A Definite Clause Grammatical Inversion of Extended Montague Semantics

ὰποθανων ἐτι λαλειται

VOLUME II

(Volume Two of Two Volumes)

Roy Ivor Bainbridge

Thesis submitted for the degree of
Doctor of Philosophy
to
The Department of Computer Science,
University of York.

May 1987.

CONTENTS

A DCG INVERSION OF EXTENDED MONTAGUE SEMANTICS

VOLUME I

INTRODUCTION 1

1. PROLEGOMENA TO MONTAGUE SEMANTICS 9

1.1. The Goals of Semantic Theory 9

1.2. Higher Order Abstraction and Type Theory 11

1.3. Extensional Semantics 14

1.3.1. Lexicon for FOL 14

1.3.2. Syntax for FOL 14

1.3.3. Lexical Semantics for FOL 15

1.3.4. Expression Semantics for FOL 16

1.4. Extensional Compositionality 18

1.5. Possible World Semantics 21

1.5.1. Lexicon for TAL 22

1.5.2. Syntax for TAL 23

1.5.3. Lexical Semantics for TAL 23

1.5.4. Expression Semantics for TAL 24

1.6. Intensions in Opaque Contexts 26

1.7. Residual Problems 33

2. MONTAGUE'S PTQ 36

2.1. Montague's IL 36

2.1.1. Semantic Types for IL 36

2.1.2. Lexicon for IL 37

2.1.3. Syntax for IL 37

2.1.4. Possible Denotations 38

2.1.5. Lexical Semantics for IL 39

2.1.6. Expression Semantics for IL 39

2.2. The Grammar of the Fragment 40

2.2.1. Syntactic Categories and Semantic Types 40

2.2.2. Rule Forms in PTQ 41

2.2.3. Colloquial Variables 44

2.2.4. The Lexicon for PTQ 44

2.2.5. The Grammar Rules 44

2.2.6. Meaning Postulates 48

2.3. Analysis Trees 49

2.3.1. Semantically Vacuous Discriminations 50

2.3.2. Relative Clauses 51

2.3.3. Intensional and Extensional Verbs 55

2.3.4. Verb Phrase and Common Noun Phrase Quantification 58

2.3.5. Prepositions and Adverbs 63

2.3.6. Disjunctive Terms 65

3. CORRECTIONS AND CONSTRAINTS 67

3.1. Catalogued Errors in PTQ 67

3.2. Friedman's Unlabeled Bracketing Solution 73

3.3. Handling Reflexivity 76

3.4. Partee's Constraints & Innovations 78

3.4.1. Constraints Ascribed to Montague 78

3.4.2. Proposed Additional Constraints 79

3.4.3. Partee's Innovations 83

3.4.4. Partee's Reformulations 86

3.5. Janssen's Hyperrules 94

3.6. Formulaic and Processing Parsimony 96

4. FUNDAMENTAL EXTENSIONS 99

4.1. Extending the Coverage of Montague's Fragment. 99

4.2. Rodman's Relative Clause Rules 99

4.2.1. Rodman's Constraint on Quantifier Scope 100

4.2.2. The Ross "Island" Constraints 102

4.2.3. Rodman Variables 103

4.2.4. Conservative Reformulations of Rodman's Rules 106

4.3. The Interrogative Theory of Karttunen & Peters 109

4.3.1. Basic Building Blocks 111

4.3.2. Choice Questions 112

4.3.3. Binary Versions of the Basic and Choice Question Rules 114

4.3.4. Interrogative and Ordinary Noun Phrases 116

4.3.5. Search Questions 119

4.3.6. Multiple Constituent Search Questions Reconsidered 122

5. PASSIVISATION, TENSE AND ASPECT 126

5.1. Bach's Account of Passivisation 126

5.2. Bach's Account of Tensed Verb Phrases and Auxiliaries 130

5.3. Dowty's Two Dimensional System 136

5.3.1. Dowty's Modifications to IL 138

5.3.2. Dowty's Grammar Rules for Tense and Aspect. 140

5.3.3. The Limitations of Two Dimensionality 142

5.4. The Reichenbachian Tradition 148

5.4.1. Reichenbach's Nine Tense System 148

5.4.2. Bull's Twelve Tense System 150

5.4.3. Bruce's Relational System 152

6. VERB PHRASES IN TMG 157

6.1. Surface Oriented Syntax 157

6.2. Lexical Modifications in TMG 161

6.3. Subcategorising Verb Phrases 163

6.4. The Verb Phrase Rules 164

6.4.1. Plural Terms and the Subject Predicate Rule 168

6.4.2. Finite Verb Phrase Rules 169

6.4.3. The Auxilliary System 169

6.4.4. Rules of Passivisation 173

6.4.5. Rules of Complementation 174

6.4.6. Verb Phrase Conjunctions 174

6.4.7. Adverbial Qualification Rules 175

6.4.8. Quantified Verb Phrase Rules 176

6.4.9. The Rules in Operation 177

6.5. Multi-Indexed Tensed Logic. 178

6.5.1. Semantic Types for TIL 178

6.5.2. Lexicon for TIL 178

6.5.3. Syntax for TIL 179

6.5.4. Denotations for TIL 180

6.5.5. Multi Indexed Evaluation 180

6.5.6. Semantics for TIL 181

6.5.7. Subordinate Time Relativisation 184

CONTENTS

A DCG INVERSION OF EXTENDED MONTAGUE SEMANTICS

VOLUME II

7. ORTHODOXY, APOSTACY AND UTILISATION 187

7.1. Sine Qua Non 187

7.2. Montague's General Theory 189

7.2.1. Recursive Compositional Syntax 190

7.2.2. Homomorphic Meaning Assignments 191

7.2.3. Model Theoretic Semantics 192

7.2.4. IL Under the General Theory 194

7.2.5. Concessionary Translation 195

7.3. Indeterminacy of Translation 197

7.4. Computational Compositional "Semantics" 201

7.4.1. On "Replacing" the Model Theoretic Interpretation 202

7.4.2. Semantics or Verification? 205

7.5. Montague Grammar as a Machine Translation Interlingua 208

8. COMPUTATIONAL INVESTIGATIONS 215

8.1. Janssen's Experimental Generator 215

8.2. The Friedman Warren Algorithm 219

8.2.1. Recursive Descent Using FVB and SA Lists 220

8.2.2. Handling Cataphora 226

8.2.3. The Zero Option 228

8.2.4. Accommodating Interrogatives 230

8.3. Equivalence Parsing 232

8.3.1. The Directed Process with Recall Table 232

8.3.2. Semantic Equivalence Parsing 238

9. INVERTED MONTAGUE GRAMMAR 240

9.1. Fundamentals of Definite Clause Grammar 240

9.1.1. Phrase Structure and Recursive Descent Reconstruction 240

9.1.2. DCG - A Grammar/Algorithm Hybrid 242

9.2. Syntactic Processing in TMDCG 244

9.2.1. Basic Templates 246

9.2.2. Feature Lists 247

9.2.3. Environments 249

9.2.4. Simulating Structural Operations: The Label Fields 251

9.2.5. Bindings on the Hold Stack 253

9.3. Replacement and Quantification 254

9.4. Multi Level Cross Referencing 256

9.5. Left Recursion and the Brough Hogger Method 258

9.6. The Lexica 262

9.7. TIL Translation under LILT 263

10. POSTSCRIPT 267

11. APPENDIX A: TIL & TMG 271

12. APPENDIX B: TMDCG 291

13. APPENDIX C: EDIT 306

14. APPENDIX D: GENLEX 309

15. APPENDIX E: LEXTMG 311

16. APPENDIX F: LILT 314

17. APPENDIX G: TBASE 320

18. APPENDIX H: SAMPLE OUTPUT - TMDCG & LILT 325

19. BIBLIOGRAPHY 384

CHAPTER 7. ORTHODOXY, APOSTACY & UTILISATION

¶ Criteria for identifying and classifying computational implementations of Montague grammar in terms of the essentials of UG are considered and computational investigations distinguished from computational utilisations. The orthogonal tradition of "computational compositional semantics", often represented misleadingly by workers in AI as computational treatment of Montague grammar, is then contrasted with true computational implementation. This chapter concludes with a discussion of computer utilisations in machine translation, while computational investigations are deferred to the next.

7.1. Sine Qua Non

In what circumstances are we entitled to describe an amendment to Montague's original proposals as a *development* of Montague grammar? An answer to this question is required if we are to distinguish a continuation of Montague's program from its abandonment in favour of some alternative theory. The distinctive features of a bona fide development are, I suggest, twofold:

(i) *Conservative Motivation.* Modifications should be designed to enhance the prospect of achieving Montague's original goal of a comprehensive model theoretic semantics for natural language.

(ii) *Strategic Orthodoxy.* Modifications should preserve the major characteristics of Montague's general strategy as expounded in *Universal Grammar* (UG, [M5]).

Strategic orthodoxy without conservative motivation might be exemplified by any inter language translation system which employed as an interface semantic representations derived from an orthodox Montagovian analysis. Such a project, although not a *development* of Montague grammar, could legitimately be classified as a *utilisation* thereof.

An example of a conservatively motivated but strategically unorthodox *alternative* to Montague's program might be Gazdar's GPSG, [G5], which abandons Montague's recursive syntax in favour of an augmented phrase structure grammar whilst preserving the model theoretic semantics. It would hardly be considered flattering to describe GPSG as a mere extension of Montague grammar: GPSG is a highly respected grammatical theory in its own right.

Finally strategic unorthodoxy without conservative motivation can only characterise an alien venture which, whatever its intrinsic merits, can have no relevance to Montague's program. To classify such an undertaking as "related research" would be little short of preposterous.

The appellation "Computer Implementation of Montague Grammar" could accordingly be legitimately ascribed in one of two circumstances:

(a) *Computational Investigation.* A conservatively motivated, strategically orthodox (partial) computer model of a Montague grammar would contribute towards a computational investigation within the subject area. Computational investigation involves employing the computer in the incontrovertibly appropriate role of a data processing tool so as to enable linguists to do what they wished to do in the first place, only to do it more expeditiously. Such employment has been classified by Thompson, [T5], as *computation in the service of linguistics*, and is exemplified by the pioneering achievements of Friedman and Warren, and also in the programs of Janssen.

(b) *Computational Utilisation.* A machine translation system not itself intended as a contribution to Montague's original goal, but employing a strategically orthodox Montague grammar as a component would constitute a computer utilisation of Montague grammar. Under this rubric therefore falls the research of Landsbergen's team , [L1, L2], working on the "Rosetta" project.

I specifically exclude from the classification "Computer Implementation of Montague Grammar" the various contributions to what has been misleadingly styled "Computational Compositional Semantics" (CCS). In so doing it is by no means my intention to denigrate CCS which, despite the inappropriateness of its title, is a bona fide and important field of research within its own domain. Nothing is to be gained by misrepresenting valuable research in one field as a putative contribution to another, and that the practitioners (or perhaps their self appointed spokespersons) should so lack confidence in the credibility of their field that they must invest it with a respectability derived from association with a famous name is a phenomenon which I find mystifying.

In this chapter I shall first seek to identify the *sine qua non* of a Montagovian program, whereafter I shall attempt to justify my claim that CCS is not research into Montague grammar. Finally I shall consider the *utilisation* of Montague grammar in a machine translation environment. Since my own implementation

falls under the rubric *computational investigation*, I shall devote a separate chapter to previous and ongoing investigations in this vein.

What then is the essential goal of Montague grammar? What are its essential characteristics as opposed to the coincidental features evident in a particular manifestation such as that of PTQ? The answers are to be found in Montague's general theory of UG.

7.2. Montague's General Theory

Montague introduces UG with the following remark:

> "There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed I consider it possible to comprehend the syntax and semantics of both kinds of language within a single natural and *mathematically precise* theory". (italics mine)

He then declares his aim to be the development of a universal syntax and semantics. Regarding the nature of semantics and its relationship with syntax his position is unequivocal:

> "The basic aim of semantics is to characterise the notions of true sentence (under a given interpretation) and of entailment while that of syntax is to characterise the various syntactic categories ... the aim of syntax could be realised in many different ways, only some of which would provide a suitable basis for semantics ... I fail to see any great interest in syntax except as a preliminary to semantics."

Ostensibly:

- **The formulation of a comprehensive *model theoretic* semantics must be acknowledged as the ultimate goal of Montague's program.**

We might however be generous and concede that there is no a priori reason why a mathematically precise theory must be model theoretic, in which case we could countenance within the Montagovian framework alternative representations *conducive to investigation with a mathematical rigour comparable to that which legitimises the theory of sets*. No other latitude is compatible with conservative motivation.

7.2.1. Recursive Compositional Syntax

A language **IL** is to be characterised as a pair:

(U1) **IL** = $< DL, R >$.

such that DL is a disambiguated language and R an ambiguating relation. A disambiguated language is itself a system:

(U2) $DL = << A, F_\gamma >, X_\delta, S, \delta_0 >_{\gamma \in \Gamma, \delta \in \Delta}$

where:

(U3) A is the set of proper expressions of DL and $A = \cup_{\delta \in \Delta} C_\delta$, where Δ is the set of categories of DL and C_δ the set of proper expressions of category δ.

(U4) F_γ is the γth. structural operation from A^n into A, where $\gamma \in \Gamma$ and Γ is the set of operation indices.

(U5) $< A, F_\gamma >_{\gamma \in \Gamma}$ is an *algebra*, ie. A is closed under all F_γ such that $\gamma \in \Gamma$.

(U6) X_δ is the set of basic expressions of category δ such that, for all $\delta \in \Delta$, $X_\delta \subseteq C_\delta \subseteq P$.

(U7) δ_0 is the category sentence.

(U8) S is the set of syntactic rules, each rule taking the form of a sequence:

(U9) $<F_\gamma, <\delta_k \ldots \delta_m>, \delta_n>$

to be interpreted "operation F_γ may take sequences of members from categories $<\delta_k \ldots \delta_m>$ in order to make a member of δ_n."

(U10) If $<\alpha_k \ldots \alpha_m> \in <C_{\delta_k} \ldots C_{\delta_m}>$ then $F_\gamma(<\alpha_k \ldots \alpha_m>) = \alpha_n \in C_{\delta_n} \subseteq A$.

As Partee complains, clause U10 places no restrictions upon the actual mechanics of a structural operation. We may however conclude that:

- **A compositional syntax taking the form of a simultaneous recursive definition of the well formed expressions of a language is an essential feature of a Montague grammar.**

With regard to the language **L** potentially ambiguated by the relation R, we may identify the following crucial definitions:

(U11) $BS_{\delta L}$ = the δth set of basic expressions of **L** = $\{\zeta' : \exists\zeta(\zeta \in X_\delta \wedge \zeta R \zeta')\}$.

(U12) $\mathrm{CAT}_{\delta \mathbf{L}}$ = the δth syntactic category of **L** = $\{\zeta' : \exists\zeta(\zeta \in \boldsymbol{C}_{\delta} \subseteq \boldsymbol{A} \wedge \zeta \boldsymbol{R} \zeta')\}$.

(U13) $\boldsymbol{ME}_{\mathbf{L}}$ = the set of meaningful expressions of **L** = $\cup_{\delta \in \Delta} \mathrm{CAT}_{\delta \mathbf{L}}$.

7.2.2. Homomorphic Meaning Assignments

In place of the elementary assignment function ***I*** of earlier models, Montague introduces an interpretation for **L** = $<\boldsymbol{DL}, \boldsymbol{R}>$ as a system:

(U14) $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}, \boldsymbol{f}>_{\gamma \in \Gamma}$

such that

(U15) ***f*** is a function assigning meanings to basic expressions of ***DL***.

(U16) ***B*** is the set of meanings corresponding to ***A***.

(U17) $\boldsymbol{G}_{\gamma}$ and $\boldsymbol{F}_{\gamma}$ are operations of common arity[†86] and indexed from a common set Γ.

(U18) $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}>_{\gamma \in \Gamma}$ is an algebra similar to $<\boldsymbol{A}, \boldsymbol{F}_{\gamma}>_{\gamma \in \Gamma}$, ie. ***B*** is closed under all $\boldsymbol{G}_{\gamma}$

If conditions U17 and U18 pertain then there is a homomorphism ***h*** from $<\boldsymbol{A}, \boldsymbol{F}_{\gamma}>_{\gamma \in \Gamma}$ *into* $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}>_{\gamma \in \Gamma}$ which constitutes the meaning assignment for **L** determined by $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}, \boldsymbol{f}>_{\gamma \in \Gamma}$, and which is defined thus:

(U19) $\boldsymbol{h} : \boldsymbol{A} \rightarrow \boldsymbol{Q} \subseteq \boldsymbol{B}$ and $\boldsymbol{F} \subseteq \boldsymbol{h}$.

(U20) $\boldsymbol{h}(\boldsymbol{F}_{\gamma}(<\alpha_k \ldots \alpha_m>)) = \boldsymbol{G}_{\gamma}(<\boldsymbol{h}(\alpha_k) \ldots \boldsymbol{h}(\alpha_m)>)$.

If in addition $\boldsymbol{Q} = \boldsymbol{B}$ then ***h*** is a homomorphism from $<\boldsymbol{A}, \boldsymbol{F}_{\gamma}>_{\gamma \in \Gamma}$ *to* $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}>_{\gamma \in \Gamma}$. Given a homomorphic meaning assignment, we may conclude:

(U21) $\boldsymbol{h}(\zeta)$ = the meaning of ζ′ in **L** under the interpretation $<\boldsymbol{B}, \boldsymbol{G}_{\gamma}, \boldsymbol{f}>_{\gamma \in \Gamma}$

iff $\zeta' \in \boldsymbol{ME}_{\mathbf{L}} \wedge \exists\zeta(\zeta \in \cup_{\delta \in \Delta} \boldsymbol{C}_{\delta} \wedge \zeta \boldsymbol{R} \zeta')$.

Although there is as yet no introduction of set theoretic entities to serve as meanings, the *rule by rule hypothesis*, ie. the convention that to every syntactic rule constructing α_n from $<\alpha_k \ldots \alpha_m>$ there should correspond a semantic rule for constructing the meaning of α_n from the meanings of $<\alpha_k \ldots \alpha_m>$, is clearly entailed by the commitment to homomorphic assignment, thus:

---

†86. That is to say, the operations share a common number of argument places.

- **The rule by rule hypothesis must be included as an essential Montagovian characteristic.**

Passing reference has already been made to a problem with Montague's formulations. Montague insists that for all sequences $<\alpha_k \ldots \alpha_m>$ in the domain of F_γ and all sequences $<\beta_k \ldots \beta_m>$ in the domain of $F_{\gamma'}$, if $F_\gamma(<\alpha_k \ldots \alpha_m>) = F_{\gamma'}(<\beta_k \ldots \beta_m>)$ then $\gamma = \gamma'$ and $<\alpha_k \ldots \alpha_m> = <\beta_k \ldots \beta_m>$. This use of Occam's razor eliminates unnecessary duplications, but fails to confront the question: what m-tuples may legitimately occur as arguments of an m placed operation F_γ?

Clause U9 tells us only that according to a given rule members of categories $<\delta_k \ldots \delta_m>$ are *permitted*, not that F_γ is restricted to the domain $C_{\delta_k} \times \ldots \times C_{\delta_m}$; thus subsequent rules in S may permit m tuples from alternative domains. As we have seen, there is in PTQ indeed a many:1 relationship between syntax rules and functional operations. Each of S14, S15, and S16 invokes f_{10}, the domain of which is variously $P_T \times P_t$, $P_T \times P_{CN}$ or $P_T \times P_{IV}$: but the price paid for this prevarication is that the correspondence between f_{10} and its translation schemas is *not* 1:1.

If interpretation is to be direct, U20 specifies a 1:1 correlation between syntactic and semantic rules, whilst it will transpire that U47 makes similar demands in the case of indirect interpretation. Arguably therefore Montague's specification in UG should be augmented by:

(U22) If $F_\gamma(<\alpha_{k_1} \ldots \alpha_{k_m}>)$ and $F_{\gamma'}(<\beta_{j_1} \ldots \beta_{j_m}>)$ are both well formed then either $\delta_{k_n} = \delta_{j_n}$ or $\gamma \neq \gamma'$.

Constraining the structural operations to specific domains in order to protect U20 would effectively enforce a 1:1 correspondence between structural operations and syntax rules. Some authors have adopted the convention of employing a common index for syntax rules and operations "for ease of reference" (*sic* Janssen). It now transpires that such uniformity should actually be mandatory if UG principles are to be inviolate.

7.2.3. Model Theoretic Semantics

An intensional model was defined in §2.1.4 as $M = < D, W, T, \leq, I >$. The orthodox interpretation function I is now defunct, but the remaining apparatus survives as:

Model base $M = < D, W, T, \leq >$.

Montague's definition of the set *Type* of types is exactly as presented in §2.1.1, and the set *den(a, M)* of

possible denotations for type a, now relative to a model base M, is defined precisely as in §2.1.4. Replacement of the intensional model by a model base is unproblematic since the interpretation function I played no part in the definition of "possible denotation".[†87]

Members of $den(<sa>, M) = den(a, M)^{W\times T}$ are to be regarded as *senses* for items of type a which are now to be distinguished from *meanings* of type a. The set of meanings of type a is defined as:

(U23) $mng(a, M, J) = den(a, M)^{W\times T\times J}$.

where J is the "set of all complexes of remaining relevant features of possible contexts of use". The intention is plainly to accommodate pragmatic features as adumbrated in Montague's earlier papers, [M2, M3], and to cater for phenomena such as those encountered in discourse analysis where interpretation is dependent upon adjacent prose.

In order to relate the new interpretation system of U14 to orthodox model theoretic semantics Montague introduces the notion of a *Fregean interpretation* for L:

(U24) $FI = <B, G_\gamma, f>_{\gamma\in\Gamma}$

such that for some model base M, some context set J and some type assignment σ (which behaves exactly like f of 3.2.1) :

(U25) $B \subseteq \cup_{a\in Type} mng(a, M, J)$.

(U26) If $\zeta \in X_\delta$ then $f(\zeta) \in mng(\sigma(\delta), M, J)$.

(U27) If $F_\gamma(<\alpha_{k_1} \ldots \alpha_{k_m}>) = \alpha_{k_n}$ and $\beta_{k_j} \in mng(\sigma(\delta_{k_j}), M, J)$ then $G_\gamma(<\beta_{k_1} \ldots \beta_{k_m}) \in mng(\sigma(\delta_{k_n}), M, J)$.

A *Fregean model structure* for L might accordingly be defined as:

$$FMS = <<B, G_\gamma, f>_{\gamma\in\Gamma}, <M, J>>.$$

and a specific model as:

(U28) $FM = <<B, G_\gamma, f>_{\gamma\in\Gamma}, <w, t, j>>$.

where w is the actual world, t is the present moment and j is the given context of use.

---

†87. In fact Montague unhelpfully uses I for W×T and, curiously, makes no mention of the linear ordering. His notation for $den(a, M)$ is $den_{a, D, I}$

7.2.4. IL Under the General Theory

The syntax and semantics for **IL** as presented in 3.1 may be subsumed under the general theory in the following manner.[†88]

(U29) $\mathbf{IL} = \langle\langle\langle A, F_\gamma\rangle, X_\delta, S, t\rangle_{\gamma \in \{0, 1, 2, 3, <4, \tau \in Type>\}, \delta \in Type \cup (\{Type\} \times Type)}, R\rangle$.

where R (the ambiguating relation) is vacuously identified as the identity relation. The lexicon for **IL** may be introduced as:

(U30) $X_\delta = Var_\delta \cup Con_\delta$ (=ILs3+ILs4).

In order to formulate the full set S of rules we must suppose that $\zeta, \eta \in A$ and that $\tau \in Type$, and let

- $F_0(\zeta,\eta) =_{def} [\zeta\eta]$.
- $F_1(\zeta,\eta) =_{def} \zeta=\eta$.
- $F_2(\zeta) =_{def} \hat{}\zeta$.
- $F_3(\zeta) =_{def} \check{}\zeta$.
- $F_{<4,\tau>}(\zeta,\eta) =_{def} [\lambda\tau\zeta\eta]$.

We may then stipulate that for $\sigma, \tau \in Type$, the following rules are included in S:

(U31) $\langle F_0, \langle\sigma\tau\rangle, \sigma, \tau\rangle$ (= ILs6).

(U32) $\langle F_1, \tau, \tau, t\rangle$ (=ILs7).

(U33) $\langle F_2, \tau, \langle\sigma\tau\rangle\rangle$ (= ILs11).

(U34) $\langle F_3, \langle\sigma\tau\rangle, \tau\rangle$ (=ILs12).

(U35) $\langle F_{<4,\sigma>}, \sigma, \tau, \langle\sigma\tau\rangle\rangle$ (= ILs5).[†89]

The context set J becomes specifically a set of value assignments relative to M which replaces the old sequence set G: but whereas $g \in G$ assigned only *variables* to members of a domain, $j \in J$ is defined

---

†88. Montague considers only a truncated version of **IL**, contending that omitted constructions may be introduced by abbreviations, viz.

$\forall\alpha\Phi =_{def} [\lambda\alpha\Phi = \lambda\alpha[\alpha=\alpha]]$
$\neg\Phi =_{def} [\Phi = \forall\beta\beta]$ where $\beta = v_{0,t}$
$\exists\alpha\Phi =_{def} \neg\forall\alpha\neg\Phi$.

†89. In (U31) ... (U35) each rule is a *triple* as indicated in (U9). The sequences constituting second arguments have however not been enclosed in angles since it would then prove difficult to distinguish between $\langle\sigma, \tau\rangle$ (a sequence of types) and $\langle\sigma\tau\rangle$ (a type).

for all members of $\cup_{\delta \in \Delta} X_\delta$. In fact the generalisation is only apparent because constants turn out to be immune to changes in context. For any lexical item $\beta_{k,\tau} \in X_\tau$ we have $j(\beta_{k,\tau}) \in den(\tau, M)$.

A logically possible model for **IL** then becomes:

$$\langle\langle B, G_\gamma, f\rangle_{\gamma \in \Gamma}, \langle w, t, j\rangle\rangle$$

such that $\langle B, G_\gamma, f\rangle_{\gamma \in \Gamma}$ is a Fregean interpretation and:

(U36) if $\upsilon \in Var_\alpha$ then $f(\upsilon)(w, t, j) = j(\upsilon)$.

(U37) If $\alpha \in Con_\alpha$ then $f(\alpha)(w, t, j) = f(\alpha)(w, t, j')$ for all j' that are α variant to j.

(U38) $G_0(\alpha,\beta)(w, t, j) = \alpha(w, t, j)[\beta(w, t, j)]$.

(U39) $G_1(\alpha,\beta)(w, t, j) = 1$ iff $\alpha(w, t, j) = \beta(w, t, j)$.

(U40) $G_2(\alpha)(w, t, j) = \eta \in den(\tau, M)^{W \times T}$ such that, for all $w', t' \in W \times T$, $\eta(w', t') = \alpha(w', t', j)$.

(U41) $G_3(\alpha)(w, t, j) = \alpha(w, t, j)(w, t)$.

(U42) $G_{<4,\tau>}(\alpha, \beta)(w, t, j) = \eta \in den(\tau, M)^{den(\sigma, M)}$ such that, for all $\zeta \in den(\sigma, M)$,

$\eta(\zeta) = \beta(w, t, j')$ for all j' which differ from j in at most that $j'(\alpha) = \zeta$.

7.2.5. Concessionary Translation

As a concession to the faint hearted Montague suggests that, although direct construction of a (Fregean) model for a (fragment of) natural language is possible, and was indeed attempted in **EFL** it may be

> "somewhat more perspicuous to translate the fragment into another language for which an interpretation is already available."

This strategy he proceeds to justify. Let

$$\mathbf{L} = \langle\langle\langle A, F_\gamma\rangle, X_\delta, S, \delta_0\rangle_{\gamma \in \Gamma, \delta \in \Delta}, R\rangle$$

and

$$\mathbf{L}' = \langle\langle\langle A', F'_\gamma\rangle, X'_\delta, S', \delta'_0\rangle_{\gamma \in \Gamma', \delta \in \Delta'}, R'\rangle.$$

Then a *translation base* $TB = \langle g, H_\gamma, q\rangle_{\gamma \in \Gamma}$ from **L** to **L′** may be derived as follows. First the basic categories of **L** are related to basic or derived categories of **L′** by means of the functions g and q[†90]

---

†90. Montague in fact employs "*f*" for "*q*", but I find this usage confusing since "*f*" is already in use to denote contexts.

(U43) $g : \Delta \rightarrow \Delta'$ and $g(\delta_0) = \delta'_0$.

(U44) $q : \cup_{\delta \in \Delta} X_\delta \rightarrow \cup_{\delta \in \Delta} C'_\delta$.

(U45) If $\zeta \in X_\delta$ then $q(\zeta) \in C'_{g(\delta)}$.

Derived expressions of **L** may then be related to derived expressions of **L′** by means of "derived syntactic rules"[†91] given that:

(U46) H_γ and F_γ have common *arity* so that H_γ is a polynomial operation over $<A', F'_\gamma>_{\gamma \in \Gamma'}$.

(U47) If $<F_\gamma<\delta_k \ldots \delta_m>,\delta_n> \in S$ then $<H_\gamma<g(\delta_k) \ldots g(\delta_m)>,g(\delta_n)>$ is a derived rule of **L′**.

A translation base determines a unique homomorphism k from $< A, F_\gamma >_{\gamma \in \Gamma}$ into $< A', H_\gamma >_{\gamma \in \Gamma}$ such that:

(U48) $q \subseteq k$.

(U49) $k : A \rightarrow E \subseteq A'$.

(U50) $k(F_\gamma(<\alpha_k \ldots \alpha_m>))) = H_\gamma(<k(\alpha_k) \ldots k(\alpha_m)>)$.

from which we may conclude that:

(U51) If $\eta \in \cup_{\delta \in \Delta} C_\delta$ and $\eta' \in \cup_{\delta' \in \Delta'} C'_{\delta'}$ and $\eta R \zeta$ and $\eta' R \zeta'$ and $k(\eta) = \eta'$ then ζ' is the translation of ζ on the basis of ***TB***.

Suppose that in addition to the translation base $TB = < g, H_\gamma, q >_{\gamma \in \Gamma}$ from **L** into **L′** we already have a Fregean interpretation:

$FI' = < B', G'_\gamma, f' >_{\gamma \in \Gamma'}$ for **L′**.

Let us also suppose that the meaning assignment h' for **L′** determined by **L′** is a homomorphism from $< A', F'_\gamma >_{\gamma \in \Gamma'}$ *to* $< B', G'_\gamma >_{\gamma \in \Gamma'}$. Note that this last assumption is non trivial since the general theory introduces meaning assignments at (U19) only as homomorphisms *into*.

If for each $\gamma \in \Gamma$ it is the case that H_γ is a polynomial operation over $< A', F'_\gamma >_{\gamma \in \Gamma'}$ then there is exactly one algebra $< B'', G_\gamma >_{\gamma \in \Gamma}$ such that:

h' is a homomorphism from $< A', H_\gamma >_{\gamma \in \Gamma}$ *to* $< B'', G_\gamma >_{\gamma \in \Gamma}$.[†92]

---

†91. The terminology is altogether unhelpful. The purpose of the derived rules is *not* to gerrymander with the syntactic definition of **L′**, which *must* be taken as given, but to construct well formed formulae within the existing constraints of that syntax.

Thus:

$$FI'' = \langle B'', G_\gamma, f'' \rangle_{\gamma \in \Gamma}$$

is the interpretation for **L** induced by **L′**, ***F′*** and ***TB*** such that:

(U52) For all $\zeta \in \cup_{\delta \in \Delta} X_\delta$ we have $h'(q(\zeta)) = f''(\zeta)$.

(U53) $h' : A' \rightarrow B''$.

(U54) $h'(H_\gamma(\langle k(a_k) \ldots k(a_m)\rangle)) = G_\gamma(\langle h'(k(a_k)) \ldots h'(k(a_m))\rangle)$.

The meaning assignment h'' determined by FI'' is accordingly a composition of two functions h' and k so that:

> "If we are given a translation from **L** into **L′** together with an interpretation for the already known language **L′** then *an* interpretation for **L** is determined." (italics mine)

7.3. Indeterminacy of Translation

Plainly Montague's exposition of the translation base strategy should not be construed dynamically: a heuristic for evolving induced interpretations has *not* been described. Rather Montague offers a static monitoring device for confirming the validity of putative translation bases. The distinction is akin to that made by Chomsky between the rules of a generative grammar conceived as operations and the same rules regarded as constraints on well formedness.

The grammarian is not being invited to devise in accordance with (U43) ... (U51) sets of purely syntactic transformations constrained only by the exigencies of type compatibility, thereafter to discover fortuitously that, in virtue of the mathematics of homomorphism, (U52) ... (U54) also hold, and that the exercise has unexpected semantic relevance. Moreover Montague grammarians do not so behave: for, as has been illustrated in Karttunen's investigation of interrogative noun phrases, syntactic decisions are typically dictated by antecedent semantic considerations throughout.

Considered even as a static constraint however, Montague's formulation of the induced translation mechanism is not unproblematic. Can any set of purely syntactic transformations, albeit constrained by the

---

†92. This is the import of the terminal *Remark* in UG section 1.

requirements of type compatibility, be guaranteed not to introduce semantic distortion? Preservation of type compatibility is undoubtedly a necessary condition for the maintenance of semantic integrity, but it could hardly be sufficient. For example the translation of a rigid designator of type <<s<et>>t> from language **L** into a type compatible correlate in language **L′** would not automatically protect its rigidity which depends upon the elimination of language specific variations at divers points of reference.

Suppose that for a given model base $< M, J >$ we have a Fregean interpretation:

$$FI' = < B', G'_\gamma, f' >_{\gamma \in \Gamma'}$$

for language **L′** and a translation base *TB* from **L** into **L′**, with:

$$FI'' = < B'', G_\gamma, f'' >_{\gamma \in \Gamma}$$

the interpretation for **L** induced by **L′**, *FI′* and *TB*. Suppose also that:

$$FI = < B, G_\gamma, f >_{\gamma \in \Gamma}$$

is the interpretation for **L** obtained by the direct method. Then we may conclude both that:

$$B \subseteq \cup_{\tau \in Type} mng(\tau, M, J)$$

and

$$B'' \subseteq \cup_{\tau \in Type} mng(\tau, M, J)$$

but it is not a necessary truth that $B = B''$. Likewise it is far from trivial to claim that:

for all $\zeta \in \cup_{\delta \in \Delta} X_\delta$ the equality $f(\zeta) = f''(\zeta)$ holds

or that there is equality between the relatia of (U20) and (U54).

Let us say that **L** and **L′** are *expressively equivalent* iff for all $< M, J >$, for all induced interpretations $< B'', G_\gamma, f'' >_{\gamma \in \Gamma'}$ and for all direct interpretations $< B, G_\gamma, f >_{\gamma \in \Gamma}$ it is the case that $B = B''$ and $f = f''$. Then my contention is that the existence of a translation base from **L** into **L′** does not guarantee expressive equivalence, and that in default of expressive equivalence between the languages involved a translation base may be employed to induce an interpretation only at the implementor's peril.

Consider now the case of a translation base from English to the language **IL** of intensional logic. English contains pronouns which may be used indexically, or which may refer anaphorically not within sentence boundaries but to previous discourse. Such phenomena present no problem for Montague's

general theory which provides in its direct interpretation mechanism a set ***J*** "as the set of all complexes of remaining relevant features of possible contexts of use".†93 A translation base from English to **IL** cannot but map *all* pronouns to expressions involving variables: but variables are to be assigned values by members of a vestigial version of ***J*** which in the interpretation of **IL** does no more than the old Tarskian sequence set **G**. The language **IL** is not equipped to handle pragmatic and discourse sensitive issues. English and **IL** are not expressively equivalent, and a translation base from one to the other may be used with impunity only for those *fragments* of English wherein pragmatic and discourse sensitive considerations do not arise.

It should accordingly come as no surprise to discover that if we employ **IL** as an intermediary in the interpretation of a sentence such as:

(150) The girl whom he loved kissed John.

then we shall lose the reading according to which "he" is used indexically, and not as an anaphoric reference to John.

Janssen's rejection, [J3, J4], of the tactic whereby the indexical reading would be obtained by allowing a syntactic variable to surface at the topmost node and then deleting its subscript, is based upon the recognition that an appropriate valuation of the free variable in the corresponding **IL** formula could not be achieved via an assignment function not designed to handle context sensitivity. Indeed Janssen's advocation of the *variable principle*" may be seen as a recommendation to restrict the employment of a translation base targetted on **IL** to subsets of English for which expressive equivalence may be maintained.

Free variables, as Janssen observes, [J4], are more like constants since the number of possible assignments in a given situation is narrowly circumscribed: thus a pragmatic language would require a mechanism for making constants sensitive to contextual features besides tense and possible world, a ploy briefly considered by Montague in *Pragmatics and Intensional Logic*, (PIL), [M3]. As noted above each function $j \in J$ applied in the semantics for **IL** has explicitly been rendered vacuous in the case of constants.

Popular folklore suggests that Montague recommended unrestricted use of a translation base from

---

†93. Objectors might of course protest that the introduction of ***J*** simply acknowledges a problem rather than solving it, but the proposed solution is at least no less arbitrary than the provision of ad hoc tailor made referents for anaphora in the form of frames and scripts.

English to **IL** as a general method for the analysis of English, a myth accidentally supported by the fact that on two occasions (**UG** and **PTQ**) he employed just such a translation base to handle *fragments*. Montague was however at pains to concentrate only upon subsets of English equivalent in expressive power to **IL**. In order to accommodate a fragment involving pragmatic features he would presumably have adopted a translation base targetted on a language embodying the characteristics of **PIL**.

The confusion engendered by uncritical acceptance of the folk version of Montague semantics is well illustrated by the following quotation from Wilks, [W3]:

> "The assumption at every stage is that there is a molecular confrontation between language and the world. This seems plausible enough perhaps for "John loves Mary" but wildly improbable for sentences whose meaning is explained by their inferential structure to other sentences. Consider how far Montague's assumption is from any AI or "frames" view of meaning on which we cannot talk about meaning independently of large structures of knowledge existing, as it were, outside the sentence examined. *There is no place for that in Montague's system ...*"
> (Italics mine)

It should now be clear that Montague's general theory makes no such assumption†94. Indeed AI "frames"

---

†94. Although not essentially *molecular*, Montague semantics is necessarily *referential*, a characteristic which Wilks contends is open to Wittgensteinian objections. In my view however Wilks mistakes the nature of Wittgenstein's reservations about the referential theory of meaning. It is not the theory itself but the epistemological confusion engendered by a particular misuse thereof in support of a bogus thesis concerning language acquisition which disturbs Wittgenstein, hence his derisive remarks about ostensive definition. Only abhorence of the philosophical chaos latent in this misuse explains and justifies his preoccupation.

The target of Wittgenstein's wrath was not Tarski (with whose works there is no reason to believe that he was familiar) but Russell's *Philosophy of Logical Atomism*, [R12], which endorses a bizarre view to which Russell's pupil Wittgenstein may have earlier subscribed:

> "It would be absolutely fatal if people meant the same by their words. It would make all intercourse impossible, and language the most hopeless and useless thing imaginable, because *the meaning you attach to your words must depend on the nature of the objects you are acquainted with*, and since different people are acquainted with different objects they would not be able to talk to each other unless they attached quite different meanings to their words." [R12 page 195] (Italics mine)

This quotation is tantamount to a commitment to *private language*, a conception to the repudiation of which a large portion of *Philosophical Investigations* is devoted, but concerning which Wilks has nothing to say. Private language provides the foundation for a number of epistemological problems associated with the British empiricist tradition, hence a spurious doctrine of language acquisition founded upon a respectable theory of reference has disastrous consequences, leading eventually to solipsism. Wittgenstein was accordingly properly concerned.

Model theoretic semantics is *not* a psychological model: it has nothing to say about how language is acquired. Plainly one need not know that the referent of "five" is the equivalence coset of $\{\{\{\{\{\varnothing\}\}\}\}\}$ in order to count up to five, indeed one may become numerate without ever being aware of this semantic truth.

Ironically it is in the AI world that the conception of private language occasionally surfaces. For example private language is presupposed by the contention that message content can be considered as an autonomous *conceptual* level which is one step removed from the details of the way in which the message is publicly expressed.

might be conceived as an engineering solution to the problem of implementing Montague's context set ***J***. An impoverished commitment to isolated sentence semantics is only the consequence of the rash adoption of a concessionary translation base from English to **IL** in circumstances where expressive equivalence patently fails.

7.4. Computational Compositional "Semantics"

The orthogonal tradition of Computational Compositional "Semantics" (CCS) originates with the work of Hobbs and Rosenschein, [H7], and includes that of Rosenschein and Scheiber, [R7], Schubert and Pelletier, [S1], and Hirst, [H6], all of whom are concerned with the construction of "Natural Language Understanding Systems". If any of these researchers were claiming to be engaged in the computational implementation of Montague grammar then my contention to the contrary would constitute a criticism. To my knowledge however no such claim has ever been registered by any of the authors themselves: the claim tends to be made gratuitously on their behalf (perhaps even to their embarrassment) by spokespersons from the AI community. What is the more surprising is that only in the case of Hobbs and Rosenschein, [H7], does the claim have even a shadow of prima facie plausibility: for they do indeed offer a computer utilisation of the *output* of a Montague grammar, although the grammar itself is not implemented.

A cursory glance at Rosenschein and Scheiber, [R7], and Schubert and Pelletier, [S1], suffices to confirm that their respective grammatical models are based not upon **UG** nor upon **PTQ** but rather upon Gazdar's **GPSG**. The former authors mention Montague's name once in a parenthetical remark whilst the latter actually *define* a Montague grammar (rather eccentrically) as essentially embodying those higher order features which they are determined *not* to implement. Hirst, [H6], provides a mechanism for translating from a transformational variation of the Marcus parser into frame based representations. That Hirst's approach is described as "Montague inspired" is no evidence that the author has embarked on research into Montague semantics: indeed he explicitly *contrasts* his approach with Friedman's "tool for understanding Montague semantics better", which is par excellence a computational implementation of Montague grammar. I shall limit my comments to establishing that the gratuitous claim of the AI fraternity is ill founded.

The four works cited differ in the degree to which they recognise Montague's influence, their assessment (if any) of the nature of Montague's program, and their acknowledged areas of affinity thereto. Where

they share a measure of agreement is in their abandonment of model theoretic interpretation (a sine qua non of Montague grammar), their definition of the role of the computational linguist and their identification of the discipline computational "semantics". Jointly they provide their own evidence that CCS is not (and should therefore not purport to be) concerned with the computational implementation of Montague semantics.

According to Hobbs and Rosenschein, Montague's method (which they take to be typified by PTQ rather than specified by UG) involves three phases:

(i) Syntactic analysis with respect to a *categorial grammar*. (Italics mine)

(ii) Translation into an expression of intensional logic.

(iii) Model theoretic interpretation of the IL expression.

Their avowed intent is to replace phase (iii) with a system of procedural analogues. Hirst, who also offers a synopsis of the method is less concerned with the sequential phases than with the properties of a Montague semantics which he identifies as:

(a) Mapping to set theoretic semantic objects optionally represented by IL expressions.

(b) Correspondence between syntactic and semantic types.

(c) A 1:1 correspondence between syntactic and semantic rules.

(d) Compositionality of the semantics.

It is left to Schubert and Pelletier to summarise the full significance of the correspondence identified as property (c) and which Montague proposes:

> "Montague grammarians ... postulate a strict homomorphism from the syntactic categories and rules of natural language to the semantic categories and rules required for its formal interpretation."

7.4.1. On "Replacing" the Model Theoretic Interpretation

The requirement for strict homomorphism imposes a constraint upon compositionality only if the latter is interpreted *mathematically*, ie. only if the meaning of a compound is to be a function of the mean-

ings of its parts in the mathematical sense of "function". Provided that phase (iii) and the corresponding property (a) are retained it is a consequence of strict homomorphism that the phrase "every man", which is incontrovertibly a syntactic component in the sentence "every man walks", must map to an identifiable semantic correlate involving higher order abstraction. Likewise, since "loves mary" is a verb phrase in the sentence "John loves Mary", the verb "loves" must be mapped not to a dyadic predicate but to a one placed function returning a monadic predicate. Without phase (iii) and property (a) the restraint upon compositionality becomes vacuous. Absolutely anything could count as a meaning representation for a component, and absolutely any mechanism for assembly as a composition.

Earlier it was observed that only Montague's rules of functional application are strictly speaking categorial. By liberally, albeit incorrectly, classifying the entire syntax as a categorial grammar Hobbs and Rosenschein focus attention upon the fact that the correlates of *all* Montague's syntax rules involve "subtle patterns of functional application" of some sort. There is however a latent ambiguity in their exposition. Must the correlates they mention behave *semantically* as functional compounds or may they be merely notational devices obeying the *syntactic* rules of typed lambda calculus? Hobbs and Rosenschein claim that:

> "Montague has made a significant contribution to the computational semanticist by showing possible *formats for the representation* of meanings of individual words, and mechanisms for the combination of meanings which are considerably more elegant than most computational alternatives now in use." (Italics mine)

For Montague however the ultimate objective was philosophical explication of semantic concepts in extra linguistic terms to which mere metalinguistic representation was but a prelude. Phase (iii) embodies the essence of his program while phase (ii) is but an option to obviate direct correlation.

The goal of any semantic theory, according to Hobbs and Rosenschein, is to express English strings in terms of an *antecedently understood metalanguage*. (Italics mine). Now antecedent understanding is a purely subjective matter: that a metalanguage be antecedently understood by a particular subject is neither a necessary nor a sufficient condition for its objective explanatory adequacy. That I understand a formalism does not guarantee its accuracy, nor does my failure to comprehend undermine its respectability. By con-

centrating on the subjective issue of antecedent understanding rather than the objective issue of extra linguistic explanatory adequacy the authors create the impression that the form of ultimate interpretation might be tailored to suit special subjects such as computers. Emphasis is shifted from extra linguistic *interpretation* to metalinguistic *formulation*; thus ironically it is phase (ii), which to Montague constitutes a concessionary option, that assumes paramount importance.

It behoves us to enquire in what sense phase (iii) could be disregarded; for when the authors propose to replace the model theoretic interpretation with procedural analogues "without destroying the overall framework of functional composition and application" the implications of the prevarication between treating "functional application" as a semantic or as a syntactic concept must be considered.

If the requirements of phase (iii) are actually *rejected* as improper then what survives is a purely notational device the interpretation of which need not be in any way *compatible* with model theoretic semantics. The direct correlation of natural language with set theoretic constructions is optional. The intermediate correlation of natural language with the language **IL** of intensional logic is optional. The correlation of a language containing symbols referring to *functions* with set theoretic entities is however mandatory. If the symbol "*f*" is to be regarded as a unary function symbol then we have *no choice* but to treat its referent as a particular set of ordered pairs; conversely if we reject such referents then "*f*" does not represent a *mathematical* function at all, and alternative notational devices might serve equally well. On this view any technique whatsoever for assembling components would qualify as a formalism which allows one "to express the composition of meaning in much the way Montague does using subtle patterns of functional application", and might be classified unhelpfully as a Montague grammar. Such a view has permeated AI and results in all manner of alien approaches being classified as Montagovian. Thus Hirst, having abandoned both model theoretic interpretation and **IL** representation acknowledges Montagovian inspiration on the grounds that his system, which employs neither Montague's syntax nor his semantics, includes a form of typing, observes a rule by rule hypothesis and subscribes to some form of compositionality principle.

Perhaps however phase (iii) is merely to be *ignored*, while remaining available in case of need. In this case the formulae constructed in phase (ii) would retain their mathematical significance, and phase (iii)

would remain an option for monitoring the suitability of proposed changes in syntax. Those who subjectively are unable antecedently to understand model theoretic interpretation would on this account be invited to pay no attention thereto, but its influence would remain intact, and only *compatible* alternative interpretations would be countenanced. By choosing not to make use of model theoretic semantics we do not invalidate it.

