy. For example the **Scanning** rule is always called to check whether the type of the current word in the string matches the current task specification before invoking **Introduction** and **Subordination** to generate new tasks; this minimizes unwanted backtracking.

2.3.2 Lexical input

Lexical entries consist of directions for the `scanning` rule and are structured as follows:

```
lexicon(String, ty(Type),len(Integer),
     (Tn, ToDo, Done),      % content of current task state
     (NewTn, NewToDo, NewDone),  % content of new task state
     Actions):-
     PreConditions.
```

Notes:

1. `String` is an NL expression to be matched with the current content of the input string.
2. `Integer` specifies by how much the string counter is to be incremented.
3. The tree node associated with the current task state is modified according to the contents of `NewTn`, `NewToDo` and `NewDone`.
4. `Actions` are performed after the current task state is updated with `NewToDo`, `NewDone`. Possible actions: invoke (sequence of) transition rule(s); create new tree node(s) at address(es) relative to `Tn`. This allows for creating and modifying tree structure apart from the 'current' tree node.
5. `PreConditions` are tested before any update takes place.

2.3.3 Example: Intransitive (1-place) Verbs

The simplest example of lexical processing is triggered when the input matches `String` and the type requirement in `ToDo` is also matched, and the result is to add the content of the lexical entry to *Done* in the current task and advance the read head (increment the string counter) by one. An example is provided by intransitive verbs like `sings`:

```
lexicon('sings', ty(e->t), len(1),
(Tn, [ty(e->t)|ToDo], Done),
(Tn, ToDo, [ty(e->t), fo(l([X]), sings(X), [ ])| Done]),
 [ ] ).
```

This has the effect of (i) removing $Ty(e \to t)$ from *ToDo* and (ii) adding $Ty(e \to t), Fo(\lambda x Sing(x))$ to *Done*. In this example there are no preconditions to be tested and no actions to execute.

2.3.4 Example: Transitive (2-place) Verbs

The following example demonstrates how several stages of processing may be projected from a single lexical entry:

```
(i)
lexicon('loves', ty(e->(e->t)), len(0),
     ([ty(e->t)|ToDo], Done),
     ([down([ty(e->(e->t))]),down([ty(e)])|ToDo],Done),
     [rule(subgoal),rule(scan) ] ).
(ii)
lexicon('loves', ty(e->(e->t)), len(1),
     ([ty(e->(e->t))|ToDo], Done),
     (ToDo, [ty(e->(e->t)), fo(l([X,Y]), love(Y,X), [ ])| Done]), [ ] ).
```

The lexical entry for `loves` has two sub-clauses, one which effectively projects a sub-tree and one which adds content to a tree node.

Clause (i) is matched when the current word in the input string is *loves* and the current requirement in *ToDo* is for type $e \rightarrow t$. The requirement is replaced by $\langle d\rangle Ty(e \rightarrow (e \rightarrow t)), \langle d\rangle Ty(e)$ and the specified transition rules **subgoal** and **scan** are called in sequence:

subgoal initiates a new daughter task with $Ty(e \rightarrow (e \rightarrow t)) \in \ ToDo$

scan at this point matches clause (ii): the current word in the input string is still *loves* (since the read head is not advanced; $len = 0$) and the current requirement in *ToDo* is for $Ty(e \rightarrow (e \rightarrow t))$. The type specification and formula are added to *NewDone*, which replaces *Done* in the current task.

Note that the string counter is only incremented after both clauses have applied (controlled by the `len` parameter).

3 Applications

3.1 Relative Clauses

Relative clause processing is triggered by the lexical entries for *wh*-items such as *who*:

```
lexicon(who, ty(e),len(1),
(Tn, [ ], [ty(cn)|Done]),
(Tn, [ ], [ty(cn)|Done]),
[rule(adjunction)]).
lexicon(who, ty(e), len(1),
```

```
(Tn, [ ], [ty(e)|Done]),
(Tn, [ ], [ty(e)|Done]),
[rule(adjunction)]).
```

The import of this is that if the word *who* is processed at a point where the current task state T has $Tn(n)$, $ToDo = \emptyset$ and $Ty(cn|e) \in Done$, the **Adjunction** rule is initiated via the *Actions* list. Adjunction adds two new tasks T', T'' to the parse state:

T' $Tn(nL0)$, $ToDo = \{Ty(t)\}$ - initiate the relative clause as a new subtree LINKed to the current node $Tn(n)$.

T'' $Tn(nL0@)$, $ToDo = \emptyset$, $Done = \{Fo(\alpha), Ty(e)\}$ - this is the 'unfixed node' which eventually instantiates a missing argument or 'gap'. The formula α is copied from the current (head) node at $Tn(n)$ and is either the formula content of *Done* if the head is type e (non-restrictive) or an individual 'metavariable' which is paired with the predicate in *DU*s of type *cn* (restrictive).

The distinction between restrictive and non-restrictive relatives is reflected in the **Completion** and **Elimination** rules, and in the `apply` predicate which is indirectly invoked by **Elimination**. The output from a sentence like *John, who cheated, failed the exam* results in two separate trees with the formula content *fail(John,exam)* and *cheat(John)*. In a restrictive example such as *A student who cheated failed the exam* the content of the linked subtree is incorporated into the predicate: $student(x) \wedge cheat(x)$. Reults can be seen in Figure 0.

3.2 Anaphora, *Wh*-resolution and Crossover

The problem of crossover in *Wh*-questions was addressed in the implementation reported in (Finger et al fc.) in a rather straightforward way. Reflecting the fact that the sequence *wh-gap-pronoun* is acceptable but not *wh-pronoun-gap*, the system resolves a pronoun by copying a labelled formula of type e which is defined to be 'visible', *wh*-items being initially placed in storage and not visible until they are retrieved to resolve a gap.

3. (a) Who$_i$ does John think e_i thinks he_i upset Mary?
   (b) *Who$_i$ does John think he_i thinks e_i upset Mary?

When we come to model *wh*-relatives however this treatment both over- and under-generates: on the one hand there are cases where a possessive pronoun is marginally acceptable before the gap, on the other there are non-restrictive relatives where the head NP apparently provides a 'visible' antecedent for the pronoun but the result is unacceptable:

```
a student who cheated failed

Task 1
 Tree Node: tn([0])
 ToDo: []
 Done: [ty(t),
 fo(l([]),(student(T2)and cheat(T1)and equate(T1,T2))and fail(T2),
 [term(T2,t(op(ep),v(x6),dep(0),
 f((student(x6)and cheat(T1)and equate(T1,x6))and fail(x6))))])|_263]

Task 5
 Tree Node: tn([0,link,1,0,0])
 ToDo: []
 Done: [ty(t),
        fo(l([]),cheat(T1)and equate(T1,x6),
        [term(T1,t(op(ep),v(x8),dep(0),f(equate(x8,x6))))])|_531]

john who cheated failed

Task 1
 Tree Node: tn([0])
 ToDo: []
 Done: [ty(t),
fo(l([]),fail(T2)and john(T2),
[term(T2,t(op(ep),v(x7),dep(0),f(john(x7))))])|_4734]

Task 3
 Tree Node: tn([0,link,0,0])
 ToDo: []
 Done: [ty(t),
fo(l([]),cheat(T1)and john(T1),
[term(T1,t(op(ep),v(x7),dep(0),f(john(x7))))])|_4893]
```

Figure 2: Parsing restrictive and non-restrictive relativs.

4. (a) ?The student$_i$ who$_i$ his$_i$ mother$_j$ helped e_i was unsuccessful.
   (b) *John$_i$, who$_i$ Sue thinks he$_i$ knows Mary likes e_i, ignores Mary.