I suggest that in fact Hobbs and Rosenschein actually replace phase (iii) only in the latter sense: their procedural analogues are designed to preserve compatibility and presume the existence of an orthodox grammar. On this view, had they provided an implementation of the grammar presupposed their proposals would have amounted to a *computer utilisation* as defined above. They appear however to take the output from such a grammar as given, providing no clue as to how it should be generated.

Model theoretic semantics makes recourse to infinite sets and functions and hence model theoretic constructions are not in practical terms computable. It has been demonstrated by Friedman, Moran and Warren, [F4], that the smallest interesting model for **PTQ**, having two points of reference and two entities, generates sixteen sets of possible denotations nine of which have a cardinality greater than or equal to 2^{32} while four contain $2^{2^{514}}$ members or more.

A complete computer model for Montague semantics accordingly presents the same kind of problem as would be encountered were one to employ Godel numbering as a hashing algorithm to generate content addressable file storage. Such considerations do nothing to undermine the philosophical respectability of either model theory or Godel numbering.

7.4.2. Semantics or Verification?

Faced with such problems, and having adopted a view of semantics according to which model theoretic interpretation may be in some sense disregarded, Hobbs and Rosenschein conclude that:

> "the computational linguist is after a quite different type of *semantic theory*, one which is ultimately machine theoretic rather than model theoretic." (Italics mine)

adding for good measure that:

> "model theoretic semantics ... are not antecedently understood by computers."

Wittgenstein has however demonstrated, [W6], that if language is to be employed as a means of communication then a common semantic interpretation must be available to *all* participants: there can be no *private* language. Thus if human beings are ever to communicate with computers using natural language then at all costs a "quite different type of semantic theory" for the computer must be avoided. The only alternative would be the ludicrous situation hypothesised in "*Philosophy of Logical Atomism*".†95 A machine oriented *semantics* would undermine the whole enterprise of human - computer communication by natural language; moreover, whatever their avowed intentions, a machine oriented *semantics* is not what Hobbs and Rosenschein provide.

Not since the benighted heyday of logical positivism has a theory of *meaning* been confused with a theory of *verification*; but that is precisely the confusion involved when the authors declare that, for the computational linguist:

> "the meaning of an expression is ... the behaviour of the procedure it is transformed into."

Later they write that:

> "the existential quantifier is a procedure which searches through the entities until it finds *[an entity]* with the required properties".

and again:

> "we cannot determine the truth and falsity of "every man seeks a frog" except after the fact, and then (unreliably) only if the seeking was successful."

In both cases the issue is not one of meaning but of verification; but a system of verification presupposes an antecedent semantic theory. We cannot set out to verify a sentence S unless we already know what S means.

Conceived as a system of verification, the computational analogues proposed by Hobbs and Rosenschein have merit, but far from requiring the rejection of phase (iii) they actually depend on the availability of the model theoretic interpretation if verification and semantic interpretation are to be compatible. The relationship between a compositional theory of verification and compositional semantics is elegantly, if

---

†95. See previous footnote.

unintentionally, expressed by Moore, [M7]:

> "The main desideratum of a system of logical form is that its semantics be compositional ...
>
> This is needed for meaning dependent computational processes"

ie. a compositional method of verification will be unavailable in default of a prior compositional semantics.

The verificational analogues introduced by Hobbs and Rosenschein provide no insights as to how to develop a Montague grammar, rather they take a viable grammar for granted. Indeed CCS is not a semantic theory at all but a theory of verification by computer, hence a fortiori it cannot contribute to the development of Montague semantics.

Dissatisfaction with the use of the term "semantics" by computational linguists is explicitly voiced by Rosenschein and Scheiber, [R7] who advocate "translation" or "transduction" for any processing phase which does no more than map sentences into a metalinguistic representation.†96 That the output from transduction is to be used for verification rather than semantic analysis is clear from the summary given of their system, the details of which need not concern us since, as mentioned above, it is oriented not towards Montague grammar but towards GPSG:

> "a simple control structure ... accepted an input, translated it into logical form, reduced the translation to first order logic and then either asserted the translation in the case of a declarative sentence *or attempted to prove it* in the case of interrogatives." (Italics mine)

Schubert and Pelletier, [S1], exclude themselves from the ranks of Montague grammarians by rejecting strict homomorphism, which they take to be definitive. The initial motivation which they offer for abandoning Montague's higher order analysis in favour of the Russellian first order alternative discussed in 2.2 is that:

> "intuitively quantified terms such as "everyone" and "no one" simply don't bear the same sort of relationship to objects in the world as names, even though the evidence for placing them in the same syntactic category is overwhelming."

to which one feels inclined to respond that as committed Russellians they might have heeded the dictum

---

†96. In this they would find allies in Halvorsen and Ladusaw, [H1].

that common sense embodies the metaphysics of the stone age.

A more plausible defence of first order methodology emerges in the authors' claim that higher order abstraction is not in fact essential for the handling of *de dicto* readings of intensional verbs because these verbs may always be lexically decomposed. Unfortunately they treat "lexical decomposition" and "paraphrase" as synonyms and offer as a decomposition of:

(151) John worships a unicorn.

the fatuous pair:

(152) John acts thinks and feels as if he worshipped a unicorn.

(153) John worships an entity which he believes to be a unicorn.

from which we may immediately generate the regress:

(154) John acts thinks and feels as if he acted thought and felt as if he worshipped a unicorn.

(155) John worships an entity which he believes to be an entity which he believes to be a unicorn.

and so on. Montague's analysis can surely survive such arguments.

My only real complaint against the practitioners of CCS is that they seek to *define* computational linguistics in terms of the construction of NLUS thus excluding Thompson's category of *computation in the service of linguistics* altogether. A complete model of Montague semantics may indeed be impossible in view of the combinatorial explosion described above: a partial model may nonetheless provide a useful tool for the investigator even if it be no more than an expert's assistant.

7.5. Montague Grammar as a Machine Translation Interlingua

The interlingual approach to machine translation, [H9], assumes that it is possible to translate source language texts into semantico-syntactic representations common to more than one language. Interlingual translation proceeds in two stages: first the source language SL is analysed into interlingual representations, and secondly these representations are synthesised into text in the target language TL. Given the need to translate to and from n distinct languages, an interlingual approach requires the construction of $2n$ programs as opposed to n^2 if recourse must be made to direct translation.

Landsbergen's "Rosetta" project at Phillips Research Laboratories, [L1, L2], is directed towards the

design of a machine translation system employing as interlingua tree representations of the (unreduced) IL expressions assigned to sentences by a variant of Montague grammar known as M-grammar.†97 Such tree representations are terned "logical derivation trees" or LD-trees. The assumption is that equivalences may be established between sentences of distinct languages to the extent that the M-grammatical analogues of their analysis trees interface with a common LD-tree. Montagovian analysis trees are in fact replaced in M-grammar by a *pair* of trees comprising a syntax tree or S-tree and a derivation tree or D-tree.

An M-grammar is defined by Landsbergen, [L1], as a triple:

$$MG = \langle G_\sigma, B, R \rangle$$

where R is a set of *M-rules*, B a set of basic expressions and G_σ a "surface" grammar. Initially G_σ is introduced as a standard loop free context free grammar (CFG) of form:

$$G_\sigma = \langle V_N, V_T, \Sigma, P \rangle$$

with V_N a set of syntactic categories, V_T a set of terminal symbols, Σ a distinguished symbol and P a set of production rules; but eventually, [L2], such a CFG is acknowledged to be but a special case of a "surface" grammar.

The crucial characteristic of G_σ is that to any surface string s it must assign an S-tree t such that LEAVES(t) = s, ie. the surface phrase must be recoverable as a linear sequence of terminal labels on the S-tree. The non terminal nodes of an S-tree will be labeled by members of V_N.

Following Partee's hint that the rules of a Montague grammar are best defined in terms of labeled bracketings (or equivalently in terms of trees), Landsbergen introduces *M-rules* as compositional syntax rules which construct S-trees out of n-tuples of smaller S-trees and basic expressions. To ensure comparability with UG†98 each M-rule in R takes the form:

---

†97. Although a *variation* of Montague grammar, M-grammar is strategically orthodox in that it embodies a compositional syntax and observes the rule by rule hypothesis. By contrast the work of Nishida and Doshita, [N1], whatever its intrinsic merits would not count as a computational utilisation of Montague grammar despite the seductive title. Nishida and Doshita propose to parse English source language by an unspecified method into an intermediary functional notation EFR (English oriented Formal Representation) which bears a passing resemblance to IL, but which is not defended on philosophical grounds. The lexical items in EFR are then replaced by "conceptual phrase structure forms" (CPSF) which constitute recipes for constructing Japanese oriented conceptual phrase structures (CPS). Target language text is finally generated from the CPS. Since the intermediate structures are designed specifically for English to Japanese translation, the approach should properly not be described as interlingual. The technique has no other affinity to Montague grammar than that the generation of Japanese from CPSF may be represented as a compositional data flow, but compositionality although a necessary condition is hardly sufficient evidence for strategic orthodoxy.

†98. See clause U9. I use S_γ where Landsbergen prefers R_i.

$$S_\gamma = \langle\langle C_\gamma, A_\gamma\rangle, \langle\delta_1, ..., \delta_n\rangle, \delta_r\rangle$$

to be interpreted "operation A_γ may combine a sequence $\langle u_1, ..., u_n\rangle$ of S-trees dominated by $\langle\delta_1, ..., \delta_n\rangle$ in order to make a new tree u_r dominated by δ_r provided that the sequence $\langle u_1, ..., u_n\rangle$ satisfies condition C_γ". The condition C_γ must be applicable to trees output by the surface grammar G_σ. Both G_σ and the M-rules generate S-trees, but whereas the first operate top down the second construct an S-tree in a bottom up fashion.

The derivational history of an S-tree constructed by means of the M-rules may be represented by means of a *derivation* tree (D-tree) having basic expressions at its terminal nodes and its non terminal nodes labeled not by a category symbol but rather by the identifier S_γ of the rule responsible for the construction. If for a given S-tree more than one D-tree may be constructed, then that S-tree must be ambiguous with regard to the M-rules.

As in **PTQ** there corresponds to each syntactical rule S_γ of M-grammar a translation rule T_γ. Accordingly a *logical* derivation tree (LD-tree) may now be identified as that tree derived from the D-tree by replacing leaf nodes with the translations of basic expressions and substituting the appropriate translation rule identifier T_γ for S_γ at each non terminal node.

Trivially, an S-tree may be reconstructed from a D-tree simply by applying the rules indexing non terminal nodes on the latter; thus for each D-tree d Landsbergen defines M-GENERATOR(d) to be the set of S-trees so generated. Recoverability of a D-tree from an S-tree is dependent upon Landsbergen's *reversibility condition* which requires that for each operation A_γ invoked by rule S_γ there must be an inverse operation A'_γ such that:

$$u_r \in A_\gamma(\langle u_1, ..., u_n\rangle) \leftrightarrow \langle u_1, ..., u_n\rangle \in A'_\gamma(u_r).$$

To each *compositional* M-rule S_γ there must correspond an analytical M-rule S'_γ such that:

$$S'_\gamma = \langle\langle C'_\gamma, A'_\gamma\rangle, \delta_r, \langle\delta_1, ..., \delta_n\rangle\rangle$$

where A'_γ is a function from S-trees dominated by δ_r for which condition C'_γ holds to *n*-tuples of S-trees dominated by $\langle\delta_1, ..., \delta_n\rangle$.

In terms of the analytical M-rules Landsbergen is able, for each S-tree t, to define M-PARSER(t) to be the set of D-trees that generate t:

"M-PARSER applies the analytical M-rules to the S-tree ... in a top to bottom fashion. ... Successful application of a rule S'_γ results in a tuple $<u_1, ..., u_n>$. M-PARSER is then applied to $u_1, ..., u_n$. Each application of M-PARSER to u_j gives a (possibly empty) set of D-trees for u_j. For each tuple of D-trees $<d_1, ..., d_n>$ in the Cartesian products of these sets a D-tree $\gamma<d_1, ..., d_n>$ is constructed." [L1]

Part of Landsbergen's earlier paper is devoted to proving that:

$t \in$ M-GENERATOR(d) $\leftrightarrow$ d $\in$ M-PARSER(t).

Assuming this to be the case, let:

S-PARSER(s) = the S-tree assigned by G_σ to a sentence s.

LOGTREE(d) = the LD-tree corresponding to D-tree d.

LOGTREE'(e) = the D-tree corresponding to LD-tree e.

Then Landsbergen is able to define the function:

$$\text{ANALYSIS(s)} =_{def} \{e:\exists t \exists d(t \in \text{S-PARSER(s)} \wedge d \in \text{M-PARSER(t)} \wedge e \in \text{LOGTREE(d)})\}$$

which maps sentences to LD-trees, and a reverse function:

$$\text{GENERATION(e)} =_{def} \{s:\exists t \exists d(d \in \text{LOGTREE'(e)} \wedge t \in \text{M-GENERATOR(d)} \wedge s \in \text{LEAVES(t)})\}.$$

mapping LD-trees to surface phrases.

Two M-grammars G_i and G_j are described by Landsbergen as *logically isomorphic* iff:

$$\forall e(\exists s(s \in \text{GENERATION}_i(e)) \leftrightarrow \exists s'(s' \in \text{GENERATION}_j(e)))$$

ie. iff for each LD-tree assigned to a sentence s by G_i there is a sentence s' to which G_j assigns the same LD-tree.

According to Landsbergen's thesis, two languages are logically isomorphic if they may be described by logically isomorphic M-grammars; moreover inter translation between logically isomorphic languages using the common LD-trees as interlingua becomes feasible.

Rules such as Montague's S3, which in PTQ contain meta variables, are accommodated in M-grammar by recourse to *rule schemes*. A rule having a parameter *p*, ie. one of form:

$$S_i = \langle\langle C_{i,p}, A_{i,p}\rangle, \langle\delta_1, ..., \delta_n\rangle, \delta_r\rangle$$

may be defined in terms of a schema:

$$\langle P_i, I_i, A_i\rangle$$

where P_i is a parameter set, $I_i(\langle u_1, ..., u_n\rangle) \subseteq P_i$, and $A_i(p, \langle u_1, ..., u_n\rangle) = u_r$.

Landsbergen requires that:

$$C_{i,p}(\langle u_1, ..., u_n\rangle) =_{def} p \in I_i(\langle u_1, ..., u_n\rangle)$$

and

$$A_{i,p}(\langle u_1, ..., u_n\rangle) =_{def} A_i(p, \langle u_1, ..., u_n\rangle).$$

A corresponding *analytical rule schema* :

$$\langle P_i, I'_i, A'_i\rangle$$

may likewise be defined such that $I'_i(u_r) \subseteq P_i$ and $A'_i(p, u_r) = \langle u_1, ..., u_n\rangle$, so as to licence the equivalences:

$$C'_{i,p}(u_r) =_{def} p \in I'_i(u_r)$$

and

$$A'_{i,p}(u_r) =_{def} A'_i(p, u_r).$$

With regard to the special case of S3 Landsbergen defines the schema:

$$\langle P_3, I_3, A_3\rangle$$

such that:

P_3 = {<g,Q,n>: g is a gender, Q a set of paths, n an index}.

$I_3(\langle u_1, u_2\rangle)$ = {<g,Q,n>: g is the gender of the first terminal CN in u_1, Q = {p: u_2.p=he_n or u_2.p=him_n}, n≥0}.

$A_3(\langle g,Q,n\rangle, \langle u_1, u_2\rangle)$ = CN[u_1 such that u'_2] where u'_2 is the result of replacing the variable at $u_2.p_i$ by an appropriate pronoun for each $p_i \in Q$.

In these formulations Q isolates all paths from the root node which terminate with a variable having the index in question, while $u_2.p_i$ is the termination of the *ith* path so identified.

Only when parameterised rules are taken into consideration does the full significance of the

distinction between PTQ style analysis trees and Landsbergen's <S-tree, D-tree> pairs become apparent. This distinction may be illustrated by comparing the sub tree dominated by 3:3,1 in fig 5 with the corresponding trees for the phrase "woman such that she loves him$_m$" produced by an M-grammar as depicted in fig 74.

****

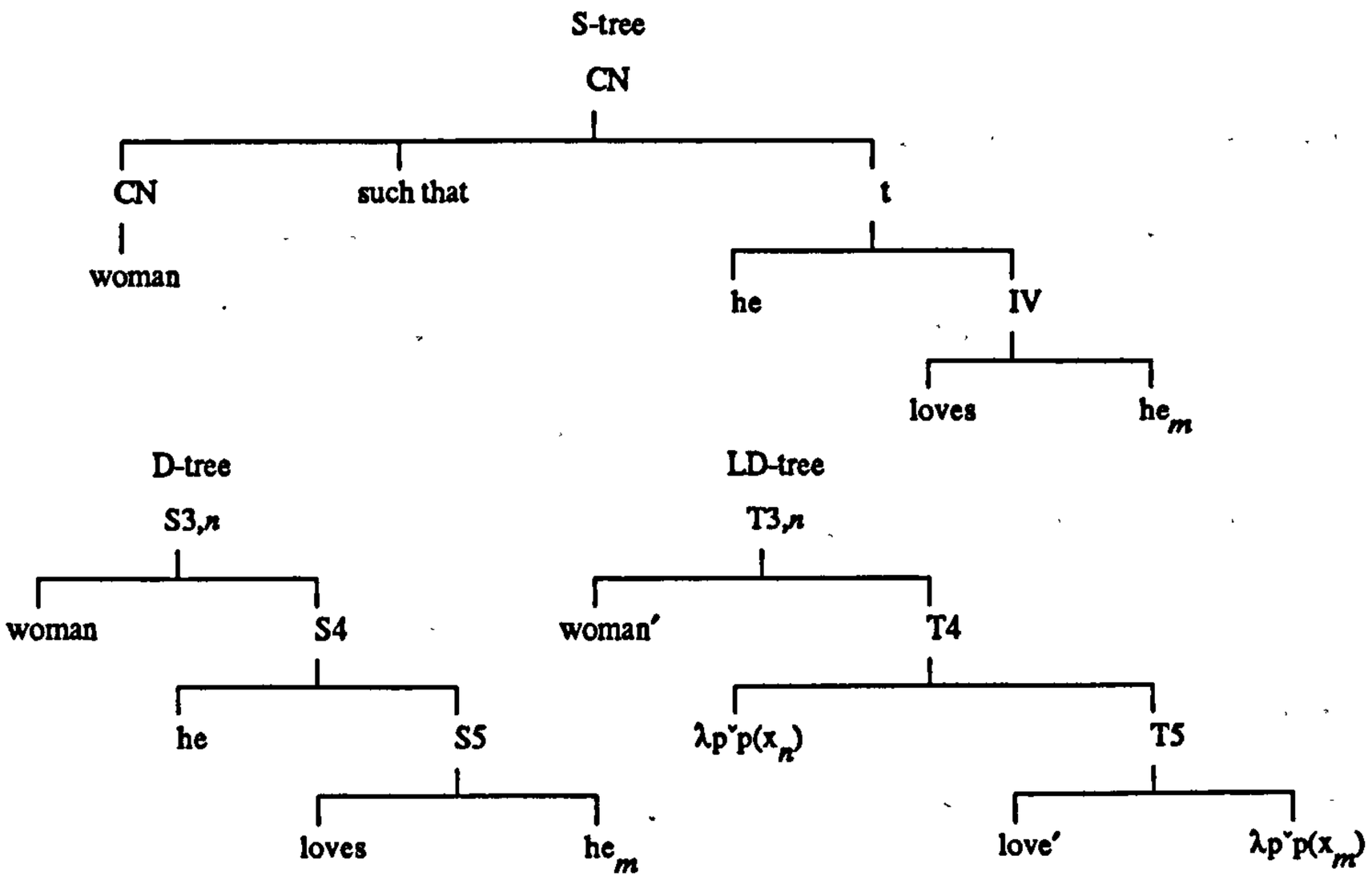


Fig 74

****

On a Montagovian analysis tree, the phrase to date is recorded as a nodal label while the leaf positions remain constant during construction. By contrast the surface phrase in an M-grammar must be recoverable as a linear sequence of *leaves* from the S-tree: accordingly the M-rules responsible for building the S-tree must be capable of *amending* leaf positions dynamically. Inspection shows that the operation A_3 defined by Landsbergen has precisely such an effect.

It is clear from Landsbergen's exposition that prior to such an amendment a leaf position on the S-tree may be occupied by a syntactic variable awaiting replacement, however S-trees are also the output of the "surface" grammar G_σ, thus a "surface" grammar must be able to produce trees having leaf sequences that are *not* surface strings. Just as a final filter is needed to eliminate analysis trees having syntactic variables surviving to the topmost node, so too some form of filter is implied which will allow S-trees characterised by variables as inputs to M-rules, but which will reject them as representations of surface sentences.

Arguably a diachronic *series* of S-trees would be needed fully to represent the generation of a sentence, together with a corresponding series of D-trees. For how is the *surface* pronoun on the illustrated D-tree to be correlated with the correct logical variable on the LD-tree? Landsbergen's solution is to invoke the analytical inverse of the M-rule equivalent of S3 and to apply it to the S-tree in order to recover the unamended leaves, ie. to resurrect an earlier D-tree. The appropriate items from the S3 *analytical rule schema* are:

$I'_3(u) =_{def}$ {<g,Q,n>: u = CN[CN[...] such that t[...]] where g is the gender of the first terminal CN in the elder daughter of u, the younger daughter does not contain variables with index n, and Q is a subset of the set of paths to pronouns with gender g in the younger daughter.

$A'_3(<g,Q,n>,u) =_{def} <u_1, u_2>$ where u_1 is the elder daughter of u, and u_2 is the result of replacement, within the younger daughter of u, of the terminals at all p_i in Q by a variable with index n.

Landsbergen's recourse to LD-trees rather than fully reduced IL expressions has much to recommend it, but we may question whether or not it is necessary to depart so radically from Montague's own method of representation in order to generate such trees. There is no obvious advantage to be gained from abandoning the practice of holding the phrase to date together with derivational information on a single analysis tree; moreover the LD-tree is analogous to the superimposed translation tree which I have been employing in previous illustrations, which suggests that LD-trees may in need be constructed directly from orthodox Montague analysis trees. Thus the success of Landsbergen's program invites the question could similar success be achieved by a less devious parsing mechanism?

CHAPTER 8. COMPUTATIONAL INVESTIGATIONS

¶ In this chapter we consider those computational investigations into Montague grammar which have the greatest affinity to the present endeavour, and which have had the most significant influence upon its development. Janssen's experimental generator includes an alternative to the language of intensional logic translator **LILT**, while the Friedman Warren algorithm has been extended and converted into DCG format for incorporation in **TMDCG**: accordingly the chapter is devoted exclusively to the work of these authors.

8.1. Janssen's Experimental Generator

The value of a computational investigation into the operation of a Montague grammar is admirably illustrated by Janssen's experimental generation program, [J1, J2]. We have already alluded to the shortcomings of the original **PTQ** structural operations $f_{3,n}$, f_4, f_5, and $f_{10,n}$, to Janssen's rejection of "left overs" and vacuous quantification, to his stipulation of the "variable principle", and to his own hyperrule reformulation of the grammar of **PTQ**: but what has not been made explicit is that most of the errors discussed in his later papers, [J3, J4], were originally drawn to his attention through the medium of his generator.

```
**********************************************************************************

        procedure make(category);
                begin
                        rule := choose-rule(category);
                        if rule ≠ choose-lexical(category)
                        then begin
                                        make(rule.argument1);
                                        if rule.argument2 ≠ nil then make(rule.argument2)
                                end
                        else choose-lexical(category)
                end.
```

Fig 75

```
**********************************************************************************
```

Janssen's computational implementation of Montague grammar is unusual in that unlike Landsbergen's, which has already been discussed, and Friedman's, to be considered shortly, it does *not* include a parser. The program generates *arbitrary* syntactic structures, comparable to Landsbergen's D-trees, by compositional application of *randomly* selected grammar rules in accordance with the recursive

algorithm of fig 75. Once a D-tree has been composed, the corresponding surface sentence may be identified by applying the structural operations corresponding to nodal labels, presumably during a post order traverse of the tree.

By generating *arbitrary* sentences through arcane albeit legitimate rule application sequences the program is able accidentally to discover anomalies: hence I classify it as an "experimental generator". Inaccuracies in **PTQ** discovered by the experimental generator include the retention of non finite forms in compound verb phrases (cf. example (63)), implausible relative clause stacking, (cf. example(56)), the absence of reflexivisation, left over variables, and the *semantic* implications of vacuous quantification, (cf. fig 24).

Janssen's programatic solution to the problem of left overs and vacuous quantification can only be regarded as ad hoc. First he removes the "else" clause from the algorithm of fig 75 so as to generate D-trees minus their leaves. Next he demands the insertion of variables with index *n* only within the scope (younger daughter) of he-*n* binding rules (ie. S3,*n*, S14,*n*, S15,*n* and S16,*n*), insisting that each such binding rule have at least one appropriate variable within its scope. Finally he inserts suitable lexical items at the remaining unoccupied leaf positions. Although this solution reflects the effect of adequate corrections to **PTQ** grammar, it does not of itself engineer such corrections, nor does it indicate the form that they should take.

The D-trees constructed by the experimental generator provide the basis for translation of the corresponding sentence into **IL**. As in Landsbergen's project, the D-tree is first mapped to a *logical* D-tree by substituting basic **IL** for lexical expressions and translation rule identifiers for those of syntax rules. An **IL** expression for the topmost node is then computed by traversing the LD-tree, applying the rule at each node to its previously computed inputs, and *fully reducing* the nodal result on a step by step basis in the manner illustrated in chapter 2.[†99]

---

†99. It may prove opportune to introduce at this point a distinction made by Friedman, [F6], between direct translation, reduced translation and extensionalised translation. In her terminology, direct translation is the immediate result of applying Montague's T rules to an analysis tree and performing no further logical operations on the results, reduced translation involves the application of principles *not* founded upon meaning postulates, while extensionalised translation incorporates the postulates. Both direct and reduced translation turn out to be reversible (ie. an analysis tree, and hence a surface sentence, can be recovered from the logical expression): and in the case of direct translation the recovery is trivial, suggesting that for machine translation purposes reduction stands in need of justification. Reduction in Friedman's sense corresponds to Janssen's principles R3, R7 and R8, while *full* reduction as implemented by Janssen, and also in **LILT** involves extensionalised translation.

Janssen's Reduction Rules			
Rule	Input	Output	Condition
R1	α^*	$\lambda p[p\{\hat{}\alpha\}]$	
R2	$\hat{}\Psi\{\eta\}$	$\Psi(\eta)$	
R3	$\Phi(\psi)(\eta)$	$\Phi(\eta,\psi)$	Φ translates a TV
R4	$\delta(\hat{}\upsilon)$	$\delta_*(\upsilon)$	δ translates IV or CN
R5	$\delta(\hat{}\upsilon,\hat{}[\nu]^*)$	$\delta_*(\upsilon,\nu)$	δ translates a TV
R6	$\delta(\hat{}\upsilon,\hat{}\lambda pp\{\check{}\nu\})$	$\delta_*(\upsilon,\nu)$	as for R5
R7	$\check{}\hat{}\Phi$	Φ	
R8	$\lambda z[..z..](\alpha)$	$[..\alpha..]$	
R9	$\neg\neg\Phi$	Φ	
R10	$\Box\Box\Phi$	$\Box\Phi$	
R11	$\delta(x)$	$\delta_*(\check{}x)$	IV $\delta \neq$ *rise, change*
R12	$\delta(x,n)$	$n\{\check{}\lambda y[\delta_*(\check{}x,\check{}y)]\}$	TV $\delta \neq$ *seek, conceive*
R13a	$\exists x[\delta(x)\wedge p\{x\}]$	$\exists\upsilon[\delta(\hat{}\upsilon)\wedge p\{\hat{}\upsilon\}]$	CN $\delta \neq$ *price, temperature*
R13b	$\exists x[\delta(x)\wedge[..x..]\wedge p\{x\}]$	$\exists\upsilon[\delta(\hat{}\upsilon)\wedge[..\hat{}\upsilon..]\wedge p\{\hat{}\upsilon\}]$	as for R13a
R14a	$\forall[\delta(x)\rightarrow p\{x\}]$	$\forall\upsilon[\delta(\hat{}\upsilon)\rightarrow p\{\hat{}\upsilon\}]$	as for R13a
R14b	$\forall[\delta(x)\wedge[..x..]\rightarrow p\{x\}]$	$\forall\upsilon[\delta(\hat{}\upsilon)\wedge[..\hat{}\upsilon..]\rightarrow p\{\hat{}\upsilon\}]$	as for R13a
R15a	$\exists y\forall x[\delta(x)\leftrightarrow x=y]\wedge p\{y\}]$	$\exists\nu[\forall\upsilon[\delta(\hat{}\upsilon)\leftrightarrow\upsilon=\nu]\wedge p\{\hat{}\nu\}]$	as for R13a
R15b	$\exists y\forall x[\delta(x)\wedge[..x..]\leftrightarrow x=y]\wedge p\{y\}]$	$\exists\nu[\forall\upsilon[\delta(\hat{}\upsilon)\wedge[..\hat{}\upsilon..]\leftrightarrow\upsilon=\nu]\wedge p\{\hat{}\nu\}]$	as for R13a
R16	$in'(n)(p)(e)$	$n\{\hat{}\lambda y[in'_*(\check{}y)(p)(e)]\}$	
R17	δ	$\lambda y\lambda x\delta(x,y)$	ad hoc

Fig 76

Much of the complexity of Janssen's reduction mechanism, which requires a total of seventeen reduction rules, (listed in fig 47), stems from his determination to preserve unamended every nuance of the grammar of **PTQ**. Montague had a curiously perverse predilection for counter productive abbreviations which introduce unnecessary processing overheads and which add little by way of perspicuity. He allows for example a "brace convention" according to which:

$\check{}\zeta(\alpha)$ may be written as $\zeta\{\alpha\}$

despite the fact that the marginal brevity so gained is more than outweighed by the nuisance value of hav-

ing to restore the original form in order to proceed with legitimate reductions. Plainly ˆζ{α} is equivalent to ˇˆζ(α) which by the principle of "down up" cancellation reduces to ζ(α); thus Janssen introduces a rule, R2, which accomplishes restoration of the unabbreviated form so that cancellation may proceed. A simpler policy, which I have implicitly been adopting from the outset, is to proscribe brace notation altogether and work with unabbreviated primitives.

A second convention adopted by Montague is "superstar" notation according to which:

john* abbreviates λpˇp{ˆjohn′) where $p = v_{0,<s<<se>t>>}$.

Recourse to Bennett's simplified system of typing together with abandonment of brace notation allows us to substitute:

john* abbreviates λpˇp(john′) where $p = v_{0,<s<et>>}$.

However again I question the value of this abbreviation. Janssen's rule R1 expands superstar notation, but as is apparent from earlier examples I have eschewed this convention and opted to use unabbreviated forms from the start hence avoiding any restoration premium.

Janssen's rules R4, R5, R6 and R11 ... R16 are all designed to accommodate the implications of Montague's meaning postulates. Six rules, (R13a ... R15b), are required to service MP2 alone, while R4 merely simplifies some of this postulates constructions. Rule R11 corresponds to MP3, R5 and R6 together to MP4, R12 to MP5 and R16 to MP8. As we have already seen, Bennett's system of typing reduces the number of postulates required by **PTQ** from nine to four, preserving only MP1, MP4, MP8 and MP9, thus rendering some of the aforementioned reduction rules redundant. Rule R17 is both anomalous and particularly ad hoc since it is to be applied only in case (a) the input corresponds to a complete sentence, (b) no other reduction applies and (c) δ does not occur in the context δ(α,β). In the language of intensional logic translator **LILT**, which processes the analysis trees output by the **TMDCG** parser, it is accordingly possible drastically to reduce the number of reduction rules needed.

Of Janssen's rules only R3 (relational notation), R7 (down up cancellation) and R8 (λ conversion) remain critical for **IL** reductions, to which **LILT** adds a single rule to drive the postulates. Janssen's R9 and R10 are not explicitly **IL** reduction rules but standard theorems, the first corresponding to double negation and the second to "S5" modal reduction. **LILT** includes a further principle, that of the *substitutivity of*

identicals which Janssen is forced to omit, despite conceding that his system is unable (without it) correctly to process identity statements.

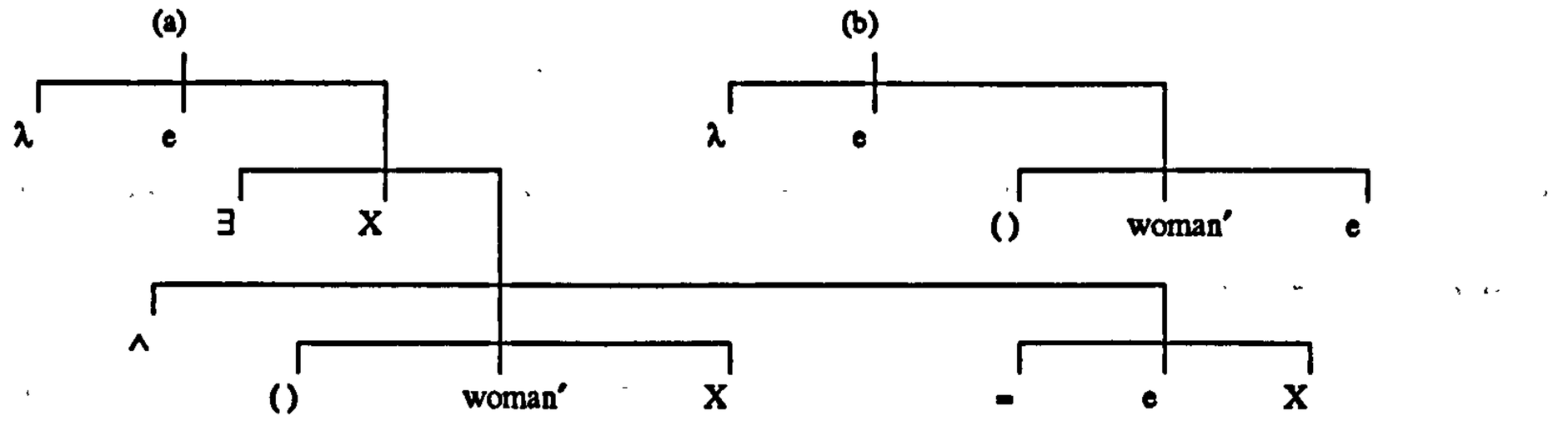


Fig 77

Janssen attributes this shortcoming to his technique for achieving reductions by means of *local* tree transformations, ie. transformations on small *contiguous* subtrees. Logical formulae are represented not by strings but as labelled trees having atomic symbols as leaves, while local transformations involve raising subtrees to ancestor nodes, thus eradicating contiguous intervening branches. This technique is unable to accomplish the reduction of:

λnλeˇn(^λZ(e=Z))[^λq∃X(woman′(X)∧ˇq(X))]

to:

λe(woman′(e))

since it would necessitate as a final stage the non-local transformation of tree (a) into tree (b) in fig 77.

In **LILT** such a reduction is achieved by recourse to PROLOG's pattern matching facilities, but no such facilities are available in Janssen's chosen implementation language, ALGOL 60. The lack of string handling, of which he explicitly complains, turns out to be not the only restriction which makes this procedural language less than felicitous for his purpose.

8.2. The Friedman Warren Algorithm

Whereas Janssen's generator will produce random ungrammatical sentences given inaccuracies in the original grammar implemented, thus serving as a monitoring device for that grammar, the parser presented by Friedman and Warren, [F3], provides a means for systematically testing the grammaticality of

a *specified* input sentence by attempting with backtracking all possible routes towards a derivation. Arguably the algorithm upon which the parser is based constitutes the most important single contribution to the computational investigation of Montague grammar to date.

8.2.1. Recursive Descent Using FVB and SA Lists

Like Landsbergen and Janssen, Friedman and Warren are content to operate not with full blown Montague analysis trees but with D-trees, the restoration of individual nodal phrases remaining as a manual exercise. The purpose of the Friedman Warren algorithm (FWA) is to handle the introduction and subsequent binding of indexed syntactic variables on D-trees. For expository purposes the authors describe FWA as a mechanism for traversing a tree constructed by the CFG rules of PTQ, ie. all rules except S3, S14, S15 and S16, and for performing such operations as will generate an alternative tree involving the variable binding rules. Thus given tree (a) of fig 78, tree (b) will be produced[†100] as an alternative.

Such a method of exposition is in fact rather misleading, suggesting as it does that the parser essentially requires two passes, the first to construct the CFG tree and the second to transform it. In fact the parser, which operates by recursive descent simulating the context free rules, constructs tree (b) in a single pass by exercising options as and when they arise: tree (b) may accordingly be returned *instead* of tree (a) rather than *subsequent* to it.

---

†100. Friedman and Warren in fact adopt a curious ternary tree representation for constructs involving variable binding rules, but nothing is lost by returning to conservative binary forms. Familiarity with the basic recursive descent parsing technique, [A2], for processing a CFG, and the incorporation therein of tree building facilities is presupposed. Using this technique, each non terminal symbol in the grammar corresponds to a parameterised procedure responsible for identifying the category concerned and returning a contribution to the parse tree, while each preterminal symbol maps to a similarly parameterised procedure which consumes appropriate items from an input string. A rule of form:

A → B A

clearly requires a recursive call from A to A; but the transformation of a simple non recursive grammar such as that required by fig 78 (a) may be illustrated by the conversion of the CFG:

(S4) S → NP VP

(S5) VP → TV NP

(Bnp) NP → John|Mary

(Btv) TV → love

into the DCG parsing program (using PROLOG "grammar rule notation"):

```
s(s4(NP,VP)) --> np(NP),vp(VP).
vp(s5(V,NP)) --> vt(V),np(NP).
np(john) --> [john].
np(mary) --> [mary].
vt(love) --> [loves].
```

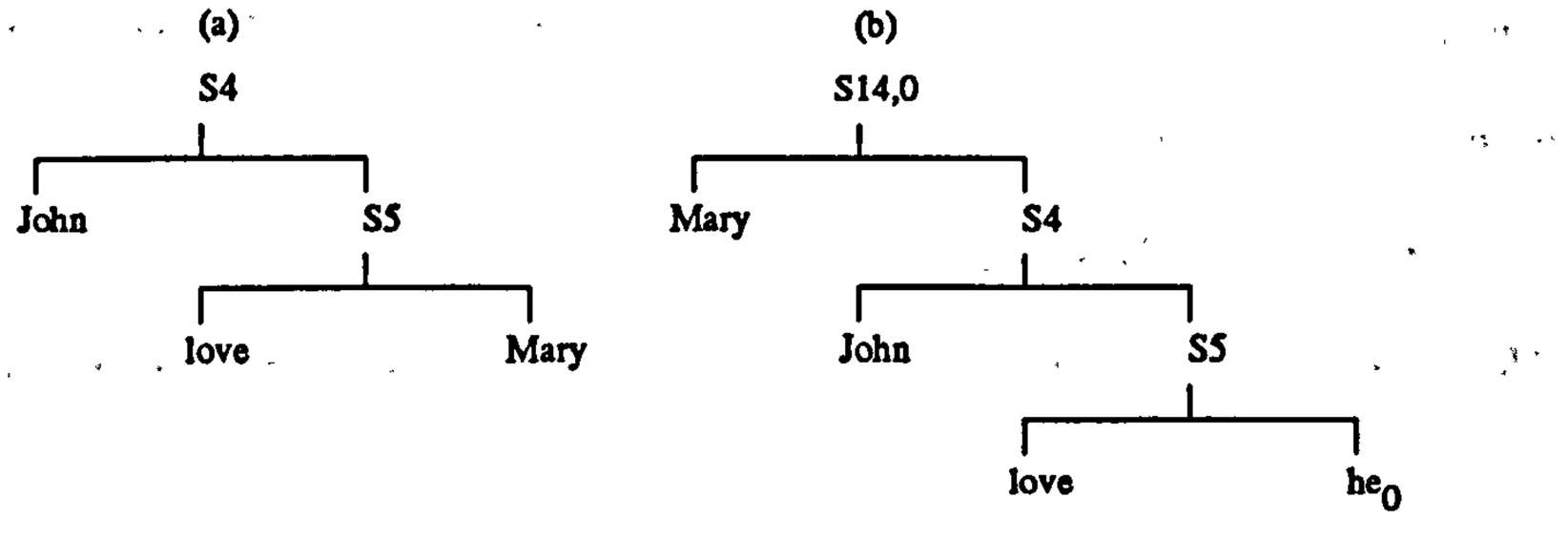


Fig 78

In recursive descent parsing, a procedure charged with processing a given category χ is responsible for returning a χ-node as its contribution to the parse tree. According to FWA, whenever a *noun phrase other than a pronoun* is parsed, the procedure responsible must exercise an *option*: it may either return a *default node* as specified by the context free rules of the grammar, or alternatively it may introduce as a *replacement node* the next available syntactic variable in sequence. Each invocation of a parsing procedure, or equivalently each node created by such an invocation, must be furnished with a list, the *free variables below* list (FVB) in which is maintained a record of all unbound variables dominated by the node in question. Should a noun phrase parsing procedure exercise the option to introduce a variable, such a record must be raised in the FVB list, which accordingly serves as an *output* parameter to the procedure.

Records in the FVB take the form of *bindings*, which in Friedman and Warren's own exposition consist of a pair <Var,NP> where Var is the syntactic variable introduced and NP the *phrase* replaced thereby. I suggest however that NP should properly be regarded not as the term phrase but as the entire *default node*; moreover NP must be feature marked with both number (if plurals are admitted) and gender. The FVB at any given node comprises the union of the FVBs of its daughters (preorder successors) plus or minus any additions or subtractions engineered at the node itself.

Procedures responsible for returning sentence, intransitive verb phrase or common noun nodes must also construct a default node as required by the context free rules, but they too are provided with an *option*: they may elect to quantify into the default node. Quantification is accomplished by selecting a binding from the current FVB list and creating a new node having the default as younger daughter, the node from the binding as elder daughter, and the new node flagged with the identifier of the quantification rule in

question together with the index subscripting the binding variable. The binding is then removed from the FVB list, whereafter the new node may itself optionally be subjected to quantification: ie. the quantification mechanism includes a recursive call. Vacuous applications of the quantification rules are automatically eliminated by this method: if a variable does *not* appear free in the younger daughter, which constitutes the *scope* of the quantification, then neither does it appear in the FVB list from which the variable of quantification is extracted.

Anaphoric references are handled by furnishing each procedure with a second list, the *substitutions above* (SA) list, which acts as an input parameter and contains bindings originating at preorder predecessors. The SA of the top level procedure must be set to the empty list whereafter the SA at any other given node will be the union of the SA of its parent and the FVB of its elder sister if any. Whenever a noun phrase parsing procedure identifies a surface pronoun with number and gender <n,g>, it may select from the SA list any binding wherein the NP node has consonant features: hence the recording of such features is imperative. The syntactic variable from the chosen binding then replaces the surface pronoun as a contribution to the parse tree.

An alternative form of binding <Var,CN>, in which a new syntactic variable is paired not with a noun phrase but with a common noun node CN, is required in order to accommodate relative clauses. The recursive descent basis for relative clause parsing becomes a CFG dilution of S3 of form:

CN → CN such that S

which of course makes no attempt to associate pronouns in the embedded sentence with the antecedent common noun.†101 On encountering "such that", the procedure which parses common nouns, having already identified the head common noun, has merely to pair it with a variable and pass the binding into the SA list of the sentence procedure charged with parsing the embedded sentence.†102 On completion the head com-

---

†101. The problems associated with recursive descent parsing of left recursive rules are addressed in the next chapter. Friedman and Warren adopt the "Well formed substring table" solution whereby a non recursive rule identifies basic common nouns and records each finding, while the "recursive" rule rather than calling itself selects a previous finding as head CN, parses a sentence and records its final result to enable further "recursion".

†102. The extension of this mechanism to non stilted relative clauses is not difficult. In TMDCG the syntactic variable together with its feature marking to date is passed to the embedded sentence in a "HOLD" list from which it may be taken to replace the extraposed noun phrase. Subsequent anaphoric references are handled by recourse to the SA list as in the original.

Friedman and Warren toy with the idea of adding the binding retrospectively to the FVB of the head CN, but since it would be immediately extracted on percolation to the S3 node which made the addition the manoeuvre is rejected as pointless.

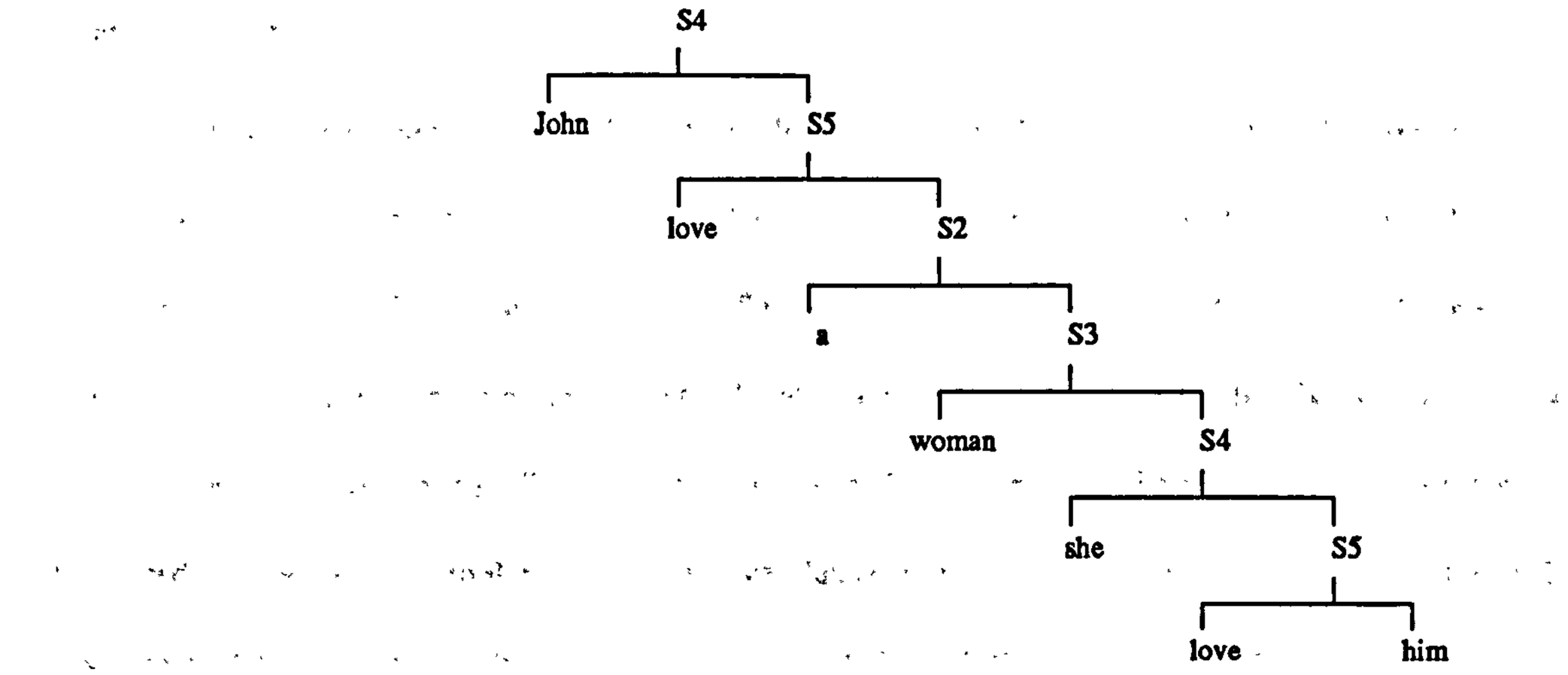

S4
John
S5
love
S2
a
S3
woman
S4
she
S5
love
him

Fig 79

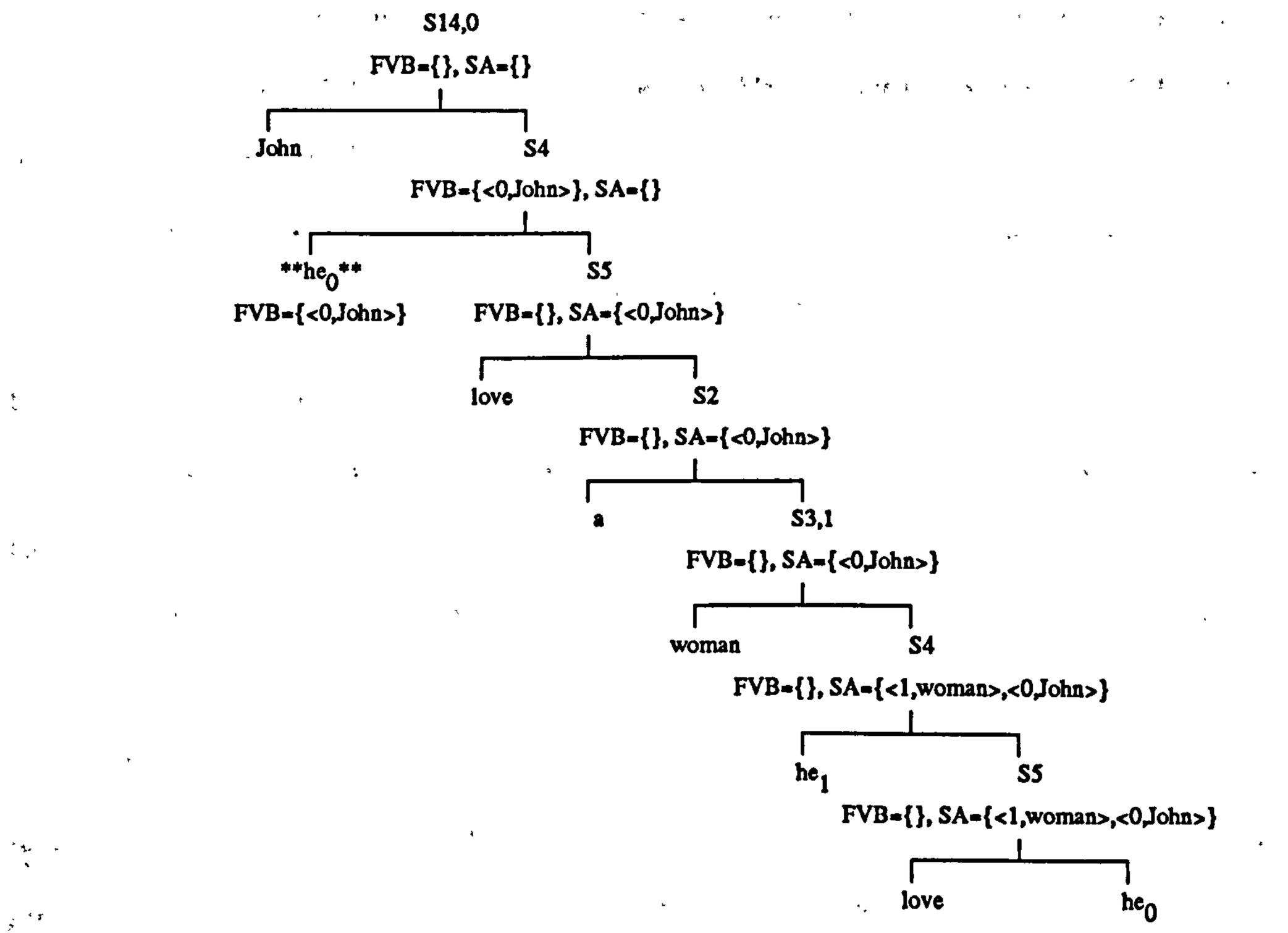

S14,0
FVB={}, SA={}
John
S4
FVB={<0,John>}, SA={}
he0
FVB={<0,John>}
S5
FVB={}, SA={<0,John>}
love
S2
FVB={}, SA={<0,John>}
a
S3,1
FVB={}, SA={<0,John>}
woman
S4
FVB={}, SA={<1,woman>,<0,John>}
he1
S5
FVB={}, SA={<1,woman>,<0,John>}
love
he0

Fig 80

mon noun becomes an elder sister of the embedded sentence, the dominating node being S3 flagged with the variable index.