The first problem is addressed by allowing a pronoun to pick up as antecedent an individual 'metavariable' associated with the predicate in a competed type *cn* task. (Since the motivating construction is marginally acceptable, and allowing both common nouns and NPs to provide antecedents leads to spurious ambiguity, this facility is made a run-time option in the implementation, controlled by the *bind* feature[1].) The second is handled by imposing constraints on the interaction of pronouns and unfixed nodes which we elaborate in section 3.2.1 below. If there is no candidate available in the parse tree the pronoun is interpreted *deictically*. The result is that a 'co-indexed' reading is always generated first if present. If only the deictic reading is generated this means that the co-indexed reading is ill-formed.

3.2.1 Strategies

The design of the program allows for experimentation with different strategies for handling the interaction between pronoun and gap resolution. The intuition we want to capture (Kempson, MS and p.c.) is that a pronoun which has identical content to the head of a relative cause and occurs before the gap has been resolved makes the task goal associated with the unfixed node 'true' and so this task may be merged with the tasl associated with the pronoun and effectively removed from the parse state. While this is a clear intuition, there is no obviously unique way for the program to prove it to be true and different possibilities have been discussed. Currently Strategies I and II below are implemented, with a run-time choice as to which strategy is followed:

Strategy I: 'Gap-checking'

The gap is resolved by identifying a 'dangling' node which has a tree node address of the form $\overline{m}@$ and content $Fo(\alpha)$, *provided* there is no existing tree node with $Fo(\alpha)$ and a fixed address which unifies with $\overline{m}@$. If there is such a node the parse fails immediately. The partial address is resolved to a complete address with @ replaced by a sequence of 0's and/or 1's.

[1]Another possibility, which I will not pursue here, is that the possessive cases are instances of *cataphora* rather than crossover, with the instantiated 'missing argument' providing an antecedent which *follows* the pronoun. Support for this approach would come from examples like the following:

(i) Who does even his mother ignore?

(ii) Even his mother wouldn't call John handsome.

where there is no prior antecedent at the point where the possessive *his* is encountered.

The effect is to rule out relative clauses like (12) in Appendix A where a pronoun precedes the gap and is resolved by copying the head; a further consequence is that resumptive pronouns are also disallowed, since if no gap is encountered the 'dangling node' will remain unfixed causing the parse to fail.

Strategy II: 'Merge'

Rather than waiting until a gap is encountered, a *merge* operation is defined as follows[2]:

Given

(i) a newly updated task state T_i with a fixed address $Tn(\overline{n})$, empty *ToDo* and $Done = \phi$;

(ii) an existing task state T_j with a partial address $Tn(\overline{m}@)$ and $Done = \psi$, where $Tn(\overline{n})$ unifies with $Tn(\overline{m}@)$ and ϕ is *identical* to ψ;

T_i and T_j may be merged resulting in a single task state with $Tn(\overline{n})$ and $Done = \phi$.

The effect is that (resumptive) pronouns are accepted in free variation with gaps in relative clauses, since they 'consume' the unfixed node which would otherwise be available for gap resolution. Note that the test for 'merge' is made every time a new task state is saved and so is potentially less efficient than the test made under Strategy I which is implemented at the point of gap-resolution. The merge operation is a run-time option controlled by the *merge* feature.

3.2.2 Example 12: Discussion

12. *John$_i$, who$_i$ Sue thinks he$_i$ knows Mary likes e_i, ignores Mary.

```
0 john 1 who 2 sue 3 thinks 4 he 5 knows 6 mary 7 likes 8 ignores
9 mary 10
```

john at position 1: the head NP is parsed as a completed e-task with content $\epsilon x(john(x))$.

```
Task 2
 Tree Node: tn([0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x7),dep(0),f(john(x7))),[]),Agr]
```

who at position 2 initiates a type T task (not shown) at $Tn(00L0)$ and a type e task at $Tn(00L0@)$:

[2]This formulation was arrived at following a discussion with Ruth Kempson and Wilfried Meyer Viol.

```
Task 4
 Tree Node: tn([@,0,link,0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x7),dep(0),f(john(x7))),[]),Agr]
```

he at position 5 is resolved by copying the content of the head NP (Task 2). NB: at this point *Merge* takes place under Strategy II only, with the unfixed node (Task 4) removed from the parse state.

```
Task 9
 Tree Node: tn([0,1,1,0,link,0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x7),dep(0),f(john(x7))),[]),Agr]
```

Gap Resolution is called to satisfy the type e requirement at position 8, since there is a mismatch between the type required and the word at this position, *ignores*:

```
Task 16
 Tree Node: tn([1,1,1,1,1,1,0,link,0,0])
 ToDo: [ty(e)]
 Done: []
```

Strategy I: Resolution fails for the following reasons:

(i) The only candidate is Task 4 with address `00L0@` and `Fo(john)`

(ii) This specification is already satisfied by Task 9 which has `Fo(john)` and address `00L0110` wich unifies with `00L0@`, i.e. $Tn(00L0), \langle u \rangle * Tn(00L0110)$ holds. So the gap resolution is blocked.

Strategy II: Resolution also fails, since the unfixed node is no longer present in the parse state. (Task 4 was removed as a result of *merge*.)

3.2.3 Resumptive example

So far it may seem that the two strategies are simply different 'tricks' which achieve the same result. However they make different predictions for examples like

5. ?John$_i$, who$_i$ Sue thinks he$_i$ knows Mary likes him$_i$, ignores Mary.

```
0 john 1 who 2 sue 3 thinks 4 he 5 knows 6 mary 7 likes 8 him
9 ignores 10 mary 11
```

which forms a 'minimal pair' with (12).

Under Strategy I, everything is the same as (12) up to *him*; at this point the pronoun may resolve to either *john* or *he* (with the same semantic effect) and the tree is completed without errors. However when the top node is complete the `error_state` predicate is satisfied since there remains a node (Task 4) with an unfixed address: under this regime unfixed addresses are only completed as part of the gap-resolution process. So the sentence is flagged as unacceptable.

Under Strategy II the result is the same until the point where the top node is completed. This time no error state is recognised since the unfixed node has been removed by *Merge* as part of the resolution of the pronoun *he*. So in this case the sentence is accepted.

3.3 *Whose* as *who* + *s*

This section motivates an implementation decision to treat the possessive *whose* as a composite of *who* and *'s*. (The discussion is restricted to relative clauses.)

Firstly, suppose we treat *whose* as a unitary lexical item. In contrast to *who* where the formula content of the head at n is simply copied across to the unfixed node at $nL0@$, the content of *whose* has to be projected to some doubly underspecified position somewhere on a left branch of $nL0@$, to enable us to model recursive constructions such as *john, whose mother's friend's babysitter's ...* which introduces a new degree of implementational complexity. Since both *who* and the possessive *'s* are independetly needed we avoid these complications by breaking down *whose* into *who* + *'s*. The possessive marker *'s* when paired with a type e node causes the latter to be 'shifted' into sub-determiner position (by extending the address from $Tn(n)$ to $Tn(n00)$), creating a new type e node with internal structure. The lexical entry is displayed in Figure 3 and the process is now explained with worked examples.

3.3.1 Dynamics

Assume the input string is

```
0 john 1 who 2 -s 3 brother 4 ...
```

At 2 we have the following nodes:

1. `[Tn(0), Ty(t), ...]`
2. `[Tn(00), ty(e), Fo(John), ...]`
3. `[Tn(00L0) Ty(t) ...]`
4. `[Tn(00L0@), ty(e), Fo(who), ...]`

```
(i) project subtree
lexicon('-s', ty(e->(cn->e)), len(0),
(tn(Path), [ ], [ty(e)|Done]),
(tn([0,0|Path]), [ ], [ty(e)|Done]),
[create_node(tn(Path), show(0),
[down([ty(cn -> e)]), down([ty(cn)])], [ ]),
 create_node(tn([0|Path]), show(0),
  [down( [ty(e)]), down([ty(e->(cn->e))])], [ ] ),
      rule(complete)]).
(ii) fill in formula content
lexicon('-s', ty(e->(cn->e)), len(1),
(Tn, [ty(e->(cn->e))|ToDo], Done),
(Tn, ToDo,
[ty(e->(cn->e)),
fo(l([X,P]),
(op(ep), v(Y),
pred( (form(P:Y) and poss(X,Y) ))), [ ]), 3@agr|Done]), [ ]).
```

Figure 3: Lexical entry for possessive *'s*

Parsing -s causes the following actions (all projected from the lexical entry shown in Fig. 0):

(i) Extend the address of (4) to `00L0@00` In fact we can see this as a *partial completion* of the address rather than a non-monotonic modification.)