The interplay between FVB and SA lists is illustrated in figs 79 and 80 where the first represents the notional output of a context free grammar and the second the result of applying the algorithm. A binding abbreviated to "<0,John>" is created at the node flagged **...**, and this is passed up to the S4 node which eventually becomes younger daughter of the top level quantification node. The binding is then passed down to all preorder successors in their SA lists where it is joined at the embedded S4 node by a second binding, abbreviated "<1,woman>" supplied by the relative clause rule. All surface pronouns in fig 79 are in due course replaced by indexed pointers to antecedents of the appropriate number and gender.

Vacuously relativised clauses are tolerated by Friedman and Warren on the grounds that unlike vacuously quantified sentences, cf. fig 24, they do not generate *semantic* nonsense. This tolerance is surprising since the elimination of vacuity may be accomplished with consummate ease. A simple traverse of any sub tree will suffice to establish whether or not a given variable is dominated by the root of the sub tree, accordingly I suggest the following constraint:

(FWA1) The relative clause procedure must reject any candidate younger daughter which does *not* dominate the variable provided in the initial binding.

be added to the algorithm.

A necessary condition for the elimination of "left overs", ie. indexed variables remaining free at the topmost node of a tree, is that the FVB list of the top level procedure be constrained to be empty. Such a constraint constitutes a final filter as countenanced, albeit with reservation, by Janssen, [J4]. The above condition is however not sufficient.

Although the FVB list at a given node is initialised as the union of the FVB lists of preorder *successors*, it is quite possible for a binding in FVB to include a node which eventually becomes a preorder *predecessor*: for the effect of the quantification option is precisely to adopt elder daughters by extracting a binding from the FVB of the younger daughter. In default of any counter measures therefore it would be possible, given the sentence:

(156) A woman such that she loves a man such that he loves her walks.

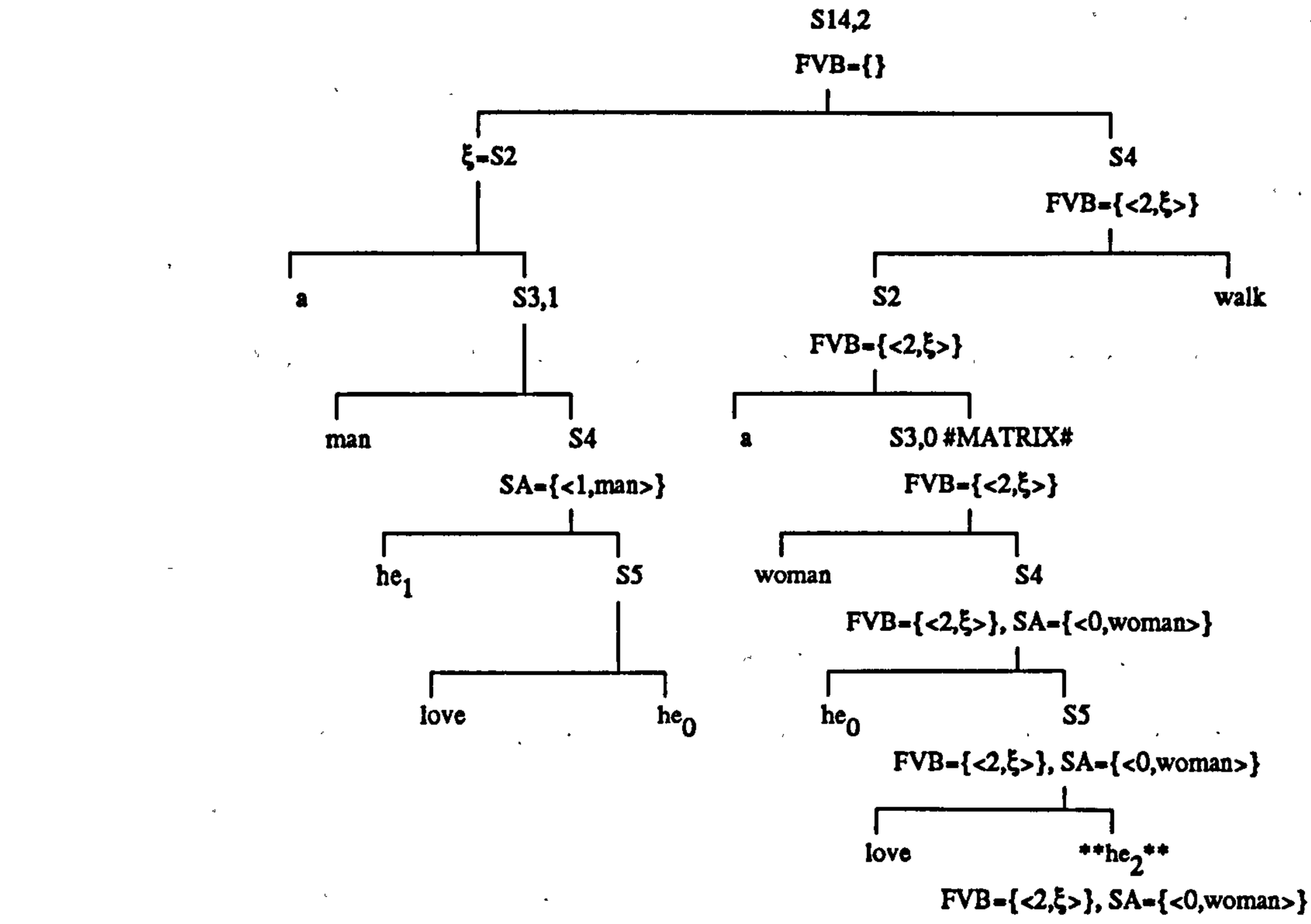


Fig 81

to derive the tree of fig 81. As Friedman and Warren point out however, manual†103 application of the syntax rules labelling the nodes would generate at the root the string:

(157) A woman such that she loves a man such that he loves her_0 walks.

which contains a "left over" variable despite the fact that the FVB list at the top node is empty as required.

When the parser attempts to process the string "a man such that he loves her", it has reached the point marked **...** in fig 81, the binding <0,woman> having been provided in the SA list by the as yet incomplete relative clause procedure call which is attempting to construct the S3 node marked MATRIX. Anaphoric connection between the surface pronoun and "woman" is established and a default node representing " a man such that he loves her_0", (identified in fig 81 as ξ), is constructed. On exercising the option to replace this default node with a variable, the binding <2,ξ> is added to the FVB list whence it proceeds to percolate upwards until control is returned to the MATRIX procedure. At this point the

†103. The fact that manual intervention at this juncture is necessary in order to appreciate the problem is in itself an incentive for introducing such modifications as would allow the automatic generation of genuine Montague analysis trees.

variable "he_0" is bound by relativisation *despite the fact that "he_0 is dominated by the node ξ which appears in the FVB list.* The FVB then continues to percolate to the highest S4 level at which point ξ is adopted as an elder daughter although containing a variable bound by a preorder successor.

To prevent such an occurrence Friedman and Warren insist that the algorithm must include a specific constraint[†104] which amounts to:

(FWA2) No variable binding rule may bind a variable dominated by the node of any other pairing in the current FVB list.

8.2.2. Handling Cataphora

In an earlier description of a prototype DCG implementation of the grammar of PTQ, [B4], I suggest a modification to the Friedman Warren algorithm which would accommodate cataphoric pronominal references and allow parses of sentences such as:

(158) The man such that he loves her kisses Mary.

The modification requires the provision of two additional lists of bindings, SRA (Substitutions Required on Arrival) and SRD (Substitutions still Required on Departure), for each parsing procedure, or equivalently at each node on the parse tree.

Whenever a surface pronoun is encountered during a parse, a check must first be made as hitherto in the SA list for a possible antecedent referent, and if a suitable candidate is found an anaphoric reference engineered in the manner already described. In the absence of a suitable antecedent referent, a check must then be made in SRA in case there has already been a cataphoric reference of matching number and gender, in which case the present surface pronoun may be replaced by the variable already available in the SRA binding. If neither SA nor SRA contains a suitable candidate then a new cataphoric reference must be established by creating a binding wherein the NP field is a *dummy* node marked with the number and gender feature of the pronoun being parsed. The union of SRA and {*dummy*} becomes SRD; moreover the

---

†104. Were we to countenance Rodman's *constraint upon quantifier scope* then where the variable binding rule is the relative clause rule an even stricter condition would be required viz:

(FWA2a) The FVB list at the embedded sentence node in a relative clause construction must be empty.

thus effectively making the subordinate clause an island. In this way the element relativised would inevitably be given wider scope than any other element in the relative clause.

dummy node must be added to the current FVB list.

Before exercising the option to replace a default NP node with a variable, procedures which parse noun phrases must now first check in SRA to ascertain whether or not there is already a dummy awaiting the arrival of a node with the features of the default NP. If there is then the variable from the dummy binding becomes the replacement variable, and the default node replaces the dummy node in *all* its occurrences including those in FVB lists. Such a global replacement is trivially easy in a PROLOG implementation given the "logical variable" facility. Finally SRD becomes the difference between SRA and the dummy binding.

At any node, SRD will be equal to SRA plus any new dummy bindings contrived at the node or less any dummy bindings completed thereat. The SRA and SRD of the root node are required to be empty, whereafter the SRA of an elder daughter is initialised as the SRA of the parent, the SRA of a younger daughter is the SRD of the elder sister and the SRD of the parent is the SRD of the younger daughter.

A new constraint upon quantification is implied by this innovation, namely:

(FWA3) No variable binding rule may extract from FVB a binding still present in the current SRD list.

for without such a constraint it would be possible to return as a putative parse of (158) the tree of fig 82 wherein variable he_2 remains free in the younger daughter of the top node.

A dummy binding is initially created at the node flagged **...** and passed back up the tree in both SRD and FVB lists. Improper extraction from FVB results in quantification with a dummy node as elder daughter at the point marked %%...%%, the dummy in SRD continuing to percolate until it is transferred to the SRA of the younger daughter of the top node whereupon it proceeds to trickle downwards. On reaching the node flagged ##...## the cataphoric referent is located and *all* instances of the dummy, including that at node %%...%%, are completed whereafter SRD becomes SRA-{<2,dummy>}, ie. an empty SRD is allowed to percolate to the top node. The net result is the spurious tree of fig 82.

By including the new constraint we ensure that not only may no younger sister dominate quantification over a variable remaining free in the family of an elder daughter (the effect of FWA2), but the elder sister must extend the same courtesy to her sibling.

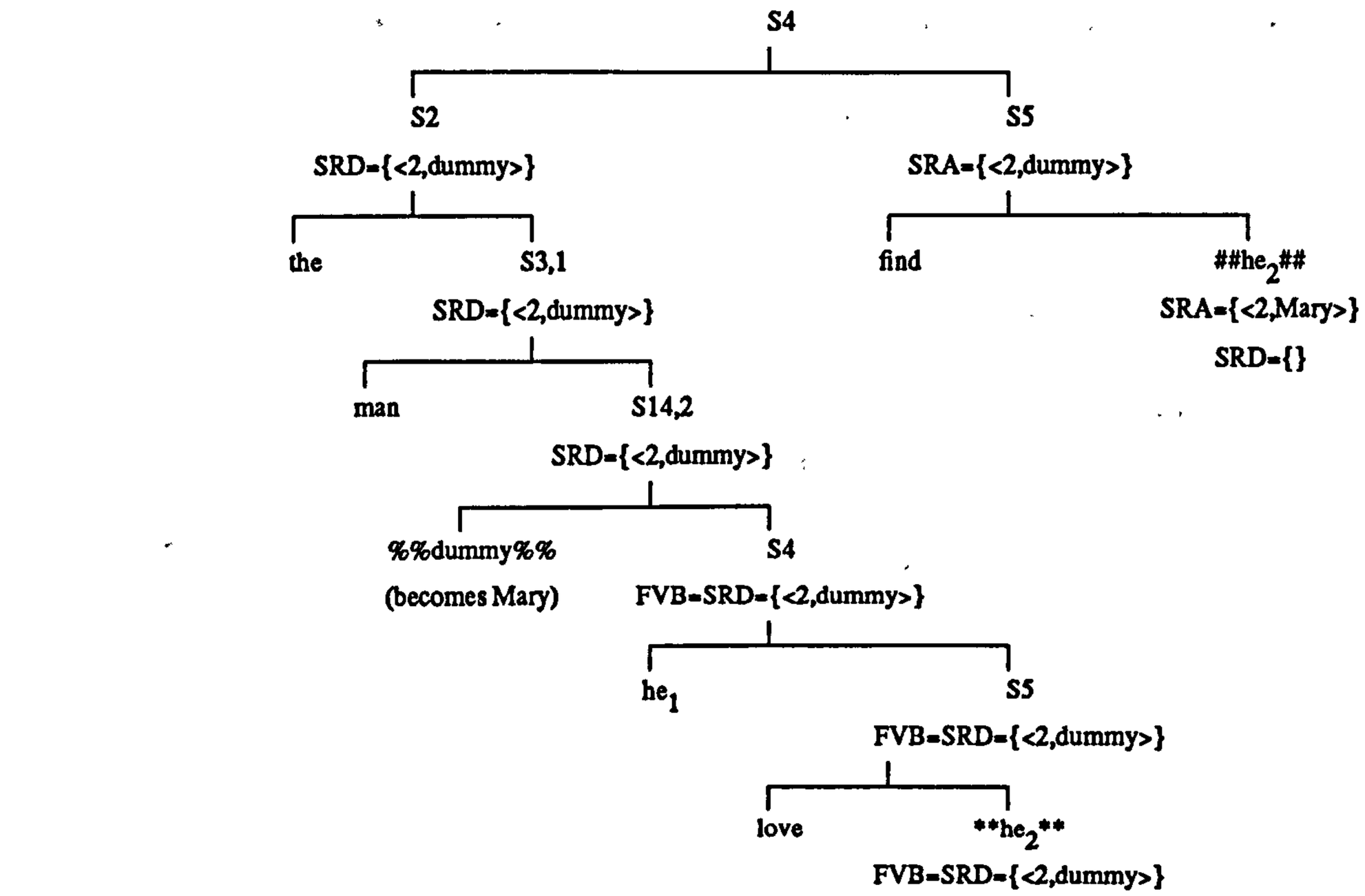


Fig 82

It might be felt that the problem could be avoided simply by delaying addition to FVB of the problematic binding until after the cataphoric reference has been finalised, but a moments reflection indicates that, were this to be countenanced, the binding <2,Mary> would be added to the FVB at the node flagged ##...##, whereupon it might be extracted by the immediate parent and adopted as an elder sister of the S5 node by the verb phrase quantification rule S16. Such a ploy would plainly result in variable he_2 remaining free in the elder daughter of the topmost node without any warning signal outstanding in its FVB, a situation avoidable if the dummy enters FVB immediately upon construction.

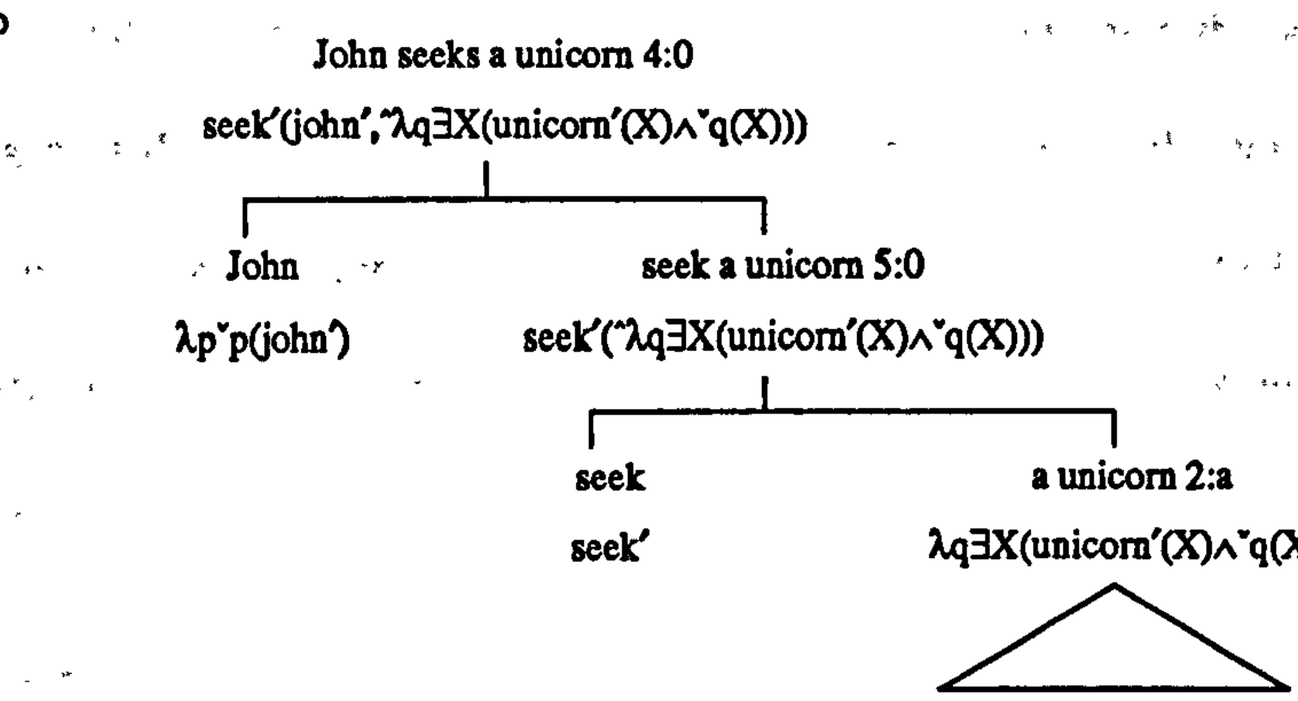
8.2.3. The Zero Option

Procedures which parse noun phrases other than pronouns may exercise an option to introduce a syntactic variable or alternatively retain the status quo and return a default node: let us call the latter choice the zero option. Whether or not the zero option is chosen will have important semantic consequences provided that the noun phrase in question occupies a position wherein accusative case marking would be appropriate. Reference to fig 8 (repeated below for convenience), confirms that a *de dicto* reading for "John seeks a

unicorn" is obtainable only in case the zero option is taken with regard to the object noun phrase.

****

(a) De dicto



(b) De re

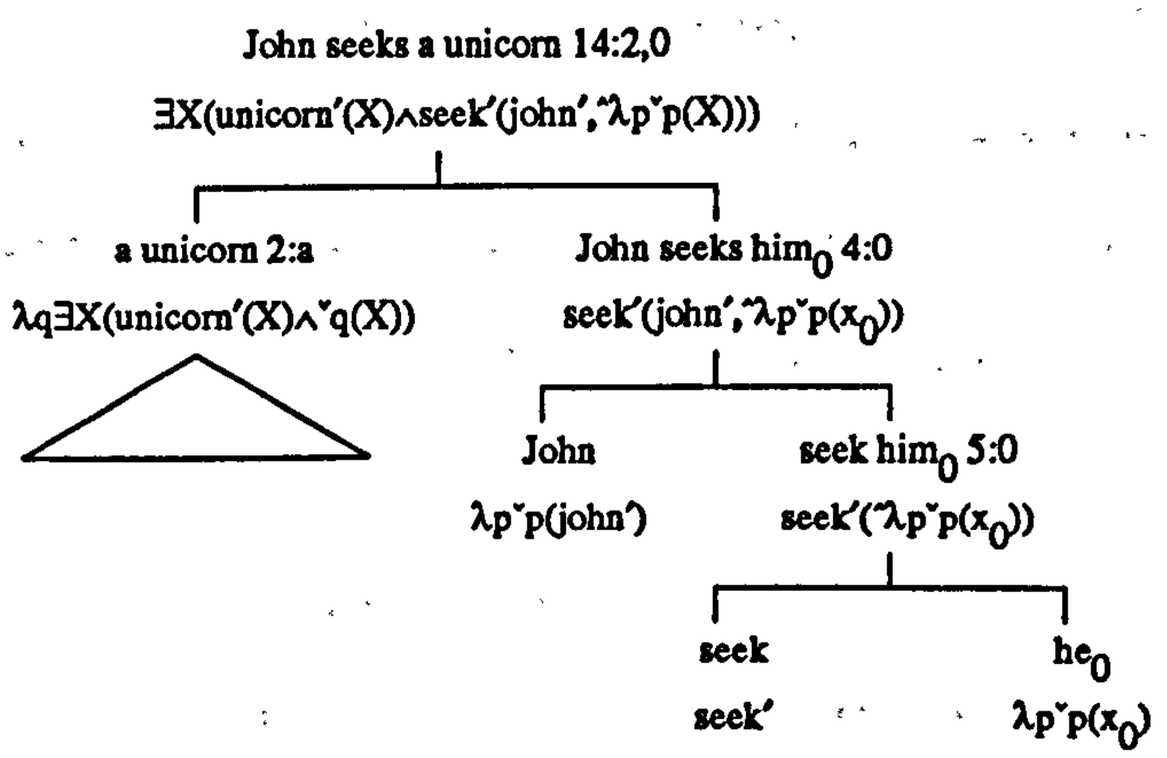


Fig 8 (repeated)

****

Where the noun phrase parsed is in subject position however no semantic significance[†105] appears to depend on the availability of the zero option, and accordingly the question arises as to whether or not it might be suppressed altogether for subject positions. If the objective of the parsing exercise is, wherever feasible,[†106] faithfully to reproduce all Montagovian variations in analysis tree irrespective of the semantic vacuity of the variation then the zero option must clearly remain uninhibited. If however we were prepared to tolerate the abandonment of some semantically ineffective alternatives then some ergonomic economy would result from insisting that all *nominative* noun phrases be introduced by quantification.

---

†105. The case is somewhat different for noun phrases with nominative case but in a predicative position, eg. as complements of the copula. See footnote 82 in connection with the effects of quantifying in such complements.

†106. An *advantage* of FWA is that it does *not* generate all *possible* variations in Montagovian analyses for a target sentence. Mere "alphabetical variants", ie. parse trees which differ only in the uniform replacement of one variable index by another *unused* one, are represented by a canonical tree employing the first available variable at all points. Without this restriction the number of semantically uninteresting variations would be infinite.

Since a Friedman Warren parser works top down, it is known at the time of encounter whether or not a given noun phrase is nominative, thus the situation in which suppression of the zero option might be advantageous is readily identifiable. As regards the corresponding Montague grammar, this would require an amended rule S4 which accepted only syntactic variables as first argument. We shall shortly be considering a strategy whereby Friedman and Warren propose to *abort* a parse upon discovery that it is semantically equivalent to an earlier one. By suppressing the zero option certain semantically equivalent parses might not even be commenced.

8.2.4. Accommodating Interrogatives

The affinities between declarative and interrogative noun phrases, as introduced by the binary revision of S2 and the innovative S29 have already been noted (§4.3.4); likewise the quantificational nature of the basic search question rule S24 (§4.3.5) and its multiple corollary S26 (§4.3.6) is evident. It should accordingly cause little surprise to discover that the Friedman Warren algorithm can be extended with minimal modification to cover Karttunen style interrogatives.

On Karttunen's analysis of search questions, the preposed interrogative is to have minimal scope: hence we require that a phrase structure analogue of the basic search question rule, ie. the preposing rule S24, have the general form:

S24′: SearchQ → PREWH S/NP

where PREWH is an interrogative noun phrase and S/NP is s sentence with a noun phrase missing (ie. marked by a "trace"). The parsing procedure corresponding to such a rule should be capable of returning a tree of the form indicated in fig 83.

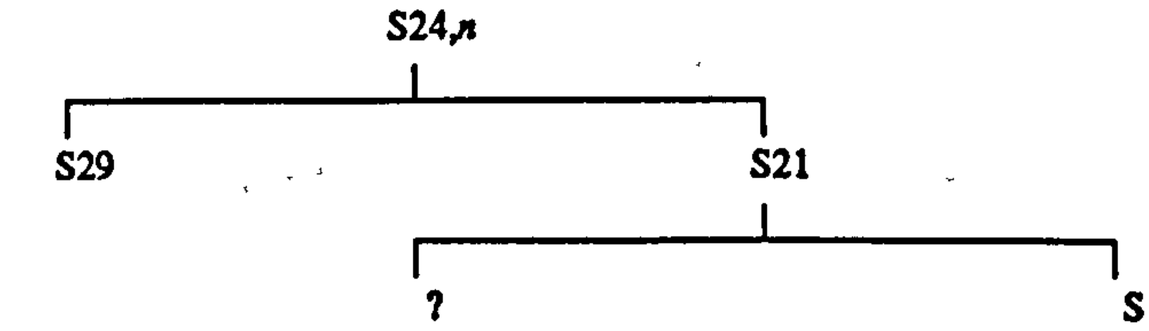


Fig 83

The node S on this tree must represent the sentence S/NP but with the missing NP now specified as

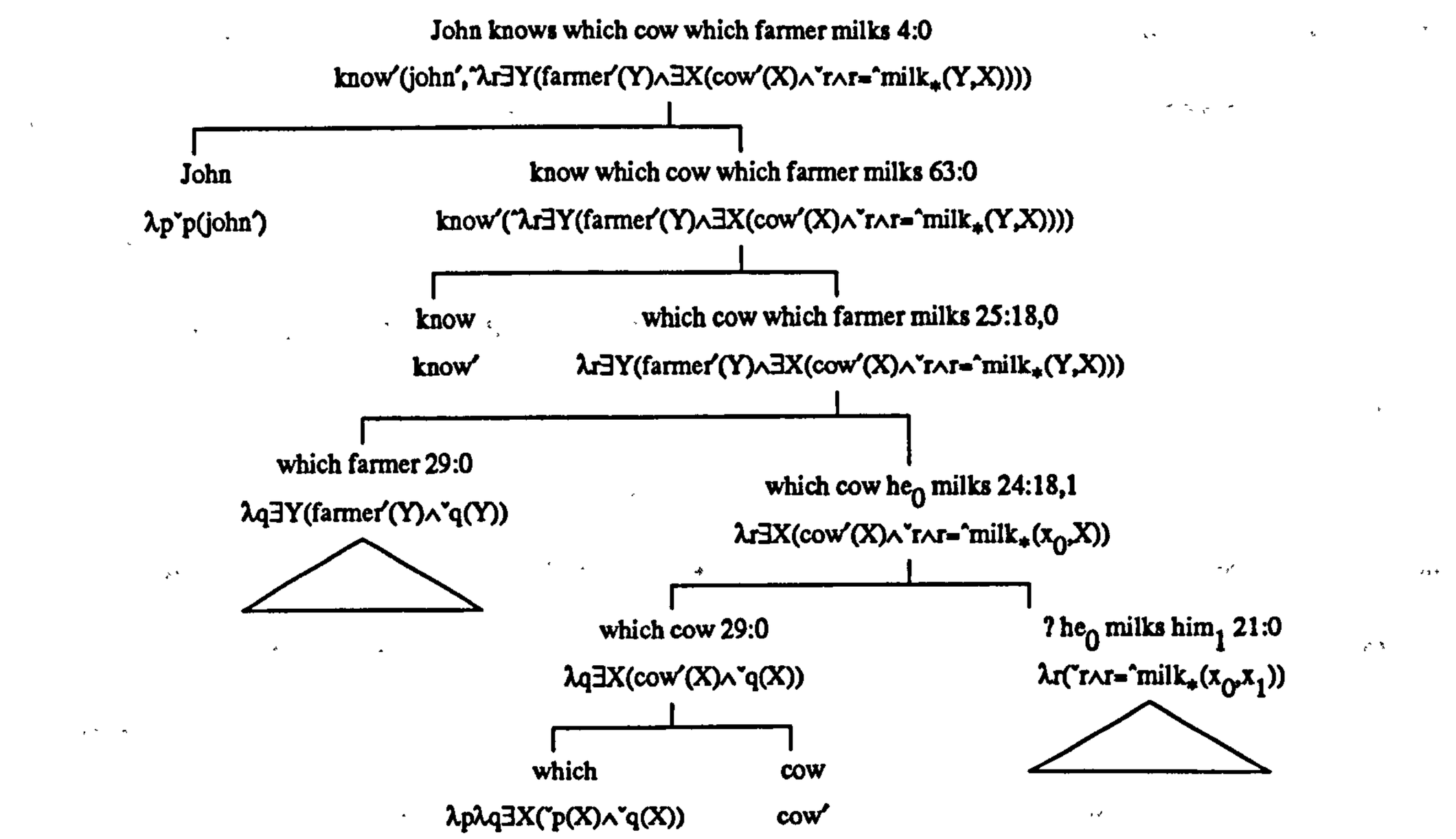


Fig 41 (repeated)

he_n and any other interrogative noun phrases replaced by further syntactic variables, precisely as illustrated in fig 41 (repeated here for convenience). In order to achieve this end an S24 parsing procedure, having parsed the preposed PREWH, must generate a binding variable he_n and pass this in a HOLD list to the procedure which deals with the embedded sentence. The latter procedure must be permitted to access the HOLD list and to extract an item when needed to plug any noun phrase gap.

Any additional interrogative noun phrases in S/NP are according to Karttunen's theory to be given wider scope than PREWH: how may this be guaranteed? We simply require that interrogative noun phrases be treated like any other noun phrases by the sentence parsing procedure and its subordinates, with the proviso that when an *interrogative* noun phrase WHNP is found by a noun phrase parsing procedure suppression of the zero option becomes *mandatory*. Accordingly a replacement variable *must* be introduced into the tree and a binding <Var,WHNP> recorded in FVB; moreover we now require that each binding in FVB be marked with a *mood* feature.

Sentence and verb phrase parsing procedures, when exercising their option to quantify, must be made to restrict their selections to bindings from FVB having nodes marked with the feature "DCL"

(declarative), while the search question parsing procedure, which must also be given the option to quantify, may be allowed to introduce either interrogative or declarative quantifiers as licensed by S25 and S26 respectively. All these provisions are realised in the version of FWA implemented in TMDCG.

8.3. Equivalence Parsing

In a later paper, [W2], Friedman and Warren illustrate their algorithm in the context of a "general execution method for non deterministic programs" which they dub "equivalence parsing". When first introduced, equivalence parsing is suggested as an economical method for accomplishing stage one of what Friedman and Warren describe as the "directed process approach" to the implementation of a Montague grammar. The directed process approach entails the sequential calling of program modules which simulate Montague's strategy :

(i) Generate analysis trees by application of the syntax rules of a Montague grammar.

(ii) Post order each analysis tree applying the appropriate translation rule at each node so as to derive an IL expression for the root.

(iii) Interpret the IL expression.

All syntactic processing is (temporally) prior to semantic processing when this approach is adopted.

8.3.1. The Directed Process with Recall Table

As a technique for investigating the internal workings of a Montague grammar the directed process approach cannot be bettered since all feasible variations are explored. Considered however as a "black box" for mapping surface sentences to semantic representations, this approach is far from optimal because in Montague grammar syntactic and semantic ambiguity are not directly correlated; hence there may be considerably more parse trees than there are non equivalent translations, and under the directed process approach all the *syntactic* variations will be investigated.

Within the framework of the directed process approach, the goal of equivalence parsing will be to produce *all* analysis trees (or alternatively skeletal D-trees) with the minimum of duplicated activity despite essential backtracking. Equivalence parsing is based upon the use of a *recall table* both to eliminate redundant processing and to enable the recursive descent parsing of left recursive constructions such as the

conjoined sentences introduced by S11. The recall table is a generalisation of the *well formed substring table* (WFST) employed by Woods, [W7], in his augmented transition network (ATN) grammar. Given such ancestry, it is unsurprising that Friedman and Warren formulate their own exposition in ATN terminology; but equivalence parsing is independent of the ATN methodology and may be equally well described in terms which lead naturally to a DCG implementation.

Each parsing procedure is to be furnished with three parameters capable of holding:

(a) The condition upon entry.

(b) The condition upon exit.

(c) The parse tree returned.

The entry condition includes the current input string awaiting parsing together with the SA list[†107] while the exit condition comprises the unconsumed residue of the string plus the FVB list. Entries in the recall table record successful procedure calls together with their parameter values.

Adopting the convention that lower case letters represent actual parameter values whilst upper case designates formal parameters, we may outline the equivalence parsing mechanism as follows. Initially a call is made in the form:

category(entry(input-string,sa-list),exit(RESIDUE,FVB),TREE).

from which return will be immediate if the recall table includes an entry indicating that the appropriate category, *given the specified entry condition*, has previously been found; for then the outstanding formal parameters may be matched with actual values from the table. If no such table entry exists then original parsing must be instituted, but either way successful exit from the call will be evidenced by the completion:

category(entry(input-string,sa-list)exit(residue,fvb),tree).

Finally, provided that this successful call is not already recorded[†108] an entry of the success must be made in the recall table.

---

†107. Friedman and Warren now refer to the SA list as the *PUSH environment*, and the FVB list as the *POP environment*. The entry condition in my description corresponds to their *PUSH identifier* while the exit condition is their *bucket*.

†108. When backtracking to find *alternative* solutions it is sometimes possible to engineer a duplicate, particularly with *semantic* equivalence parsing.

```
/* category(entry(Input,SA),exit(Residue,FVB),Parse-tree) */

sentence(entry(I,SA),E,S) :- table(sentence(entry(I,SA),E,S)).

/* S4 */
sentence(entry(I,SA),exit(Z,FVB),S) :-
        nounphrase(entry(I,SA),exit(I1,FVB1),NP),
        join(SA,FVB1,SA1),
        verbphrase(entry(I1,SA1),exit(Z,FVB2),VP),
        join(FVB1,FVB2,FVB3),
        quantify(s4(NP,VP),S,FVB3,FVB),
        add(table(sentence(entry(I,SA),exit(Z,FVB),S))).

/* S11 */
sentence(entry(I,SA),exit(Z,FVB),S) :-
        table(sentence(entry(I,SA),exit(I1,FVB1),S1)),
        join(SA,FVB1,SA1),scan(and,I1,I1a),
        sentence(entry(I1a,SA1),exit(Z,FVB2),S2),
        join(FVB1,FVB2,FVB3),
        quantify(s11(S1,S2),S,FVB3,FVB),
        add(table(sentence(entry(I,SA),exit(Z,FVB),S))).

nounphrase(entry(I,SA),E,S) :- table(nounphrase(entry(I,SA),E,S)).

/* Bnp */
nounphrase(entry(I,SA),exit(Z,FVB),NP) :-
        scan(N,I,Z),
        propername(N),
        replace(N,NP,FVB),
        add(table(nounphrase(entry(I,SA),exit(Z,FVB),NP))).

verbphrase(entry(I,SA),E,VP) :- table(verbphrase(entry(I,SA),E,VP)).

/* Biv */
verbphrase(entry(I,SA),exit(Z,FVB),VP) :-
        scan(V,I,Z),
        verbform(V,VP),
        add(table(verbphrase(entry(I,SA),exit(Z,FVB),VP))).
```

Fig 84

At the top level a parsing request must take the form:

sentence(entry(input-string,[]),exit(RESIDUE,FVB),TREE).

while a *successful* parse will be signalled by the return condition:

sentence(entry(input-string,[]),exit([],[]),tree).

A small ATN subnet corresponding to Montague's rules S4, S11 and S14, with only proper names,

intransitive verbs and conjunctions as basic lexical items, is used by Friedman and Warren to demonstrate equivalence parsing. The essential features of a DCG equivalent for this ATN are illustrated in fig 84. This DCG incorporates an integrated recall table mechanism similar to that employed in my earlier implementation of **PTQ**, [B4]. Of the procedures *not* defined in detail, "scan" is merely a lexical scanner which strips the head from the list provided as second argument leaving the third as residue, while "add" asserts its argument into the data base provided that it is not already there. The procedures "quantify" and "replace" are designed to implement FWA, the appropriate templates being:

```
/* quantify(Default-Node,FVB-Received,Returned-Node,FVB-Returned) */

/* replace(Default-Node,Replacement-Variable,FVB-Returned) */.
```

Since only intransitive verb phrases are admitted in this grammar, the question of verb phrase quantification does not arise.

Like Friedman and Warren's ATN, the DCG parses left to right, depth first, with backtracking to the most recent choice point when errors are encountered, or when a request for alternatives is explicitly made. The procedure for S4 includes two choice points:

(c1) The embedded call to noun phrase must exercise an option to *replace* the default NP node.

(c2) The final call made is to *quantify* which may or may not retain the default node unmodified.

Obviously c2 cannot be exercised unless c1 has introduced a variable, thus there remain three effective choices when parsing a simple sentence of form NP VERB. If the zero option is taken then a tree of form s4(NP,VERB) must be returned, but if a variable has been introduced the choice at c2 will determine whether or not the final tree is s4(he_n,VERB) or s14:*n*(NP,s4(he_n,VERB)).

The S11 procedure contains $n*m+1$ choice points where n is the number of alternatives pertaining to the first conjunct, m the number associated with the second and the additional option involves the overt call to "quantify".

Inspection of the entries made in the recall table of fig 85 during a parse of Friedman and Warren's example sentence:

(159) Bill walks and Mary runs.

****

1	nounphrase(entry([b,w,m,r],[]),exit([w,m,r],[]),b)
2	verbphrase(entry([w,m,r],[]),exit([m,r],[]),w)
3	sentence(entry([b,w,m,r],[]),exit([m,r],[]),s4(b,w))
4	nounphrase(entry([b,w,m,r],[]),exit([w,m,r],[<0,b>]),he_0)
5	verbphrase(entry([w,m,r],[<0,b>]),exit([m,r],[]),w)
6	sentence(entry([b,w,m,r],[]),exit([m,r],[<0,b>]),s4(he_0,w))
7	sentence(entry([b,w,m,r],[]),exit([m,r],[]),s14:0(b,s4(he_0,w)))
8	nounphrase(entry([m,r],[]),exit([r],[]),m)
9	verbphrase(entry([r],[]),exit([],[]),r)
10	sentence(entry([m,r],[]),exit([],[]),s4(m,r))
11	sentence(entry([b,w,m,r],[]),exit([],[]),s11(s4(b,w),s4(m,r)))
12	nounphrase(entry([m,r],[]),exit([r],[<1,m>]),he_1)
13	verbphrase(entry([r],[<1,m>]),exit([],[]),r)
14	sentence(entry([m,r],[]),exit([],[<1,m>]),s4(he_1,r))
15	sentence(entry([b,w,m,r],[]),exit([],[<1,m>]),s11(s4(b,w),s4(he_1,r)))
16	sentence(entry([b,w,m,r],[]),exit([],[]),s14:1(m,s11(s4(b,w),s4(he_1,r))))
17	sentence(entry([m,r],[]),exit([],[]),s14:1(m,s4(he_1,r)))
18	sentence(entry([b,w,m,r],[]),exit([],[]),s11(s4(b,w),s14:1(m,s4(he_1,r))))
19	sentence(entry([b,w,m,r],[]),exit([],[<0,b>]),s11(s4(he_0,w),s4(m,r)))
20	sentence(entry([b,w,m,r],[]),exit([],[]),s14:0(b,s11(s4(he_0,w),s4(m,r))))
21	sentence(entry([b,w,m,r],[]),exit([],[<0,b>,<1,m>]),s11(s4(he_0,w),s4(he_1,r)))
24	sentence(entry([b,w,m,r],[]),exit([],[]),s14:1(m,s14:0(b,s11(s4(he_0,w),s4(he_1,r)))))
25	sentence(entry([b,w,m,r],[]),exit([],[]),s14:0(b,s14:1(m,s11(s4(he_0,w),s4(he_1,r)))))
26	sentence(entry([b,w,m,r],[])exit([],[<0,b>]),s11(s4(he_0,w),s14:1(m,s4(he_1,r))))
27	sentence(entry([b,w,m,r],[]),exit([],[]),s14:0(b,s11(s4(he_0,w),s14:1(m,s4(he_1,r)))))
28	sentence(entry([b,w,m,r],[]),exit([],[]),s11(s14:0(b,s4(he_0,w)),s4(m,r)))
29	sentence(entry([b,w,m,r],[]),exit([],[<1,m>]),s11(s14:0(b,s4(he_0,w)),s4(he_1,r)))
30	sentence(entry([b,w,m,r],[]),exit([],[]),s14:1(m,s11(s14:0(b,s4(he_0,w)),s4(he_1,r))))
31	sentence(entry([b,w,m,r],[]),exit([],[]),s11(s14:0(b,s4(he_0,w)),s14:1(m,s4(he_1,r))))

Fig 85

****

(which following their precedent I abbreviate b,w,m,r) indicates the economy of effort resulting from equivalence parsing. Lines 1...7 represent the parsers attempts to consume the entire string using S4. The first conjunct is identified as a sentence at line 3, but since the residue is non empty the parser backs up to

the nearest executable choice point in the S4 rule. Advantage cannot be taken of the quantification option because FVB is empty so the replacement option in the noun phrase sub goal is exercised and in due course a second parse for the first conjunct obtained at line 6. This too fails to consume the whole string, but it does leave a binding in FVB so the quantification option may now be taken as a remedy is sought. As a result yet a third version of the first conjunct is recorded at line 7, but like its predecessors it leaves a residue outstanding.

No more alternatives are available to the S4 rule so control must pass to S11: the S4 procedure has however left three possible parses for the first conjunct in the recall table. Were the S11 rule to make a direct call to a sentence procedure to identify its first conjunct it would in due course invoke S11 itself thus generating an infinite regress. The availability of candidate first conjuncts in the recall table obviates this problem typically associated with left recursion. Extracting its first conjunct from line 3, the S11 procedure parses the second conjunct using S4 and obtains a result on line 10, thus permitting a first successful parse of the entire sentence on line 11.

Asked for alternative solutions, the parser is again unable to activate the quantification option in S11 in the absence of a suitable binding, so a second attempt at the second conjunct is made by the S4 rule. This time the replacement option is taken by the noun phrase sub goal and a parse of the conjunct completed on line 14. By combining lines 3 and 14 and then quantifying a second parse of the whole sentence emerges on line 16; but since the FVB list at line 14 is non empty a third alternative for the second conjunct is also possible by quantifying within the S4 rule so as to derive line 17, whereupon lines 3 and 17 may be combined to produce a third parse for the complete sentence on line 18.

At this juncture all original parsing has been accomplished: three parses are available for the first conjunct (3, 6, 7), and three for the second (10, 14, 17), while by combining 3 in turn with 10, 14 and 17 we obtain the first three full parses at lines 11, 16 and 18; but more are forthcoming. After backing right up to its first sub goal the S11 procedure may attempt more parses by extracting its first conjunct from line 6. The combination of 6+10 followed by quantification gives a fourth parse at line 20, the conjunction of 6+14 followed by two quantifications in either order allows fifth and sixth parses at lines 24 and 25[†109]

†109. I ignore the unsuccessful attempts involving a single quantification only.

while quantifying the result of joining 6 and 17 gives a seventh at line 27.

When line 7 is extracted as first conjunct three further parses are obtained. An eighth results immediately from the conjunction of 7+10 at line 28, quantification of the result of combining 7+14 generates a ninth parse on line 30, and finally another immediate and tenth parse is derived by joining lines 7 and 17.

8.3.2. Semantic Equivalence Parsing

Although much effort has been saved by the utilisation of previous results retrieved from the recall table, the directed process orientation of the program still results in the return of ten parse trees for the example sentence. Unless we are particularly interested in observing the structure of all these trees (and we may well be if our concern is to understand the ramifications of Montague's method), this duplication may cause some concern since in *all* cases the trees translate to a common IL expression:

walk′(bill′)∧run′(mary′).

Friedman and Warren suggest that this duplication may be prevented if instead of calling the translation module only upon completion of all syntactic processing, the module be made available as a subroutine to the parser to be called on a node by node basis.[†110] The net effect of such a manoeuvre would be to replace the returned parse tree with an expression of IL as the final parameter in every recall table entry.

For purposes of comparison the recall table for a semantic equivalence parse of example (159) is illustrated in fig 86. Two parses of the first conjunct are recorded on lines c and f, corresponding to lines 3 and 6 of the previous table; but there is no entry corresponding to line 7 since:

$s14{:}0(b,s4(he_0,w)) \rightsquigarrow walk'(bill')$

so that when in line 7 the parse tree is replaced by its reduced translation the result is equivalent to line c of the new table.

By parity of reasoning we may predict that lines i and m of the new table will correspond to lines 10

---

†110. Such a mode of operation would parallel the calling of lexical scanner and code generator as subroutines of the parser during the single pass compilation of a PASCAL program. Semantic equivalence parsing implies a translation module performing at least *reduced* translation (in Friedman's sense). The economy in parsing effort now suggests that such reduction, given its recoverability, might be justified in a machine translation environment provided that canonical forms in the target language were acceptable.