(ii) Create a new node at `00L0@` with `ToDo = <d>ty(cn->e), <d>ty(cn)` - add `create_node` to Actions list: this predicate partially implements the i^n, f^n functions of (Meyer Viol MS).

(iii) Create a new node at `00L0@0` with `Done = <d>(ty(e), Fo(who)), ToDo = <d>ty(e->(cn->e))` (ditto)

(iv) Scan `-s` as `ty(e->(cn->e))` with formula content $\lambda x \lambda P \epsilon y(Py \wedge poss(x, y))$

The parse state is now:

1. `[Tn(0), Ty(t), ...]`
2. `[Tn(00), ty(e), Fo(John), ...]`
3. `[Tn(00L0) Ty(t) ...]`

4. `[Tn(00L0@00), ty(e), Fo(who), ...]`

5. `[Tn(00L0@), ToDo = <d>ty(cn->e), <d>ty(cn)]`

6. `[Tn(00L0@0), Done = <d>(ty(e), Fo(who)), ToDo = <d>ty(e->(cn->e))]`

7. `[Tn(00L0@01), ty(e->(cn->e)), Fo(-s), ...]`

(I assume that 'whose' is broken down into 'who + s' by morphological rule and the particle doesn't appear on its own, so we do not predict well-formedness for e.g. 'who did Jan steal *e*'s bike?')

3.3.2 Example 10: Discussion

10. John$_i$, whose$_i$ mother$_j$ Sue thinks he$_i$ knows Mary likes e$_j$, ignores Mary.
Some snapshots:
(NB: tree addresses to be read backwards)
"whose mother"

```
Task 5
 Tree Node: tn([@,0,link,0,0])
 ToDo: []
 Done: [ty(e), fo(t(op(ep),v(x18),dep(0),f((mother(x18)and
poss(T1,x18))and john(T1))),[]),Agr]
```

"who"

```
Task 4
 Tree Node: tn([0,0,@,0,link,0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x7),dep(0),f(john(x7))),[]),Agr]
```

"hei"

```
Task 13
 Tree Node: tn([0,1,1,0,link,0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x7),dep(0),f(john(x7))),[]),Agr]
```

Gap Resolution

```
Task 5
 Tree Node: tn([1,1,1,1,1,1,0,link,0,0])
 ToDo: []
 Done: [ty(e),fo(t(op(ep),v(x18),dep(0),f((mother(x18)and
  poss(T1,x18))and john(T1))),[]),Agr]
```

Remarks:

(i) This is the only candidate to resolve the gap since only Task 5 (see above) had a tree node address ending with @.

(ii) The presence of the pronoun equated to john does not block the gap resolution since it does not have the same content as the resolvent.

Goal State:

```
Task 1
 Tree Node: tn([0])
 ToDo: []
 Done: [ty(t),fo(l([]),(ignore(T7,T6)and mary(T6))and john(T7),
[term(T7,t(op(ep),v(x7),dep(0),f(john(x7)))),
term(T6,t(op(ep),v(x57),dep(0),f(mary(x57))))])|_41180]

Task 3
 Tree Node: tn([0,link,0,0])
 ToDo: []
 Done: [ty(t),fo(l([]),think(T5,know(T4,(like(T3,T2)and(mother(T2) and
  poss(T1,T2))and john(T1))and mary(T3))and john(T4))and sue(T5),
[term(T5,t(op(ep),v(x31),dep(0),f(sue(x31)))),
term(T4,t(op(ep),v(x7),dep(0),f(john(x7)))),
term(T3,t(op(ep),v(x51),dep(0),f(mary(x51)))),
term(T2,t(op(ep),v(x18),dep(0),f((mother(x18)and
  poss(T1,x18))and john(T1))))])|_41498]
```

(Note: the name "mary" has different variables under tasks 1 and 3 since the name is introduced twice.)