****

a	nounphrase(entry([b,w,m,r],[]),exit([w,m,r],[]),λp˘p(bill′))
b	verbphrase(entry([w,m,r],[]),exit([m,r],[]),walk′)
c	sentence(entry([b,w,m,r],[]),exit([m,r],[]),walk′(bill′))
d	nounphrase(entry([b,w,m,r],[]),exit([w,m,r],[<0,b>]),λp˘p(x_0))
e	verbphrase(entry([w,m,r],[<0,b>]),exit([m,r],[]),walk′)
f	sentence(entry([b,w,m,r],[]),exit([m,r],[<0,b>]),walk′(x_0))
g	nounphrase(entry([m,r],[]),exit([r],[]),λp˘p(mary′))
h	verbphrase(entry([r],[]),exit([],[]),run′)
i	sentence(entry([m,r],[]),exit([],[]),run′(mary′))
j	sentence(entry([b,w,m,r],[]),exit([],[]),walk′(bill′)∧run′(mary′))
k	nounphrase(entry([m,r],[]),exit([r],[<1,m>]),λp˘p(x_1))
l	verbphrase(entry([r],[<1,m>]),exit([],[]),run′)
m	sentence(entry([m,r],[]),exit([],[<1,m>]),run′(x_1))
n	sentence(entry([b,w,m,r],[]),exit([],[<1,m>]),walk′(bill′)∧run′(x_1))

Fig 86

****

and 14 of the old, representing two attempts at the second conjunct, and once again there will be no entry corresponding to line 17, the third alternative previously discovered. Consequently only two alternatives are available for each conjunct in the recall table, permitting four possible combinations. The conjunction of c with i gives the result on line j which is now the sole acceptable parse of the entire sentence. In seeking further representations the parser will indeed *attempt* to combine c+m, f+i and f+m, but in each case the attempt will be aborted once it is confirmed that the result is equivalent to that on line j: for duplicate entries are never recorded in the recall table.

The alternative parses of the individual conjuncts, which derive from the availability of *optional* replacement in the procedure which parses the subject noun phrases, turn out to serve no useful purpose: the contention that in subject position the zero option has no semantic significance is accordingly confirmed.

CHAPTER 9. INVERTED MONTAGUE GRAMMAR

¶ Prior to a description of the central characteristics of the mechanics of **TMDCG** and **LILT** together with their respective subordinate modules, this chapter commences with a brief exposition of the definite clause grammar technique which combines a phrase structure grammar with a recursive descent recognition algorithm. Investigation of the program modules takes the form of an examination of the templates of all the principal procedures called during syntactic and semantic processing, with particular attention given to the treatments of extraposition, reflexion and left recursion.

9.1. Fundamentals of Definite Clause Grammar

TMDCG is a definite clause grammar (DCG) simulation of extended Montague syntax originally developed in DEC-10 PROLOG but now running under CPROLOG on the University of York VAX-780 and GEC-63 computers. Familiarity with the PROLOG programming language, [B12, C3], and the basic philosophy underlying *logic programming*, [K8], must be presupposed; but a brief exposition of the basic definite clause grammar technique might not be untoward.

9.1.1. Phrase Structure and Recursive Descent Reconstruction

An alphabet Σ may be defined as a non empty finite set of symbols, and a *string* over Σ as any finite sequence of members from Σ with repetitions allowed: the set of strings of length n is denoted by Σ^n. The closure Σ^* of Σ is the set of *all* finite length strings over Σ, and the *positive* closure Σ^+ of Σ the set of all *non empty* finite length strings over Σ. Formally:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i \text{ and } \Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

where ε is the empty string.

A *phrase structure grammar* (PSG) may be defined as a quintuple:

$$G = \langle V^t, V^{nt}, V^{Pt}, S, P \rangle$$

where V^t is an alphabet of terminal symbols (ie. a lexicon), V^{nt} is a set of non terminal category symbols with $V^t \cap V^{nt} = \varnothing$, V^{Pt} is a set of preterminal symbols with $V^{Pt} \subseteq V^{nt}$, S is a distinguished "start" symbol

with $S \in V^{nt}$, and P is a set of *productions* or rewrite rules. When expressed in "Chomsky normal form", rewrite rules take the form:

$$\alpha \Phi \beta \rightarrow \alpha \Psi \beta$$

where $\alpha, \beta \in (V^t \cup V^{nt})^*$ provide the context of application and either:

[A] $\Phi \in V^{nt} - V^{pt}$ and $\Psi \in V^{nt*}$

or

[B] $\Phi \in V^{pt}$ and $\Psi \in V^{t*}$.

If χ may be derived from η by the application of zero or more production rules we write $\eta \Rightarrow * \chi$. Accordingly the language (or set of sentences) L(G) generated by G may be defined as:

$$L(G) = \{t \in V^{t*} \mid S \Rightarrow * t\}.$$

Finally if both α and β are empty in *every* rewrite rule, then the phrase structure grammar is *context free*, otherwise it is *context sensitive*.

Whereas a phrase structure grammar (PSG) is a device for *generating* sentences, the purpose of a parser or recognition algorithm must be to *reconstruct*, for any valid input sentence, the steps by which it has been legitimately generated. Recursive descent parsing implements a top down, depth first, left to right recognition algorithm according to which each non terminal symbol in a phrase structure grammar[†111] corresponds to a procedure responsible for identifying strings of the appropriate category. If a production rule rewrites a non terminal symbol in terms of others (case [A]), then the procedure corresponding to Φ must call sub procedures corresponding to the non terminals in Ψ. Each parsing procedure must be provided at call time with a pointer to the head of the unconsumed remainder of the input string, and must ensure that the pointer is suitably advanced upon successful exit: in this way the matrix sentence procedure will, if successful, consume the entire string.

Incrementing the pointer is ultimately the responsibility of procedures corresponding to *preterminal* categories (case [B]); for they alone are charged with direct inspection of the terminal sequence under

---

†111. No caveat need be entered if the grammar is context free. Context sensitivity is customarily handled by constraining procedures to maintain and inspect "symbol tables". In the case of a PASCAL like programming language, for example, concordance between function call and function declaration is so monitored.

investigation. A preterminal procedure must both *test* the scanned symbol for category membership and *advance* the pointer if the test succeeds.

9.1.2. DCG - A Grammar/Algorithm Hybrid

A definite clause grammar combines a context free phrase structure grammar with a recursive descent recognition algorithm implemented in PROLOG, and accordingly must make the requirements for input consumption explicit. The minimal augmentation implied by such a combination involves:

(i) Provision of input and output buffers as arguments on the category symbols.

(ii) Explicit calls to a lexical scanner in preterminal rewrite rules.

(iii) Incorporation of category testing in preterminal rewrite rules.

Context free rules of type [A] will then take the form:

phi(I0,I*n*) :- psi1(I0,I1), ..., psi*k*(I*k*-1,I*k*), ... psi*n*(I*n*-1,I*n*). for $0<k\leq n\geq 1$.

with rules of type [B] becoming:

phi(I0,I1) :- scan(W,I0,I1), psi_test(W).

which presupposes a lexical scanner of form:

scan(Word,[Word|Balance],Balance).

Ironically certain aspects of context sensitivity, such as the requirement that a singular finite verb phrase appear only in the environment of a singular grammatical subject, defy convenient representation in terms of a *context sensitive* PSG as defined above, but may be accommodated within an *augmented context free* PSG of DCG format by including additional argument places on appropriate goals and sub goals to record "features". Such arguments, when within the scope of a single PROLOG clause, will be constrained to match; however the polymodal nature of PROLOG logical variables makes it a matter of indifference whether the instantiated value of an earlier occurrence "trickles" to a later one, or whether a value not discovered until the later appearance "percolates" to the earlier occurrence. Additional argument places may also be employed to hold the *results* of analysis such as parse trees†112 or semantic representations.

---

†112. As indeed was illustrated in footnote 100.

Supplementary (non input consuming) goals may undertake not only category membership testing but also such functions as structure building: the predicates "quantify" and "replace" in the simulation of *equivalence parsing* shown in fig 84 are precisely such supplementary goals.†113

---

†113. A simple DCG which includes all the above features together with its output is listed hereunder:

```
:-op(500,xfy,:).
:-op(800,xfy,&).
:-op(900,xfy,=>).

sentence(Form,I0,In) :-
                nounphrase(NP,[Var,T,Num],I0,I1),
                verbintrans(VP,[Var,Num],I1,In),
                construct(NP,VP,[Var,T,Num],Form).

nounphrase(NP,[Var,T,Num],I0,In) :-
                determiner([T,Num],I0,I1),
                common(NP,[Var,Num],I1,In).

determiner([T,Num],I0,In) :-
                scan(D,I0,In),article(D,[T,Num]).

common(NP,[Var,Num],I0,In) :-
                scan(N,I0,In),noun(N,[Var,Num],NP).

verbintrans(VP,[Var,Num],I0,In) :-
                scan(V,I0,In),verb(V,[Var,Num],VP).

article(a,[indef,sg]).
article(the,[def,_]).
noun(man,[X,sg],man(X)).
noun(men,[X,pl],man(X)).
noun(sheep,[X,_],sheep(X)).
verb(walks,[X,sg],walk(X)).
verb(walk,[X,pl],walk(X)).

construct(NP,VP,[X,indef,sg],exists(X):(NP&VP)).
construct(NP,VP,[X,def,sg],unique(X):(NP&VP)).
construct(NP,VP,[X,def,pl],exists(w):(all(X):(w(X)=>(NP&VP)))).
scan(W,[W|In],In).

sentence(F,[a,man,walks],[]).
F = exists(_51):(man(_51)&walk(_51))
sentence(F,[the,man,walks],[]).
F = unique(_59):(man(_59)&walk(_59))
sentence(F,[the,sheep,walk],[]).
F = exists(w):all(_63):(w(_63)=>sheep(_63)&walk(_63))
sentence(exists(X):(man(X)&walk(X)),S,[]).
S = [a,man,walks]
sentence(unique(X):(man(X)&walk(X)),S,[]).
S = [the,man,walks]
sentence(exists(w):all(X):(w(X)=>sheep(X)&walk(X)),S,[]).
S = [the,sheep,walk]
```

The polymodality of variables may be illustrated by tracing the history of "Num" during the parsing of the three sentences shown; for the variable serves as a "trickling input" or "percolating output" parameter depending on circumstances. In the first example "Num" is instantiated by the "article" procedure whereupon it percolates to "determiner", trickles to "common" and "noun", percolates to "nounphrase" and trickles to "verbintrans", "verb" and "construct". There is no instantiation in the second case until the success of "noun" whence the value percolates to "common", "determiner" and "nounphrase" and then trickles to "verbphrase", "verb" and "construct". Finally in example three instantiation is deferred until "verb" succeeds, whereupon percolation is to "verbphrase" and "nounphrase" (thence permeating to families of earlier offspring) with trickling restricted to "construct".

Features trickle to preorder successors or percolate to preorder predecessors depending upon the point of instantiation. In the terminology of Knuth, [K9], the former are inherited and the latter synthesised attributes. Note that this grammar is fully polymodal in

The use of a notation directly implementable in PROLOG confers the advantage that any improvement in performance of the interpreter guarantees automatic improvement in performance of the parser: the grammar writer inherits the benefits of first order research in the field of logic programming. A more fundamental advantage accrues however when the target grammar is *categorial* in nature; for the behaviour of PROLOG variables makes it possible effortlessly to synthesise an analysis tree in quasi bottom up fashion during the course of a top down, depth first, left to right parse.

Most PROLOG implementations include a "grammar rule" preprocessor which expands into raw PROLOG clauses written in the "grammar rule notation" (GRN) as follows:

phi --> psi1, ..., psi*n*.	becomes	phi(I0,I*n*) :- psi1(I0,I1), ..., psi*n*(I*n*-1,I*n*).
phi --> [W], {psi_test(W)}.	becomes	phi(I0,I*n*) :- scan(W,I0,I*n*), psi_test(W).

All clauses written with the GRN operator "-->" are furnished with I/O buffer arguments on all goals not protected by inhibitory curly brackets which accordingly indicate supplementary goals. Calls to the scanner are signalled by square bracket notation.

Ostensibly GRN reduces noise[†114] and makes DCG notation (marginally) more like the original PSG notation: it is however of limited utility since it effectively prohibits explicit reference to I/O buffers, and the ability to make such references is often important. In the DCG representation of Warren and Friedman's equivalence parsing algorithm (vide fig 84), for instance, recall table maintenance using the "add" predicate requires explicit mention of an "I" field on entry and a "Z" field on exit: hence such entries cannot be engineered using GRN.

A cataloguing facility similar to recall table maintenance is introduced in TMDCG and accordingly GRN is eschewed, all lexical scanning being accomplished in raw PROLOG.

9.2. Syntactic Processing in TMDCG

In a *strictly* categorial grammar, if $\eta \in A/B$ and $\zeta \in B$ then $\eta\zeta \in A$, where $\eta\zeta$ is formed by simple concatenation. Formation of a context free phrase structural inverse for such a rule is a trivial exercise; for

---

that, given a "logical form", it generates the corresponding sentence.

†114. GRN is employed in footnote 100 for this reason.

the PSG rule is simply:

$$A \rightarrow A/B\ A$$

As we have seen (§2.2.2) Montague grammar is not *strictly* categorial in so far that structural operations are not limited to concatenation; but provided that the operation does no more than introduce either syncategorematic additions in determinate locations or morphological variations of existing elements inversion remains unproblematic. Rules of the former kind might be typified by examples such as:

$$\text{if } \eta \in A/B \text{ and } \zeta \in B \text{ then } \eta\chi\zeta \in A$$

which has the phrase structural inverse:†115

$$A \rightarrow A/B\ \chi\ B.$$

A rule introducing morphological variation has, in categorial terms, a form:

$$\text{if } \eta[F] \in A/B \text{ and } \zeta[F'] \in B \text{ then } \eta\zeta[F''] \in A$$

where F, F′, F″ are feature markings. The inverse must accordingly be a quasi phrase structural rule which we might schematise as:

$$A\ [F''] \rightarrow A/B\ [F]\ B\ [F'].$$

Such a rule is unproblematically formulable in PROLOG DCG format since the feature markings become argument places.

So far as inversion is concerned, the only problematic Montagovian rules are those with structural operations which introduce variable binding, but a solution to the problem is readily available in the form of the Friedman Warren algorithm. **TMDCG** implements aversion of the Friedman Warren algorithm under the direction of the immediately invertible generalised categorial rules of **TMG** thus achieving a definite clause grammatical inversion of the extended Montague grammar. During the course of parsing **TMDCG** employs three sub modules **EDIT, GENLEX** and **LEXTMG.** Output from the parser consists of *complete* Montagovian analysis trees represented as "vine diagrams", where a vine diagram results from rotation of the original tree ninety degrees anticlockwise, transposition across the north west - south east axis and minor distortion of branch lengths.

---

†115. Strictly speaking this is an elision of two rules viz $A \rightarrow A/B\ C\ B$ and $C \rightarrow \chi$.

Semantic processing is achieved by passing each analysis tree produced to a language of intensional logic translator **LILT** which traverses trees in "galilean"†116 post order and generates fully reduced formulae of **TIL** with the assistance of its slave **TBASE.** Full listings of all programs are available in appendices B ... G.

9.2.1. Basic Templates

The nodes of an analysis tree are represented in **TMDCG** by structures†117 having the form:

node(N, F, L, D)

where the arguments are to be interpreted as follows.

N is an identifier of form #*n*:*k* or, in the case of a node created by a variable binding rule, #*n*:(*k*:*i*) where *n* is the index of syntax rule S*n*, *k* is the index of the semantic operation g_k invoked by translation rule T*n* and *i* is the index of the syntactic variable subject to binding. Lexical items receive the identifier "#1:=" and pseudo lexical items the identifier "#0:=".

F is a list of those features regarded as *intrinsic* to the node, where an intrinsic feature is one independent of grammatical function in a particular configuration.

L is the expression associated with the node and held as a list of feature marked lexical items. This field is referred to as the *overt* label.

D is dependent on the nature of the node in question. In the case of a non terminal node, **D** is a binary list of daughters each of which herself a node structure; otherwise **D** is a structure of the form **sense(Item,F).** A daughter field of the latter form is accessed by the "translate" predicate in **LILT** which calls the **TBASE** predicate "sense" in order to introduce basic **TIL** translations.

Procedures which parse grammatical categories during the recursive descent usually have hendecatuple arguments and take the form:

category(N, F, E, L, Ia, Iz, H, FVB, SA, SRa, SRz)

the arguments of which are described seriatim.

---

†116. That is to say, the order is younger daughter's family, elder daughter's family, parent: in other words the "mirror image" of post order.

†117. Originally suggested by the "syn" structures of McCord, [M1].

N is a node structure as described above returned as a representation of the category member identified.

F is a list of features of the category - in - context and often constitutes a superset of the intrinsic features of the node.

E is a skeletal tree structure representing the environment in which the category is being sought and relative to which the node returned must be unique if the parse is not to be aborted.

L is the contribution transmitted upwards to the parent procedure for inclusion in the overt label field of the parent's node structure. This field is known as the *transmission* label.

H is a "hold stack" used for passing left extraposed noun phrases downwards to subordinate clauses.

Ia and **Iz** are the input and output buffers which hold the remainder of the candidate string both before and after parsing since lexical scanning is to be explicit.

FVB and **SA** are the free variables below and substitutions above lists as described in §8.2.1.

SRa and **SRz** are the substituend required lists on arrival and on departure as described in §8.2.2.

9.2.2. Feature Lists

In the interests of both uniformity and ease of extension all nodes are given intrinsic features even if they prove vestigial, and all features are held as lists even if they be singleton. The feature lists associated with parsing procedures are made available so that the procedures may monitor issues of concord.

Sentence nodes and sentence parsing procedures have a common singleton feature list which records [Mood]. At present only "dcl" is an allowed value since direct questions do not appear in **TMG**. Indirect question procedures carry the feature [q] which is echoed in the node structures of all but yes-no and proto questions which are flagged [?yn] and [proto] respectively.

Noun phrase parsing procedures include the quartet [Mood, Gender, Case, Number], where "Mood" is set to "dcl" in the case of declarative pronouns and proper names, "q" in the case of interrogative pronouns, and constrained by the nature of the determiner in all other cases. The mood feature is required for onward transmission to the "replace" routine of the Friedman Warren algorithm since, as argued in §8.2.4, the zero option *must* be suppressed in the case of interrogative noun phrases. When a noun phrase procedure is called in an accusative position, Case is set to "acc", otherwise it is "nom". Intrinsic features on

noun phrase nodes are limited to [Gender, Number], for in Montague grammar leaf positions are not typically case marked.

Both finite verb phrase nodes and the procedures which create them record a feature [fin(Num,Time)], but when the procedure calls a routine to identify members of P_{QIV} the feature list passed on takes the form [Kind, fin(Num,Tense), Comp]. The significance of the variable "Time" as opposed to "Tense" lies in its function in eliminating inappropriate combinations of finite verb phrase and temporal adverbial. As a subordinate clause

(160) John would run tomorrow.

is unexceptionable while

(161) * John had run tomorrow.

is absurd, although in both cases the *tense* of the leading auxiliary is "past": a constraint upon the occurrence of "tomorrow" cannot accordingly be formulated in terms of tense.

In **TMDCG** the "Time" feature is determined by reference to both the root of the verb and its tense. It has the possible values "ante", "nunc", "post", "janus" (in the case of conjunctions referring in opposite directions) or, as in the case of "would", no setting. Reference to **GENLEX** will confirm that "had run" gets the feature "ante" which is incompatible with the "post" feature of "tomorrow", but "would run" remains unmarked for time direction thus is eligible for combination with either "tomorrow", "today" or "yesterday".†118

Procedures which parse non finite verb phrases all record the feature list [Kind, Inflection, Complement], where "Inflection" denotes the constraint imposed by the predecessor and "Complement" denotes the constraint to be imposed on the successor once a value for "Kind" has been determined. Intrinsic features on non finite verb phrase nodes are limited to the categorial identifier.

---

†118. This form of selectional restriction takes us beyond the powers of TMG but has been introduced as an economy measure to abort worthless parses. When a program is designed both to simulate and to refine a target grammar one or other tends to be marginally in advance at any given point in their development. The systematic incorporation of selectional restrictions is a prospect for the future.

9.2.3. Environments

Analysis trees are constructed by TMDCG in post order, hence at any given stage of parsing either nothing has yet been determined (the environment is empty) or there exists an elder sister of anonymous parentage: furthermore an anonymous parent may herself have an elder sister of like anonymous parentage.

****

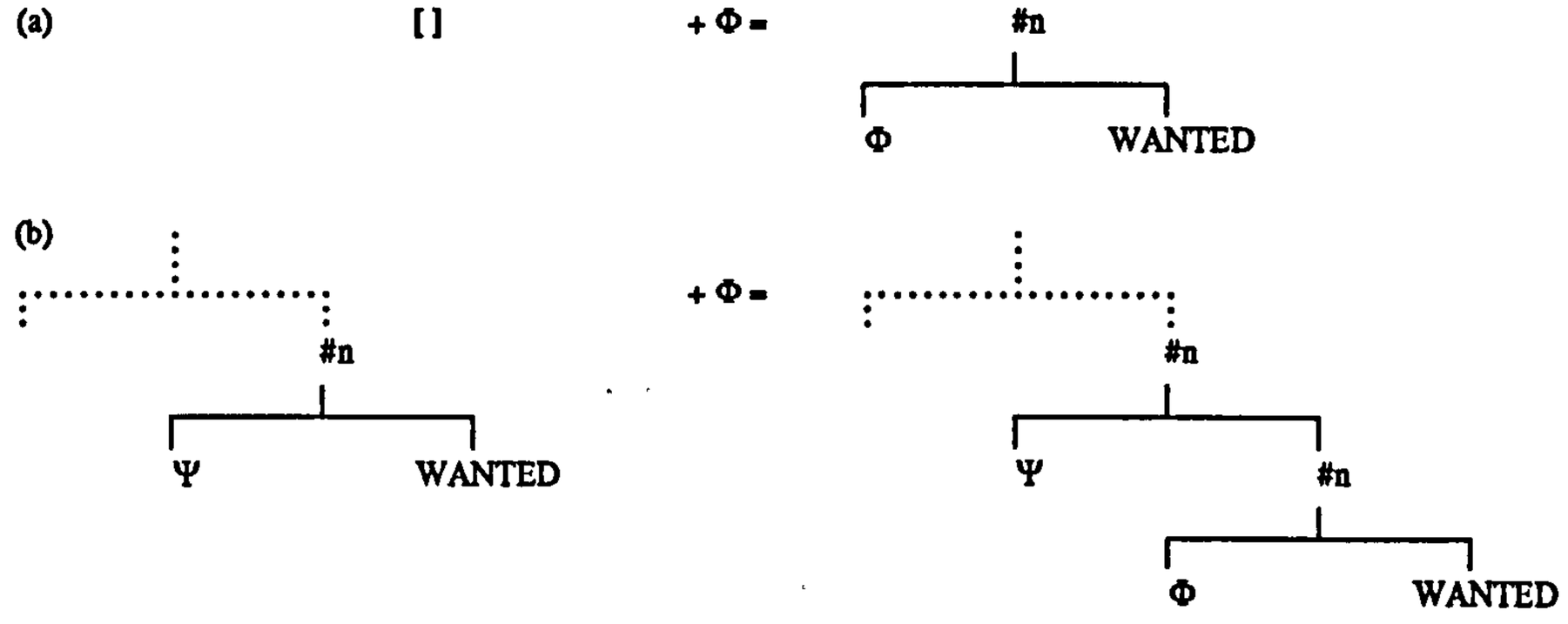


Fig 87

****

Environments may accordingly be represented either as empty sets or as skeletal trees in which each sub tree has a complete elder daughter, an unspecified parent and a skeletal younger daughter. The right-most leaf of a skeletal tree marks the point of attachment for the next node completed.

A matrix sentence procedure must be called in an empty environment, whereafter the environment of the parent will become the environment of the elder daughter, while the environment of the younger daughter must be constructed by combining the environment of the elder sister with the node created by that sister. Suppose the node created by the elder sister to be Φ, then if the elder sister's environment was empty the younger sister will inherit the nuclear environment illustrated in fig 87 (a), where "WANTED" marks the current point of attachment for the node to be constructed. If however the environment of the elder sister is already a skeletal tree with a "WANTED" leaf awaiting completion then the new environment must be constructed as in fig 87 (b).

When a parsing procedure in TMDCG successfully returns a node structure then, provided that the output buffer is not empty, an attempt must be made to catalogue the success in a well formed substring table. A catalogue entry must constitute a *unique* pairing <E,N> where E is the environment and N the

node returned. Should the pairing be non unique†119 then the procedure aborts since plainly a previously trodden path has been explored.

****

The woman who married John has beared a son.

Unsuccessful

Attempt :
****
Environment
Empty

Node
#2:0 the woman who married john
#1:= the
#31:3:1 woman who married john
#1:= woman
#14:2:3 she1 married john
#1:= john
#40:1 she1 married him3
#1:= he1
#42:0 married him3
#0:= past
#60:0 marry him3
#1:= marry
#1:= he3

Fig 88

****

The catalogue entries do not however serve merely to eliminate duplication. Left to its own devices, a DCG presented with invalid input will backtrack to attempt all possible paths, but will eventually terminate with the not altogether helpful response "no". Maintenance of a catalogue allows TMDCG to provide some elementary diagnostic information in so far that, on failure, catalogue entries may be displayed thus indicating those nodes successfully built and their environments.

The information displayed on correctly rejecting the sentence

(162) * The woman who married John has beared a son

includes the <environment, node> pair illustrated in fig 88, where the parser's best attempt is indicated by

---

†119. Should a single pass mode of operation, with LILT called on a node by node basis, be subsequently adopted then the abortion facility will prove crucial since otherwise many syntactic routes to the same semantic goal may be fruitlessly explored. Meanwhile in multi pass (ie. directed process) operation the facility is a benign but unnecessary precaution.

an empty environment together with a tree of depth greater than any other occurring in an empty environment. Procedures for implementing the environment and cataloguing mechanism are listed in appendix B under the heading "Well Formed Sub String Maintenance". Finally it may be observed that in TMDCG the well formed sub string table is *not* used to overcome the problems inherent in recursive descent parsing of left recursive constructions.

9.2.4. Simulating Structural Operations: The Label Fields

All structural operations in TMG have been formulated in terms of PROLOG predicates in an attempt to accord with the constraints of Partee, [P6]: it might therefore seem surprising that these predicates are not directly incorporated in TMDCG. The reason is simply that the structural operation predicates are "conceptual" in nature, that is to say they are designed to demonstrate that the mathematical requirements are *in principle* realisable rather than to provide a cost effective means to an end. Since legibility rather than optimal performance is the primary criterion of a conceptual predicate considerable redundancy in processing may be tolerable, thus although all the conceptual predicates execute in CPROLOG they are scarcely efficient.

Structural operations which concatenate one argument with a morphological variation in another are accommodated in TMDCG by recourse to the distinction between overt and transmission labels, the latter being where necessary a morphological variation of the former. Since the parser works top down, the nature of the variation is known at the time each node is created, thus the transmission label is readily predictable: the overt label of a parent node may then be constructed by concatenating the transmission labels of the daughters.

The structural operations involved in variable binding rules are simulated in TMDCG by recourse to the string editor module EDIT. Prior to the binding of any variable with index *k*, a call is made to the EDIT procedure "makevars" which has the template:

makevars(Nom, Nom1, Acc, Acc1, Rf, Suffix, Subj, Obj, Ref, [G,Num])

where Suffix is the index number and the final argument is matched by the gender and number of the noun phrase to be quantified in. The first five arguments of "makevars" are instantiated as syntactic variables of the appropriate index, gender and number being respectively the nominative, nominative cataphòric,

accusative, accusative cataphoric and reflexive, while Subj, Obj and Ref are the corresponding nominative, accusative and reflexive surface forms. Thus a total of eight constants are prefabricated.

When variable binding is accomplished by the option to quantify under the Friedman Warren algorithm, the candidate term is selected from an eligible binding in the FVB list and "makevars" duly called whereupon control is passed to the procedure "edit" having the template:

edit(Nom, Nom1, Acc, Acc1, Rf, Subj, Obj, Ref, L, Y, Y1)

where L is the candidate term, Y the string to be edited and Y1 the result of editing, other arguments being the same as in "makevars". The purpose of "edit" is to scan the input string looking for the *first* occurrence of an appropriate nominative or accusative syntactic variable.†120 If the first occurrence is *cataphoric* it is replaced by the corresponding surface pronoun and "edit" called recursively on the balance of the string, otherwise the variable is replaced by the term L and responsibility for handling the balance passed to "editline".

The template for "editline" is:

editline(Nom, Acc, Rf, Subj, Obj, Ref, Y, Y1)

Like "edit", "editline" scans the input string looking for suitable syntactic variables which it replaces with surface forms unless they are *parenthesised* in which case it simply eliminates them. Parenthesised variables are recorded whenever a noun phrase parsing procedure has recourse to the "hold stack" and in effect constitute "traces".

Reference to appendix B will reveal that on two occasions "editline" is called directly by parsing procedures, namely those procedures which parse search questions and relative clauses respectively. In both cases a term is to be prefixed to a result of editing which must involve the elimination of a trace.

Note that the clauses of both "edit" and "editline" are mutually exclusive and each can succeed at most once, hence the somewhat unusual double cut mechanism.

---

†120. The *first* occurrence cannot be reflexive in most idiolects.

9.2.5. Bindings on the Hold Stack

In **TMDCG** the bindings required for implementation of the Friedman Warren algorithm take the form:

bind(Variable, Index, [Mood, Case, Ref-Case, Level], Node).

Variables are paired with *complete* node structures rather than nodal phrases alone so that those dominance relationships prescribed or proscribed by (FWA1), (FWA2) and (FWA3) may the more easily be investigated, while the variable index is held separately in order to obviate unnecessary extraction from the variable itself. The significance of Ref-Case and Level will become apparent once the cross referencing mechanism for handling reflexion has been discussed. Four noun phrase features are available in each binding, for in addition to the mood and case shown specifically the gender and number may be determined from the feature field of the node itself.

When a preposed noun phrase is sought by the procedure which parses search questions the mood is constrained to be interrogative and accordingly the FVB list returned by the embedded noun phrase call will contain a binding. If the noun phrase is an interrogative pronoun then the binding is constructed directly by the procedure dealing with such pronouns, whereas if it has the form "WDET NP" then there is mandatory introduction of a binding by the Friedman Warren algorithm "replace" routine, the zero option being prohibited. The variable from the binding, paired with a list of salient features, is then added to the inherited hold stack of the search question procedure in the form:

hold([Gender, Case, Number], Variable).

and the new hold stack passed to an embedded "sentence" procedure, the binding itself appearing in that procedures SA list.

As a last resort, the procedure which seeks nuclear noun phrases may access a non empty hold stack in order to create a variable node; but in such cases it will return as its transmission label a case marked variation of the variable *in parentheses*, thus indicating to the program that the item is in the nature of a trace. Since the inherited hold stack is itself made available to the embedded sentence procedure, **TMDCG** successfully parses sentences such as:

(163) John wonders which cow which farmer knows which maid milks.

but should multi level extraposition be considered objectionable it may be removed by initialising the embedded sentence hold stack to the new item alone.

Such a singleton hold stack is indeed presently passed to the embedded sentence procedure which parses relative clauses, although in this case a binding for inclusion in the "sentence" procedures SA list must be constructed explicitly. At the top level "sentence" call the hold stack must plainly be empty.

9.3. Replacement and Quantification

A description of the extended Friedman Warren algorithm incorporated in **TMDCG** has already been provided since the program implements all the modifications introduced in chapter 8: moreover the nature of the bindings now included in the algorithm lists has been discussed in the previous sub section. It remains therefore but to consider the dual procedures "replace" and "quantify" which are charged with execution of the algorithm.

The "replace" procedure has the template:

replace(DN, RN, [M, G, C, Num], OTL, NTL, OFVB, NFVB, OSR, NSR).

where DN and RN are the default and replacement nodes, [M,G,C,Num] is the calling noun phrase procedure's feature list, OTL and NTL are old and new transmission labels, OFVB and NFVB are old and new FVB lists and OSR and NSR old and new SR lists.

The first clause for "replace" checks the substituend required list to ascertain whether or not there is a suitable incomplete cataphoric reference to an item with matching gender and number features. If there is then the default node is used to complete the dummy binding, the variable from the binding becomes the overt label of the replacement node, and the binding is removed from OSR to give NSR. A new transmission label is finally constructed via a call to the **EDIT** procedure "copyvar" which conforms to the pattern:

copyvar([G, C, Num, Type], Anaphoric, Cataphoric, Suffix).

where Suffix is the ASCII code for a variable index and Anaphoric and Cataphoric become case and gender marked syntactic variables of that index. If the Type field is set to "p" a non reflexive pronominal reference is involved while an "r" indicates reflexion.

Provided that the mood feature M is set to "dcl", a second clause for "replace" allows a zero option

to be exercised, otherwise control is passed to the third clause which handles standard replacement. In standard replacement a new syntactic variable is first generated and bound to the default node, the binding then being introduced at a *random* position in Old_FVB to give New_FVB. The standard replacement clause also establishes the current *level number* (to be explained in the next sub section) and records this as a feature in the binding unless the default node is an interrogative noun phrase in which case the level is incremented by 1. Preposed interrogative noun phrases thus acquire the level number of the earliest embedded sentence from which they might have been extraposed. An unset case feature infallibly identifies preposed interrogative noun phrases for which the generation of a transmission label by "copy-var" would be inappropriate, the search question procedure having no use for this field. Accordingly a dummy "trace" field is returned as transmission label whenever the case variable is unset.

Variable binding in TMDCG is handled by a predicate "quantify" having the template:

quantify(Type, DN, RN, OOL, NOL, OTL, NTL, OFVB, NFVB, Skip, SR).

The Type field accepts values from the set {cn, s, [lex,Infl,Comp], [piv], q}. If the type is "cn", "s", "[lex,_,_]" or "[piv]" then quantification using a binding from OFVB having the features [dcl,_,_,_] will be licensed by S14, S15, S16 and S18 respectively. A type field of "q" may trigger S25 quantification if the binding features are [q,_,_,_] or S26 quantification otherwise.

In cases of verb phrase quantification both old overt (OOL) and old transmission (OTL) label fields must be edited whenever a variable binding node is introduced by quantification, the results being returned in NOL and NTL respectively; but in all other cases only overt label fields need be maintained, the corresponding transmission labels merely echoing them, thus OTL and NTL may be set to [] to inhibit unnecessary editing.

Of the remaining argument places only "Skip" requires explanation. Three clauses[†121] govern the behaviour of the "quantify" predicate of which the first merely preserves the status quo whenever the incoming FVB list is empty. A single option to effect variable binding is exercised by the second clause which considers for eligibility and extraction only the *head* of the FVB list. Before introducing the node

---

†121. In the present implementation there is in fact a prefix clause which switches off quantification under S15 since in the majority of cases this is semantically vacuous.

from the head binding as an elder daughter however, the clause issues a recursive call to "quantify", passing down the *tail* of the FVB list for consideration and accepting the node returned as its own younger daughter.

Constraints (FWA2) and (FWA3) prohibit extraction from an FVB list of any binding the variable of which is dominated by the node of any other binding in the FVB list or by any binding in the SR list. Each recursive call to "quantify" invokes a test to confirm that variable he_n from the head binding bind(he_n,n,_,ϕ) is not so dominated by any node ψ in other bindings, but such a test need not and does not consider *inverse* domination: hence there may be some binding bind(he_m,m,_,ψ) in the tail of FVB such that he_m is dominated by ϕ. In such circumstances he_m must be bound higher on the analysis tree than he_n, thus node ψ must not be introduced by the an embedded call to "quantify" into the family of the younger daughter. Apparently each embedded call must be provided with details of all bindings removed by earlier calls in the recursion since these may inhibit extraction from the current FVB. The "Skip" field, which is set to [] at top level, provides such a list of earlier removals and this together with the current FVB and SR lists constitutes the data to be inspected by the eligibility testing mechanism.

An option *not* to introduce variable binding is exercised by the third clause of "quantify" which simply removes the head of the FVB list, prefixes it to the skip chain and issues a recursive call designed not to generate a younger daughter but to construct the return node for the calling procedure itself. Conformity with (FWA1) is monitored directly by the "recurse" procedure which handles relative clauses.

9.4. Multi Level Cross Referencing

Reference to appendix B will confirm that **TMDCG** contains only one "sentence" clause called not only at top level but also within the recursive procedure which handles conjoined sentences, in the procedure which prefixes sentential adverbs and in the procedures which parse search questions and relative clauses. Each time "sentence" is called a new index number is generated and automatically recorded as the current *level number* in the ongoing parse. The procedures which handle anaphora and cataphora may access the level number in order to determine the proper circumstances for reflexion.

Pronouns are recorded in the general (ie. topic neutral) lexicon **GENLEX** in the form:

pronoun(Form, [Gender, Case, Number, Type]).

where Type is set to either "r" (reflexive) or "p". The "nounphrase" procedure which identifies surface pronouns includes a call having the pattern:

crossref([Gender, Case, Number, Type], He, Var, FVB, SA, SR).

in which He and Var are to become the overt and transmission labels returned by the calling procedure, SA and SR are the SA and incoming SR lists of that calling procedure and FVB will be the FVB list returned as a singleton dummy binding should the pronoun be deemed to make a first cataphoric reference or as [ ] otherwise.

As required by section §8.2.2, the first clause for "crossref" must check the SA list provided for a possible antecedent referent and must also be capable of correctly handling examples such as:

(164) John shaves himself.

(165) John hopes that Mary believes that he shaves himself.

(166) Mary loves John and he loves himself.

Having extracted from the SA list a binding of form:

bind(He, Inx, [Md, C1, RC, Lev], node(N, [G, Num], L, D)).

the procedure reasons as follows.

(i) If the surface pronoun is nominative (Case=nom) then the level of the head noun phrase (which is recorded as a feature in the binding by the "replace" predicate) should in effect be regarded as the current level, moreover irrespective of the *actual* case (C1) of the head noun phrase we have a *nominative reference* thereto, hence RC must be set to "nom". In both (164) and (165) the phrase "he shaves himself" will be treated as on a level with "John", to whom there has in either case been a nominative reference.

(ii) If the surface pronoun is reflexive then the levels of pronoun and head noun phrase must (after any adjustment engineered under (i)) agree and the case of the pronoun must be accusative. Reflexion demands either a nominative head noun phrase (C1=nom) or a previous nominative reference thereto (RC=nom).

(iii) On any other occasion when the level of the head noun phrase is the current level, the head noun phrase must be accusative, nor must there have been a nominative reference thereto.

The second clause of "crossref" handles non initial cataphoric references and selects as overt label the variable from that binding chosen from the SR list, while the third clause, having established that there are no suitable bindings in either SA or SR list, generates a new syntactic variable, binds it to a dummy node and returns the dummy as a singleton list in FVB. In all cases "crossref" calls "copyvar" to construct the appropriate transmission label.

9.5. Left Recursion and the Brough Hogger Method

A number of syntax rules in Montague's PTQ have inverses which are explicitly left recursive viz:

(S3') CN → CN such that S

(S10')VP → VP VPADV

(S11')S → S [and|or] S

(S12')VP → VP [and|or] VP

(S13')NP → NP or NP.

Such rules defy naive recursive descent parsing in so far that their corresponding procedures become infinitely regressive.

Recourse to such established techniques for removing left recursion as result in only *weakly*†122 equivalent grammars, [W4], would jeopardise the principle of compositionality; for if the meaning of a compound is demonstrably a function of the meanings of specified elements then the syntactic representation of the compound must combine precisely the syntactic representations of those elements albeit in a left recursive configuration.†123

---

†122. Grammars which recognise the same set of sentences but assign different structural analyses are weakly equivalent. Strong equivalence requires the assigned structures to be the same.

†123. Partee, [P3] has argued for example that a "CN → CN S" analysis of restrictive relative clauses is necessary if the Russellian semantics for the definite singular determiner, according to which "the" implies uniqueness,are to be accepted.

An alternative "NP → NP S" analysis would, given a noun phrase of the form "the man who broke the bank at Monte Carlo", require identification of a unique man *prior* to consideration of the restriction imposed by the relative clause.

Yet a third choice considered by Janssen, [J4], is "NP → DET CN, DET → DET S"; but even were we to accept one of the alternatives the present problem would survive since all are left recursive.

```
sentence(S,I0,In) :-
        atomic_sentence(S,I0,In),
        assertz(wfst(sentence(S,I0,In))).

sentence(s(S1,S2),I0,In) :-
        wfst(sentence(S1,I0,I1)),
        scan(C,I1,I2),conjunction(C),
        sentence(S2,I2,In),
        assertz(wfst(sentence(s(S1,S2),I0,In))).
```

Fig 89

In my earlier prototype, [B4], left recursive constructions were handled with the assistance of a well formed substring table (or WFST), [W7], a conventional solution also adopted by Friedman and Warren. Every left recursive rule must have a left recursive extremal counterpart, thus for example the conjoined sentence rule must constitute one option from the pair:

S → S [and|or] S | atomic-S.

By employing a well formed substring table (and ignoring for the time being the ramifications entailed by the introduction of variable binding rules) such a pair may be handled in a DCG by the corresponding clauses outlined in fig 89. Both extremal and recursive clauses record their latest successes at the *foot* of the WFST, whereupon the last recorded entry becomes available when, on backtracking, any call to "sentence" falls through to the recursive clause.

One major disadvantage of the WFST technique outlined is that it fails for *competing* left recursive rules, for one or other must be given priority. The inverses of (S12) and (S10) which were adopted in my prototype **MDCG** and are illustrated in fig 90 fail for the example:

(167) John runs rapidly and swims slowly frequently.

Although "runs rapidly" is successfully entered in the WFST by the "S10" clause, it is impossible thereafter to backtrack to the "S12" clause so the parse fails at "and": were the clauses to be reversed failure would occur at "slowly".

An elegant mechanism for reducing left recursion to right recursion whilst preserving strong equivalence has recently been devised by Brough and Hogger, [B14]. Among other advantages, this

```
/* VP CONJUNCTION/DISJUNCTION RULE (WFST METHOD) */

verbphrase(node(N0,VF1,L0,D0),VF1,E,L,Ia,Iz,FVB,SA,SRa,SRz) :-
        wfst(verbphrase(node(R1,VF1,L1,D1),VF1,
                E,La,Ia,Ib,FVBa,SA,SRa,SRb)),
        scan(Conj,Ib,Ic),
        choose(vp,Rule,Conj),
        mix(FVBa,SA,SAa),
        join(E,node(R1,VF1,L1,D1),E1),
        verbphrase(node(R2,VF1,L2,D2),VF1,E1,Lb,Ic,Iz,
                FVBb,SAa,SRb,SRz),
        join(La,[Conj|Lb],Lc),
        join(L1,[Conj|L2],L3),
        mix(FVBa,FVBb,FVBc),
        quantify(vp,node(Rule,VF1,L3,
                        [node(R1,VF1,L1,D1),
                        node(R2,VF1,L2,D2)]),
                #node(N0,VF1,L0,D0),Lc,L,L3,L0,FVBc,FVB,[],SRz),
        recordz(wfst(verbphrase(node(N0,VF1,L0,D0),VF1,
                        E,L,Ia,Iz,FVB,SA,SRa,SRz))).

/* VP ADVERB RULE (WFST METHOD) */

verbphrase(node(N0,F0,L0,D0),VF,E,L,Ia,Iz,FVB,SA,SRa,SRz) :-
        wfst(verbphrase(node(N1,F1,L1,D1),VF,E,La,Ia,Ib,
                FVBa,SA,SRa,SRb)),
        mix(FVBa,SA,SAa),
        join(E,node(N1,F1,L1,D1),E1),
        vpadverb(VPADV,AV,E1,Lb,Ib,Iz,FVBb,SAa,SRa,SRz),
        join(La,Lb,Lc),
        join(L1,Lb,L2),
        mix(FVBa,FVBb,FVBc),
        quantify(vp,node(#10:7,VF,L2,
                        [VPADV,node(N1,F1,L1,D1)]),
                node(N0,F0,L0,D0),Lc,L,L2,L0,FVBc,FVB,[],SRz),
        recordz(wfst(verbphrase(node(N0,F0,L0,D0),VF,
                E,L,Ia,Iz,FVB,SA,SRa,SRz))).
```

Fig 90

mechanism overcomes the difficulty caused by competing clauses, hence a generalisation thereof has been adopted in **TMDCG.**

According to the Brough Hogger method (BHM), any left recursive rule representable by an augmented PSG pair like that illustrated in fig 91 (a) may be reduced to the triple shown in fig 91 (b). This triple can in turn be converted into raw DCG format as indicated in fig 91 (c) where the arguments of the "recurse" predicate are the structure built to date, the structure returned, the incoming string and the uncon-

```
(a)
category(c(C,A)) → category(C) appendage(A)
category(B) → basic(B).

(b)
category(Z) --> basic(B),recurse(B,Z).
recurse(B,B).
recurse(B,Z) --> appendage(A),recurse(c(B,A),Z).

(c)
category(Z,I0,In) :- basic(B,I0,I1),recurse(B,Z,I1,In).
recurse(B,B,I,I).
recurse(B,Z,I0,In) :- appendage(A,I0,I1),recurse(c(B,A),Z,I1,In).
```

Fig 91

sumed remainder respectively. A triple argument category predicate with places for structure and I/O buffers requires a quadruple argument "recurse" predicate; for both structure and string undergo modification during recursion and "recurse" requires two parameters for each factor that varies.

Parsing procedures in TMDCG typically involve hendecatuple arguments, thus it behoves us to enquire how the Brough Hogger method could be generalised to accommodate additional argument places. Clearly not all of the arguments will prove sensitive to recursion: the environment, hold stack and SA list inherited by a procedure seeking category ξ represent constant inputs irrespective of whether the instance of ξ identified is basic or one involving recursion. Sensitive arguments will include the node returned, the transmission label returned, the unconsumed string, the FVB list and the SR list on exit.

A generalisation of the "recurse" predicate which proves compatible with category procedures of any acity should take the form outlined in fig 92 (a), while in TMDCG a standard call to "recurse" should follow the pattern shown in fig 92 (b) where N, F, E, L, Iz, H, FVB, SA and SRz are as described seriatim in the discussion of parsing procedure templates in section §9.2.1 and the fields flagged with "1" are the default output values.†124 The functor ξ is instantiated to a flag for the category involved ("s", "np", "cn" or "vp") and is designed to eliminate inappropriate investigations.

Whether or not the extremal clause for "recurse" precedes or succeeds the recursive clause in the

†124. The "recurse" clause called by "nounphrase" is exceptional in that it splits the feature list into invariants and variants. Mood and case are insensitive to recursion, but the gender and number features of a noun phrase must be allowed to vary in the manner described in section §6.4.1

```
(a)
recurse(Given,Old_Variants,Old_Variants).

recurse(given(← Insensitives →),
            variants(← Old_Sensitives →),
            variants(← New_Sensitives →)) :-
                  /* Sub goals to convert Old_Sensitives
                        to Intermediates */
                  recurse(given(← Insensitives →),
                        variants(← Intermediates →),
                        variants(← New_Sensitives →)).

(b)
recurse(g(F, E, H, SA),
            ξ(N1, L1, Iz1, FVB1, SRz1),
            ξ(N, L, Iz, FVB, SRz)).
```

Fig 92

Brough Hogger method has no effect on termination, but the choice reflects alternative strategies. An "extremal clause first" option indicates a decision to be content with the shortest identifiable member of a category until this proves unacceptable while the alternative order represents a determination to press on for a longer string wherever possible.

In terms of processing efficiency it is interesting to compare the benchmarks derived when sample sentences were parsed using in turn the techniques WFST, E-First (BHM with extremal clause first) and E-Last (BHM with extremal clause last) to handle left recursion. Run times in milliseconds registered under the different techniques for a chosen selection of sentences are tabulated in fig 93. In All cases BHM outperforms the WFST technique in handling left recursion and in all but three cases the "extremal clause first" option proves the more advantageous: accordingly it is BHM (E-First) which is adopted in TMDCG.

9.6. The Lexica

Following a distinction dating back to Aristotle, topic neutral vocabulary is assigned to the general lexicon **GENLEX** which also contains the procedures for verb inflection and noun declension anticipated in chapter 6.

The module **LEXTMG** contains vocabulary specific to the domain areas of the various sample sentences under investigation. Both lexica are listed as appendices D and E and demand no further exposition.

Benchmarks (milliseconds)			
Sentence	WFST	E-First	E-Last
John runs and Mary walks	473	263	404
Every man loves a woman	455	316	460
Mary believes that a unicorn will not run in the park	881	696	953
John believes that the men and the women have eaten a fish	973	881	1131
John believes that Mary seeks a unicorn	476	451	648
John tries to catch a fish and eat it	1677	2004	1394
John loves the woman who runs	1055	853	735
The man who runs loves Mary	716	602	688
The man who loves the woman who runs walks	4206	2178	1523
John runs or Bill swims and Mary walks	1717	545	741
John runs rapidly and swims slowly frequently	FAIL	580	590
John knows what grade every student deserves	1178	806	962
John knows which professor teaches every student	1120	818	956
John knows which man Mary and Emily love	1517	901	982
John knows which woman who loves him Mary hates	3928	2061	2215
John knows which man loves which woman	1529	1074	1222
John knows which woman which man loves	1582	1013	1219

Fig 93

9.7. TIL Translation under LILT

Galilean post order traversal of analysis trees is undertaken in **LILT** by the procedure "translate" which is called in the form:

translate(Step, Tree, TIL)

where "Tree" is the analysis tree input, "TIL" is the language of intensional logic translation and "Step" is the step number of the composition and simplification process at which that translation appears as illustrated in examples (H-1) ... (H-4) of appendix H.

Given an analysis tree of form node(N, F, L, [D1, D2]), recursive calls to "translate" are first made on behalf of the younger and elder daughters D2 and D1, their reduced translations being returned as T2 a T1 respectively. Thereafter a call is made to the **TBASE** procedure "formulate" with template:

formulate(node(N, F, L, [T1, T2]), UIL)

so that the semantic operation *k* recorded in N, which has the form #*n*:*k* or #*n*:(*k*:*i*), may be applied to T1 and T2 to give the unreduced logical form UIL.

Since simple juxtaposition does not generate a well formed PROLOG expression, any semantic operation requiring *juxtaposition* of variations upon T1 and T2 is required by **TBASE** to introduce a functor "eval" for the purpose of legitimising the marriage. Thus for example:

g_0(T1, T2)	generates	eval(T1, ^T2)
g_{21}(T1, T2)	generates	eval(T1, ^lambda(Z: eval(T2, Z)))

Should the analysis tree have the form node(N, F, L, [sense(Item, Feature)]) then control is passed not to "formulate" but to the **TBASE** procedure "sense" with template:

sense(Item, Feature, TIL).

which returns a basic **TIL** translation for the lexical or pseudo lexical item concerned.

Once the unreduced formula UIL has been identified, a message announcing the application of the appropriate rule T*n* is issued whereafter UIL is subjected to *simplification* in order to derive the reduced **TIL** expression TIL. A call to the **LILT** predicate "simplify" has the form:

simplify(StepIn, StepOut, UIL, TIL).

where StepIn and StepOut are the reported composition and simplification numbers at the commencement and termination of simplification, UIL is the unreduced expression and TIL a fully reduced expression in the language of tensed intensional logic. The initial clauses of "simplify" do no more than impose the requirement for simplification on embedded formulae governed by divers operators; but then come clauses which undertake the evaluation of both lambda expressions and any expression not yet in relational form,[†125] the substitution of identicals and the application of meaning postulates.

The reduction of a lambda expression of form:

λVar(Body)[Value]

---

†125. **LILT** will simplify original **PTQ** formulae in **IL**, formulae in Gallin's **Ty2**, [G4], or **TIL** formulae with equal ease. The choice of representation lies in the assignment to lexical items by **TBASE**.

where the argument denoted by "Value" has already undergone simplification is accomplished by recourse to a predicate "convert"†[126] having the template:

convert(StepIn, StepOut, Value, Var, Body, Result).

Only an atom can stand in predicative position in PROLOG, hence all higher order lambda abstraction must be formulated in terms of *lower case* "variables". For individual variables in the range $v_{j,e}$, where $j>0$ was selected as the next available odd subscript (see fig 2), *upper case* may prudently be employed so that responsibility for choosing an appropriate variable may be delegated to the PROLOG interpreter. Accordingly if Body is a PROLOG atom or variable a check must be made to ascertain whether or not it matches the lambda variable Var. If it matches then Body must be replaced by Value, ie Result = Value, otherwise Result = Body. Whenever Body is a complex formula, "convert" issues a call to its slave procedure "insert" in the form:

insert(StepIn, StepOut, Value, Var, Body Result).

so that the internal structure of Body may be investigated and Value inserted at appropriate points. Direct action by "insert" is actually limited to cases where Var turns out to match either the principal functor of Body or the principal functor of Body1 where Body = ^Body1 or Body = ˇBody1. In all other cases "insert" simply converts Body into the form [*head*|Tail] and returns as its result the simplification of the reconverted repacement [*head*|Tail1], where responsibility for deriving Tail1 from Tail devolves on a slave procedure "substitute" having the template:

substitute(StepIn, StepOut, Value, Var, Tail, Tail1).

A list of form [H|T] is reduced by "substitute" to [H1|T1] by the expedient of recalling "convert" to generate H1 from H and issuing a recursive call to "substitute" to derive T1 from T.

Although **LILT** is designed primarily to generate **TIL** expressions with an ongoing commentary (as in H-1 ... H-4), the commentary may be switched off at will by inserting:

```
message(_,_,_) :- !.
```

before the commencement of the substantive "message" predicates.

---

†126. Originally inspired by the "substitute" predicate of Clocksin and Mellish, [C3].

The current version of **LILT**, unlike its predecessor, [B4], is a "stand alone" language of intensional logic translator capable of handling alternative logics if formulated in terms of higher order lambda abstraction. For this reason the assignment of primitive formulae in a particular logic, like the specification of meaning postulates, devolves on the separate module **TBASE** which also accepts the onus for preserving type compatibility by implementing the semantic operations of an orthodox Montague grammar.

POSTSCRIPT

¶ After summarising the strategic features and tactical scope of **TMDCG/LILT** we consider alternative demands that might legitimately be made of a simulation during computational investigation of Montague grammar and speculate concerning the possibilities for accession thereto. Prima facie areas for extension, improvement and utilisation are finally identified.

Summary

A computer implementation designed both to simulate and to refine a target grammar should be compact, self documenting and sensitive to change. Where the target grammar is a generalised categorial grammar in the Montagovian idiom these characteristics are realised by the DCG technique adopted in **TMDCG.**

Those rules of a generalised categorial grammar which do not involve variable binding have immediate inverses which may or may not prove to be left recursive. Most non left recursive inverse rules can be unproblematically formulated as rules of a definite clause grammar which merely combines a phrase structure grammar with a recursive descent recognition algorithm. In preparation for the introduction of variable binding rules, inverses which parse non pronominal noun phrases must be given both the opportunity to effect *replacement* by a syntactic variable as demanded by the Friedman Warren algorithm and also the privilege of accessing a "hold stack" if extraposition is to be accommodated. Rules dealing with surface pronouns interface with a multi level cross referencing mechanism capable of handling anaphora, cataphora and reflexion when provided with suitable *substitution above* and *substituens required* lists.

Left recursion may be handled elegantly and efficiently by the Brough Hogger method while the *quantification* facility of the Friedman Warren algorithm serves to introduce variable binding rules.

Provided that a conventional multi pass (ie. directed process) mode of analysis is to be adopted, the modification of **TMDCG** and **TBASE** so as to accommodate absent grammatical constructions, thus allowing experimentation with proposed extensions to **TMG,** would entail no major overhaul, while the separation of **LILT** from **TBASE** would continue to provide scope for experimentation with alternative logics. Multi pass operation with tabulated composition and simplification is indeed the preferred mode for

an implementation designed to facilitate investigation into the *behaviour* of Montague grammars on the assumption that the most obvious requests from the investigator will be one or other of:

(i) Provide a parse and interpretation for sentence S.

(ii) Provide *all* parses and interpretations for sentence S.

The examples listed in appendix H constitute responses to just such requests, hence TMDCG and LILT constitute tools custom built for this purpose.

Prognosis

Could alternative purposes be fulfilled with comparable facility by a definite clause grammatical inversion of Montague grammar? My concern in this postscript is to reflect upon the nature of such alternative requirements and thus identify areas for future exploration, although prognosis must perforce be speculative.

As regards further *computational investigations*, it seems not unreasonable to consider as legitimate interests of the researcher such requests as:

(iii) Provide all *semantically distinct* analyses of sentence S.

(iv) Provide analyses of sentence S subject to limitation L.

(v) Provide the *most plausible* analysis of sentence S in context C.

Requirement (iii) is no more than a demand for semantic equivalence parsing accession to which would involve switching to a single pass mode of operation. Modifications implied by such a switch would include the return of a TIL translation instead of the present overt label L in each node structure and a call to a laconic variation of the LILT procedure "translate", in place of maintenance of now obsolete transmission labels, by every parsing procedure responsible for returning a node.

No recursion would be required by this "translate" procedure; for the TIL field of a node of form:

node(N, F, TIL, [sense(Item, Features)])

could be completed following a call, as by the present extremal clause, to the TBASE procedure "sense", and a reduced TIL expression for a node of form:

node(N, F, TIL, [node(N1, F1, T1, D1), node(N2, F2, T2, D2)])

could be derived through the calls

formulate(node(N, F, TIL, [T1, T2]), UIL), simplify(S0, S1, UIL, TIL).

Since the commentary would no longer be germane, the "Step" arguments of "simplify" could now be eliminated altogether. As a consequence of such modifications **TMDCG** would continue to return *complete* analysis trees, but with **TIL** expressions in place of conventional nodal phrases. The abortion facility built into the cataloguing mechanism would effectively eliminate duplicate findings, although cataloguing would now become necessary even when the entire string had been consumed.

A request to provide all parses of:

(168) Some professor believes that every student studies some language.

having "some language" in outer scope might fall under our requirement (iv). A possible approach might be to provide the limitation as a parameter to the cataloguing procedure so that trees showing unacceptable dominance relationships might be eliminated as soon as detected.

Some kinship between requirements (iv) and (v) is evident, for whereas the limitation in the former is directly supplied, that pertaining to the latter must be derived from context. Thus *provided that the limitation could be so derived* a common approach might suffice in both instances. Deriving phenomena such as plausible scope limitations from context is however no trivial task and represents a research area in its own right. As I have argued, Montague's general theory was *not* intended, as is often mistakenly claimed, to apply only to individual declarative sentences in isolation, hence context sensitive Montague grammar is a bona fide field of exploration. Whether or not data structures such as frames or conceptual dependency representations could indeed be adapted to simulate Montague's context set "J" remains to be seen, but it is interesting to note that tentative steps have been taken, [J5], to reconcile MG and CD analyses in a *context free* environment.

The system of multi indexed tense logic adopted in chapter 6 is designed to demonstrate that a natural syntax for tense and aspect is in principle interpretable, but the coverage is by no means exhaustive. No attempt has been made in my treatment of verb phrases to distinguish between episodic and dispositional locutions, between states, processes, accomplishments and achievements or between stative and

dynamic verbs. However it is hoped that the suggestions made will provide a foundation for a more thoroughgoing investigation in these areas and that they may stimulate a more elegant solution in due course: thus **TMG** may serve as a prototype for further extensions to the coverage of verb phrases in the Montagovian idiom. Subcategorisation and selectional restriction are notoriously absent in Montague's writings, so priority must be given to their introduction in any more sophisticated analysis of verb phrase constructions and their interpretation.

Whatever the form of enquiry envisaged during a computational investigation, provision must be made for diagnostic information in the event of failure to respond as expected, but the facility incorporated in **TMDCG** is undeniably rudimentary. An optimal solution is not obviously to be found using the standard first* and follow list error recovery technique appropriate to recursive descent parsing of *deterministic* grammars and doubtless the whole question of recovery in the context of a top down, depth first, left to right backtracking definite clause grammar requires independent research.

Whether a Montague grammar such as **TMG** could prudently be employed in a computer *utilisation* other than machine translation seems to me to remain problematic. A full range of English tense form nuances is hardly germane to an answering system in which direct questions in the present or present perfect constitute the standard user input, while the utility of a genuine Montague grammar in discourse analysis is plainly dependent upon a solution to the problem of representing Montague's context set "J" in terms of cost effectively machine processable data structures: a point which may be made without recourse to misleading anthropomorphic extravaganza concerning the machine's capacity to understand. It would however certainly be of interest to determine whether or not the natural tense system of **TIL** could provide a more catholic and flexible interlingual vehicle than any presently available.

In facilitating the development of **TMG**, the **TMDCG/LILT** suite has fulfilled its original purpose and now remains as a tool for extension or adaptation in pursuance of any of the above suggested enquiries, thus computational investigation of Montague grammar will continue. *Et non sentitur sedulitate labor.*

APPENDIX A: TIL & TMG

The Language TIL

Semantic Types and Denotations for TIL

The set *Type* of semantic types is the smallest set satisfying the definition:

e ∈ *Type*.

t ∈ *Type*.

i ∈ *Type*.

If **a, b** ∈ *Type* then **<ab>** ∈ *Type*.

If **a** ∈ *Type* then **<sa>** ∈ *Type*.

Possible denotations are assigned as follows:

$M = \langle D, W, T, Int, \leq, I \rangle$

where D is a domain of possible individuals, W a set of possible worlds, T a set of moments in time ordered by $\leq$, $Int \subset Power(T)$ such that if $\mathbf{i} \in Int$ then for all $\mathbf{m}, \mathbf{m}', \mathbf{m}'' \in T$, if $\mathbf{m} \in \mathbf{i}$ and $\mathbf{m}'' \in \mathbf{i}$ and $\mathbf{m} < \mathbf{m}' < \mathbf{m}''$ then $\mathbf{m}' \in \mathbf{i}$, and I an interpretation function.

$den(\mathbf{e},M) = D$.

$den(\mathbf{t},M) = \{0,1\}$.

$den(i,M) = Int$.

$den(\langle \mathbf{ab} \rangle,M) = den(\mathbf{b},M)^{den(\mathbf{a},M)}$.

$den(\langle s,\mathbf{a} \rangle,M) = den(\mathbf{a},M)^{W \times Int}$.

$sen(\mathbf{a},M) = den(\langle \mathbf{sa} \rangle,M)$

Lexicon for TIL

(TILs1) $Var_{\mathbf{a}} = \{v_{n,\mathbf{a}} : n \geq 0\}$

(TILs2) $Con_{\mathbf{a}} = \{c_{n,\mathbf{a}} : n \geq 0\}$.

where $Var_{\mathbf{a}}$ is the set of variables of type **a**, and $Con_{\mathbf{a}}$ is the set of (non logical) constants of type **a**. By convention $v_{0,i}$, $v_{1,i}$, $v_{2,i}$ ... may be abbreviated to **t**, **t′**, **t″** ... , while $c_{0,i}$ is written as **t*** and $c_{1,i}$ as **t@**.

Syntax for TIL

The meaningful expressions of **TIL** are the members of $\cup_{\mathbf{a} \in Type} \mathbf{ME_a}$:

(TILs3) If $\alpha \in Var_{\mathbf{a}}$ then $\alpha \in \mathbf{ME_a}$.

(TILs4) If $\alpha \in Con_{\mathbf{a}}$ then $\alpha \in \mathbf{ME_a}$.

(TILs5) If $\alpha \in \mathbf{ME_a}$ and $\upsilon \in Var_{\mathbf{b}}$ then $\lambda\upsilon\alpha \in \mathbf{ME_{<ba>}}$.

(TILs6) If $\alpha \in \mathbf{ME_{<ab>}}$ and $\beta \in \mathbf{ME_a}$ then $\alpha(\beta) \in \mathbf{ME_b}$.

(TILs7) If $\alpha, \beta \in \mathbf{ME_a}$ then $\alpha{=}\beta \in \mathbf{ME_t}$.

(TILs8) If $\zeta,\xi \in \mathbf{ME_i}$ then $\lceil \zeta \subseteq \xi \rceil$ and $\lceil \zeta < \xi \rceil \in \mathbf{ME_t}$.

(TILs9) If $\alpha \in \mathbf{ME_a}$ then $\lceil \hat{}\alpha \rceil \in \mathbf{ME_{<sa>}}$.

(TILs10) If $\alpha \in \mathbf{ME_{<sa>}}$ then $\lceil \check{}\alpha \rceil \in \mathbf{ME_a}$.

(TILs11) If $\Phi, \Psi \in \mathbf{ME_t}$ then:

$\lceil \neg\Phi \rceil \in \mathbf{ME_t}$.

$\lceil (\Phi \wedge \Psi) \rceil \in \mathbf{ME_t}$.

$\lceil (\Phi @ \Psi) \rceil \in \mathbf{ME_t}$.

$\lceil (\Phi \vee \Psi) \rceil \in \mathbf{ME_t}$.

$\lceil (\Phi \rightarrow \Psi) \rceil \in \mathbf{ME_t}$.

$\lceil (\Phi \leftrightarrow \Psi) \rceil \in \mathbf{ME_t}$.

(TILs12) If $\Phi \in \mathbf{ME_t}$ and for some **a**, $\upsilon \in Var_{\mathbf{a}}$ then:

$\lceil \forall\upsilon\Phi \rceil \in \mathbf{ME_t}$.

$\lceil \exists\upsilon\Phi \rceil \in \mathbf{ME_t}$.

(TILs13) If $\Phi \in \mathbf{ME_t}$ then:

$\lceil \Psi(\Phi) \rceil \in \mathbf{ME_t}$

where Ψ = *nec, poss, pres, past, fut, perf,* or *prog.*

(ILs14) If $\tau \in \mathbf{ME_i}$ and $\Phi \in \mathbf{ME_t}$ then

$\lceil \mathrm{at}(\tau,\Phi) \rceil \in \mathbf{ME_t}$ and $\lceil \mathrm{ab}(\tau,\Phi) \rceil \in \mathbf{ME_t}$.

Semantics for TIL

(TILt1) If $\upsilon \in Var_{\mathbf{a}}$ and $g \in G$ then $g(\upsilon) \in den(\mathrm{a},M)$.

(TILt2) If $\alpha \in Con_{\mathbf{a}}$ then $I(\alpha) \in sen(\mathrm{a},M) = den(\mathrm{a},M)^{W\times Int}$.

In particular $I(\mathrm{t}^*)(w,i) = i$.

(TILt3) If $\upsilon \in Var_{\mathbf{a}}$ then $[\![\upsilon]\!]^{M,w,<i,j,k,(h)>,g} = g(\upsilon)$.

(TILt4) $[\![\mathrm{t}@]\!]^{M,w,<i,j,k,(h)>,g} = [\![\mathrm{t}^*]\!]^{M,w,<k,k,k,(h)>,g}$ otherwise

If $\alpha \in Con_{\mathbf{a}}$ then $[\![\alpha]\!]^{M,w,<i,j,k,(h)>,g} = I(\alpha)(w,i)$.

(TILt5) If $\alpha \in \mathbf{ME}_{\mathbf{a}}$ and $\upsilon \in Var_{\mathbf{b}}$ then $[\![\lambda\upsilon\alpha]\!]^{M,w,<i,j,k,(h)>,g} = \eta \in den(\mathrm{a},M)^{den(\mathrm{b},M)}$

such that for all $\beta \in den(\mathrm{b},M)$, $\eta(\beta) = [\![\alpha]\!]^{M,w,<i,j,k,(h)>,g'}$

where g and g' differ *at most* in that $g'(\upsilon) = \beta$.

(TILt6) If $\alpha \in \mathbf{ME}_{<\mathbf{ab}>}$ and $\beta \in \mathbf{M}_{\mathbf{a}}$ then, for $i \subseteq j$, $[\![\alpha(\beta)]\!]^{M,w,<i,j,k,(h)>,g} =$

$[\![\alpha]\!]^{M,w,<i,j,k,(h)>,g}([\![\beta]\!]^{M,w,<i,j,k,(h)>,g})$.

(TILt7) If α and $\beta \in \mathbf{ME}_{\mathbf{a}}$ then $[\![\lceil \alpha{=}\beta \rceil]\!]^{M,w,<i,j,k,(h)>,g} = 1$

iff $[\![\alpha]\!]^{M,w,<i,j,k,(h)>,g} = [\![\beta]\!]^{M,w,<i,j,k,(h)>,g}$, 0 otherwise.

(TILt8) $[\![\lceil \zeta{\subseteq}\xi \rceil]\!]^{M,w,<i,j,k,(h)>,g} = 1$ iff $[\![\zeta]\!]^{M,w,<i,j,k,(h)>,g} \subseteq [\![\xi]\!]^{M,w,<i,j,k,(h)>,g}$.

$[\![\lceil \zeta{<}\xi \rceil]\!]^{M,w,<i,j,k,(h)>,g} = 1$ iff for all $m \in [\![\zeta]\!]^{M,w,<i,j,k,(h)>,g}$

and all $m' \in [\![\xi]\!]^{M,w,<i,j,k,(h)>,g}$, $m < m'$.

(TILt9) If $\alpha \in \mathbf{ME}_{\mathbf{a}}$ then $[\![\lceil \hat{}\alpha \rceil]\!]^{M,w,<i,j,k,(h)>,g} = \eta \in den(\mathrm{a},M)^{W\times Int}$ such that,

for all $<w,i> \in W\times Int$, $\eta(w,i) = [\![\alpha]\!]^{M,w,<i,j,k,(h)>,g}$.

(TILt10) If $\alpha \in \mathbf{ME}_{<\mathbf{sa}>}$ then $[\![\lceil \check{}\alpha \rceil]\!]^{M,w,<i,j,k,(h)>,g} = [\![\alpha]\!]^{M,w,<i,j,k,(h)>,g_{(w,i)}}$.

(TILt11) If Φ and $\Psi \in \mathbf{ME}_{\mathbf{t}}$ then:

$[\![\lceil \neg\Phi \rceil]\!]^{M,w,<i,j,k,(h)>,g} = 1$ iff $[\![\Phi]\!]^{M,w,<i,j,k,(h)>,g} = 0$.

$[\![\lceil (\Phi\wedge\Psi) \rceil]\!]^{M,w,<i,j,k,(h)>,g} = 1$ iff both $[\![\Phi]\!]^{M,w,<i,j,k,(h)>,g} = 1$

and $[\![\psi]\!]^{M,w,<i,j,k,(h)>,g} = 1$.

$[\![\lceil (\Phi@\Psi) \rceil]\!]^{M,w,<ijk,(h)>,g} = 1$ iff for some i',i'' $[\![\Phi]\!]^{M,w,<i'jk,(i')>,g} = 1$

and $[\![\Psi]\!]^{M,w,<i''jk,(i')>,g} = 1$ and $i' \subseteq i$ and $i'' \subseteq i$.

$$\text{[}\ulcorner(\Phi\vee\Psi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff either } \text{[}\Phi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1$$

or $\text{[}\psi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1$.

$$\text{[}\ulcorner(\Phi\rightarrow\Psi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff either } \text{[}\Phi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 0$$

or $\text{[}\psi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1$.

$$\text{[}\ulcorner(\Phi\leftrightarrow\Psi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff}$$

$$\text{[}\Phi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = \text{[}\Psi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g}.$$

(TILt12) If $\Phi \in \mathbf{ME}_t$ and for some a, $\upsilon \in Var_a$ then:

$$\text{[}\ulcorner\forall\upsilon\Phi\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff}$$

$\text{[}\Phi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g'} = 1$ for all g' that are υ-variant to g.

$$\text{[}\ulcorner\exists\upsilon\Phi\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff}$$

$\text{[}\Phi\text{]}^{M,w,\langle i,j,k,(h)\rangle,g'} = 1$ for some g' that is υ-variant to g.

(TILt13) If $\Phi \in \mathbf{ME}_t$ then:

$$\text{[}\ulcorner nec\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w',\langle i,j,k,(h)\rangle',g} = 1$$

for all $w' \in W$ and all i', j', k' and $h' \in Int$.

$$\text{[}\ulcorner poss\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w',\langle i,j,k,(h)\rangle',g} = 1$$

for some $w' \in W$ and some i', j', k' and $h' \in Int$.

$$\text{[}\ulcorner pres(\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w,\langle i,j',k,(h)\rangle,g} = 1 \text{ and } j'=j.$$

$$\text{[}\ulcorner past(\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w,\langle i,j',k,(h)\rangle,g} = 1 \text{ and } j'<j.$$

$$\text{[}\ulcorner fut(\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}P\text{]}^{M,w,\langle i,j',k,(h)\rangle,g} = 1 \text{ and } j'>j.$$

$$\text{[}\ulcorner perf(\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w,\langle i,j',k,(h)\rangle,g} = 1$$

and j is a final subinterval of j'.

$$\text{[}\ulcorner prog(\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w,\langle i,j',k,(h)\rangle,g} = 1 \text{ and } j\sqsubseteq j'.$$

(TILt14) $\text{[}\ulcorner at(\tau,\Phi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\Phi\text{]}^{M,w,\langle i',j,k,(h)\rangle,g} = 1$

and $i' = \text{[}t\text{]}^{M,w,\langle i,j,k,(h)\rangle,g}$.

$\text{[}\ulcorner ab(\tau,X)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g}$ is defined by the *first* matching case from the following:

$\text{[}\ulcorner ab(\tau,\Phi\wedge\Psi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1 \text{ iff } \text{[}\ulcorner ab(\tau,\Phi)\wedge ab(\tau,\Psi)\urcorner\text{]}^{M,w,\langle i,j,k,(h)\rangle,g} = 1$.

$\llbracket \lceil ab(\tau,\Phi@\Psi)\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket \exists t(ab(\tau,\Phi))\wedge\exists t(ab(\tau,\Psi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$.

$\llbracket \lceil ab(\tau,\Sigma(\Phi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket \Sigma(ab(\tau,\Phi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ where Σ is a quantifier.

$\llbracket \lceil ab(\tau,at(\tau',\Phi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket \lceil ab(\tau',\Phi)\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$.

$\llbracket \lceil ab(\tau,pres(\Phi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket \lceil ab(\tau,\Phi)\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$.

$\llbracket \lceil ab(\tau,fut(\Phi)\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket \lceil fut(\Phi)\rceil \rrbracket^{M,w,<i',j',k,(h)>,g} = 1$

where $j' = \llbracket \tau \rrbracket^{M,w,<i,j,k,(h)>,g} \geq \llbracket t@ \rrbracket^{M,w,<i,j,k,(h)>,g}$.

$\llbracket \lceil ab(\tau,past(\Phi))\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff either

(i) $h<k$ and $\llbracket \Phi \rrbracket^{M,w,<i',j',k,(h)>,g} = 1$ with $h \subseteq j' = \llbracket \tau \rrbracket^{M,w,<i,j,k,(h)>,g}$

(ii)$h \geq k$ and $\llbracket \lceil past(\Phi)\rceil \rrbracket^{M,w,<i',j',k,(h)>,g} = 1$ where $j' = \llbracket \tau \rrbracket^{M,w,<i,j,k(h)>,g}$

$\leq \llbracket t@ \rrbracket^{M,w,<i,j,k,(h)>,g}$.

$\llbracket \lceil ab(\tau,Other)\rceil \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$ iff $\llbracket Other \rrbracket^{M,w,<i',j,k,(h)>,g} = 1$

where $j = \llbracket \tau \rrbracket^{M,w,<i,j,k,(h)>,g}$.

Formula Φ is true at speech point k iff for some i $\llbracket \Phi \rrbracket^{M,w,<i,j,k,(h)>,g} = 1$.

Colloquial Variables	
Colloquial Form	Pure Form
e	$v_{0,e}$
x_n	$v_{2*(n+1),e}$
X, Y, Z	$v_{j,e}$ where j>0 is the next unused odd subscript.
r, s	$v_{0,<st>}$ and $v_{1,<st>}$
p, q	$v_{0,<s<et>>}$ and $v_{1,<s<et>>}$
n	$v_{0,<s<<s<et>>t>>}$
b	$v_{0,<s<<s<<s<et>>t>><et>>>}$
f	$v_{0,<s<<s<<s<et>>t>>t>>}$
u	$v_{0,<s<<st>t>>}$

The Grammar TMG

Categories and Basic Assignments of TMG

Category	Mnemonic	Type	Category	Mnemonic	Type
t	-	t	LIV//T	COP	<<s,*f*(T)><et>>
CN	-	<et>	LIV/t	SCVERB	<<st><et>>
CLV⊆LIV ⊆IV⊆QIV	-	<et>	LIV/LIV	ICVERB	<<s<et>><et>>
CPV⊆PIV	-	<et>	LIV/Q	QCVERB	<<s,*f*(Q)><et>>
t/IV	T	<<s<et>>t>	IV/LIV	LAUX	<<s<et>><et>>
t//IV	WH	<<s<et>>t>	IV/$_1$IV	MAUX1	<<s<et>><et>>
T/CN	DET	<<s<et>>,*f*(T)>	IV/$_2$IV	MAUX2	<<s<et>><et>>
WH/CN	WDET	<<s<et>>,*f*(WH)>	IV/$_3$IV	TAUX	<<s<et>><et>>
t/t	Q	<<st>t>	IV/$_4$IV	NAUX	<<s<et>><et>>
(t/t)/t	QMARK	<<st><<st>t>>	IV/$_5$IV	GAUX	<<s<et>><et>>
t/T	FV	<<s,*f*(T)>t>	IV/PIV	PAUX	<<s<et>><et>>
FV/QIV	TENSE	<<s<et>>,*f*(FV)>	IV/$_6$IV	IAV	<<s<et>><et>>
LIV/T	TV	<<s,*f*(T)><et>>	IAV/T	PREP	<<s,*f*(T)>,*f*(IAV)>
PIV/T	PTV	<<s,*f*(T)><et>>	(t/T)/(t/T)	TMADV	<<s,*f*(FV)>,*f*(FV)>
PTV/TV	PASS	<<s,*f*(TV)>,*f*(PTV)>	t/t	SADV	<<st>t>

Categories are mapped onto semantic types by the function ***f***:

f(t) = **t**.

f(QIV) = ***f***(LIV) = ***f***(PIV) = **<et>**.

f(CN) = **<et>**.

f(A/$_n$B) = **<<s,*f*(B)>*f*(A)>**.

Pseudo Lexical Assignments (Defined by S0)			
Marker	Semantic Type	Basic of Category	Translation
present	<<s<et>><<s<<s<et>>t>>t>>	TENSE = (t/T)/QIV	λpλn(pres(ˇn(p)))
past	<<s<et>><<s<<s<et>>t>>t>>	TENSE = (t/T)/QIV	λpλn(past(ˇn(p)))
?yn	<<st><<st>t>>	QMARK = (t/t)/t	λsλr(ˇr∧(r=^ˇs∨r=^¬ˇs))
?	<<st><<st>t>>	QMARK = (t/t)/t	λsλr(ˇr∧r=s)
passive	<<s<<s<<s<et>>t>><et>> <<s<<s<et>>t>><et>>	PASS = (PIV/T)/(LIV/T)	λbλnλX[(ˇn)^ˇb(^λpˇp(X)))]
agent	<<s<et>>t>	T = t/IV	λpΞYˇp(Y)

Lexical Assignments in **TMG** (Defined by S1)		
If $\xi \in \mathbf{B}_{CN}$	then ξ ≈»	λe(ξ′(e))
If $\xi \in \mathbf{B}_{LIV}$	then ξ ≈»	λe(ξ′(e))
If $\xi \in \mathbf{B}_{TV}$	then ξ ≈»	λnλe(ξ′(e,n))
If $\xi \in \mathbf{B}_{COP}$	then ξ ≈»	λnλeˇn(^λY(e=Y))
If $\xi \in \mathbf{B}_{SCVERB}$	then ξ ≈»	λrλe(ξ′(e,r))
If $\xi \in \mathbf{B}_{ICVERB}$	then ξ ≈»	λpλe(ξ′(e,p))
If $\xi \in \mathbf{B}_{QCVERB}$	then ξ ≈»	λuλe(ξ′(e,u))
If $\xi \in \mathbf{B}_{IAV}$	then ξ ≈»	λpλe(ξ′(e,p))
If $\xi \in \mathbf{B}_{IV/nIV}$ (n<6)	then ξ ≈»	λpλe(ξ′(ˇp(e)))
If $\xi \in \mathbf{B}_{LAUX}$ or $\mathbf{B}_{PAUX}$	then ξ ≈»	λpλe(ˇp(e))
If $\xi \in \mathbf{B}_{TMADV}$	then ξ ≈»	λfλn(ˇn)[^λe(at(t*,ˇf(^λpˇp(e))∧t*⊆ξ′))]
If $\xi \in \mathbf{B}_{PREP}$	then ξ ≈»	λnλpλe(ξ′(e,^λZˇp(Z),n))
If $\xi \in \mathbf{B}_{SADV}$	then ξ ≈»	λp(ξ′(ˇp))
If $\xi \in \mathbf{B}_{WH}$	then ξ ≈»	λpΞXˇp(X)
If $\xi \in \mathbf{B}_{WDET}$	then ξ ≈»	λpλqΞX(ˇp(X)@ˇq(X))
If ξ = every	then ξ ≈»	λpλq∀X(¬[ˇp(X)@¬ˇq(X)])
If ξ = a	then ξ ≈»	λpλqΞX(ˇp(X)@ˇq(X))
If ξ = the (sg)	then ξ ≈»	λpλqΞY(∀X(ˇp(X)↔X=Y)@ˇq(Y))
If ξ = the (pl)	then ξ ≈»	λpλqΞw∀X(ˇw(X)→(ˇp(X)@ˇq(X)))

Semantic Operations

The following table correlates the semantic operations of **TMG** with the syntax rules corresponding to the translation rules invoking them.

Semantic Op.	Definition	Invoking Rule
$g_0(\theta',\eta')$	$\theta'(\hat{}\eta')$	Default
$g_1(\theta',\eta')$	$\eta'(\hat{}\theta')$	S40, S77
$g_{2,n}(\alpha',\phi')$	$\alpha'(\hat{}\lambda x_n\phi')$	S14
$g_{3,n}(\zeta',\phi')$	$\lambda x_n(\zeta'(x_n)\wedge\exists t(ab(t,\phi'))$	S30, S31
$g_{4,n}(\alpha',\theta')$	$\lambda Y\alpha'(\hat{}\lambda x_n[\theta'(Y)])$	S15, S16, S18
$g_5(\gamma',\delta')$	$\lambda e(\gamma'(e)@\delta'(e))$	S71, S72
$g_6(\gamma',\delta')$	$\lambda e(\gamma'(e)\vee\delta'(e))$	S71,S72
$g_7(\phi',\psi')$	$\lambda n(\phi'(n)@\psi'(n))$	S70
$g_8(\phi',\psi')$	$\lambda n(\phi'(n)\vee\psi'(n))$	S70
$g_9(\lambda n\lambda X[(\check{}n)\phi],\lambda n\lambda X[(\check{}n)\psi])$	$\lambda n\lambda X[(\check{}n)\hat{}\lambda e(\check{}\phi(e)@\psi(e))]$	S73
$g_{10}(\lambda n\lambda X[(\check{}n)\phi],\lambda n\lambda X[(\check{}n)\psi])$	$\lambda n\lambda X[(\check{}n)\hat{}\lambda e(\check{}\phi(e)\vee\psi(e))]$	S73
$g_{11}(\phi',\psi')$	$(\phi'\wedge\psi')$	S11
$g_{12}(\phi',\psi')$	$(\phi'\vee\psi')$	S11
$g_{13}(\alpha',\beta')$	$\lambda q(\alpha'(q)\wedge\beta'(q))$	S13
$g_{14}(\alpha',\beta')$	$\lambda q(\alpha'(q)\vee\beta'(q))$	S13
$g_{15}(\lambda p\lambda e(\phi,\psi')$	$\lambda p\lambda e\neg\phi(\hat{}\psi')$	S43, S45, S47, S49, S53, S55
$g_{16}(\lambda p\lambda e\delta(\xi),\psi')$	$\lambda p\lambda e\delta(\neg\xi)(\hat{}\psi')$	S51
$g_{17}(\phi',\psi')$	$\lambda r(\phi'(r)\vee\psi'(r))$	S22,S23
$g_{18,n}(\phi',\psi')$	$\lambda r(\phi'(\hat{}\lambda x_n(\psi'(r))))$	S24, S25
$g_{19,n}(\phi',\psi')$	$\lambda r(\neg\phi'(\hat{}\lambda x_n(\neg\psi'(r))))$	S26
$g_{20}(\delta',\phi')$	$\delta'(\hat{}\exists t(ab(t,\phi')))$	S61
$g_{21}(\delta',\phi')$	$\delta'(\hat{}\lambda Z\phi'(Z))$	S74, S75
$g_{22}(\delta',\lambda n\lambda X[(\check{}n)\hat{}\phi])$	$\lambda n\lambda X(\check{}n)\hat{}\delta'(\hat{}\lambda Z\phi(Z))$	S76

Syntactic Operation Predicates

Each syntactic rule S*n* invokes a structural operation having *n* as an integral part of its index. Structural operations are defined in terms of the following predicates.

```
/* Arg1 is a syntactic variable with index N */
/* for specified or arbitrary N             */

synvar(term([V],[Case]),N) :-
                nonvar(N),name(N,Suffix),
                varstem(Case,Stem),
                append(Stem,Suffix,Ascii),
                name(V,Ascii).

synvar(term([V],[Case]),N) :-
                name(V,Ascii),
                varstem(Case,Stem),
                append(Stem,Suffix,Ascii),
                name(N,Suffix).

/* Arg1 is a rodmanised variable with index N */

rsynvar(term([R],[C]),N) :-
                        name(N,Suffix),
                        rstem(C,Stem),
                        append(Stem,Suffix,Ascii),
                        name(R,Ascii).

/* V is leading variable with index N in Arg2 and V has case C */
/* else fail if none found                                     */

leadvar(N,[term([V],[C])|B],V,C) :- synvar(term([V],[C]),N),!.
leadvar(N,[H|T],V,C) :- leadvar(N,H,V,C),!.
leadvar(N,[H|T],V,C) :- leadvar(N,T,V,C).
leadvar(N,B,V,C) :- lform(B,[P,B1,F]),leadvar(N,B1,V,C).

/* V is the set of rodmanised variables with index N in Arg2 */

rvariables(N,[term([R],[C])|B],V) :-
                        rsynvar(term([R],[C]),N),!,
                        rvariables(N,B,V1),
                        union([R],V1,V).
rvariables(N,[H|T],V) :-
                        rvariables(N,H,V1),
                        rvariables(N,T,V2),
                        union(V1,V2,V).

rvariables(N,B,V) :- lform(B,[P,B1,F]),rvariables(N,B1,V).

rvariables(N,B,[]).
```

```
/* Arg1 is a syntactic variable with index N, Arg3 is a term, and Arg4 is a      */
/* pronoun marked with the case of Arg1 and the number and gender of Arg3 */

pform(term([V],[Case]),N,term([A],[Gen,Num,_]),term([P],[Gen,Num,Case])) :-
                                synvar(term([V],[Case]),N),
                                pronoun(P,[Gen,Num,Case]).

/* Arg3 results from rodmanising variable Arg1 with index N */

rform(term([V],[C]),N,term([R],[C])) :-
                                name(V,W),
                                freestem(C,Stem),
                                append(Stem,Suffix,W),
                                name(N,Suffix),integer(N),
                                append(Stem,[82|Suffix],W1),
                                name(R,W1).

/* R results from reflexivising accusative variable V with index N */

xform(term([V],[acc]),N,term([R],[acc])) :-
                        name(V,W),
                        freestem(acc,Stem),
                        append(Stem,Suffix,W),
                        name(N,Suffix),integer(N),
                        append(Stem,[115,101,108,102|Suffix],W1),
                        name(R,W1).

/* Y is the result of deleting the leading variable with index N in X */
/* else fail if none found                                            */

delete(N,X,Y) :- leadvar(N,X,V,C),erase(term([V],[C]),X,Y).
erase(V,[V|T],T) :- !.
erase(V,[H|T],[H1|T]) :- erase(V,H,H1),!.
erase(V,[H|T],[H|T1]) :- erase(V,T,T1).
erase(V,X,Y) :- lform(X,[P,X1,F]),erase(V,X1,Y1),lform(Y,[P,Y1,F]).

/* D1 and D2 are the binary daughters of M */

daughters(M,D1,D2) :- lform(M,[P,[D1,D2],F]).

/* W is the first word in Arg1 */

firstword([H|T],W) :- firstword(H,W).
firstword(B,W) :- lform(B,[P,B1,F]),firstword(B1,W).
firstword(W,W).

/* Substitute A for 1st occurrence of B in Arg3 leaving Arg4 else fail */
/* if no such occurrence */

sub(A,B,[B|T],[A|T]) :- !.
sub(A,B,[H|T],[H1|T]) :- sub(A,B,H,H1).
sub(A,B,[H|T],[H|T1]) :- sub(A,B,T,T1).
sub(A,B,X,Y) :- lform(X,[P,X1,F]), sub(A,B,X1,Y1), lform(Y,[P,Y1,F]).
```

```
/* Substitute A for first occurrence of a variable with index N in Arg3 */
/* leaving Arg4 else fail if no such occurrence                          */

psub(A,N,[V|T],[A|T]) :- synvar(V,N),!.
psub(A,N,[H|T],[H1|T]) :- psub(A,N,H,H1).
psub(A,N,[H|T],[H|T1]) :- psub(A,N,T,T1).
psub(A,N,X,Y) :- lform(X,[P,X1,F]),psub(A,N,X1,Y1),lform(Y,[P,Y1,F]).

/* Substitute a pronoun with number and gender of A and case of variable for */
/* each variable in Arg3 with index N leaving Arg4 */

esub(A,N,[V|T],[P|T1]) :- pform(V,N,A,P),esub(A,N,T,T1).
esub(A,N,[H|T],[H1|T1]) :- esub(A,N,H,H1),esub(A,N,T,T1).
esub(A,N,X,Y) :- lform(X,[P,X1,F]),esub(A,N,X1,Y1),lform(Y,[P,Y1,F]).
esub(A,N,X,X).

/* Substitute A for first occurrence of variable with index N in X and */
/* a suitable pronominal form for subsequent occurrences leaving Y     */
/* else fail if no occurrences.                                        */

qsub(A,N,X,Y) :-
                        psub(np,N,X,X1),
                        esub(A,N,X1,X2),
                        sub(A,np,X2,Y).

/* Arg2 is the result of rodmanising all variables in Arg1 */

rsub([V|T],[R|T1]) :- rform(V,_,R),!,rsub(T,T1).
rsub([H|T],[H1|T1]) :- rsub(H,H1),rsub(T,T1).
rsub(X,Y) :- lform(X,[P,X1,F]),rsub(X1,Y1),lform(Y,[P,Y1,F]).
rsub(X,X).

/* Given input X, Y is the result of deleting the leading variable with */
/* index N, replacing all other N-indexed variables by surface pronouns */
/* of like case and matching A in number and gender, and rodmanising    */
/* all other variables                                                  */

dsub(A,N,X,Y) :-
                        delete(N,X,X1),
                        esub(A,N,X1,X2),
                        rsub(X2,Y).

/* Y is the result of reflexivising all accusative variables in X */
/* having index N */

xsub([V|T],N,[R|T1]) :- xform(V,N,R),!,xsub(T,N,T1).
xsub([H|T],N,[H1|T1]) :- xsub(H,N,H1),xsub(T,N,T1).
xsub(X,N,Y) :- lform(X,[P,X1,F]),xsub(X1,N,Y1),lform(Y,[P,Y1,F]).
xsub(X,N,X).
```

```
/* Arg2 is Arg1 with each head verb marked as [+@@@form] */
/* else fail if no such form (@@@ ∈ {simple, s, inf, ing, en}) */

@@@form(vp([VP1,Conj,VP2],K),vp([VP3,Conj,VP4],K1)) :- !,
        (Conj=and ; Conj=or),
        @@@form(VP1,VP3),@@@form(VP2,VP4),
        join(K,'+@@@',K1).

@@@form(vp([v([Root],F)|C],K),vp([v([Form],F)|C],K1)) :-
        verbform(Form,@@@,Root),not(Form=[]),
        join(K,'+@@@',K1).

/* Arg2 is Arg1 with each head verb marked as [+past] */
/* else fail if no such form */

pastform(vp([VP1,Conj,VP2],K),vp([VP3,Conj,VP4],K1),Infl) :- !,
        (Conj=and ; Conj=or),
        pastform(VP1,VP3,Infl),pastform(VP2,VP4,Infl),
        join(K,'+past',K1).

pastform(vp([v([Root],F)|C],K),vp([v([Form],F)|C],K1),Infl) :-
        (Infl==fin(sg,past);Infl==fin(pl,past)),
        verbform(Form,Infl,Root),not(Form=[]),
        join(K,'+past',K1).

/* Arg3 is Arg1 with each first variable of index N marked accusative */

accform(term([T1,Conj,T2],K),N,term([T3,Conj,T4],K1)) :-
    (Conj=and;Conj=or),
    accform(T1,N,T3),accform(T2,N,T4),
    join(K,'+acc',K1).

accform(term([V],[nom]),N,term([V1],[acc])) :-
    synvar(term([V],[nom]),N),!,
    synvar(term([V1],[acc]),N).

accform(T,N,T).

/* Arg3 is Arg1 with each head noun marked with number of Arg2 */

decline(cn(CN,K),sg,cn(CN,K1)) :-
        join(K,'+sg',K1).

decline(cn([n([S],F)|C],K),pl,cn([n([P],F)|C],K1)) :-
        noun(S,P,F),
        join(K,'+pl',K1).
```

Syntax and Semantic Rules for TMG

Modified PTQ Rules

(S2) If $\alpha \in P_{DET}$ and $\zeta \in P_{CN}$ then $f_2(\alpha,\zeta) \in P_T$.

$f_2(\alpha,\zeta)$ = term([α,ξ],F) where number(α,N), decline(ζ,N,ξ), features(ξ,F).

(T2) If α $\approx\!\!\gg$ α' and ζ $\approx\!\!\gg$ ζ' then $f_2(\alpha,\zeta)$ $\approx\!\!\gg$ $g_0(\alpha',\zeta')$.

(S9) If $\delta \in P_{SADV}$ and $\phi \in P_t$ then $f_9(\delta,\phi) \in P_t$.

$f_9(\delta,\phi)$ = t([δ,ϕ],F) where features(ϕ,F).

(T9) If δ $\approx\!\!\gg$ δ' and ϕ $\approx\!\!\gg$ ϕ' then $f_9(\delta,\phi)$ $\approx\!\!\gg$ $g_0(\delta',\phi')$.

(S11) If ϕ and $\psi \in P_t$ and SA11 is fulfilled then $f_{11.1}(\phi,\psi)$ and $f_{11.2}(\phi,\psi) \in P_t$.

SA11: features(ϕ,F), features(ψ,F), dcl $\in$ F.

$f_{11.1}(\phi,\psi)$ = t([ϕ,and,ψ],[dcl]) and $f_{11.2}(\phi,\psi)$ = t([ϕ,or,ψ],[dcl]).

(T11) If ϕ $\approx\!\!\gg$ ϕ' and ψ $\approx\!\!\gg$ ψ' then: $f_{11.1}(\phi,\psi)$ $\approx\!\!\gg$ $g_{11}(\phi',\psi')$ and $f_{11.2}(\phi,\psi)$ $\approx\!\!\gg$ $g_{12}(\phi',\psi')$.

(S13) If α and $\beta \in P_T$ then $f_{13.1}(\alpha,\beta) \in P_T$ and $f_{13.2}(\alpha,\beta) \in P_T$.