3.4 Summary

Table 1 shows the results of parsing a selection of the test suite in Appendix A with combinations of the run-time options *bind* and *merge* (see Sect. 3.2). Comparison with the judgments indicated in the appendix show that the clear cases produce consistent results across all four combinations, while the marginal cases flagged with a question mark vary according to the options. As mentioned in the introduction, this finding may go some way toward modelling dialectal and cross-linguistic differences.

Bind option	0	1	1	0
Merge option	0	0	1	1
Ex 2: X-over in Wh-Qs	*	*	*	*
Ex 10: 2ary Strong X-over, Nonres Rels:	OK	OK	OK	OK
Ex 12: Strong X-over, Nonres Rels:	*	*	*	*
Ex 13: Strong X-over, Res Rels:	*	*	*	*
Ex 14: No X-over, Res Rels:	OK	OK	OK	OK
Ex 16: Weak X-over, Nonres Rels:	OK	OK	OK	OK
Ex 17: Weak X-over, Nonres Rels:	OK	OK	OK	OK
Ex 20: 2ary Strong X-over, Res Rels:	*	OK	OK	*
Ex 21: Resumptive Pronoun, Res Rel:	*	*	OK	*
Ex 27: Resumptive Pronoun, Nonres Rel:	*	*	OK	OK

Table 1: Results of parse runs

4 Conclusion

Gorrell (1995) describes an 'incremental' parser with a deterministic component involving 'primary relations' *dominates* and *precedes* which are defined over a fully specified tree structure and have to be preserved when new material is incorporated into a tree. with limited restructuring of 'secondary relations' when adding adjuncts or resolving local ambiguity. The LDS_{NL} account is closer to the D-Theory of (Marcus 1983, Weir & Carroll, Rogers & Vijay-Shankar...) in that what corresponds to the primary relations are descriptions in a tree logic which allow partial specification of a tree structure.

As stated in section 1.1 the parser does not aim at 'completeness' but returns a single preferred reading: this may lead to failure with garden-path sentences or where a default pronoun resolution strategy leads to an unacceptable reading. This paper has attempted to avoid taking a position on the various linguistic issues dealt with but demonstrates how theoretical assumptions may be tested by setting various fairly general runtime options which turn out to have interesting empirical consequences.

Acknowledgements

This research was supported by the UK Engineering and Physical Sciences Research Council under grant reference GR/K67397, "A Labelled Deductive System for Natural Language Understanding". The test suite listed in Appendix A is based on example sentences provided by Ruth Kempson. This work is a result of close collaboration with Kempson and Wilfried Meyer Viol; I have been careful to acknowledge their work in the text and I apologise in advance if any of their

thoughts have crept into this paper without due credit.

A Test Suite

A.1 Crossover in Questions:

1. *Who$_i$ did he$_i$ think that Bill liked e_i?
2. *Who$_i$ did he$_i$ think Bill liked e_i?
3. * Who$_i$ did his $_i$ mother ignore e_i?
4. *Whose$_i$ mother did you tell him$_i$ e_j had refused the operation?
5. *Whose$_i$ results$_j$ was he$_i$ certain e_j would be good?
6. *whose$_i$ results$_j$ was he$_i$ certain John would ignore e_j?
7. Who$_i$ thought he$_i$ was ill?

A.2 Secondary Strong Crossover: Nonrestrictive Relatives

8. John$_i$, whose$_i$ mother$_j$ I told him$_i$ e_j had refused the operation, was upset.
9. John$_i$, whose$_i$ results$_j$ he$_i$ had been certain e_j would be good, failed dismally.
10. John$_i$, whose$_i$ mother$_j$ Sue thinks he$_i$ knows Mary likes e_j, ignores Mary.
11. John$_i$, whose$_i$ results$_j$ indicated that he$_i$ was good, got the job.