$f_{13.1}(\alpha,\beta)$ = term([α,and,β],F) where features(α,A), number(α,N), F=(A-N)$\cup$['+pl']

and $f_{13.2}(\alpha,\beta)$ = term([α,or,β],F) where features(β,B), gender(β,G), F=(B-G)$\cup$['+whichever']

(T13) If α $\approx\!\!\gg$ α' and β $\approx\!\!\gg$ β' then $f_{13.1}(\alpha,\beta)$ $\approx\!\!\gg$ $g_{13}(\alpha',\beta')$ and $f_{13.2}(\alpha,\beta)$ $\approx\!\!\gg$ $g_{14}(\alpha',\beta')$.

(S14) If $\alpha \in P_T$ and $\phi \in P_t$ and SA14 is fulfilled then $f_{14,n}(\alpha,\phi) \in P_t$.

SA14: qsub(α,*n*,ϕ,ψ) succeeds.

$f_{14,n}(\alpha,\phi)$ = ψ such that qsub(α,*n*,ϕ,ψ).

(T14) If α $\approx\!\!\gg$ α' and ϕ $\approx\!\!\gg$ ϕ' then $f_{14,n}(\alpha,\phi)$ $\approx\!\!\gg$ $g_2(\alpha',\phi')$.

(S15) If $\alpha \in P_T$ and $\zeta \in P_{CN}$ and SA15 is fulfilled then $f_{15,n}(\alpha,\zeta) \in P_{CN}$.

SA15: qsub(α,*n*,ζ,ξ) succeeds.

$f_{15,n}(\alpha,\zeta)$ = ξ such that qsub(α,*n*,ζ,ξ).

(T15) If α $\approx\!\!\gg$ α' and ζ $\approx\!\!\gg$ ζ' then $f_{15,n}(\alpha,\zeta)$ $\approx\!\!\gg$ $g_4(\alpha',\zeta')$.

Verb Phrase Quantification Rules

(S16) If $\alpha \in P_T$ and $\delta \in P_{CLV}$ and SA16 is fulfilled then $f_{16,n}(\alpha,\delta) \in P_{CLV}$.

SA16: qsub(α,n,δ,γ) succeeds.

$f_{16,n}(\alpha,\delta) = \gamma$ such that qsub(α,n,δ,γ).

(T16) If $\alpha \Rightarrow \alpha'$ and $\delta \Rightarrow \delta'$ then $f_{16,n}(\alpha,\delta) \Rightarrow g_4(\alpha',\delta')$.

(S18) If $\alpha \in P_T$ and $\delta \in P_{CPV}$ and SA18 is fulfilled then $f_{18,n}(\alpha,\delta) \in P_{CPV}$.

SA18: qsub(α,n,δ,γ) succeeds.

$f_{18,n}(\alpha,\delta) = \gamma$ such that qsub(α,n,δ,γ).

(T18) If $\alpha \Rightarrow \alpha'$ and $\delta \Rightarrow \delta'$ then $f_{18,n}(\alpha,\delta) \Rightarrow g_4(\alpha',\delta')$.

Interrogative Rules

(S20) If α = ?yn and $\phi \in P_t$ then $f_{20.1}(\alpha,\phi)$, $f_{20.2}(\alpha,\phi)$, and $f_{20.3}(\alpha,\phi) \in P_Q$.

$f_{20.1}(\alpha,\phi) = {}_q$[whether ϕ], $f_{20.2}(\alpha,\phi) = {}_q$[whether or not ϕ], $f_{20.3}(\alpha,\phi) = {}_q$[whether ϕ or not].

(T20) If $\alpha \Rightarrow \alpha'$ and $\phi \Rightarrow \phi'$ then $f_{20.n}(\alpha,\phi) \Rightarrow g_0(\alpha',\phi') = \alpha'(\hat{}\phi')$.

(S21) If α = ? and $\phi \in P_t$ then $f_{21}(\alpha,\phi) \in P_Q$.

$f_{21}(\alpha,\phi) = {}_q[\alpha\ \phi]$.

(T21) If $\alpha \Rightarrow \alpha'$ and $\phi \Rightarrow \phi'$ then $f_{21}(\alpha,\phi) \Rightarrow g_0(\alpha',\phi') = \alpha'(\hat{}\phi')$.

(S22) If ψ_1 and $\psi_2 \in P_Q$ and SA22 is fulfilled then $f_{22}(\psi_1,\psi_2)$ P_Q.

SA22: daughters(ψ_1,?,ϕ_1), daughters(ψ_2,?,ϕ_2).

$f_{22}(\psi_1,\psi_2) = {}_q$[whether ϕ_1 or ϕ_2].

(T22) If $\psi_n \Rightarrow \psi'_n$then $f_{22}(\psi_1,\psi_2) \Rightarrow g_{17}(\psi'_1\psi'_2) = \lambda r(\psi'_1(r) \vee \psi'_2(r))$.

(S23) If ψ_1 and $\psi_2 \in P_Q$ and SA23 is fulfilled then $f_{23}(\psi_1,\psi_2) \in P_Q$.

SA23: daughters(ψ_1,whether,ϕ_1), daughters(ψ_2,?,ϕ_2).

$f_{23}(\psi_1,\psi_2) = {}_q[\psi_1$ or $\phi_2]$.

(T23) If $\psi_n \Rightarrow \psi'_n$then $f_{23}(\psi_1,\psi_2) \Rightarrow g_{17}(\psi'_1\psi'_2) = \lambda r(\psi'_1(r) \vee \psi'_2(r))$.

(S24) If $\alpha \in P_{WH}$ and $\phi \in P_Q$ and SA24 is fulfilled then $f_{24,n}(\alpha,\phi) \in P_Q$.

SA24: firstword(ϕ,?), leadvar(n,ϕ,V,C), α = wh([W],[G,N,C]).

$f_{24,n}(\alpha,\phi) = \alpha\,\xi$ where delete(n,ϕ,ψ), esub(α,n,ψ,ξ).

(T24) If α ⇝ α' and ϕ ⇝ ϕ' then $f_{24,n}(\alpha,\phi)$ ⇝ $g_{18,n}(\alpha',\phi') = \lambda r(\alpha'(\hat{}\lambda x_n(\phi'(r))))$.

(S25) If $\alpha \in P_{WH}$ and $\phi \in P_Q$ and SA25 is fulfilled then $f_{25,n}(\alpha,\phi) \in P_Q$.

SA25: firstword(ϕ,Q), Q ∉ {?, whether}, leadvar(n,ϕ,V,C), α = wh([W],[G,N,C]).

$f_{25,n}(\alpha,\phi) = \psi$ where qsub(α,n,ϕ,ψ).

(T25) If α ⇝ α' and ϕ ⇝ ϕ' then $f_{25,n}(\alpha,\phi)$ ⇝ $g_{18,n}(\alpha',\phi') = \lambda r(\alpha'(\hat{}\lambda x_n(\phi'(r))))$.

(S26) If $\alpha \in P_T$ and $\phi \in P_Q$ and SA26 is fulfilled then $f_{26,n}(\alpha,\phi) \in P_Q$.

SA26: ¬firstword(ϕ,whether), leadvar(n,ϕ,V,C).

$f_{26,n}(\alpha,\phi) = \psi$ such that qsub(α,n,ϕ,ψ).

(T26) If α ⇝ α' and ϕ ⇝ ϕ' then $f_{26,n}(\alpha,\phi)$ ⇝ $g_{19,n}(\alpha',\phi') = \lambda r(\neg\alpha'(\hat{}\lambda x_n(\neg\phi'(r))))$.

(S29) If $\alpha \in P_{WDET}$ and ζ P_{CN} then $f_{29}(\alpha,\zeta) \in P_{WH}$.

$f_{29}(\alpha,\zeta)$ = wh([α,ζ],F) where features(ζ,F).

(T29) If α ⇝ α' and ζ ⇝ ζ' then $f_{29}(\alpha,\zeta)$ ⇝ $g_0(\alpha',\zeta') = \alpha'(\hat{}\zeta')$.

Relative Clause Rules

(S30) If $\alpha \in P_{CN}$ and $\phi \in P_t$ and SA30 is fulfilled then $f_{30,n}(\alpha,\phi) \in P_{CN}$.

SA30 leadvar(n,ϕ,V,C), rvariables(n,ϕ,R), and V ∉ R.

$f_{30,n}(\alpha,\phi)$ = cn([α that ψ],[G,N,_]) given dsub(α,n,ϕ,ψ) and α = cn([β],[G,N,_]).

(T30) If α ⇝ α' and ϕ ⇝ ϕ' then $f_{30,n}(\alpha,\phi)$ ⇝ $g_{3,n}(\alpha',\phi')$.

(S31) If $\alpha \in P_{CN}$ and $\phi \in P_t$ and SA31 is fulfilled then $f_{31,n}(\alpha,\phi) \in P_{CN}$.

SA31 leadvar(n,ϕ,V,C), rvariables(n,ϕ,R), and V ∉ R.

$f_{31,n}(\alpha,\phi)$ = cn([α ω ψ],[G,N,_])

given dsub(α,n,ϕ,ψ) and α = cn([β],[G,N,_]) and relpron(ω,[G,N,C]).

(T31) If α ⇝ α' and ϕ ⇝ ϕ' then $f_{31,n}(\alpha,\phi)$ ⇝ $g_{3,n}(\alpha',\phi')$.

Subject Predicate Rule

(S40) If $\alpha \in P_T$ and $\delta \in P_{FV}$ and SA40 is fulfilled then $f_{40}(\alpha,\delta) \in P_t$.

SA40: number(α,N), number(δ,M), N=M.

$f_{40}(\alpha,\delta)$ = t([α,γ],[dcl]) where if α = he_k then xsub(δ,k,η), rsub(η,γ) else rsub(δ,γ).

(T40) If α $\approx\!\!\rightarrow$ α' and δ $\approx\!\!\rightarrow$ δ' then $f_{40}(\alpha,\delta)$ $\approx\!\!\rightarrow$ $g_1(\alpha',\delta')$.

Verb Phrase Rules

(S41) If α = present and $\delta \in P_{QIV}$ then $f_{41.1}(\alpha,\delta) \in P_{FV}$ and $f_{41.2}(\alpha,\delta) \in P_{FV}$.

$f_{41.1}(\alpha,\delta) = \xi$ where simple(δ,ξ) and $f_{41.2}(\alpha,\delta) = \xi$ where sform(δ,ξ).

(T41) If α $\approx\!\!\rightarrow$ α' and δ $\approx\!\!\rightarrow$ δ' then $f_{41.n}(\alpha,\delta)$ $\approx\!\!\rightarrow$ $g_0(\alpha',\delta')$.

(S42) If α = past and $\delta \in P_{QIV}$ then $f_{42.1}(\alpha,\delta) \in P_{FV}$ and $f_{42.2}(\alpha,\delta) \in P_{FV}$.

$f_{42.1}(\alpha,\delta) = \xi$ where pastform(δ,ξ,fin(sg,past)) and

$f_{42.2}(\alpha,\delta) = \xi$ where pastform(δ,ξ,fin(pl,past)).

(T42) If α $\approx\!\!\rightarrow$ α' and δ $\approx\!\!\rightarrow$ δ' then $f_{42.n}(\alpha,\delta)$ $\approx\!\!\rightarrow$ $g_0(\alpha',\delta')$.

(S43) If $\delta \in P_{LAUX}$ and $\gamma \in P_{LIV}$ then $f_{43}(\delta,\gamma) \in P_{QIV}$.

$f_{43}(\delta,\gamma)$ = iv([δ,not,ξ],[laux]) where infform(γ,ξ).

(T43) If δ $\approx\!\!\rightarrow$ δ' and γ $\approx\!\!\rightarrow$ γ' then $f_{43}(\delta,\gamma)$ $\approx\!\!\rightarrow$ $g_{15}(\delta',\gamma')$.

(S44) If $\delta \in P_{LAUX}$ and $\gamma \in P_{LIV}$ then $f_{44}(\delta,\gamma) \in P_{IV}$.

$f_{44}(\delta,\gamma)$ = iv([δ,ξ],[laux]) where infform(γ,ξ).

(T44) If δ $\approx\!\!\rightarrow$ δ' and γ $\approx\!\!\rightarrow$ γ' then $f_{44}(\delta,\gamma)$ $\approx\!\!\rightarrow$ $g_0(\delta',\gamma')$.

(S45) If $\delta \in P_{PAUX}$ and $\gamma \in P_{PIV}$ then $f_{45}(\delta,\gamma) \in P_{QIV}$.

$f_{45}(\delta,\gamma)$ = iv([δ,not,ξ],[paux]) .

(T45) If δ $\approx\!\!\rightarrow$ δ' and γ $\approx\!\!\rightarrow$ γ' then $f_{45}(\delta,\gamma)$ $\approx\!\!\rightarrow$ $g_{15}(\delta',\gamma')$.

(S46) If $\delta \in P_{PAUX}$ and $\gamma \in P_{PIV}$ then $f_{46}(\delta,\gamma) \in P_{IV}$.

$f_{46}(\delta,\gamma)$ = iv([δ,ξ],[paux]) .

(T46) If δ $\approx\!\!\rightarrow$ δ' and γ $\approx\!\!\rightarrow$ γ' then $f_{46}(\delta,\gamma)$ $\approx\!\!\rightarrow$ $g_0(\delta',\gamma')$.

(S47) If $\delta \in P_{TAUX}$ and $\gamma \in P_{IV}$ and SA47 is fulfilled then $f_{47}(\delta,\gamma) \in P_{QIV}$.

SA47: infform(γ,ξ) succeeds

$f_{47}(\delta,\gamma)$ = iv([δ,not,to,ξ],[taux]) where infform(γ,ξ).

(T47) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{47}(\delta,\gamma) \rightsquigarrow g_{15}(\delta',\gamma')$.

(S48) If $\delta \in P_{TAUX}$ and $\gamma \in P_{IV}$ and SA48 is fulfilled then $f_{48}(\delta,\gamma) \in P_{IV}$.

SA48: infform(γ,ξ) succeeds

$f_{48}(\delta,\gamma)$ = iv([δ,to,ξ],[taux]) where infform(γ,ξ).

(T48) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{48}(\delta,\gamma) \rightsquigarrow g_{0}(\delta',\gamma')$.

(S49) If $\delta \in P_{MAUX1}$ and $\gamma \in P_{IV}$ and SA49 is fulfilled then $f_{49}(\delta,\gamma) \in P_{QIV}$.

SA49: infform(γ,ξ) succeeds

$f_{49}(\delta,\gamma)$ = iv([δ,not,ξ],[maux1]) where infform(γ,ξ).

(T49) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{49}(\delta,\gamma) \rightsquigarrow g_{15}(\delta',\gamma')$.

(S50) If $\delta \in P_{MAUX1}$ and $\gamma \in P_{IV}$ and SA50 is fulfilled then $f_{50}(\delta,\gamma) \in P_{IV}$.

SA50: infform(γ,ξ) succeeds

$f_{50}(\delta,\gamma)$ = iv([δ,ξ],[maux1]) where infform(γ,ξ).

(T50) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{50}(\delta,\gamma) \rightsquigarrow g_{0}(\delta',\gamma')$.

(S51) If $\delta \in P_{MAUX2}$ and $\gamma \in P_{IV}$ and SA51 is fulfilled then $f_{51}(\delta,\gamma) \in P_{QIV}$.

SA51: infform(γ,ξ) succeeds

$f_{51}(\delta,\gamma)$ = iv([δ,not,ξ],[maux2]) where infform(γ,ξ).

(T51) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{51}(\delta,\gamma) \rightsquigarrow g_{16}(\delta',\gamma')$.

(S52) If $\delta \in P_{MAUX2}$ and $\gamma \in P_{IV}$ and SA52 is fulfilled then $f_{52}(\delta,\gamma) \in P_{IV}$.

SA52: infform(γ,ξ) succeeds

$f_{52}(\delta,\gamma)$ = iv([δ,ξ],[maux2]) where infform(γ,ξ).

(T52) If $\delta \rightsquigarrow \delta'$ and $\gamma \rightsquigarrow \gamma'$ then $f_{52}(\delta,\gamma) \rightsquigarrow g_{0}(\delta',\gamma')$.

(S53) If $\delta \in P_{NAUX}$ and $\gamma \in P_{IV}$ and SA53 is fulfilled then $f_{53}(\delta,\gamma) \in P_{QIV}$.

SA53: enform(γ,ξ) succeeds

$f_{53}(\delta,\gamma)$ = iv([δ,not,ξ],[naux]) where enform(γ,ξ).

(T53) If δ $\approx\!\!\gg$ δ' and γ $\approx\!\!\gg$ γ' then $f_{53}(\delta,\gamma)$ $\approx\!\!\gg$ $g_{15}(\delta',\gamma')$.

(S54) If $\delta \in P_{NAUX}$ and $\gamma \in P_{IV}$ and SA54 is fulfilled then $f_{54}(\delta,\gamma) \in P_{IV}$.

SA54: enform(γ,ξ) succeeds

$f_{54}(\delta,\gamma)$ = iv([δ,ξ],[naux]) where enform(γ,ξ).

(T54) If δ $\approx\!\!\gg$ δ' and γ $\approx\!\!\gg$ γ' then $f_{54}(\delta,\gamma)$ $\approx\!\!\gg$ $g_0(\delta',\gamma')$.

(S55) If $\delta \in P_{GAUX}$ and $\gamma \in P_{IV}$ and SA55 is fulfilled then $f_{55}(\delta,\gamma) \in P_{QIV}$.

SA55: ingform(γ,ξ) succeeds

$f_{55}(\delta,\gamma)$ = iv([δ,not,ξ],[gaux]) where ingform(γ,ξ).

(T55) If δ $\approx\!\!\gg$ δ' and γ $\approx\!\!\gg$ γ' then $f_{55}(\delta,\gamma)$ $\approx\!\!\gg$ $g_{15}(\delta',\gamma')$.

(S56) If $\delta \in P_{GAUX}$ and $\gamma \in P_{IV}$ and SA56 is fulfilled then $f_{56}(\delta,\gamma) \in P_{IV}$.

SA56: ingform(γ,ξ) succeeds

$f_{56}(\delta,\gamma)$ = iv([δ,ξ],[gaux]) where ingform(γ,ξ).

(T56) If δ $\approx\!\!\gg$ δ' and γ $\approx\!\!\gg$ γ' then $f_{56}(\delta,\gamma)$ $\approx\!\!\gg$ $g_0(\delta',\gamma')$.

Passivisation Rules

(S57) If $\delta \in P_{PTV}$ and $\beta \in P_T$ and SA57 is fulfilled then $f_{57}(\delta,\beta) \in P_{PIV}$.

SA57: $\beta \neq$ agent

$f_{57}(\delta,\beta)$ = piv([δ,by,ξ],F) where accform(β,ξ,_) and features(δ,F).

(T57) If δ $\approx\!\!\gg$ δ' and β $\approx\!\!\gg$ β' then $f_{57}(\delta,\beta)$ $\approx\!\!\gg$ $g_0(\delta',\beta')$.

(S58) If $\delta \in P_{PTV}$ and $\beta \in P_T$ and SA58 is fulfilled then $f_{58}(\delta,\beta) \in P_{PIV}$.

SA58: β = agent

$f_{58}(\delta,\beta)$ = piv([δ],F) where features(δ,F).

(T58) If δ $\approx\!\!\gg$ δ' and β $\approx\!\!\gg$ β' then $f_{58}(\delta,\beta)$ $\approx\!\!\gg$ $g_0(\delta',\beta')$.

(S59) If α = passive and $\delta \in P_{TV}$ then $f_{59}(\alpha,\delta) \in P_{PTV}$.

$f_{59}(\alpha,\delta)$ = ptv([ξ],F) where enform(δ,ξ) and features(δ,F).

(T59) If δ $\approx\!\!\rightarrow$ δ' and β $\approx\!\!\rightarrow$ β' then $f_{59}(\delta,\beta)$ $\approx\!\!\rightarrow$ $g_0(\delta',\beta')$.

Rules of Complementation

(S60) If $\delta \in P_{TV}$ and $\beta \in P_T$ then $f_{60}(\delta,\beta) \in P_{LIV}$.

$f_{60}(\delta,\beta)$ = iv([δ,ξ],[liv]) where accform(β,ξ,_).

(T60) If δ $\approx\!\!\rightarrow$ δ' and β $\approx\!\!\rightarrow$ β' then $f_{60}(\delta,\beta)$ $\approx\!\!\rightarrow$ $g_0(\delta',\beta')$.

(S61) If $\delta \in P_{SCVERB}$ and $\phi \in P_t$ then $f_{61}(\delta,\phi) \in P_{LIV}$.

$f_{61}(\delta,\phi)$ = iv([δ,ϕ],[liv]).

(T61) If δ $\approx\!\!\rightarrow$ δ' and ϕ $\approx\!\!\rightarrow$ ϕ' then $f_{61}(\delta,\phi)$ $\approx\!\!\rightarrow$ $g_{20}(\delta',\phi')$.

(S62) If $\delta \in P_{ICVERB}$ and $\gamma \in P_{IV}$ then $f_{62}(\delta,\gamma) \in P_{LIV}$.

$f_{62}(\delta,\gamma)$ = iv([δ,γ],[liv]).

(T62) If δ $\approx\!\!\rightarrow$ δ' and γ $\approx\!\!\rightarrow$ γ' then $f_{62}(\delta,\gamma)$ $\approx\!\!\rightarrow$ $f_0(\delta',\gamma')$.

(S63) If $\delta \in P_{QCVERB}$ and $\phi \in P_Q$ and SA63 is fulfilled then $f_{63}(\delta,\phi) \in P_{LIV}$.

SA63: ϕ does not begin with "?".

$f_{63}(\delta,\phi) = \delta\phi$.

(T63) If δ $\approx\!\!\rightarrow$ δ' and ϕ $\approx\!\!\rightarrow$ ϕ' then $f_{63}(\delta,\phi)$ $\approx\!\!\rightarrow$ $g_0(\delta',\phi') = \delta'(\hat{}\phi')$.

(S64) If $\delta \in P_{COP}$ and $\beta \in P_T$ then $f_{64}(\delta,\beta) \in P_{LIV}$.

$f_{64}(\delta,\beta)$ = iv([δ,β],[iv]).

(T64) If δ $\approx\!\!\rightarrow$ δ' and β $\approx\!\!\rightarrow$ β' then $f_{64}(\delta,\beta)$ $\approx\!\!\rightarrow$ $f_0(\delta',\beta')$.

Verb Phrase Conjunction Rules

(S70) If $\gamma, \delta \in P_{FV}$ then $f_{70.1}(\gamma,\delta) \in P_{FV}$ and $f_{70.2}(\gamma,\delta) \in P_{FV}$.

$f_{70.1}(\gamma,\delta)$ = fv([γ,and,δ],[fv]) and $f_{70.2}(\gamma,\delta)$ = fv([γ,or,δ],[fv]).

(T70) If γ $\approx\!\!\rightarrow$ γ' and δ $\approx\!\!\rightarrow$ δ' then $f_{70.1}(\gamma,\delta)$ $\approx\!\!\rightarrow$ $g_7(\gamma',\delta')$ and $f_{70.2}(\gamma,\delta)$ $\approx\!\!\rightarrow$ $g_8(\gamma',\delta')$.

(S71) If $\gamma, \delta \in P_{LIV}$ then $f_{71.1}(\gamma,\delta) \in P_{CLV}$ and $f_{71.2}(\gamma,\delta) \in P_{CLV}$.

$f_{71.1}(\gamma,\delta)$ = iv([γ,and,δ],[clv]) and $f_{71.2}(\gamma,\delta)$ = iv([γ,or,δ],[clv]).

(T71) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{71.1}(\gamma,\delta) \approx\!\!\gg g_5(\gamma',\delta')$ and $f_{71.2}(\gamma,\delta) \approx\!\!\gg g_6(\gamma',\delta')$.

(S72) If $\gamma, \delta \in P_{PIV}$ then $f_{72.1}(\gamma,\delta) \in P_{CPV}$ and $f_{72.2}(\gamma,\delta) \in P_{CPV}$.

$f_{72.1}(\gamma,\delta)$ = iv([γ,and,δ],[cpv]) and $f_{72.2}(\gamma,\delta)$ = iv([γ,or,δ],[cpv]).

(T72) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{72.1}(\gamma,\delta) \approx\!\!\gg g_5(\gamma',\delta')$ and $f_{72.2}(\gamma,\delta) \approx\!\!\gg g_6(\gamma',\delta')$.

(S73) If $\gamma, \delta \in P_{PTV}$ then $f_{73.1}(\gamma,\delta) \in P_{PTV}$ and $f_{73.2}(\gamma,\delta) \in P_{PTV}$.

$f_{73.1}(\gamma,\delta)$ = iv([γ,and,δ],[ptv]) and $f_{73.2}(\gamma,\delta)$ = iv([γ,or,δ],[ptv]).

(T73) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{73.1}(\gamma,\delta) \approx\!\!\gg g_9(\gamma',\delta')$ and $f_{73.2}(\gamma,\delta) \approx\!\!\gg g_{10}(\gamma',\delta')$.

Rules of Adverbial Qualification

(S74) If $\gamma \in P_{IAV}$ and $\delta \in P_{LIV}$ then $f_{74}(\gamma,\delta) \in P_{LIV}$.

$f_{74}(\gamma,\delta)$ = iv([γ,δ],[liv]).

(T74) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{74}(\gamma,\delta) \approx\!\!\gg g_{21}(\gamma',\delta')$.

(S75) If $\gamma \in P_{IAV}$ and $\delta \in P_{PIV}$ then $f_{75}(\gamma,\delta) \in P_{PIV}$.

$f_{75}(\gamma,\delta)$ = piv([γ,δ],[piv]).

(T75) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{75}(\gamma,\delta) \approx\!\!\gg g_{21}(\gamma',\delta')$.

(S76) If $\gamma \in P_{IAV}$ and $\delta \in P_{PTV}$ then $f_{76}(\gamma,\delta) \in P_{PTV}$.

$f_{76}(\gamma,\delta)$ = ptv([γ,δ],[ptv]).

(T76) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{76}(\gamma,\delta) \approx\!\!\gg g_{22}(\gamma',\delta')$.

(S77) If $\gamma \in P_{FV}$ and $\delta \in P_{TMADV}$ then $f_{77}(\gamma,\delta) \in P_{FV}$.

$f_{77}(\gamma,\delta)$ = iv([γ,δ],F) where features(γ,F).

(T77) If $\gamma \approx\!\!\gg \gamma'$ and $\delta \approx\!\!\gg \delta'$ then $f_{77}(\gamma,\delta) \approx\!\!\gg g_1(\gamma',\delta')$.

(S78) If $\delta \in P_{PREP}$ and $\beta \in P_T$ then $f_{78}(\delta,\beta) \in P_{IAV}$.

$f_{78}(\delta,\beta)$ = iav([δ,ξ],[]) where accform(β,ξ,_).

(T78) If $\delta \approx\!\!\gg \delta'$ and $\beta \approx\!\!\gg \beta'$ then $f_{78}(\delta,\beta) \approx\!\!\gg g_0(\delta',\beta')$.

APPENDIX B: TMDCG

```
/* ########################################## */
/* ##### MONTAGUE GRAMMAR PARSER #### */
/* ########################################## */


:-op(500,xfy,:).
:-op(500,fy,#).
:-op(500,fy,^).
:-op(500,fy,`). /* ` is used as hacek accent */
:-op(500,fy,~).
:-op(750,xfy,@). /* Cresswellian "and" */
:-op(800,xfy,&).
:-op(800,xfy,v).
:-op(900,xfx,=>).
:-op(900,xfx,<=>).

/* WELL FORMED SUB-STRING TABLE MAINTENANCE */

environment([],N,node(#n,[N,slot])) :- !.

environment(Env,Node,New) :- plug(Env,Node,New).

plug(node(#n,[N,slot]),Node,
    node(#n,[N,node(#n,[Node,slot])])) :- !.

plug(node(#n,[N1,N2]),Node,node(#n,[N1,N3])) :- plug(N2,Node,N3).

catalog([],Env,Node) :- !.

catalog(_,Env,Node) :-
            \+ (wfst([Env,Node])),
            asserta(wfst([Env,Node])),!.


/* LEXICAL SCANNER */

scan(Word,[Word|Balance],Balance) :- atom(Word),!.

scan([],Balance,Balance) :- !.

scan([Word|Others],[Word|Remainder],Balance) :-
            scan(Others,Remainder,Balance),!.

/* LEFT RECURSION TERMINATOR */

recurse(G,N,N).
```

```
/* ##############      AUGMENTED      ############## */
/* ######## FRIEDMAN-WARREN ALGORITHM ######## */

quantify(cn,Node,Node,L,L,M,M,F,F,Sk,SR) :- !. /* switch off S15 */

quantify(_,Node,Node,L,L,M,M,[],[],Sk,SR) :- !.

quantify(T,node(N0,F0,L0,D0),
    node(#R,F0,Lx,[node(N1,F1,L1,D1),Node1]),
    La,L,Ma,M,
    [B|FVB],FVB1,Sk,SR) :-
        B=bind(He,Inx,[Md,_,_,_],node(N1,F1,L1,D1)),
        variants(Md,T,R,Inx,L,M,Lx),
        quantify(T,node(N0,F0,L0,D0),Node1,La,Lb,Ma,Mb,
            FVB,FVB1,[B|Sk],SR),
        eligible(B,FVB1,Sk,SR),
        name(He,[104,101|Suffix]),
        makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,Subj,Obj,Ref,F1),
        edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L1,Lb,L),
        edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L1,Mb,M).

quantify(T,Node,Node1,L,L1,M,M1,[H|FVB],[H|FVB1],Sk,SR) :-
        quantify(T,Node,Node1,L,L1,M,M1,FVB,FVB1,[H|Sk],SR).

variants(dcl,s,14:(2:Inx),Inx,L,_,L).
variants(dcl,cn,15:(4:Inx),Inx,L,_,L).
variants(dcl,[lex,_,_],16:(4:Inx),Inx,_,M,M).
variants(dcl,[piv],18:(4:Inx),Inx,_,M,M).
variants(q,q,25:(18:Inx),Inx,L,_,L).
variants(dcl,q,26:(19:Inx),Inx,L,_,L).

replace(node(N0,[G,Num],L0,D0),
    [dcl,G,C,Num],
    node(#1:'=',[G,Num],[He],[sense(He,[v])]),
    L0,[Var],FVB,FVB,SR,SR1) :-
        member(bind(He,Inx,[dcl,_,_,_],node(N0,[G,Num],L0,D0)),SR),
        \+ dominated(He,node(N0,[G,Num],L0,D0)),!,
        extract(bind(He,Inx,[dcl,_,_,_],node(N0,[G,Num],L0,D0)),SR,SR1),
        name(Inx,Suffix),
        copyvar([G,C,Num,p],Var,_,Suffix).

replace(Node,[dcl,G,C,Num],Node,NP,NP,FVB,FVB,SR,SR).

replace(Node,[Md,G,C,Num],node(#1:'=',[G,Num],[He],[sense(He,[v])]),
    NP,[Var],
    FVB,FVB1,SR,SR) :-
        gensym(he,He,Inx,Suffix),!,
        ((atom(C),copyvar([G,C,Num,p],Var,_,Suffix));
         (var(C),Var=trace)),
        currentnum(lev,Lev),
        ((Md=q,Lev1 is Lev+1) ; Lev1=Lev),
        mix(bind(He,Inx,[Md,C,_,Lev1],Node),FVB,FVB1).
```

```
eligible(B,FVB,Sk,SR) :-
        \+ (member(B,SR)),
        \+ (embedded(B,Sk)),
        \+ (embedded(B,FVB)).

embedded(bind(He,Inx,Mx,Node),FVB) :-
        getnext(bind(H,I,M,node(N,F,L,[D1,D2])),FVB),
        ((nonvar(D1),dominated(He,D1));(nonvar(D2),dominated(He,D2))).

dominated(H,node(N,F,[H],D)) :- !.

dominated(H,node(N,F,L,[D1,D2])) :-
        (dominated(H,D1);dominated(H,D2)).
```

```
/* ######## TMDCG PARSER ######## */


/* SENTENCE RULES */

sentence(Node,[Md],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        getnum(lev,Lev),
        sentence0(S1,[Md],E,La,Ia,Ib,H,FVBa,SA,SRa,SRb),
        catalog(Ib,E,S1),
        recurse(g([Md],E,H,SA),
                s(S1,La,Ib,FVBa,SRb),
                s(Node,L,Iz,FVB,SRz)).

recurse(g([Md],E,H,SA),
s(S1,La,Ib,FVBa,SRb),
s(Node,L,Iz,FVB,SRz)) :-
    scan(Conj,Ib,Ic),
    choose(s,Rule,Conj),
    join(FVBa,SA,SAa),
    environment(E,S1,E1),
    sentence(S2,[Md],E1,Lb,Ic,Id,H,FVBb,SAa,SRb,SRc),
    join(La,[Conj|Lb],Lc),
    mix(FVBa,FVBb,FVBc),
    quantify(s,node(Rule,[Md],Lc,[S1,S2]),
            Node0,Lc,Ld,[],[],FVBc,FVBq,[],SRc),
    catalog(Id,E,Node0),
    recurse(g([Md],E,H,SAa),
            s(Node0,Ld,Id,FVBq,SRc),
            s(Node,L,Iz,FVB,SRz)).

sentence0(Node,[Md],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        sadverb(SADV,La,Ia,Ib),!,
        environment(E,SADV,E1),
        sentence(S,[Md],E1,Lb,Ib,Iz,H,FVBa,SA,SRa,SRz),
        join(La,Lb,Lc),
        quantify(s,node(#9:0,[Md],Lc,[SADV,S]),
                Node,Lc,L,[],[],FVBa,FVB,[],SRz).

sentence0(Node,[Md],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        nounphrase(NP,[Md,G,nom,Num],E,La,Ia,[W|Ib],H,FVBa,
                SA,SRa,SRb),
        \+ ( (W==and);(W==or) ),
        join(FVBa,SA,SAa),
        environment(E,NP,E1),
        verbphrase(FVP,[fin(Num,Tense)],E1,Lb,[W|Ib],Iz,
                H,FVBb,SAa,SRb,SRz),
        join(La,Lb,Lc),
        mix(FVBa,FVBb,FVBc),
        quantify(s,node(#40:1,[Md],Lc,[NP,FVP]),
                Node,Lc,L,[],[],FVBc,FVB,[],SRz).

sadverb(node(#1:'=',[sadv],[SADV],[sense(SADV,[sadv])]),
        [SADV],Ia,Iz) :-
        scan(SADV,Ia,Iz),
        sadverb(SADV).
```

```
/* EMBEDDED QUESTION RULES */

question(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        choicequestion(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz),
        catalog(Iz,E,Q).

question(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        polarquestion(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz),
        catalog(Iz,E,Q).

question(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        searchquestion(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz),
        catalog(Iz,E,Q).

choicequestion(node(#22:17,[q],L,[C1,CR]),[q],E,L,Ia,Iz,
        H,FVB,SA,SRa,SRz) :-
         scan(whether,Ia,Ib),
         proto(C1,F1,E,L1,['?'|Ib],[or|Ic],H,FVBa,SA,SRa,SRb),
         catalog(Ic,E,C1),
         join(FVBa,SA,SAa),
         environment(E,C1,E1),
         alterquestion(CR,F2,E1,L2,[or|Ic],Iz,H,FVBb,SAa,SRb,SRz),
         mix(FVBa,FVBb,FVB),
         join([whether|L1],L2,L).

alterquestion(Node,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
         scan(or,Ia,Ib),
         proto(C1,F1,E,La,['?'|Ib],Ic,H,FVBa,SA,SRa,SRb),
         join([or],La,Lb),
         catalog(Ic,E,C1),
          recurse(g([q],E,H,SA),
                aq(C1,Lb,Ic,FVBa,SRb),
                aq(Node,L,Iz,FVB,SRz)).

recurse(g([q],E,H,SA),
        aq(C1,Lb,Ic,FVBa,SRb),
        aq(Node,L,Iz,FVB,SRz)) :-
           scan(or,Ic,Id),
           join(FVBa,SA,SAa),
           environment(E,C1,E1),
           proto(C2,Ft,E1,Lc,['?'|Id],Ie,H,FVBb,SAa,SRb,SRc),
           mix(FVBa,FVBb,FVBc),
           join(Lb,[or|Lc],Ld),
           catalog(Ie,E,node(#23:17,[q],L3,[C1,C2])),
           recurse(g([q],E,H,SAa),
                aq(node(#23:17,[q],L3,[C1,C2]),Ld,Ie,FVBc,SRc),
                aq(Node,L,Iz,FVB,SRz)).

proto(node(#21:0,[proto],['?'|L],[node(#0:'=',[query],['?'],
        [sense('?',[query])]),S]),[proto],E,L,['?'|Ib],Iz,
        H,FVB,SA,SRa,SRz) :-
        sentence(S,[dcl],E,L,Ib,Iz,H,FVB,SA,SRa,SRz).
```

```
polarquestion(node(#20:0,['?yn'],L,[node(#0:'=',[yn],['?yn'],
                        [sense('?yn',[yn])])],S]),
        [q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        qprefix(L0,Flag,Ia,Ib),
        sentence(S,[dcl],E,L1,Ib,Ic,H,FVB,SA,SRa,SRz),
        join(L0,L1,L2),
        qsuffix(L3,Flag,Ic,Iz),
        join(L2,L3,L).

qprefix([whether],0,Ia,Iz) :- scan(whether,Ia,Iz).

qprefix([whether,or,not],1,Ia,Iz) :- scan([whether,or,not],Ia,Iz).

qsuffix([or,not],0,Ia,Iz) :- scan([or,not],Ia,Iz).

qsuffix([],_,Ia,Ia).

searchquestion(Q,[q],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        nounphrase(node(N,F,[He],D),[q,G,C,Num],
              E,L0,Ia,Ib,H,FVBa,SA,SRa,SRb),
        FVBa=[bind(He,Inx,[q,_,_,_],node(N1,F1,L1,D1))],
        environment(E,node(N,F,[He],D),E1),
        join(FVBa,SA,SAa),
        join([hold([G,C,Num],He)],H,Hold),
        sentence(S,T,E1,L2,Ib,Iz,Hold,
             FVBb,SAa,SRb,SRz),
        name(He,[104,101|Suffix]),
        makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,Subj,Obj,Ref,[G,Num]),
        editline(Nom,Acc,Rf,Subj,Obj,Ref,L2,L3),
        join(L1,L3,L4),
    quantify(q,node(#24:(18:Inx),[q],L4,
                [node(N1,F1,L1,D1),
                 node(#21:0,[proto],['?'|L2],
                     [node(#0:'=',[query],['?'],
                         [sense('?',[query])])],S])]),
            Q,L4,L,[],[],FVBb,FVB,[],SRz).
```

```
/* NOUN PHRASE RULES */

nounphrase(Node,[Md,G,C,Num],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        nounphrase0(NP1,[Md,G1,C,Num1],E,La,Ia,Ib,
                H,FVBa,SA,SRa,SRb),
        catalog(Ib,E,NP1),
        recurse(g([Md,C],E,H,SA),
                np(NP1,G1,Num1,La,Ib,FVBa,SRb),
                np(Node,G,Num,L,Iz,FVB,SRz)).

recurse(g([Md,C],E,H,SA),
        np(NP1,G1,Num1,La,Ib,FVBa,SRb),
        np(Node,G,Num,L,Iz,FVB,SRz)) :-
        scan(Conj,Ib,Ic),
        choose(np,Rule,Conj,G1,G3,Num1,Num2,Num3),
        join(FVBa,SA,SAa),
        environment(E,NP1,E1),
        nounphrase(NP2,[Md,G2,C,Num2],E1,Lb,Ic,Id,H,FVBb,SAa,SRb,SRc),
        join(La,[Conj|Lb],Lc),
        mix(FVBa,FVBb,FVBc),
        replace(node(Rule,[G3,Num3],Lc,[NP1,NP2]),
            [Md,G3,C,Num3],Node0,Lc,Ld,FVBc,FVBd,SRc,SRd),
        catalog(Id,E,Node0),
        recurse(g([Md,C],E,H,SAa),
                np(Node0,G3,Num3,Ld,Id,FVBd,SRd),
                np(Node,G,Num,L,Iz,FVB,SRz)).

nounphrase0(node(#1:'=',[G,Num],[He],[sense(He,[v])]),
        [dcl,G,C,Num],E,[Var],Ia,Iz,H,FVB,SA,SRa,SRz) :-
        scan(P,Ia,Iz),
        pronoun(P,[G,C,Num,K]),
        crossref([G,C,Num,K],He,Var,FVB,SA,SRa),
        join(FVB,SRa,SRz).

nounphrase0(node(#1:'=',[G,Num],[He],[sense(He,[v])]),
        [q,G,C,Num],E,[Var],Ia,Iz,H,FVB,SA,SRa,SRz) :-
        scan(W,Ia,Iz),
        wh_pronoun(W,[G,C,Num]),
        gensym(he,He,Inx,Suffix),
        copyvar([G,C,Num,p],Var,_,Suffix),
        currentnum(lev,Lev),Lev1 is Lev+1,
        FVB=[bind(He,Inx,[q,C,_,Lev1],node(#1:'=',[G,Num],[W],
            [sense(W,[whpron])]))].

nounphrase0(Node,[dcl,G,C,Num],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        scan(Pn,Ia,Iz),
        proper(Pn,[G,Num]),
        replace(node(#1:'=',[G,Num],[Pn],[sense(Pn,[pn])]),
            [dcl,G,C,Num],Node,[Pn],L,[],FVB,SRa,SRz).
```

```
nounphrase0(Node,[Md,G,C,Num],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        determiner(node(N0,[Md,Num],L0,D0),R,L0,Ia,Ib),
        environment(E,node(N0,[Md,Num],L0,D0),E1),
        common(CN,[G,Num],E1,Lb,Ib,Iz,FVBa,SA,SRa,SRb),
        consonant(L0,Lb),
        join(L0,Lb,Lc),
        replace(node(#R:0,[G,Num],Lc,
                [node(N0,[Md,Num],L0,D0),CN]),
            [Md,G,C,Num],Node,Lc,L,FVBa,FVB,SRb,SRz).

nounphrase0(node(#1:'=',[G,Num],[He],[sense(He,[v])]),
      [dcl,G,C,Num],E,['(',Var,')'],Ia,Ia,Hold,[],SA,SRa,SRa) :-
        nonvar(Hold),getnext(hold([G,C,Num],He),Hold),
        name(He,[104,101|Suffix]),
        copyvar([G,C,Num,p],Var,_,Suffix).

determiner(node(#1:'=',[Md,Num],[DET],[sense(DET,[Md,Num])]),
        R,[DET],Ia,Iz) :-
        scan(DET,Ia,Iz),
        determiner(DET,R,[Md,Num]).

crossref([G,C,Num,K],He,Var,[],SA,SRa) :-
        currentnum(lev,Lev),
        getnext(bind(He,Inx,[Md,C1,Cn,Lev1],node(N,[G,Num],L,D)),SA),
        ((C=nom,!,setnum(lev,Lev1),Cn=nom);C=acc),
        ((K=r,!,Lev=Lev1,(C1=nom;Cn==nom))
         ;(Lev=Lev1,!,C=acc,C1=acc,\+ Cn==nom)
         ;K=p),
        name(Inx,Suffix),
        copyvar([G,C,Num,K],Var,_,Suffix).

crossref([G,C,Num,p],He,Var,[],SA,SRa) :-
        getnext(bind(He,Inx,Mx,node(N,[G,Num],L,D)),SRa),
        name(Inx,Suffix),
        copyvar([G,C,Num,p],_,Var,Suffix).

crossref([G,C,Num,p],He,Var,[bind(He,Inx,Mx,node(N,[G,Num],L,D))],SA,SRa) :-
        not(free(bind(He,Inx,Mx,node(N,[G,Num],L,D)),SA,SRa)),
        gensym(he,He,Inx,Suffix),
        copyvar([G,C,Num,p],_,Var,Suffix),!.

free(B,SA,SR) :-
        (member(B,SA);member(B,SR)).

consonant(an,[H|_]) :- !,name(H,[H1|_]),
            member(H1,[97,101,105,111]).

consonant(a,[H|_]) :- !,name(H,[H1|_]),
            \+ member(H1,[97,101,105,111]).

consonant(_,_).
```

```
/* COMMON NOUN PHRASE RULES */

common(Node,Ft,E,L,Ia,Iz,FVB,SA,SRa,SRz) :-
        common0(CN,Ft,E,La,Ia,Ib,FVBa,SA,SRa,SRb),
        catalog(Ib,E,CN),
        recurse(g(Ft,E,SA),
            cn(CN,La,Ib,FVBa,SRb),
            cn(Node,L,Iz,FVB,SRz)).

recurse(g([G,Num],E,SA),
      cn(CN,La,Ib,FVBa,SRb),
      cn(Node,L,Iz,FVB,SRz)) :-
        scan(Rel,Ib,Ic),
        relpronoun(Rel,R,[G,C]),
        gensym(he,He,Inx,Suffix),
        currentnum(lev,Lev),Lev1 is Lev+1,
        join([bind(He,Inx,[dcl,C,_,Lev1],CN)|FVBa],SA,SAa),
        environment(E,CN,E1),
        sentence(S,[dcl],E1,Lb,Ic,Id,[hold([G,C,Num],He)],FVBb,
            SAa,SRb,SRc),
        eligible(bind(He,_,_,_),FVBb,[],[]),
        dominated(He,S),
        makevars(Nom,_,Acc,_,Rf,Suffix,Subj,Obj,Ref,[G,Num]),
        editline(Nom,Acc,Rf,Subj,Obj,Ref,Lb,Lc),
        join(La,[Rel|Lc],Ld),
        mix(FVBa,FVBb,FVBc),
        quantify(cn,node(#R:(3:Inx),[G,Num],Ld,[CN,S]),
            Node0,Ld,Le,[],[],FVBc,FVBd,[],SRc),
        catalog(Id,E,Node0),
        recurse(g([G,Num],E,SAa),
            cn(Node0,Le,Id,FVBd,SRc),
            cn(Node,L,Iz,FVB,SRz)).

common0(node(#1:'=',Ft,[Sg],[sense(Sg,[cn])]),Ft,E,[CN],Ia,Iz,[],
        SA,SR,SR) :-
        scan(CN,Ia,Iz),
        decline(CN,Ft,Sg).
```

```
/* VERB PHRASE RULES */

verbphrase(FVP,[fin(Num,Time)],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        finitevp(FVP1,[fin(Num,Time)],
                E,La,Ia,Ib,H,FVBa,SA,SRa,SRb),
        catalog(Ib,E,FVP1),
        recurse(g([fin(Num,_)],E,H,SA),
            vp(FVP1,La,Ib,FVBa,SRb),
            vp(FVP,L,Iz,FVB,SRz)).

finitevp(node(R,[fin(Num,Time)],L,[node(N0,F0,L0,D0),VP]),
        [fin(Num,Time)],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        tense(node(N0,F0,L0,D0),[K,fin(Num,Tense),C],Root,Ia),
        direction(Tense,Root,Time),
        qiverbphrase(VP,[K,fin(Num,Tense),C],E,L,Ia,Iz,
            H,FVB,SA,SRa,SRz),
        choose(R,L0,Tense).

qiverbphrase(node(#43:15,[qiv],[L0,not|Lb],[node(N0,F0,[L0],D0),VP]),
        [laux,Infl,inf],
        E,[V,not|Lb],[V,not|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[laux,Infl,inf],45,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[lex,inf,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

qiverbphrase(node(#45:15,[qiv],[L0,not|Lb],[node(N0,F0,[L0],D0),VP]),
        [paux,Infl,en],
        E,[V,not|Lb],[V,not|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[paux,Infl,en],47,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[piv],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

qiverbphrase(node(#N:15,[qiv],[L0,not,to|Lb],[node(N0,F0,[L0],D0),VP]),
        [taux,Infl,inf],
        E,[V,not,to|Lb],[V,not,to,Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[taux,Infl,inf],M,V),N is M-1,
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[K,inf,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

qiverbphrase(node(#N:Tn,[qiv],[L0,not|Lb],[node(N0,F0,[L0],D0),VP]),
        [aux,Infl,C],E,[V,not|Lb],[V,not|Ib],Iz,
        H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[aux,Infl,C],M,V),N is M-1,
        ((N=51,Tn=16);Tn=15),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[K,C,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

qiverbphrase(N,F,E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        verbphrase(N,F,E,L,Ia,Iz,H,FVB,SA,SRa,SRz).
```

```
verbphrase(node(#44:0,[ivp],[L0|Lb],[node(N0,F0,[L0],D0),VP]),
        [laux,Infl,inf],
        E,[V|Lb],[V|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[laux,Infl,inf],N,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[lex,inf,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

verbphrase(node(#46:0,[ivp],[L0|Lb],[node(N0,F0,[L0],D0),VP]),
        [paux,Infl,en],
        E,[V|Lb],[V|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[paux,Infl,en],N,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[piv],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

verbphrase(node(#N:0,[ivp],[L0,to|Lb],[node(N0,F0,[L0],D0),VP]),
        [taux,Infl,inf],
        E,[V,to|Lb],[V,to|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[taux,Infl,inf],N,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[K,inf,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

verbphrase(node(#N:0,[ivp],[L0|Lb],[node(N0,F0,[L0],D0),VP]),
        [aux,Infl,C],E,[V|Lb],[V|Ib],Iz,H,FVB,SA,SRa,SRz) :-
        auxverb(node(N0,F0,[L0],D0),[aux,Infl,C],N,V),
        environment(E,node(N0,F0,[L0],D0),E1),
        verbphrase(VP,[K,C,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz).

verbphrase(PIV,[piv],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        piverbphrase(PIV1,[piv],E,La,Ia,Ib,H,FVBa,SA,SRa,SRb),
        catalog(Ib,E,PIV1),
        recurse(g([piv],E,H,SA),
            vp(PIV1,La,Ib,FVBa,SRb),
            vp(PIV,L,Iz,FVB,SRz)).

verbphrase(LVP,[lex,Infl,C],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        lexverbphrase(LVP1,[lex,Infl,C],E,La,Ia,Ib,
            H,FVBa,SA,SRa,SRb),
        catalog(Ib,E,LVP1),
        recurse(g([lex,Infl,C1],E,H,SA),
            vp(LVP1,La,Ib,FVBa,SRb),
            vp(LVP,L,Iz,FVB,SRz)).

verbphrase(IVP,[cop,Infl,trans],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        copulative(CVP,[cop,Infl,trans],E,La,Ia,Ib,H,
            FVBa,SA,SRa,SRb),
        (K=cop;K=lex),
        catalog(Ib,E,CVP),
        recurse(g([K,Infl,C1],E,H,SA),
            vp(CVP,La,Ib,FVBa,SRb),
            vp(IVP,L,Iz,FVB,SRz)).

verbphrase(PTV,[ptv],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        ptverbphrase(PTV,[ptv],E,L,Ia,Iz,H,FVB,SA,SRa,SRz).
```

```
piverbphrase(node(N,[piv],L,[PTV,node(N0,F0,L0,D0)]),[piv],
        E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        ptverbphrase(PTV,[ptv],E,La,Ia,Ib,H,FVBa,SA,SRa,SRb),
        environment(E,PTV,E1),
        join(FVBa,SA,SAa),
        agent(node(N0,F0,L0,D0),F1,E1,Lb,Ib,Iz,H,FVB,SAa,SRb,SRz),
        ((N0==(#0:'=') , N=(#58:0)) ; N=(#57:0)),
        join(La,Lb,L).

ptverbphrase(PTV,[ptv],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        ptverb(PTV1,[lex,en,trans],La,Ia,Ib),
        catalog(Ib,E,PTV1),
        recurse(g([ptv],E,H,SA),
            vp(PTV1,La,Ib,[],SRa),
            vp(PTV,L,Iz,FVB,SRz)).

lexverbphrase(VP,[lex,Infl,intrans],E,L,Ia,Iz,H,[],SA,SR,SR) :-
        lexverb(VP,[lex,Infl,intrans],L,Ia,Iz).

lexverbphrase(node(#60:0,[liv],L1,[node(N0,F0,L0,D0),NP]),
        [lex,Infl,trans],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        lexverb(node(N0,F0,L0,D0),[lex,Infl,trans],La,Ia,Ib),
        environment(E,node(N0,F0,L0,D0),E1),
        nounphrase(NP,[Md,G,acc,Num],E1,Lb,Ib,Iz,H,FVB,
            SA,SRa,SRz),
        join(L0,Lb,L1),
        join(La,Lb,L).

lexverbphrase(node(#61:20,[liv],L1,[node(N0,F0,L0,D0),S]),
        [lex,Infl,scomp],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        lexverb(node(N0,F0,L0,D0),[lex,Infl,scomp],
            La,Ia,[that|Ib]),
        environment(E,node(N0,F0,L0,D0),E1),
        sentence(S,[dcl],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz),
        join(L0,[that|Lb],L1),
        join(La,[that|Lb],L).

lexverbphrase(node(#62:21,[liv],L1,[node(N0,F0,L0,D0),VP]),
        [lex,Infl,icomp],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        lexverb(node(N0,F0,L0,D0),[lex,Infl,icomp],
            La,Ia,[to|Ib]),
        environment(E,node(N0,F0,L0,D0),E1),
        verbphrase(VP,[K,inf,C1],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz),
        join(L0,[to|Lb],L1),
        join(La,[to|Lb],L).

lexverbphrase(node(#63:0,[liv],L1,[node(N0,F0,L0,D0),Q]),
[lex,Infl,qcomp],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
lexverb(node(N0,F0,L0,D0),[lex,Infl,qcomp],
    La,Ia,Ib),
environment(E,node(N0,F0,L0,D0),E1),
question(Q,[q],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz),
join(L0,Lb,L1),
join(La,Lb,L).
```

```
copulative(node(#64:0,[cop],L1,[node(N0,F0,L0,D0),NP]),[cop,Infl,trans],
        E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        copula(node(N0,F0,L0,D0),[cop,Infl,trans],La,Ia,Ib),
        environment(E,node(N0,F0,L0,D0),E1),
        nounphrase(NP,[Md,G,nom,Num],E1,Lb,Ib,Iz,H,FVB,SA,SRa,SRz),
        join(L0,Lb,L1),
        join(La,Lb,L).

recurse(g(Ft,E,H,SA),
      vp(node(N1,F1,L1,D1),La,Ib,FVBa,SRb),
      vp(VP,L,Iz,FVB,SRz)) :-
         scan(Conj,Ib,[Vb|Ic]),
         choose(Ft,Rule,Conj),
         verbform(Vb,_,_),
         join(FVBa,SA,SAa),
         environment(E,node(N1,F1,L1,D1),E1),
         ((Ft=[fin(Num,T1)],Ft2=[fin(Num,T2)]);Ft=Ft2),
         verbphrase(node(N2,F2,L2,D2),Ft2,E1,Lb,[Vb|Ic],Id,
              H,FVBb,SAa,SRb,SRc),
         join(La,[Conj|Lb],Lc),
         join(L1,[Conj|L2],L3),
         mix(FVBa,FVBb,FVBc),
         quantify(Ft,node(Rule,Ft,L3,
                     [node(N1,F1,L1,D1),
                      node(N2,F2,L2,D2)]),
              node(N0,F0,L0,D0),Lc,Ld,L3,L0,
                 FVBc,FVBd,[],SRc),
         catalog(Id,E,node(N0,L0,L0,D0)),
         ((Ft=[fin(Num,T1)],\+ T1=T2,Ft3=[fin(Num,janus)])
         ;Ft3=Ft),
          recurse(g(Ft3,E,H,SAa),
                vp(node(N0,F0,L0,D0),Ld,Id,FVBd,SRc),
                vp(VP,L,Iz,FVB,SRz)).

recurse(g(Ft,E,H,SA),
      vp(node(N1,F1,L1,D1),La,[W|Ib],FVBa,SRb),
      vp(VP,L,Iz,FVB,SRz)) :-
         (preposition(W);vpadverb(W)),
         \+ Ft=[fin(_,_)],
         join(FVBa,SA,SAa),
         environment(E,node(N1,F1,L1,D1),E1),
         vpadverb(VPADV,AV,E1,Lb,[W|Ib],Ic,H,FVBb,SAa,SRb,SRc),
         join(La,Lb,Lc),
         join(L1,Lb,L2),
         mix(FVBa,FVBb,FVBc),
         choose(Ft,R,vpadv),
         VP1=node(R,Ft,L2,[VPADV,node(N1,F1,L1,D1)]),
         catalog(Ic,E,VP1),
         recurse(g(Ft,E,H,SAa),
               vp(VP1,Lc,Ic,FVBc,SRc),
               vp(VP,L,Iz,FVB,SRz)).
```

```
recurse(g([fin(Num,Time)],E,H,SA),
        vp(node(N1,F1,L1,D1),La,[W|Ib],FVBa,SRb),
        vp(VP,L,Iz,FVB,SRz)) :-
            join(FVBa,SA,SAa),
            environment(E,node(N1,F1,L1,D1),E1),
            timeadverb(TMA,[tma(Time)],E1,Lb,[W|Ib],Ic,H,FVBb,SAa,
                    SRb,SRc),
            join(La,Lb,Lc),
            join(L1,Lb,L2),
            mix(FVBa,FVBb,FVBc),
            VP1=node(#77:1,fin(Num,Time),L2,[node(N1,F1,L1,D1),TMA]),
            catalog(Ic,E,VP1),
            recurse(g([fin(Num,Time)],E,H,SAa),
                    vp(VP1,Lc,Ic,FVBc,SRc),
                    vp(VP,L,Iz,FVB,SRz)).

agent(NP,[Md,G,acc,Num],E,[by|L],[by|Ia],Iz,H,FVB,SA,SRa,SRz) :-
            nounphrase(NP,[Md,G,acc,Num],E,L,Ia,Iz,
                    H,FVB,SA,SRa,SRz).

agent(node(#0:'=',[agent],[agent],[sense(agent,[agent])]),
            [agent],E,[],Ia,Ia,H,[],SA,SR,SR).

tense(node(#0:'=',[tense],[T],[sense(T,[tense])]),
            [K,fin(Num,Tense),C],Root,Ia) :-
            scan(Form,Ia,Iz),
            verbform(Form,fin(Num,Tense),Root),
            verb(Root,[K,C]).

auxverb(node(#1:'=',[K,C],[L],[sense(Root,[K,C])]),
            [K,Infl,C],N,V) :-
            verbform(V,Infl,Root),
            auxilliary(Root,L,N,[K,C]).

ptverb(node(#59:0,[ptv],[Form],
          [node(#0:'=',[pflag],[passive],[sense(passive,[pflag])]),
           node(#1:'=',[lex,trans],[Root],[sense(Root,[lex,trans])])]),
       [lex,en,trans],[Form],Ia,Iz) :-
            scan(Form,Ia,Iz),
            verbform(Form,en,Root),
            verb(Root,[lex,trans]).

lexverb(node(#1:'=',[lex,C],[Root],[sense(Root,[lex,C])]),
            [lex,Infl,C],[Form],Ia,Iz) :-
            scan(Form,Ia,Iz),
            verbform(Form,Infl,Root),
            verb(Root,[lex,C]).

copula(node(#1:'=',[cop,trans],[be],[sense(be,[cop,trans])]),
            [cop,Infl,trans],[Form],Ia,Iz) :-
            scan(Form,Ia,Iz),
            verbform(Form,Infl,be),
            verb(be,[cop,trans]).
```

```
/* ADVERBIAL PHRASE RULES */

vpadverb(node(#1:'=',[adv],[VPADV],[sense(VPADV,[adv])]),[adv],E,
        [VPADV],Ia,Iz,H,[],SA,SR,SR) :-
        scan(VPADV,Ia,Iz),
        vpadverb(VPADV).

vpadverb(node(#78:0,[aph],L,
        [P,NP]),[aph],E,L,Ia,Iz,H,FVB,SA,SRa,SRz) :-
        preposition(P,La,Ia,Ib),
        environment(E,P,E1),
        nounphrase(NP,[Md,G,acc,Num],E1,Lb,Ib,Iz,
            H,FVB,SA,SRa,SRz),
        join(La,Lb,L).

preposition(node(#1:'=',[prep],[P],[sense(P,[prep])]),[P],Ia,Iz) :-
        scan(P,Ia,Iz),
        preposition(P).

timeadverb(node(#1:'=',Ft,[TMA],[sense(TMA,Ft)]),Ft,E,[TMA],Ia,Iz,
        H,[],SA,SR,SR) :-
        scan(TMA,Ia,Iz),
        timeadverb(TMA,Ft).


/* SELECTORS */

choose(s,#11:11,and) :- !.
choose(s,#11:12,or) :- !.
choose(np,#13:13,and,G1,_,Num1,Num2,pl) :- !.
choose(np,#13:14,or,G1,odd,Num1,Num2,Num2) :- !.

choose(#41:0,[present],pres).
choose(#42:0,[past],past).

choose([fin(_,_)],#70:7,and).
choose([fin(_,_)],#70:8,or).
choose([lex,_,_],#71:5,and).
choose([lex,_,_],#71:6,or).
choose([piv],#72:5,and).
choose([piv],#72:6,or).
choose([ptv],#73:9,and).
choose([ptv],#73:10,or).

choose([lex,_,_],#74:21,vpadv).
choose([piv],#75:21,vpadv).
choose([ptv],#76:22,vpadv).
```

APPENDIX C: EDIT

```
/* ######################## */
/* #### TEXT EDITOR #### */
/* ######################## */



edit(_,_,_,_,_,_,_,_,_,[],[]) :- !.

edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,[Nom|Y],Y2) :- !,
        editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),
        join(L,Y1,Y2),!.

edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,[Acc|Y],Y2) :- !,
        editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),
        join(L,Y1,Y2),!.

edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,[Nom1|Y],[Subj|Y1]) :- !,
        edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,Y,Y1),!.

edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,[Acc1|Y],[Obj|Y1]) :- !,
        edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,Y,Y1),!.

edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,[X|Y],[X|Y1]) :-
        edit(Nom,Nom1,Acc,Acc1,Rf,Subj,Obj,Ref,L,Y,Y1),!.


editline(_,_,_,_,_,_,[],[]) :- !.

editline(Nom,Acc,Rf,Subj,Obj,Ref,['(',Nom,')'|Y],Y1) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.

editline(Nom,Acc,Rf,Subj,Obj,Ref,['(',Acc,')'|Y],Y1) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.

editline(Nom,Acc,Rf,Subj,Obj,Ref,[Nom|Y],[Subj|Y1]) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.

editline(Nom,Acc,Rf,Subj,Obj,Ref,[Acc|Y],[Obj|Y1]) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.

editline(Nom,Acc,Rf,Subj,Obj,Ref,[Rf|Y],[Ref|Y1]) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.

editline(Nom,Acc,Rf,Subj,Obj,Ref,[X|Y],[X|Y1]) :-
        !,editline(Nom,Acc,Rf,Subj,Obj,Ref,Y,Y1),!.
```

```
makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,he,him,himself,[m,sg]) :-
        !,name(Nom,[104,101|Suffix]),
        name(Nom1,[72,69|Suffix]),
        name(Acc,[104,105,109|Suffix]),
        name(Acc1,[72,73,77|Suffix]),
        name(Rf,[104,105,109,115,101,108,102|Suffix]),!.

makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,she,her,herself,[f,sg]) :-
        !,name(Nom,[115,104,101|Suffix]),
        name(Nom1,[83,72,69|Suffix]),
        name(Acc,[104,101,114|Suffix]),
        name(Acc1,[72,69,82|Suffix]),
        name(Rf,[104,101,114,115,101,108,102|Suffix]),!.

makevars(Nom,Nom1,Nom,Nom1,Rf,Suffix,it,it,itself,[n,sg]) :-
        !,name(Nom,[105,116|Suffix]),
        name(Nom1,[73,84|Suffix]),
        name(Rf,[105,116,115,101,108,102|Suffix]),!.

makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,they,them,themselves,[_,pl]) :-
        !,name(Nom,[116,104,101,121|Suffix]),
        name(Nom1,[84,72,69,89|Suffix]),
        name(Acc,[116,104,101,109|Suffix]),
        name(Acc1,[84,72,69,77|Suffix]),
        name(Rf,[116,104,101,109,115,101,108,118,101,115|Suffix]),!.

makevars(Nom,Nom1,Acc,Acc1,Rf,Suffix,whichever,whichever,whichever,[odd,Num]) :-
        !,name(Nom,[104,101|Suffix]),
        name(Nom1,[72,69|Suffix]),
        name(Acc,[104,105,109|Suffix]),
        name(Acc1,[72,73,77|Suffix]),
        name(Rf,[104,105,109,115,101,108,102|Suffix]),!.
```

```
copyvar([m,nom,sg,p],He,HE,Suffix) :-
        !,name(He,[104,101|Suffix]),
        name(HE,[72,69|Suffix]),!.

copyvar([m,acc,sg,p],Him,HIM,Suffix) :-
        !,name(Him,[104,105,109|Suffix]),
        name(HIM,[72,73,77|Suffix]),!.

copyvar([m,acc,sg,r],Himself,_,Suffix) :-
        !,name(Himself,[104,105,109,115,101,108,102|Suffix]),!.

copyvar([f,nom,sg,p],She,SHE,Suffix) :-
        !,name(She,[115,104,101|Suffix]),
        name(SHE,[83,72,69|Suffix]),!.

copyvar([f,acc,sg,p],Her,HER,Suffix) :-
        !,name(Her,[104,101,114|Suffix]),
        name(HER,[72,69,82|Suffix]),!.

copyvar([f,acc,sg,r],Herself,_,Suffix) :-
        !,name(Herself,[104,101,114,115,101,108,102|Suffix]),!.

copyvar([n,_,sg,p],It,IT,Suffix) :-
        !,name(It,[105,116|Suffix]),
        name(IT,[73,84|Suffix]),!.

copyvar([n,acc,sg,r],Itself,_,Suffix) :-
        !,name(Itself,[105,116,115,101,108,102|Suffix]),!.

copyvar([_,nom,pl,p],They,THEY,Suffix) :-
        !,name(They,[116,104,101,121|Suffix]),
        name(THEY,[84,72,69,89|Suffix]),!.

copyvar([_,acc,pl,p],Them,THEM,Suffix) :-
        !,name(Them,[116,104,101,109|Suffix]),
        name(THEM,[84,72,69,77|Suffix]),!.

copyvar([_,acc,pl,r],Themselves,_,Suffix) :-
    !,name(Themselves,[116,104,101,109,115,101,108,118,101,115|Suffix]),!.

copyvar([odd,C,N,T],Nom,Acc,S) :-
    copyvar([m,C,N,T],Nom,Acc,S).
```

APPENDIX D: GENLEX

```
/* ###################################### */
/* ######## GENERAL LEXICON ######## */
/* ###################################### */


determiner(every,2,[dcl,sg]).
determiner(the,2,[dcl,_]).
determiner(a,2,[dcl,sg]).
determiner(which,29,[q,_]).
determiner(what,29,[q,_]).

pronoun(he,[m,nom,sg,p]).
pronoun(him,[m,acc,sg,p]).
pronoun(himself,[m,acc,sg,r]).
pronoun(she,[f,nom,sg,p]).
pronoun(her,[f,acc,sg,p]).
pronoun(herself,[f,acc,sg,r]).
pronoun(it,[n,_,sg,p]).
pronoun(itself,[n,acc,sg,r]).
pronoun(they,[_,nom,pl,p]).
pronoun(them,[_,acc,pl,p]).
pronoun(themselves,[_,acc,pl,r]).
pronoun(whichever,[odd,_,_,p]).

relpronoun(that,30,[_,_]).
relpronoun(who,31,[G,nom]) :- (G=m);(G=f).
relpronoun(whom,31,[G,acc]) :- (G=m);(G=f).
relpronoun(which,31,[n,_]).

wh_pronoun(who,[m,nom,sg]).
wh_pronoun(who,[m,nom,pl]).
wh_pronoun(who,[f,nom,sg]).
wh_pronoun(who,[f,nom,pl]).
wh_pronoun(whom,[m,acc,sg]).
wh_pronoun(whom,[m,acc,pl]).
wh_pronoun(whom,[f,acc,sg]).
wh_pronoun(whom,[f,acc,pl]).
wh_pronoun(what,[n,_,sg]).

decline(Sg,[G,sg],Sg) :- noun(Sg,_,[G]).
decline(Pl,[G,pl],Sg) :- noun(Sg,Pl,[G]).

preposition(in).
preposition(about).

timeadverb(yesterday,[tma(ante)]).
timeadverb(today,[tma(_)]).
timeadverb(tomorrow,[tma(post)]).
```

```
sadverb(necessarily).

direction(pres,willaux,post) :- !.
direction(pres,bmod,post) :- !.
direction(pres,shall,post) :- !.
direction(past,willaux,_) :- !.
direction(past,shall,_) :- !.
direction(past,bemod,_) :- !.
direction(pres,_,nunc).
direction(past,_,ante).

verbform(Vr,root,Vr) :- inflect(Vr,_,_,_,_,_,_).
verbform(Vo,fin(pl,pres),Vr) :- inflect(Vr,Vo,_,_,_,_,_).
verbform(Vs,fin(sg,pres),Vr) :- inflect(Vr,_,Vs,_,_,_,_).
verbform(Vd,fin(_,past),Vr)  :- inflect(Vr,_,_,Vd,_,_,_).
verbform(Vto,inf,Vr)         :- inflect(Vr,_,_,_,Vto,_,_).
verbform(Vg,ing,Vr)          :- inflect(Vr,_,_,_,_,Vg,_).
verbform(Vn,en,Vr)           :- inflect(Vr,_,_,_,_,_,Vn).

verbform(was,fin(sg,past),B) :- getnext(B,[be,bepass,beprog,bemod]).
verbform(were,fin(pl,past),B) :- getnext(B,[be,bepass,beprog,bemod]).

verb(R,[K,C]) :- auxiliary(R,_,_,[K,C]).

auxiliary(doaux,do,44,[laux,inf]).
auxiliary(bepass,be,46,[paux,en]).
auxiliary(bemod,be,48,[taux,inf]).
auxiliary(goaux,go,48,[taux,inf]).
auxiliary(useaux,use,48,[taux,inf]).
auxiliary(shall,shall,50,[aux,inf]).
auxiliary(willaux,will,50,[aux,inf]).
auxiliary(canaux,can,50,[aux,inf]).
auxiliary(must,must,52,[aux,inf]).
auxiliary(haveaux,have,54,[aux,en]).
auxiliary(beprog,be,56,[aux,ing]).

inflect(doaux,do,does,did,[],[],[]).
inflect(bepass,are,is,[],be,being,been).
inflect(bemod,are,is,[],[],[],[]).
inflect(beprog,are,is,[],be,[],been).
inflect(goaux,[],[],[],[],going,[]).
inflect(useaux,[],[],used,[],[],[]).
inflect(shall,shall,shall,should,[],[],[]).
inflect(willaux,will,will,would,[],[],[]).
inflect(canaux,can,can,could,[],[],[]).
inflect(must,must,must,[],[],[],[]).
inflect(haveaux,have,has,had,have,[],[]).

verb(be,[cop,trans]).
inflect(be,are,is,[],be,being,been).
```

APPENDIX E: LEXTMG

```
/* ################################## */
/* ########  LEXICON  ######## */
/* ################################## */

proper(bill,[m,sg]).
proper(emily,[f,sg]).
proper(henry,[m,sg]).
proper(john,[m,sg]).
proper(lucy,[f,sg]).
proper(mary,[f,sg]).
proper(ninety,[n,sg]).
proper(scott,[m,sg]).
proper(susan,[f,sg]).
proper(waverley,[n,sg]).

vpadverb(allegedly).
vpadverb(frequently).
vpadverb(happily).
vpadverb(passionately).
vpadverb(rapidly).
vpadverb(sadly).
vpadverb(secretly).
vpadverb(slowly).
vpadverb(tenderly).
vpadverb(voluntarily).

noun(boy,boys,[m]).
noun(cat,cats,[n]).
noun(child,children,[_]).
noun(cow,cows,[n]).
noun(daughter,daughters,[f]).
noun(dog,dogs,[n]).
noun(farmer,farmers,[m]).
noun(fish,fishes,[n]).
noun(girl,girls,[f]).
noun(grade,grades,[n]).
noun(horse,horses,[n]).
noun(king,kings,[m]).
noun(maid,maids,[f]).
noun(man,men,[m]).
noun(park,parks,[n]).
noun(pen,pens,[n]).
noun(price,prices,[n]).
noun(professor,professors,[_]).
noun(son,sons,[m]).
noun(student,students,_).
noun(traitor,traitors,_).
noun(unicorn,unicorns,[n]).
noun(vet,vets,[_]).
noun(woman,women,[f]).
```

```
verb(ask,[lex,qcomp]).
verb(assert,[lex,scomp]).
verb(bear,[lex,trans]).
verb(believe,[lex,scomp]).
verb(catch,[lex,trans]).
verb(change,[lex,intrans]).
verb(conceive,[lex,trans]).
verb(cuddle,[lex,trans]).
verb(date,[lex,trans]).
verb(deserve,[lex,trans]).
verb(die,[lex,intrans]).
verb(eat,[lex,trans]).
verb(find,[lex,trans]).
verb(hate,[lex,trans]).
verb(hope,[lex,scomp]).
verb(kick,[lex,trans]).
verb(kiss,[lex,trans]).
verb(know,[lex,qcomp]).
verb(lose,[lex,trans]).
verb(love,[lex,trans]).
verb(marry,[lex,trans]).
verb(meet,[lex,trans]).
verb(milk,[lex,trans]).
verb(recommend,[lex,trans]).
verb(rise,[lex,intrans]).
verb(run,[lex,intrans]).
verb(see,[lex,trans]).
verb(seek,[lex,trans]).
verb(sell,[lex,trans]).
verb(shave,[lex,trans]).
verb(swim,[lex,intrans]).
verb(talk,[lex,intrans]).
verb(teach,[lex,trans]).
verb(try,[lex,icomp]).
verb(walk,[lex,intrans]).
verb(wish,[lex,icomp]).
verb(wonder,[lex,qcomp]).
verb(write,[lex,trans]).
```

```
inflect(ask,ask,asks,asked,ask,asking,asked).
inflect(assert,assert,asserts,asserted,assert,asserting,asserted).
inflect(bear,bear,bears,bore,bear,bearing,born).
inflect(believe,believe,believes,believed,believe,believing,
      believed).
inflect(catch,catch,catches,caught,catch,catching,caught).
inflect(change,change,changes,changed,change,changing,changed).
inflect(conceive,conceive,conceives,conceived,conceive,
      conceiving,conceived).
inflect(cuddle,cuddle,cuddles,cuddled,cuddle,cuddling,cuddled).
inflect(date,date,dates,dated,date,dating,dated).
inflect(deserve,deserve,deserves,deserved,deserve,deserving,
      deserved).
inflect(die,die,dies,died,die,dying,died).
inflect(eat,eat,eats,ate,eat,eating,eaten).
inflect(find,find,finds,found,find,finding,found).
inflect(hate,hate,hates,hated,hate,hating,hated).
inflect(hope,hope,hopes,hoped,hope,hoping,hoped).
inflect(kick,kick,kicks,kicked,kick,kicking,kicked).
inflect(kiss,kiss,kisses,kissed,kiss,kissing,kissed).
inflect(know,know,knows,knew,know,knowing,known).
inflect(lose,lose,loses,lost,lose,losing,lost).
inflect(love,love,loves,loved,love,loving,loved).
inflect(marry,marry,marries,married,marry,marrying,married).
inflect(meet,meet,meets,met,meet,meeting,met).
inflect(milk,milk,milks,milked,milk,milking,milked).
inflect(teach,teach,teaches,taught,teach,teaching,taught).
inflect(recommend,recommend,recommends,recommended,
      recommend,recommending,recommended).
inflect(rise,rise,rises,rose,rise,rising,risen).
inflect(run,run,runs,ran,run,running,run).
inflect(see,see,sees,saw,see,seeing,seen).
inflect(seek,seek,seeks,sought,seek,seeking,sought).
inflect(sell,sell,sells,sold,sell,selling,sold).
inflect(shave,shave,shaves,shaved,shave,shaving,shaved).
inflect(swim,swim,swims,swam,swim,swimming,swum).
inflect(talk,talk,talks,talked,talk,talking,talked).
inflect(try,try,tries,tried,try,trying,tried).
inflect(walk,walk,walks,walked,walk,walking,walked).
inflect(wish,wish,wishes,wished,wish,wishing,wished).
inflect(wonder,wonder,wonders,wondered,wonder,wondering,wondered).
inflect(write,write,writes,wrote,write,writing,written).
```

APPENDIX F: LILT

```
/* ################################################## */
/* #### LANGUAGE OF INTENTIONAL LOGIC TRANSLATOR #### */
/* ################################################## */

/* GALILEAN POST ORDER TRAVERSAL */

translate(J,node(N,F,L,[sense(R,T)]),S) :-
        !,sense(R,T,S),
        message(J,basic(L),S),!.

translate(J,Tree,IL) :-
        structure(Tree,node(N,F,L,_),Lsub,Rsub),
        translate(J1,Rsub,Rnew),
        translate(J2,Lsub,Lnew),
        construct(node(N,F,L,_),Lnew,Rnew,Tree1),
        formulate(Tree1,IL1),
        message(J3,cons(N,J1,J2),IL1),!,
        simplify(J3,J,IL1,IL).

structure(node(N,F,L,[Lsub,Rsub]),node(N,F,L,_),Lsub,Rsub).

construct(node(N,F,L,_),Lnew,Rnew,node(N,F,L,[Lnew,Rnew])).

/* SIMPLIFICATION OF EMBEDDED FORMULAE */

simplify(J,J,X,X) :- var(X),!.

simplify(Ja,Jz,(P & Q),(P1 & Q1)) :-
        simplify(Ja,Jb,P,P1),
        simplify(Ja,Jz,Q,Q1).

simplify(Ja,Jz,(P @ Q),(P1 @ Q1)) :-
        simplify(Ja,Jb,P,P1),
        simplify(Ja,Jz,Q,Q1).

simplify(Ja,Jz,(P v Q),(P1 v Q1)) :-
        simplify(Ja,Jb,P,P1),
        simplify(Ja,Jz,Q,Q1).

simplify(Ja,Jz,~(P),~(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,future(P),future(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,past(P),past(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,perfect(P),perfect(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,progressive(P),progressive(Q)) :-
        simplify(Ja,Jz,P,Q).
```

```
simplify(Ja,Jz,poss(P),poss(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,nec(P),nec(Q)) :- simplify(Ja,Jz,P,Q).

simplify(Ja,Jz,'(^Body),Result) :-
        message(Jb,du(Ja),Body),!,
        simplify(Jb,Jz,Body,Result).

simplify(Ja,Jz,^Body,^Result) :- !,
        simplify(Ja,Jz,Body,Result).

simplify(Ja,Jz,'Body,'Result) :- !,
        simplify(Ja,Jz,Body,Result).

simplify(Ja,Jz,eval('(^P),A),Result) :-
        message(Jb,du(Ja),eval(P,A)),!,
        simplify(Jb,Jz,eval(P,A),Result).

simplify(Ja,Jz,eval(^P,A),^Result) :-
        !,simplify(Ja,Jz,eval(P,A),Result).

simplify(Ja,Jz,eval('P,A),'Result) :-
        !,simplify(Ja,Jz,eval(P,A),Result).

simplify(Ja,Jz,B,Result) :-
        B=..[Pred,Z],
        nonvar(Z),
        Z=eval(P,Q),!,
        Result=..[Pred,W],
        simplify(Ja,Jz,Z,W).

simplify(Ja,Jz,lambda(X:A),lambda(X:Z)) :-
        nonvar(A),
        reducible(A),!,
        simplify(Ja,Jz,A,Z).

/* EVALUATE LAMBDA EXPRESSION */

simplify(Ja,Jz,eval(lambda(Var:Body),Z),Result) :-
        nonvar(Z),reducible(Z),
        !,simplify(Ja,Jb,Z,Arg),
        convert(Ja,Jc,Arg,Var,Body,R),
        message(Jd,conv(Ja),R),!,
        simplify(Jd,Jz,R,Result).

simplify(Ja,Jz,eval(lambda(Var:Body),Arg),Result) :-
        !,convert(Ja,Jb,Arg,Var,Body,R),
        message(Jc,conv(Ja),R),!,
        simplify(Jc,Jz,R,Result).
```

```
/* EXPRESS IN RELATIONAL NOTATION */

simplify(Ja,Jz,eval(P,A),Result) :-
        simplify(Ja,Jaa,A,B),
        evaluate(Jaa,Jb,eval(P,B),R),
        message(Jc,rel(Jb),R),!,
        simplify(Jc,Jz,R,Result).

/* SUBSTITUTE IDENTICALS */

simplify(Ja,Jz,exists(X:(Z @ equals(Y,X))),Result) :-
        nonvar(Z),
        Z=..[Pred,X],
        R=..[Pred,Y],
        message(Jb,iden(Ja),R),!,
        simplify(Jb,Jz,R,Result).

simplify(Ja,Jz,exists(X:(Z & equals(Y,X))),Result) :-
        nonvar(Z),
        Z=..[Pred,X],
        R=..[Pred,Y],
        message(Jb,iden(Ja),R),!,
        simplify(Jb,Jz,R,Result).

/* APPLY MEANING POSTULATE */

simplify(Ja,Jz,Old,New) :-
        postulate(pos(N),Old,Eq),
        message(Jb,pos(N,Ja),Eq),!,
        simplify(Jb,Jz,Eq,New).

simplify(J,J,X,X).

/* CONVERT LAMBDA EXPRESSION: λVar(Body)[Value] */

convert(J,J,Value,Var,Body,Value) :-
        atomic(Body),Body==Var,!. /* match predicate vble */

convert(J,J,Value,Var,Body,Value) :-
        var(Body),Body==Var,!.  /* match individual vble */

convert(J,J,Value,Var,Body,Body) :-
        (atomic(Body);var(Body)),!. /* non-matching element */

convert(Ja,Jz,Value,Var,Body,Result) :-
        insert(Ja,Jz,Value,Var,Body,Result). /* not elementary Body */
```

```
/* INSTANTIATE A PRINCIPLE FUNCTOR */

insert(Ja,Jz,Value,Var,Body,Result) :-
        Body=..[^,B],
        B=..[V|Tail],
        V==Var,!,
        ((Tail==[],R=..[^,Value]);
        R=..[eval,^Value|Tail]),
        message(Jb,inst(Ja),R),
        simplify(Jb,Jz,R,Result).

insert(Ja,Jz,Value,Var,Body,Result) :-
        Body=..['',B],
        B=..[V|Tail],
        V==Var,!,
        ((Tail==[],R=..['',Value]);
        R=..[eval,'Value|Tail]),
        message(Jb,inst(Ja),R),
        simplify(Jb,Jz,R,Result).

insert(Ja,Jz,Value,Var,Body,Result) :-
        Body=..[V|Tail],
        V==Var,!,
        ((Tail==[],R=Value);
        R=..[eval,Value|Tail]),
        message(Jb,inst(Ja),R),
        simplify(Ja,Jz,R,Result).

insert(Ja,Jz,Value,Var,Body,Result) :-
        Body=..[Head|Tail],
        substitute(Ja,Jb,Value,Var,Tail,Tail1),
        R=..[Head|Tail1],
        simplify(Ja,Jz,R,Result).

/* INSTANTIATE A VARIABLE WITHIN A LIST */

substitute(J,J,_,_,[],[]) :- !.

substitute(Ja,Jz,Value,Var,[H|T],[H1|T1]) :-
        convert(Ja,Jb,Value,Var,H,H1),
        substitute(Jb,Jz,Value,Var,T,T1).
```

```
/* EVALUATE TO A RELATIONAL EXPRESSION */

evaluate(Ja,Jz,eval(P,A),Result) :-
        atom(P),!,
        R=..[P,A],
        simplify(Ja,Jz,R,Result).

evaluate(Ja,Jz,eval(P,A),Result) :-
        P=..[F|Args],
        R=..[F,A|Args],
        simplify(Ja,Jz,R,Result).

/* CHECK THAT EXPRESSION IS PRIMA FACIE REDUCIBLE */

reducible(eval(X,Y)).

reducible(^lambda(X:B)).

reducible(lambda(X:B)).

reducible('P) :-
        P=..[Pred,Q],
        nonvar(Q),
        Q=eval(A,B).

reducible((T1 & T2)).

reducible((T1 @ T2)).

reducible((T1 v T2)).
```

```
/* COMPOSITION AND SIMPLIFICATION MESSAGES */

message(J,basic(L),S) :- nl,
        format(J,' Lexicon','Basic expression ',L,S).

message(J,cons(#N:M,J1,J2),S) :- nl,
        format(J,[J1,J2],'Construction by T',N,S).

message(J,pos(N,J1),S) :-
        format(J,[J1],'Postulate ',N,S).

message(J,rel(J1),S) :-
        format(J,[J1],'Relational notation',S).

message(J,du(J1),S) :-
        format(J,[J1],'Down-up conversion',S).

message(J,conv(J1),S) :-
        format(J,[J1],'Lambda conversion',S).

message(J,inst(J1),S) :-
        format(J,[J1],'Instantiate variable',S).

message(J,iden(J1),S) :-
        format(J,[J1],'Substitute identicals',S).

format(J,K,T,N,S) :-
        getnum(step,J),nl,
        write([J]),write(' from '),write(K),write(': '),
        write(T),write(N),write(' =>'),
        nl,tab(15),write(S).

format(J,K,T,S) :-
        getnum(step,J),nl,
        write([J]),write(' from '),write(K),write(': '),
        write(T),
        nl,tab(15),write(S).
```

```
/* ############################################## */
/* #   BASIC TIL ASSIGNMENTS AND POSTULATES   # */
/* ############################################## */

/* TRANSLATION RULE FORMULATIONS */

formulate(node(#N:0,F,L,[T1,T2]),eval(T1,^T2)).

formulate(node(#N:1,F,L,[T1,T2]),eval(T2,^T1)).

formulate(node(#N:(2:Inx),F,L,[T1,T2]),
          eval(T1,^lambda(Var:T2))) :-
          anaphoric(Var,Inx).

formulate(node(#N:(3:Inx),F,L,[T1,T2]),
        lambda(Var:(eval(T1,Var) & exists(t:ab(t,T2))))) :-
          anaphoric(Var,Inx).

formulate(node(#N:(4:Inx),F,L,[T1,T2]),
          lambda(Y:eval(T1,^lambda(Var:eval(T2,Y))))) :-
          anaphoric(Var,Inx).

formulate(node(#N:5,F,L,[T1,T2]),
          lambda(e:(eval(T1,e) @ eval(T2,e)))).

formulate(node(#N:6,F,L,[T1,T2]),
          lambda(e:(eval(T1,e) v eval(T2,e)))).

formulate(node(#N:7,F,L,[T1,T2]),
          lambda(n:(eval(T1,n) @ eval(T2,n)))).

formulate(node(#N:8,F,L,[T1,T2]),
          lambda(n:(eval(T1,n) v eval(T2,n)))).

formulate(node(#N:9,F,L,[lambda(n:lambda(X:(‘n(T1)))),
                   lambda(n:lambda(X:(‘n(T2))))]),
        lambda(n:lambda(X:(‘n(^lambda(e:(eval(‘T1,e) @
                                eval(‘T2,e)))))))).

formulate(node(#N:10,F,L,[lambda(n:lambda(X:(‘n(T1)))),
                   lambda(n:lambda(X:(‘n(T2))))]),
        lambda(n:lambda(X:(‘n(^lambda(e:(eval(‘T1,e) v
                                eval(‘T2,e)))))))).
```

```
formulate(node(#N:11,F,L,[T1,T2]),(T1 @ T2)).

formulate(node(#N:12,F,L,[T1,T2]),(T1 v T2)).

formulate(node(#N:13,F,L,[T1,T2]),
        lambda(p:(eval(T1,p) & eval(T2,p)))).

formulate(node(#N:14,F,L,[T1,T2]),
        lambda(p:(eval(T1,p) v eval(T2,p)))).

formulate(node(#N:15,F,L,[lambda(p:lambda(e:T1)),T2]),
      eval(lambda(p:lambda(e:(~T1))),^T2)).

formulate(node(#N:16,F,L,[lambda(p:lambda(e:T1)),T2]),
      eval(lambda(p:lambda(e:T3)),^T2)) :-
         T1 =.. [M,S],
         T3 =.. [M,~S].

formulate(node(#N:17,F,L,[T1,T2]),lambda(r:(eval(T1,r) v
                              eval(T2,r)))).

formulate(node(#N:(18:Inx),F,L,[T1,T2]),
      lambda(r:eval(T1,^lambda(Var:eval(T2,r)))) :-
         anaphoric(Var,Inx).

formulate(node(#N:(19:Inx),F,L,[T1,T2]),
      lambda(r:eval(~T1,^lambda(Var:eval(~T2,r)))) :-
         anaphoric(Var,Inx).

formulate(node(#N:20,F,L,[T1,T2]),
        eval(T1,^exists(t,ab(t,T2)))).

formulate(node(#N:21,F,L,[T1,T2]),
      eval(T1,^lambda(Z:eval(T2,Z)))).

formulate(node(#N:22,F,L,[T1,lambda(n:lambda(X:('n(^T2))))]),
   lambda(n:lambda(X:('n(eval(^T1,^lambda(Z:eval(T2,Z))))))).

anaphoric(Var,Inx) :-
        name(Inx,Suffix),
        name(Var,[120|Suffix]).
```

```
/* LEXICAL SENSES */


sense(every,[dcl,sg],
lambda(p:lambda(q:all(X:(~('p(X)@(~('q(X))))))))) :- !.

sense(the,[dcl,sg],
lambda(p:lambda(q:exists(Y:all(X:('p(X)<=>equals(X,Y)) @
                        ('q(Y))))))) :- !.

sense(the,[dcl,pl],
lambda(p:lambda(q:exists(w:all(X:('w(X)=>('p(X)@('q(X))))))))) :- !.

sense(a,[dcl,sg],
lambda(p:lambda(q:exists(X:('p(X)@('q(X))))))) :- !.

sense(W,[whpron],lambda(p:exists(X:('p(X))))) :- !.

sense(W,[q,_],lambda(p:lambda(q:exists(X:
          ('p(X) @ ('q(X))))))) :- !.

sense(N,[pn],lambda(p:('p(N)))) :- !.

sense(V,[v],lambda(p:('p(Var)))) :-
          !,name(V,[104,101|S]),
          name(Var,[120|S]).

sense(necessarily,[sadv],lambda(p:nec('p))) :- !.

sense(A,[adv],lambda(p:lambda(e:V))) :- !,V=..[A,e,p].

sense(W,[tma(_)],
lambda(f:lambda(n:('n(^lambda(e:at('t*','f(^lambda(p:('p(e)
                    & incl(W,'t*'))))))))))) :- !.

sense(P,[prep],lambda(n:lambda(p:lambda(e:K)))) :-
          K=..[P,e,^lambda(Z:('p(Z))),n].

sense(W,[cn],lambda(e:V)) :- !,V=..[W,e].

sense(W,[lex,intrans],lambda(e:V)) :- !,V=..[W,e].

sense(be,[cop,trans],
lambda(n:lambda(e:('n(^lambda(Y:equals(e,Y))))))) :- !.

sense(W,[lex,trans],lambda(n:lambda(e:V))) :- !,V=..[W,e,n].

sense(W,[lex,scomp],lambda(r:lambda(e:V))) :- !,V=..[W,e,r].

sense(W,[lex,icomp],lambda(p:lambda(e:V))) :- !,V=..[W,e,p].

sense(W,[lex,qcomp],lambda(u:lambda(e:V))) :- !,V=..[W,e,u].
```

```
sense(doaux,[laux,inf],lambda(p:lambda(e:(‘p(e))))) :- !.

sense(bepass,[paux,en],lambda(p:lambda(e:(‘p(e))))) :- !.

sense(bemod,[taux,inf],lambda(p:lambda(e:future(‘p(e))))) :- !.

sense(goaux,[taux,inf],lambda(p:lambda(e:future(‘p(e))))) :- !.

sense(shall,[aux,inf],lambda(p:lambda(e:future(‘p(e))))) :- !.

sense(willaux,[aux,inf],lambda(p:lambda(e:future(‘p(e))))) :- !.

sense(canaux,[aux,inf],lamda(p:lambda(e:poss(‘p(e))))) :- !.

sense(must,[aux,inf],lambda(p:lambda(e:nec(‘p(e))))) :- !.

sense(haveaux,[aux,en],lambda(p:lambda(e:perfect(‘p(e))))) :- !.

sense(beprog,[aux,ing],lambda(p:lambda(e:progressive(‘p(e))))) :- !.


/* PSEUDO LEXICAL SENSES */


sense(agent,[agent],lambda(p:exists(Y:(‘p(Y))))) :- !.

sense(Q,[yn],lambda(s:lambda(r:(‘r & equals(r,^(‘s)) v
                                equals(r,^~(‘s)))))) :- !.

sense(Q,[query],lambda(s:lambda(r:(‘r & equals(r,s))))) :- !.

sense(present,
[tense],lambda(p:lambda(n:(present(‘n(p)))))) :- !.

sense(past,
[tense],lambda(p:lambda(n:(past(‘n(p)))))) :- !.

sense(passive,[pflag],
lambda(b:lambda(n:lambda(X:(‘n(^(‘b(^lambda(p:(‘p(X))))))))))) :- !.


/* DEFAULT */

sense(Word,_,Word).
```

```
/* MEANING POSTULATES */

postulate(pos(4),lambda(e:Old),lambda(e:New)) :-
        Old=..[Pred,e,N],!,
        verb(Pred,_),\+ intensional(Pred),
        nonvar(N),N=(^Term),
        name(Pred,Ascii),
        name(Star,[42|Ascii]),
        P=..[Star,e,Y],
        New=eval(Term,^lambda(Y:P)).

postulate(pos(8),lambda(e:Old),lambda(e:New)) :-
        Old=..[Pred,e,S,N],
        preposition(Pred), \+ intensional(Pred),
        nonvar(N),N=(^Term),
        nonvar(S),S=(^Sen),
        name(Pred,Ascii),
        name(Star,[42|Ascii]),
        P=..[Star,e,S,Y],
        New=eval(Term,^lambda(Y:P)).

postulate(pos(9),seek(e,^T),try(e,^lambda(Z:Body))) :-
        name(Findstar,[42,102,105,110,100]),
        P=..[Findstar,Z,Y],
        Body=eval(T,^lambda(Y:P)).

intensional(believe).
intensional(hope).
intensional(know).
intensional(seek).
intensional(try).
intensional(wish).

intensional(about).
```

/* ## */
/* SAMPLE PARSES, FULL REDUCTIONS AND LOGICAL FORMS */
/* ## */

(H-1) John was kissed by Mary and cuddled by Lucy.