A.3 Strong Crossover in Relatives

12. *John$_i$, who$_i$ Sue thinks he$_i$ knows Mary likes e_i, ignores Mary.
13. *The student who$_i$ Sue believed he$_i$ knew e_i had cheated, failed.
14. The student who$_i$ Sue believed e_i knew he$_i$ had cheated, failed.

A.4 Weak Crossover Nonrestrictive Relatives

15. John$_i$, who his$_i$ mother ignored e_i, was ill.
16. John$_i$, who$_i$ Sue thinks his$_i$ mother knows Mary likes e_i, ignores Mary.
17. John$_i$, who$_i$ Sue thinks e_i knows his$_i$ mother likes Mary , ignores Mary.

A.5 Secondary Strong Crossover Restrictive Relatives

18. ?The student$_i$ whose$_i$ friend$_j$ he$_i$ helped e_j cheated.

19. ?The applicant$_i$ whose$_i$ candidacy$_j$ he$_i$ helped his$_i$ MP to fix e_j was unsuccessful.

20. ?The student$_i$ who$_i$ his$_i$ mother$_j$ helped e_i was unsuccessful.

A.6 Resumptive Pronouns

21. ?The student$_i$ who$_i$ Sue says he$_i$ should have failed has turned up to class.

22. ?The student$_i$ who$_i$ Sue says she wished she had failed him$_i$ has applied to the PhD program.

23. ?The son$_i$ who$_i$ wish I could win an argument with him$_i$ has finished his$_i$ degree.

24. ?My son$_i$ who he$_i$ agrees he$_i$'s been overworking is taking a rest.

A.7 Resumptive Pronouns with *even*

25. ?The Chairman$_i$, who Sue distrusted even him$_i$, was sympathetic.

26. ?The Chairman$_i$, who even he$_i$ distrusted Mary, was sympathetic.

A.8 Resumptive Pronouns in Non-restrictive Relatives

27. ?John$_i$, who$_i$ Sue thinks he$_i$ knows his$_i$ mother likes Mary , ignores Mary.

References

Blackburn, S. & Meyer Viol, W. (1994) Linguistics, logic and finite trees. *Bulletin of Interest Group in Pure and Applied Logics* 2 (1), 3 – 29.

Finger, M., Kibble, R., Kempson, R. & Gabbay, D. (forthcoming) Parsing natural language using LDS: a prototype. *Bulletin of Interest Group in Pure and Applied Logics.*

Gabbay, D., 1996, *Labelled Deductive Systems*, OUP.

Gabbay, D. & Kempson, R. (1992) Natural language content: a proof–theoretic perspective. *Proceedings of 8th Amsterdam Semantics Colloquium.* Amsterdam, 173 – 96.

Gorrell, 1995, *Syntax and Parsing*, CUP.

Kempson, R., 1996, Semantics, Pragmatics and Natural Language Interpretation, in Lappin (ed), *The Handbook of Contemporary Semantic Theory*, Blackwell, Oxford.

Kempson, R., MS, Crossover: A Dynamic Perspective.

Meyer Viol, W., MS, Parsing as Tree Construction.

Marcus, M....

Meyer Viol, W., 1995, Instantial Logic. Utrecht: PhD dissertation.

Meyer Viol & Kempson 1996: Language Understanding, A Procedural Perspective. Proceedins of LACL, Nancy.

Meyer Viol, W. & Kempson, R., 1997, Parsing as Tree Construction in LDS, in G.-J. Kruijff et al. (eds), *Proceedings of the Formal Grammar Conference*, Aix-en-Provence.

Meyer Viol, W., Kibble, R., Kempson, R. & Gabbay, D. 1997, 'Indefinites as Epsilon Terms' in H. Bunt et al. (eds), Proceedings Meyer Viol et al 1997 II, Tilburg.

Shieber, S., Schabes, Y. & Pereira, F. (1995) Principles and implementation of deductive parsing. *Journal of Logic Programming* 24 (1–2), 3 – 36.

Tomita, 1986, *Efficient Parsing for Natural Language.*

Vijay-Shankay & Rogers, ...