Analysis Tree
****

#40:1 john was kissed by mary and cuddled by lucy
  #1:= john
  #42:0 was kissed by mary and cuddled by lucy
    #0:= past
    #46:0 be kissed by mary and cuddled by lucy
      #1:= be
      #72:5 kissed by mary and cuddled by lucy
        #57:0 kissed by mary
          #59:0 kissed
            #0:= passive
            #1:= kiss
          #1:= mary
        #57:0 cuddled by lucy
          #59:0 cuddled
            #0:= passive
            #1:= cuddle
          #1:= lucy

Composition & Simplification
****

[0] from Lexicon: Basic expression [lucy] =>
lambda(p:'p(lucy))

[1] from Lexicon: Basic expression [cuddle] =>
lambda(n:lambda(e:cuddle(e,n)))

[2] from Lexicon: Basic expression [passive] =>
lambda(b:lambda(n:lambda(_1535:'n(^'b(^lambda(p:'p(_1535)))))))

[3] from [1,2]: Construction by T59 =>
eval(lambda(b:lambda(n:lambda(_1535:'n(^'b(^lambda(p:'p(_1535))))))),^lambda(n:lambda(e:cuddle(e,n))))
[4] from [3]: Instantiate variable
eval('^lambda(n:lambda(e:cuddle(e,n))),^lambda(p:'p(_1535)))
[5] from [4]: Down-up conversion
eval(lambda(n:lambda(e:cuddle(e,n))),^lambda(p:'p(_1535)))
[6] from [5]: Postulate 4 =>
lambda(e:eval(lambda(p:'p(_1535)),^lambda(_1983:*cuddle(e,_1983))))
[7] from [6]: Instantiate variable
eval('^lambda(_1983:*cuddle(e,_1983)),_1535)

[8] from [7]: Down-up conversion
eval(lambda(_1983:*cuddle(e,_1983)),_1535)
[9] from [8]: Lambda conversion
*cuddle(e,_1535)
[10] from [6]: Lambda conversion
*cuddle(e,_1535)
[11] from [5]: Lambda conversion
lambda(e:*cuddle(e,_1535))
[12] from [3]: Lambda conversion
lambda(n:lambda(_1535:^n(^lambda(e:*cuddle(e,_1535)))))

[13] from [0,12]: Construction by T57 =>
eval(lambda(n:lambda(_1535:^n(^lambda(e:*cuddle(e,_1535))))),^lambda(p:^p(lucy)))
[14] from [13]: Instantiate variable
eval(^^lambda(p:^p(lucy)),^lambda(e:*cuddle(e,_1535)))
[15] from [14]: Down-up conversion
eval(lambda(p:^p(lucy)),^lambda(e:*cuddle(e,_1535)))
[16] from [15]: Instantiate variable
eval(^^lambda(e:*cuddle(e,_1535)),lucy)
[17] from [16]: Down-up conversion
eval(lambda(e:*cuddle(e,_1535)),lucy)
[18] from [17]: Lambda conversion
*cuddle(lucy,_1535)
[19] from [15]: Lambda conversion
*cuddle(lucy,_1535)
[20] from [13]: Lambda conversion
lambda(_1535:*cuddle(lucy,_1535))

[21] from Lexicon: Basic expression [mary] =>
lambda(p:^p(mary))

[22] from Lexicon: Basic expression [kiss] =>
lambda(n:lambda(e:kiss(e,n)))

[23] from Lexicon: Basic expression [passive] =>
lambda(b:lambda(n:lambda(_2861:^n(^^b(^lambda(p:^p(_2861)))))))

[24] from [22,23]: Construction by T59 =>
eval(lambda(b:lambda(n:lambda(_2861:^n(^^b(^lambda(p:^p(_2861)))))))),^lambda(n:lambda(e:kiss(e,n))))
[25] from [24]: Instantiate variable
eval(^^lambda(n:lambda(e:kiss(e,n))),^lambda(p:^p(_2861)))
[26] from [25]: Down-up conversion
eval(lambda(n:lambda(e:kiss(e,n))),^lambda(p:^p(_2861)))
[27] from [26]: Postulate 4 =>
lambda(e:eval(lambda(p:^p(_2861)),^lambda(_3309:*kiss(e,_3309))))
[28] from [27]: Instantiate variable
eval(^^lambda(_3309:*kiss(e,_3309)),_2861)
[29] from [28]: Down-up conversion
eval(lambda(_3309:*kiss(e,_3309)),_2861)
[30] from [29]: Lambda conversion
*kiss(e,_2861)
[31] from [27]: Lambda conversion
*kiss(e,_2861)
[32] from [26]: Lambda conversion
lambda(e:*kiss(e,_2861))

[33] from [24]: Lambda conversion
lambda(n:lambda(_2861:'n(^lambda(e:*kiss(e,_2861)))))

[34] from [21,33]: Construction by T57 =>
eval(lambda(n:lambda(_2861:'n(^lambda(e:*kiss(e,_2861))))),^lambda(p:'p(mary)))
[35] from [34]: Instantiate variable
eval('^lambda(p:'p(mary)),^lambda(e:*kiss(e,_2861)))
[36] from [35]: Down-up conversion
eval(lambda(p:'p(mary)),^lambda(e:*kiss(e,_2861)))
[37] from [36]: Instantiate variable
eval('^lambda(e:*kiss(e,_2861)),mary)
[38] from [37]: Down-up conversion
eval(lambda(e:*kiss(e,_2861)),mary)
[39] from [38]: Lambda conversion
*kiss(mary,_2861)
[40] from [36]: Lambda conversion
*kiss(mary,_2861)
[41] from [34]: Lambda conversion
lambda(_2861:*kiss(mary,_2861))

[42] from [20,41]: Construction by T72 =>
lambda(e:(eval(lambda(_2861:*kiss(mary,_2861)),e)@eval(lambda(_1535:*cuddle(lucy,_1535)),e)))
[43] from [42]: Lambda conversion
*kiss(mary,e)
[44] from [42]: Lambda conversion
*cuddle(lucy,e)

[45] from Lexicon: Basic expression [be] =>
lambda(p:lambda(e:'p(e)))

[46] from [44,45]: Construction by T46 =>
eval(lambda(p:lambda(e:'p(e))),^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))))
[47] from [46]: Instantiate variable
eval('^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))),e)
[48] from [47]: Down-up conversion
eval(lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))),e)
[49] from [48]: Lambda conversion
*kiss(mary,e)@*cuddle(lucy,e)
[50] from [46]: Lambda conversion
lambda(e:(*kiss(mary,e)@*cuddle(lucy,e)))

[51] from Lexicon: Basic expression [past] =>
lambda(p:lambda(n:past('n(p))))

[52] from [50,51]: Construction by T42 =>
eval(lambda(p:lambda(n:past('n(p)))),^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))))
[53] from [52]: Lambda conversion
lambda(n:past('n(^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))))))

[54] from Lexicon: Basic expression [john] =>
lambda(p:'p(john))

[55] from [53,54]: Construction by T40 =>
eval(lambda(n:past('n(^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e)))))),^lambda(p:'p(john)))
[56] from [55]: Instantiate variable

eval(^lambda(p:^p(john)),^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))))

[57] from [56]: Down-up conversion

eval(lambda(p:^p(john)),^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))))

[58] from [57]: Instantiate variable

eval(^lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))),john)

[59] from [58]: Down-up conversion

eval(lambda(e:(*kiss(mary,e)@*cuddle(lucy,e))),john)

[60] from [59]: Lambda conversion

*kiss(mary,john)@*cuddle(lucy,john)

[61] from [57]: Lambda conversion

*kiss(mary,john)@*cuddle(lucy,john)

[62] from [55]: Lambda conversion

past(*kiss(mary,john)@*cuddle(lucy,john))

Logical Form

****

past(*kiss(mary,john)@*cuddle(lucy,john))

(H-2) John was kissed tenderly by Lucy.

Analysis Tree
****

```
#40:1 john was kissed tenderly by lucy
  #1:= john
  #42:0 was kissed tenderly by lucy
    #0:= past
    #46:0 be kissed tenderly by lucy
      #1:= be
      #57:0 kissed tenderly by lucy
        #76:22 kissed tenderly
          #1:= tenderly
        #59:0 kissed
          #0:= passive
          #1:= kiss
      #1:= lucy
```

Composition & Simplification
****

```
[0] from Lexicon: Basic expression [lucy] =>
        lambda(p:'p(lucy))

[1] from Lexicon: Basic expression [kiss] =>
        lambda(n:lambda(e:kiss(e,n)))

[2] from Lexicon: Basic expression [passive] =>
        lambda(b:lambda(n:lambda(_1185:'n(^'b(^lambda(p:'p(_1185)))))))

[3] from [1,2]: Construction by T59 =>
        eval(lambda(b:lambda(n:lambda(_1185:'n(^'b(^lambda(p:'p(_1185)))))),^lambda(n:lambda(e:kiss(e,n))))
[4] from [3]: Instantiate variable
        eval('^lambda(n:lambda(e:kiss(e,n))),^lambda(p:'p(_1185)))
[5] from [4]: Down-up conversion
        eval(lambda(n:lambda(e:kiss(e,n))),^lambda(p:'p(_1185)))
[6] from [5]: Postulate 4 =>
        lambda(e:eval(lambda(p:'p(_1185)),^lambda(_1633:*kiss(e,_1633))))
[7] from [6]: Instantiate variable
        eval('^lambda(_1633:*kiss(e,_1633)),_1185)
[8] from [7]: Down-up conversion
        eval(lambda(_1633:*kiss(e,_1633)),_1185)
[9] from [8]: Lambda conversion
        *kiss(e,_1185)
[10] from [6]: Lambda conversion
        *kiss(e,_1185)
[11] from [5]: Lambda conversion
        lambda(e:*kiss(e,_1185))
[12] from [3]: Lambda conversion
        lambda(n:lambda(_1185:'n(^lambda(e:*kiss(e,_1185)))))

[13] from Lexicon: Basic expression [tenderly] =>
        lambda(p:lambda(e:tenderly(e,p)))
```

[14] from [12,13]: Construction by T76 =>
lambda(n:lambda(_1185:'n(eval(^lambda(p:lambda(e:tenderly(e,p))),^lambda(_2044:eval(lambda(e:*kiss(e,_1185)),_2044))))))
[15] from [14]: Lambda conversion
*kiss(_2044,_1185)
[16] from [14]: Lambda conversion
lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185))))

[17] from [0,16]: Construction by T57 =>
eval(lambda(n:lambda(_1185:'n(^lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185)))))),^lambda(p:'p(lucy)))
[18] from [17]: Instantiate variable
eval('^lambda(p:'p(lucy)),^lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185)))))
[19] from [18]: Down-up conversion
eval(lambda(p:'p(lucy)),^lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185)))))
[20] from [19]: Instantiate variable
eval('^lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185)))),lucy)
[21] from [20]: Down-up conversion
eval(lambda(e:tenderly(e,^lambda(_2044:*kiss(_2044,_1185)))),lucy)
[22] from [21]: Lambda conversion
tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185)))
[23] from [19]: Lambda conversion
tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185)))
[24] from [17]: Lambda conversion
lambda(_1185:tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185))))

[25] from Lexicon: Basic expression [be] =>
lambda(p:lambda(e:'p(e)))

[26] from [24,25]: Construction by T46 =>
eval(lambda(p:lambda(e:'p(e))),^lambda(_1185:tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185)))))
[27] from [26]: Instantiate variable
eval('^lambda(_1185:tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185)))),e)
[28] from [27]: Down-up conversion
eval(lambda(_1185:tenderly(lucy,^lambda(_2044:*kiss(_2044,_1185)))),e)
[29] from [28]: Lambda conversion
tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))
[30] from [26]: Lambda conversion
lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e))))

[31] from Lexicon: Basic expression [past] =>
lambda(p:lambda(n:past('n(p))))

[32] from [30,31]: Construction by T42 =>
eval(lambda(p:lambda(n:past('n(p)))),^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))))
[33] from [32]: Lambda conversion
lambda(n:past('n(^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))))))

[34] from Lexicon: Basic expression [john] =>
lambda(p:'p(john))

[35] from [33,34]: Construction by T40 =>
eval(lambda(n:past('n(^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))))),^lambda(p:'p(john)))
[36] from [35]: Instantiate variable
eval('^lambda(p:'p(john)),^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))))
[37] from [36]: Down-up conversion
eval(lambda(p:'p(john)),^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))))

[38] from [37]: Instantiate variable
eval(^lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))),john)
[39] from [38]: Down-up conversion
eval(lambda(e:tenderly(lucy,^lambda(_2044:*kiss(_2044,e)))),john)
[40] from [39]: Lambda conversion
tenderly(lucy,^lambda(_2044:*kiss(_2044,john)))
[41] from [37]: Lambda conversion
tenderly(lucy,^lambda(_2044:*kiss(_2044,john)))
[42] from [35]: Lambda conversion
past(tenderly(lucy,^lambda(_2044:*kiss(_2044,john))))

Logical Form
****

past(tenderly(lucy,^lambda(_2044:*kiss(_2044,john))))

(H-3) Mary believes that John shaves himself.

Analysis Tree
****

#40:1 mary believes that john shaves himself
#1:= mary
#41:0 believes that john shaves himself
#0:= present
#61:20 believe that john shaves himself
#1:= believe
#14:2:0 john shaves himself
#1:= john
#40:1 he0 shaves himself0
#1:= he0
#41:0 shaves himself0
#0:= present
#60:0 shave himself0
#1:= shave
#1:= he0

Composition & Simplification
****

[0] from Lexicon: Basic expression [he0] =>
lambda(p:^p(x0))

[1] from Lexicon: Basic expression [shave] =>
lambda(n:lambda(e:shave(e,n)))

[2] from [0,1]: Construction by T60 =>
eval(lambda(n:lambda(e:shave(e,n))),^lambda(p:^p(x0)))
[3] from [2]: Postulate 4 =>
lambda(e:eval(lambda(p:^p(x0)),^lambda(_1737:*shave(e,_1737))))
[4] from [3]: Instantiate variable
eval(^lambda(_1737:*shave(e,_1737)),x0)
[5] from [4]: Down-up conversion
eval(lambda(_1737:*shave(e,_1737)),x0)
[6] from [5]: Lambda conversion
*shave(e,x0)
[7] from [3]: Lambda conversion
*shave(e,x0)
[8] from [2]: Lambda conversion
lambda(e:*shave(e,x0))

[9] from Lexicon: Basic expression [present] =>
lambda(p:lambda(n:present(^n(p))))

[10] from [8,9]: Construction by T41 =>
eval(lambda(p:lambda(n:present(^n(p)))),^lambda(e:*shave(e,x0)))
[11] from [10]: Lambda conversion
lambda(n:present(^n(^lambda(e:*shave(e,x0)))))

[12] from Lexicon: Basic expression [he0] =>

lambda(p:'p(x0))

[13] from [11,12]: Construction by T40 =>
eval(lambda(n:present('n(^lambda(e:*shave(e,x0))))),^lambda(p:'p(x0)))
[14] from [13]: Instantiate variable
eval('^lambda(p:'p(x0)),^lambda(e:*shave(e,x0)))
[15] from [14]: Down-up conversion
eval(lambda(p:'p(x0)),^lambda(e:*shave(e,x0)))
[16] from [15]: Instantiate variable
eval('^lambda(e:*shave(e,x0)),x0)
[17] from [16]: Down-up conversion
eval(lambda(e:*shave(e,x0)),x0)
[18] from [17]: Lambda conversion
*shave(x0,x0)
[19] from [15]: Lambda conversion
*shave(x0,x0)
[20] from [13]: Lambda conversion
present(*shave(x0,x0))

[21] from Lexicon: Basic expression [john] =>
lambda(p:'p(john))

[22] from [20,21]: Construction by T14 =>
eval(lambda(p:'p(john)),^lambda(x0:present(*shave(x0,x0))))
[23] from [22]: Instantiate variable
eval('^lambda(x0:present(*shave(x0,x0))),john)
[24] from [23]: Down-up conversion
eval(lambda(x0:present(*shave(x0,x0))),john)
[25] from [24]: Lambda conversion
present(*shave(john,john))
[26] from [22]: Lambda conversion
present(*shave(john,john))

[27] from Lexicon: Basic expression [believe] =>
lambda(r:lambda(e:believe(e,r)))

[28] from [26,27]: Construction by T61 =>
eval(lambda(r:lambda(e:believe(e,r))),^exists(t,ab(t,present(*shave(john,john)))))
[29] from [28]: Lambda conversion
lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john))))))

[30] from Lexicon: Basic expression [present] =>
lambda(p:lambda(n:present('n(p))))

[31] from [29,30]: Construction by T41 =>
eval(lambda(p:lambda(n:present('n(p)))),^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john)))))))
[32] from [31]: Lambda conversion
lambda(n:present('n(^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john))))))))

[33] from Lexicon: Basic expression [mary] =>
lambda(p:'p(mary))

[34] from [32,33]: Construction by T40 =>
eval(lambda(n:present('n(^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john))))))))),^lambda(p:'p(mary)))
[35] from [34]: Instantiate variable

```
        eval(^lambda(p:^p(mary)),^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john)))))))
[36] from [35]: Down-up conversion
        eval(lambda(p:^p(mary)),^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john)))))))
[37] from [36]: Instantiate variable
        eval(^lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john)))))),mary)
[38] from [37]: Down-up conversion
        eval(lambda(e:believe(e,^exists(t,ab(t,present(*shave(john,john)))))),mary)
[39] from [38]: Lambda conversion
        believe(mary,^exists(t,ab(t,present(*shave(john,john)))))
[40] from [36]: Lambda conversion
        believe(mary,^exists(t,ab(t,present(*shave(john,john)))))
[41] from [34]: Lambda conversion
        present(believe(mary,^exists(t,ab(t,present(*shave(john,john))))))

Logical Form
************

present(believe(mary,^exists(t,ab(t,present(*shave(john,john))))))
```

(H-4) Mary loves John and he loves himself.

Analysis Tree
****

#14:2:1 mary loves john and he loves himself
  #1:= john
  #11:11 mary loves him1 and he1 loves himself1
    #40:1 mary loves him1
      #1:= mary
      #41:0 loves him1
        #0:= present
        #60:0 love him1
          #1:= love
          #1:= he1
    #40:1 he1 loves himself1
      #1:= he1
      #41:0 loves himself1
        #0:= present
        #60:0 love himself1
          #1:= love
          #1:= he1

Composition & Simplification
****

[0] from Lexicon: Basic expression [he1] =>
lambda(p:'p(x1))

[1] from Lexicon: Basic expression [love] =>
lambda(n:lambda(e:love(e,n)))

[2] from [0,1]: Construction by T60 =>
eval(lambda(n:lambda(e:love(e,n))),^lambda(p:'p(x1)))
[3] from [2]: Postulate 4 =>
lambda(e:eval(lambda(p:'p(x1)),^lambda(_1985:*love(e,_1985))))
[4] from [3]: Instantiate variable
eval('^lambda(_1985:*love(e,_1985)),x1)
[5] from [4]: Down-up conversion
eval(lambda(_1985:*love(e,_1985)),x1)
[6] from [5]: Lambda conversion
*love(e,x1)
[7] from [3]: Lambda conversion
*love(e,x1)
[8] from [2]: Lambda conversion
lambda(e:*love(e,x1))

[9] from Lexicon: Basic expression [present] =>
lambda(p:lambda(n:present('n(p))))

[10] from [8,9]: Construction by T41 =>
eval(lambda(p:lambda(n:present('n(p)))),^lambda(e:*love(e,x1)))
[11] from [10]: Lambda conversion
lambda(n:present('n(^lambda(e:*love(e,x1)))))

[12] from Lexicon: Basic expression [he1] =>
lambda(p:'p(x1))

[13] from [11,12]: Construction by T40 =>
eval(lambda(n:present('n(^lambda(e:*love(e,x1))))),^lambda(p:'p(x1)))
[14] from [13]: Instantiate variable
eval('^lambda(p:'p(x1)),^lambda(e:*love(e,x1)))
[15] from [14]: Down-up conversion
eval(lambda(p:'p(x1)),^lambda(e:*love(e,x1)))
[16] from [15]: Instantiate variable
eval('^lambda(e:*love(e,x1)),x1)
[17] from [16]: Down-up conversion
eval(lambda(e:*love(e,x1)),x1)
[18] from [17]: Lambda conversion
*love(x1,x1)
[19] from [15]: Lambda conversion
*love(x1,x1)
[20] from [13]: Lambda conversion
present(*love(x1,x1))

[21] from Lexicon: Basic expression [he1] =>
lambda(p:'p(x1))

[22] from Lexicon: Basic expression [love] =>
lambda(n:lambda(e:love(e,n)))

[23] from [21,22]: Construction by T60 =>
eval(lambda(n:lambda(e:love(e,n))),^lambda(p:'p(x1)))
[24] from [23]: Postulate 4 =>
lambda(e:eval(lambda(p:'p(x1)),^lambda(_3206:*love(e,_3206))))
[25] from [24]: Instantiate variable
eval('^lambda(_3206:*love(e,_3206)),x1)
[26] from [25]: Down-up conversion
eval(lambda(_3206:*love(e,_3206)),x1)
[27] from [26]: Lambda conversion
*love(e,x1)
[28] from [24]: Lambda conversion
*love(e,x1)
[29] from [23]: Lambda conversion
lambda(e:*love(e,x1))

[30] from Lexicon: Basic expression [present] =>
lambda(p:lambda(n:present('n(p))))

[31] from [29,30]: Construction by T41 =>
eval(lambda(p:lambda(n:present('n(p)))),^lambda(e:*love(e,x1)))
[32] from [31]: Lambda conversion
lambda(n:present('n(^lambda(e:*love(e,x1)))))

[33] from Lexicon: Basic expression [mary] =>
lambda(p:'p(mary))

[34] from [32,33]: Construction by T40 =>
eval(lambda(n:present('n(^lambda(e:*love(e,x1))))),^lambda(p:'p(mary)))
[35] from [34]: Instantiate variable

eval(^lambda(p:^p(mary)),^lambda(e:*love(e,x1)))

[36] from [35]: Down-up conversion

eval(lambda(p:^p(mary)),^lambda(e:*love(e,x1)))

[37] from [36]: Instantiate variable

eval(^lambda(e:*love(e,x1)),mary)

[38] from [37]: Down-up conversion

eval(lambda(e:*love(e,x1)),mary)

[39] from [38]: Lambda conversion

*love(mary,x1)

[40] from [36]: Lambda conversion

*love(mary,x1)

[41] from [34]: Lambda conversion

present(*love(mary,x1))

[42] from [20,41]: Construction by T11 =>

present(*love(mary,x1))@present(*love(x1,x1))

[43] from Lexicon: Basic expression [john] =>

lambda(p:^p(john))

[44] from [42,43]: Construction by T14 =>

eval(lambda(p:^p(john)),^lambda(x1:(present(*love(mary,x1))@present(*love(x1,x1)))))

[45] from [44]: Instantiate variable

eval(^lambda(x1:(present(*love(mary,x1))@present(*love(x1,x1)))),john)

[46] from [45]: Down-up conversion

eval(lambda(x1:(present(*love(mary,x1))@present(*love(x1,x1)))),john)

[47] from [46]: Lambda conversion

present(*love(mary,john))@present(*love(john,john))

[48] from [44]: Lambda conversion

present(*love(mary,john))@present(*love(john,john))

Logical Form

****

present(*love(mary,john))@present(*love(john,john))

```
/* ######################################## */
/* SAMPLE PARSES AND LOGICAL FORMS */
/* ######################################## */
```

(H-5) John hopes that Mary believes that he shaves himself.

Analysis Tree
****

```
#14:2:0 john hopes that mary believes that he shaves himself
  #1:= john
  #40:1 he0 hopes that mary believes that he0 shaves himself0
    #1:= he0
    #41:0 hopes that mary believes that he0 shaves himself0
      #0:= present
      #61:20 hope that mary believes that he0 shaves himself0
        #1:= hope
        #14:2:1 mary believes that he0 shaves himself0
          #1:= mary
          #40:1 she1 believes that he0 shaves himself0
            #1:= he1
            #41:0 believes that he0 shaves himself0
              #0:= present
              #61:20 believe that he0 shaves himself0
                #1:= believe
                #40:1 he0 shaves himself0
                  #1:= he0
                  #41:0 shaves himself0
                    #0:= present
                    #60:0 shave himself0
                      #1:= shave
                      #1:= he0
```

Logical Form
****

```
present(hope(john,^exists(t,ab(t,present(believe(mary,
            ^exists(t,ab(t,present(*shave(john,john))))))))))
```

(H-6) John wishes to date a girl and marry her.

Analysis Tree
****

#40:1 john wishes to date a girl and marry her
#1:= john
#41:0 wishes to date a girl and marry her
#0:= present
#62:21 wish to date a girl and marry her
#1:= wish
#16:4:3 date a girl and marry her
#2:0 a girl
#1:= a
#1:= girl
#71:5 date her3 and marry her3
#60:0 date her3
#1:= date
#1:= he3
#60:0 marry her3
#1:= marry
#1:= he3

Logical Form
****

present(wish(john,^lambda(_3369:exists(_2539:(girl(_2539)@*date(_3369,_2539)@*marry(_3369,_2539))))))

Analysis Tree
****

#14:2:3 john wishes to date a girl and marry her
#2:0 a girl
#1:= a
#1:= girl
#40:1 john wishes to date her3 and marry her3
#1:= john
#41:0 wishes to date her3 and marry her3
#0:= present
#62:21 wish to date her3 and marry her3
#1:= wish
#71:5 date her3 and marry her3
#60:0 date her3
#1:= date
#1:= he3
#60:0 marry her3
#1:= marry
#1:= he3

Logical Form
****

exists(_3476:(girl(_3476)@present(wish(john,^lambda(_2503:(*date(_2503,_3476)@*marry(_2503,_3476)))))))

(H-7) Lucy wishes to be dated by a man and loved by him.

Analysis Tree
****

```
#40:1 lucy wishes to be dated by a man and loved by him
  #1:= lucy
  #41:0 wishes to be dated by a man and loved by him
    #0:= present
    #62:21 wish to be dated by a man and loved by him
      #1:= wish
      #46:0 be dated by a man and loved by him
        #1:= be
        #18:4:1 dated by a man and loved by him
          #2:0 a man
            #1:= a
            #1:= man
          #72:5 dated by him1 and loved by him1
            #57:0 dated by him1
              #59:0 dated
                #0:= passive
                #1:= date
              #1:= he1
            #57:0 loved by him1
              #59:0 loved
                #0:= passive
                #1:= love
              #1:= he1
```

Logical Form
****

```
present(wish(lucy,^lambda(_5312:exists(_4093:(man(_4093)
      @*date(_4093,_5312)@*love(_4093,_5312))))))
```

(H-8) Lucy has married a man secretly and born a son.

Analysis Tree
****

```
#40:1 lucy has married a man secretly and born a son
  #1:= lucy
  #41:0 has married a man secretly and born a son
    #0:= present
    #54:0 have married a man secretly and born a son
      #1:= have
      #71:5 marry a man secretly and bear a son
        #74:21 marry a man secretly
          #1:= secretly
          #60:0 marry a man
            #1:= marry
            #2:0 a man
              #1:= a
              #1:= man
        #60:0 bear a son
          #1:= bear
          #2:0 a son
            #1:= a
            #1:= son
```

Logical Form
****

```
present(perfect(secretly(lucy,^lambda(_3365:exists(_2585:(man(_2585)
    @*marry(_3365,_2585)))))@exists(_1764:(son(_1764)@*bear(lucy,_1764)))))
```

(H-9) John was kissed by Mary tenderly and cuddled by Emily passionately.

Analysis Tree
****

```
#40:1 john was kissed by mary tenderly and cuddled by emily passionately
  #1:= john
  #42:0 was kissed by mary tenderly and cuddled by emily passionately
    #0:= past
    #46:0 be kissed by mary tenderly and cuddled by emily passionately
      #1:= be
      #72:5 kissed by mary tenderly and cuddled by emily passionately
        #75:21 kissed by mary tenderly
          #1:= tenderly
          #57:0 kissed by mary
            #59:0 kissed
              #0:= passive
              #1:= kiss
            #1:= mary
        #75:21 cuddled by emily passionately
          #1:= passionately
          #57:0 cuddled by emily
            #59:0 cuddled
              #0:= passive
              #1:= cuddle
            #1:= emily
```

Logical Form
****

```
past(tenderly(john,^lambda(_3899:*kiss(mary,_3899)))
    @passionately(john,^lambda(_2793:*cuddle(emily,_2793))))
```

(H-10) John was kissed tenderly and cuddled passionately by Lucy.

Analysis Tree
****

```
#14:2:0 john was kissed tenderly and cuddled passionately by lucy
  #1:= john
  #40:1 he0 was kissed tenderly and cuddled passionately by lucy
    #1:= he0
    #42:0 was kissed tenderly and cuddled passionately by lucy
      #0:= past
      #46:0 be kissed tenderly and cuddled passionately by lucy
        #1:= be
        #72:5 kissed tenderly and cuddled passionately by lucy
          #75:21 kissed tenderly
            #1:= tenderly
            #58:0 kissed
              #59:0 kissed
                #0:= passive
                #1:= kiss
              #0:= agent
          #57:0 cuddled passionately by lucy
            #76:22 cuddled passionately
              #1:= passionately
              #59:0 cuddled
                #0:= passive
                #1:= cuddle
            #1:= lucy
```

Logical Form
****

```
past(tenderly(john,^lambda(_5220:exists(_3800:*kiss(_3800,_5220))))
                @passionately(lucy,^lambda(_2939:*cuddle(_2939,john))))
```

/* Note: two passive intransitive verb phrases, the first agentless, conjoined */

(H-11) John was kissed tenderly and cuddled by Lucy.

Analysis Tree
****

```
#40:1 john was kissed tenderly and cuddled by lucy
  #1:= john
  #42:0 was kissed tenderly and cuddled by lucy
    #0:= past
    #46:0 be kissed tenderly and cuddled by lucy
      #1:= be
      #57:0 kissed tenderly and cuddled by lucy
        #73:9 kissed tenderly and cuddled
          #76:22 kissed tenderly
            #1:= tenderly
            #59:0 kissed
              #0:= passive
              #1:= kiss
          #59:0 cuddled
            #0:= passive
            #1:= cuddle
          #1:= lucy
```

Logical Form
****

```
past(tenderly(lucy,^lambda(_2721:*kiss(_2721,john)))@*cuddle(lucy,john))
```

/* Note: two passive transitive verb phrases conjoined */

(H-12) Mary will not be being dated by John tomorrow.

Analysis Tree

****

```
#14:2:0 mary will not be being dated by john tomorrow
  #1:= mary
  #40:1 she0 will not be being dated by john tomorrow
    #1:= he0
    #77:1 will not be being dated by john tomorrow
      #41:0 will not be being dated by john
        #0:= present
        #49:15 will not be being dated by john
          #1:= will
          #56:0 be being dated by john
            #1:= be
            #46:0 be dated by john
              #1:= be
              #57:0 dated by john
                #59:0 dated
                  #0:= passive
                  #1:= date
                #1:= john
        #1:= tomorrow
```

Logical Form

****

```
at(t*,present(~future(progressive(*date(john,mary))))&incl(tomorrow,t*))
```

(H-13) John cuddled Emily yesterday and will kiss Lucy tomorrow.

Analysis Tree
****

```
#14:2:0 john cuddled emily yesterday and will kiss lucy tomorrow
  #1:= john
  #40:1 he0 cuddled emily yesterday and will kiss lucy tomorrow
    #1:= he0
    #70:7 cuddled emily yesterday and will kiss lucy tomorrow
      #77:1 cuddled emily yesterday
        #42:0 cuddled emily
          #0:= past
          #60:0 cuddle emily
            #1:= cuddle
            #1:= emily
        #1:= yesterday
      #77:1 will kiss lucy tomorrow
        #41:0 will kiss lucy
          #0:= present
          #50:0 will kiss lucy
            #1:= will
            #60:0 kiss lucy
              #1:= kiss
              #1:= lucy
          #1:= tomorrow
```

Logical Form
****

```
at(t*,past(*cuddle(john,emily))&incl(yesterday,t*))
        @at(t*,present(future(*kiss(john,lucy)))&incl(tomorrow,t*))
```

(H-14) A man who was a traitor will die.

Analysis Tree
****

#40:1 a man who was a traitor will die
#2:0 a man who was a traitor
#1:= a
#31:3:1 man who was a traitor
#1:= man
#40:1 he1 was a traitor
#1:= he1
#42:0 was a traitor
#0:= past
#64:0 be a traitor
#1:= be
#2:0 a traitor
#1:= a
#1:= traitor
#41:0 will die
#0:= present
#50:0 will die
#1:= will
#1:= die

Logical Form
****

present(exists(_3603:((man(_3603)&exists(t:ab(t,past(traitor(_3603)))))
@future(die(_3603)))))

(H-15) Scott was the man who wrote Waverley.

Analysis Tree
****

#40:1 scott was the man who wrote waverley
#1:= scott
#42:0 was the man who wrote waverley
#0:= past
#64:0 be the man who wrote waverley
#1:= be
#2:0 the man who wrote waverley
#1:= the
#31:3:1 man who wrote waverley
#1:= man
#40:1 he1 wrote waverley
#1:= he1
#42:0 wrote waverley
#0:= past
#60:0 write waverley
#1:= write
#1:= waverley

Logical Form
****

past(exists(_2726:all(_2727:(man(_2727)
&exists(t:ab(t,past(*write(_2727,waverley))))<=>equals(_2727,_2726))
@equals(scott,_2726))))

(H-16) The man who loves her finds Mary.

Analysis Tree
****

#14:2:2 the man who loves her finds mary
#1:= mary
#40:1 the man who loves HER2 finds her2
  #2:0 the man who loves HER2
    #1:= the
    #31:3:1 man who loves HER2
      #1:= man
      #40:1 he1 loves HER2
        #1:= he1
        #41:0 loves HER2
          #0:= present
          #60:0 love HER2
            #1:= love
            #1:= he2
  #41:0 finds her2
    #0:= present
    #60:0 find her2
      #1:= find
      #1:= he2

Logical Form
****

present(exists(_3453:all(_3454:(man(_3454)
    &exists(t:ab(t,present(*love(_3454,mary))))<=>equals(_3454,_3453))
      @*find(_3453,mary))))

(H-17) Mary believed that Lucy would be kissed by Bill tomorrow.

Analysis Tree
****

#14:2:0 mary believed that lucy would be kissed by bill tomorrow
#1:= mary
#40:1 she0 believed that lucy would be kissed by bill tomorrow
#1:= he0
#77:1 believed that lucy would be kissed by bill tomorrow
#42:0 believed that lucy would be kissed by bill
#0:= past
#61:20 believe that lucy would be kissed by bill
#1:= believe
#14:2:1 lucy would be kissed by bill
#1:= lucy
#40:1 she1 would be kissed by bill
#1:= he1
#42:0 would be kissed by bill
#0:= past
#50:0 will be kissed by bill
#1:= will
#46:0 be kissed by bill
#1:= be
#57:0 kissed by bill
#59:0 kissed
#0:= passive
#1:= kiss
#1:= bill
#1:= tomorrow

Logical Form
****

at(t*,past(believe(mary,^exists(t,ab(t,past(future(*kiss(bill,lucy)))))))&incl(tomorrow,t*))

(H-18) John would not have been going to have been kissed by Emily yesterday.

Analysis Tree
****

#14:2:0 john would not have been going to have been kissed by emily yesterday
#1:= john
#40:1 he0 would not have been going to have been kissed by emily yesterday
#1:= he0
#77:1 would not have been going to have been kissed by emily yesterday
#42:0 would not have been going to have been kissed by emily
#0:= past
#49:15 will not have been going to have been kissed by emily
#1:= will
#54:0 have been going to have been kissed by emily
#1:= have
#56:0 be going to have been kissed by emily
#1:= be
#48:0 go to have been kissed by emily
#1:= go
#54:0 have been kissed by emily
#1:= have
#46:0 be kissed by emily
#1:= be
#57:0 kissed by emily
#59:0 kissed
#0:= passive
#1:= kiss
#1:= emily
#1:= yesterday

Logical Form
****

at(t*,past(~future(perfect(progressive(future(perfect(*kiss(emily,john)))))))&incl(yesterday,t*))

(H-19) John knows which student deserves which grade.

Analysis Tree
****

```
#14:2:0 john knows which student deserves which grade
  #1:= john
  #40:1 he0 knows which student deserves which grade
    #1:= he0
    #41:0 knows which student deserves which grade
      #0:= present
      #63:0 know which student deserves which grade
        #1:= know
        #25:18:2 which student deserves which grade
          #29:0 which grade
            #1:= which
            #1:= grade
          #24:18:1 which student deserves it2
            #29:0 which student
              #1:= which
              #1:= student
            #21:0 ? he1 deserves it2
              #0:= ?
              #40:1 he1 deserves it2
                #1:= he1
                #41:0 deserves it2
                  #0:= present
                  #60:0 deserve it2
                    #1:= deserve
                    #1:= he2
```

Logical Form
****

```
present(know(john,^lambda(p:exists(_5492:(grade(_5492)@exists(_4168:(student(_4168)
            @(^p&equals(p,^present(*deserve(_4168,_5492)))))))))))
```

(H-20) John knew whether Lucy would have been kissed by Bill.

Analysis Tree
****

```
#14:2:0 john knew whether lucy would have been kissed by bill
  #1:= john
  #40:1 he0 knew whether lucy would have been kissed by bill
    #1:= he0
    #42:0 knew whether lucy would have been kissed by bill
      #0:= past
      #63:0 know whether lucy would have been kissed by bill
        #1:= know
        #20:0 whether lucy would have been kissed by bill
          #0:= ?yn
          #14:2:3 lucy would have been kissed by bill
            #1:= lucy
            #40:1 she3 would have been kissed by bill
              #1:= he3
              #42:0 would have been kissed by bill
                #0:= past
                #50:0 will have been kissed by bill
                #1:= will
                #54:0 have been kissed by bill
                  #1:= have
                  #46:0 be kissed by bill
                    #1:= be
                    #57:0 kissed by bill
                      #59:0 kissed
                        #0:= passive
                        #1:= kiss
                      #1:= bill
```

Logical Form
****

```
past(know(john,^lambda(r:('r&equals(r,^past(future(perfect(*kiss(bill,lucy)))))
          vequals(r,^~(past(future(perfect(*kiss(bill,lucy))))))))))
```

(H-21) John knows whether Mary walks or Emily runs or Lucy swims.

Analysis Tree
****

#40:1 john knows whether mary walks or emily runs or lucy swims
#1:= john
#41:0 knows whether mary walks or emily runs or lucy swims
#0:= present
#63:0 know whether mary walks or emily runs or lucy swims
#1:= know
#22:17 whether mary walks or emily runs or lucy swims
#21:0 ? mary walks
#0:= ?
#40:1 mary walks
#1:= mary
#41:0 walks
#0:= present
#1:= walk
#23:17
#21:0 ? emily runs
#0:= ?
#40:1 emily runs
#1:= emily
#41:0 runs
#0:= present
#1:= run
#21:0 ? lucy swims
#0:= ?
#40:1 lucy swims
#1:= lucy
#41:0 swims
#0:= present
#1:= swim

Logical Form
****

present(know(john,^lambda(r:((‘r&equals(r,^present(walk(mary))))
v(‘r&equals(r,^present(run(emily))))
v‘r&equals(r,^present(swim(lucy)))))))

(H-22) John walks in the park and talks about a unicorn.

Analysis Tree
****

#40:1 john walks in the park and talks about a unicorn
#1:= john
#70:7 walks in the park and talks about a unicorn
#41:0 walks in the park
#0:= present
#74:21 walk in the park
#78:0 in the park
#1:= in
#2:0 the park
#1:= the
#1:= park
#1:= walk
#41:0 talks about a unicorn
#0:= present
#74:21 talk about a unicorn
#78:0 about a unicorn
#1:= about
#2:0 a unicorn
#1:= a
#1:= unicorn
#1:= talk

Logical Form
****

present(exists(_3432:all(_3433:(park(_3433)<=>equals(_3433,_3432))
@*in(john,^lambda(_3911:walk(_3911)),_3432))))
@present(about(john,^lambda(_2193:talk(_2193)),^lambda(q:exists(_1852:(unicorn(_1852)
@'q(_1852))))))

(H-23) John or Mary catches a fish and whichever eats it.

Analysis Tree
****

#14:2:12 john or mary catches a fish and whichever eats it
#2:0 a fish
#1:= a
#1:= fish
#14:2:9 john or mary catches it12 and whichever eats it12
#13:14 john or mary
#1:= john
#1:= mary
#11:11 he9 catches it12 and he9 eats it12
#40:1 he9 catches it12
#1:= he9
#41:0 catches it12
#0:= present
#60:0 catch it12
#1:= catch
#1:= he12
#40:1 he9 eats it12
#1:= he9
#41:0 eats it12
#0:= present
#60:0 eat it12
#1:= eat
#1:= he12

Logical Form
****

exists(_4902:(fish(_4902)@(present(*catch(john,_4902))@present(*eat(john,_4902))
vpresent(*catch(mary,_4902))@present(*eat(mary,_4902)))))

(H-24) A cow or the horses are running.

Analysis Tree
****

#40:1 a cow or the horses are running
#13:14 a cow or the horses
#2:0 a cow
#1:= a
#1:= cow
#2:0 the horses
#1:= the
#1:= horse
#41:0 are running
#0:= present
#56:0 be running
#1:= be
#1:= run

Logical Form
****

present(exists(_2171:(cow(_2171)@progressive(run(_2171))))
vexists(w:all(_1646:('w(_1646)=>horse(_1646)@progressive(run(_1646))))))

(H-25) The horses or the cow is running.

Analysis Tree
****

#40:1 the horses or a cow is running
#13:14 the horses or a cow
#2:0 the horses
#1:= the
#1:= horse
#2:0 a cow
#1:= a
#1:= cow
#41:0 is running
#0:= present
#56:0 be running
#1:= be
#1:= run

Logical Form
****

present(exists(w:all(_2018:('w(_2018)=>horse(_2018)@progressive(run(_2018)))))
vexists(_1642:(cow(_1642)@progressive(run(_1642)))))

(H-26) The men and the women swim.

Analysis Tree
****

#40:1 the men and the women swim
  #13:13 the men and the women
    #2:0 the men
      #1:= the
      #1:= man
    #2:0 the women
      #1:= the
      #1:= woman
  #41:0 swim
    #0:= present
    #1:= swim

Logical Form
****

present(exists(w:all(_1848:('w(_1848)=>man(_1848)@swim(_1848))))
&exists(w:all(_1323:('w(_1323)=>woman(_1323)@swim(_1323)))))

(H-27) John seeks a unicorn.

Analysis Tree
****

#40:1 john seeks a unicorn
  #1:= john
  #41:0 seeks a unicorn
    #0:= present
    #60:0 seek a unicorn
      #1:= seek
      #2:0 a unicorn
        #1:= a
        #1:= unicorn

Logical Form
****

present(try(john,^lambda(_1233:exists(_770:(unicorn(_770)@*find(_1233,_770))))))

Analysis Tree
****

#14:2:0 john seeks a unicorn
  #2:0 a unicorn
    #1:= a
    #1:= unicorn
  #40:1 john seeks it0
    #1:= john
    #41:0 seeks it0
      #0:= present
      #60:0 seek it0
        #1:= seek
        #1:= he0

Logical Form
****

exists(_2080:(unicorn(_2080)@present(try(john,^lambda(_1099:*find(_1099,_2080))))))

(H-28) Lucy knows which man loves a girl who loves him.

Analysis Tree
****

```
#40:1 lucy knows which man loves a girl who loves him
  #1:= lucy
  #41:0 knows which man loves a girl who loves him
    #0:= present
    #63:0 know which man loves a girl who loves him
      #1:= know
      #24:18:0 which man loves a girl who loves him
        #29:0 which man
          #1:= which
          #1:= man
        #21:0 ? he0 loves a girl who loves him0
          #0:= ?
          #40:1 he0 loves a girl who loves him0
            #1:= he0
            #41:0 loves a girl who loves him0
              #0:= present
              #60:0 love a girl who loves him0
                #1:= love
                #2:0 a girl who loves him0
                  #1:= a
                  #31:3:2 girl who loves him0
                    #1:= girl
                    #40:1 she2 loves him0
                      #1:= he2
                      #41:0 loves him0
                        #0:= present
                        #60:0 love him0
                          #1:= love
                          #1:= he0
```

Logical Form
****

```
present(know(lucy,^lambda(p:exists(_5827:(man(_5827)
    @('p&equals(p,^present(exists(_3662:((girl(_3662)
        &exists(t:ab(t,present(*love(_3662,_5827)))))@*love(_5827,_3662)))))))))))
```

(H-29) Lucy knows which man whom Emily loves is loved by Mary.

Analysis Tree
****

#40:1 lucy knows which man whom emily loves is loved by mary
#1:= lucy
#41:0 knows which man whom emily loves is loved by mary
#0:= present
#63:0 know which man whom emily loves is loved by mary
#1:= know
#24:18:2 which man whom emily loves is loved by mary
#29:0 which man whom emily loves
#1:= which
#31:3:1 man whom emily loves
#1:= man
#40:1 emily loves him1
#1:= emily
#41:0 loves him1
#0:= present
#60:0 love him1
#1:= love
#1:= he1
#21:0 ? he2 is loved by mary
#0:= ?
#40:1 he2 is loved by mary
#1:= he2
#41:0 is loved by mary
#0:= present
#46:0 be loved by mary
#1:= be
#57:0 loved by mary
#59:0 loved
#0:= passive
#1:= love
#1:= mary

Logical Form
****

present(know(lucy,^lambda(p:exists(_5436:((man(_5436)
&exists(t:ab(t,present(*love(emily,_5436)))))
@('p&equals(p,^present(*love(mary,_5436)))))))))

/* ################################## */
/* ALL PARSES AND LOGICAL FORMS */
/* ################################## */

(H-30) John knows which professor teaches every student.

Parse No. 1
****

#40:1 john knows which professor teaches every student
  #1:= john
  #41:0 knows which professor teaches every student
    #0:= present
    #63:0 know which professor teaches every student
      #1:= know
      #24:18:0 which professor teaches every student
        #29:0 which professor
          #1:= which
          #1:= professor
        #21:0 ? he0 teaches every student
          #0:= ?
          #40:1 he0 teaches every student
            #1:= he0
            #41:0 teaches every student
              #0:= present
              #60:0 teach every student
                #1:= teach
                #2:0 every student
                  #1:= every
                  #1:= student

Logical Form
****

present(know(john,^lambda(p:exists(_3990:(professor(_3990)
    @('p&equals(p,^present(all(_1875:~(student(_1875)
    @~*teach(_3990,_1875))))))))))

Parse No. 2
****

#40:1 john knows which professor teaches every student
  #1:= john
  #41:0 knows which professor teaches every student
    #0:= present
#63:0 know which professor teaches every student
  #1:= know
  #24:18:0 which professor teaches every student
    #29:0 which professor
      #1:= which
      #1:= professor
    #21:0 ? he0 teaches every student
      #0:= ?
      #14:2:1 he0 teaches every student
        #2:0 every student
          #1:= every
          #1:= student
        #40:1 he0 teaches him1
          #1:= he0
          #41:0 teaches him1
            #0:= present
            #60:0 teach him1
              #1:= teach
              #1:= he1

Logical Form
****

present(know(john,^lambda(p:exists(_4021:(professor(_4021)
    @('p&equals(p,^all(_2994:~(student(_2994)
        @~present(*teach(_4021,_2994)))))))))))

Parse No. 3
****

#40:1 john knows which professor teaches every student
  #1:= john
  #41:0 knows which professor teaches every student
    #0:= present
    #63:0 know which professor teaches every student
      #1:= know
      #26:19:1 which professor teaches every student
        #2:0 every student
          #1:= every
          #1:= student
        #24:18:0 which professor teaches him1
          #29:0 which professor
            #1:= which
            #1:= professor
          #21:0 ? he0 teaches him1
            #0:= ?
            #40:1 he0 teaches him1
              #1:= he0
              #41:0 teaches him1
                #0:= present
                #60:0 teach him1
                  #1:= teach
                  #1:= he1

Logical Form
****

present(know(john,^lambda(p:~(^lambda(x1:~(p,lambda(p:exists(_3249:(professor(_3249)
    @('p&equals(p,^present(*teach(_3249,x1)))))))),lambda(q:all(_4274:~(student(_4274)
    @~'q(_4274))))))))

Parse No. 4
****

#14:2:1 john knows which professor teaches every student
  #2:0 every student
    #1:= every
    #1:= student
#40:1 john knows which professor teaches him1
  #1:= john
  #41:0 knows which professor teaches him1
    #0:= present
    #63:0 know which professor teaches him1
      #1:= know
      #24:18:0 which professor teaches him1
        #29:0 which professor
          #1:= which
          #1:= professor
        #21:0 ? he0 teaches him1
          #0:= ?
          #40:1 he0 teaches him1
            #1:= he0
            #41:0 teaches him1
              #0:= present
              #60:0 teach him1
                #1:= teach
                #1:= he1

Logical Form
****

all(_5810:~(student(_5810)
    @~present(know(john,^lambda(p:exists(_3277:(professor(_3277)
      @('p&equals(p,^present(*teach(_3277,_5810))))))))))

Parse No. 5
****

#14:2:2 john knows which professor teaches every student
  #1:= john
  #40:1 he2 knows which professor teaches every student
    #1:= he2
#41:0 knows which professor teaches every student
  #0:= present
  #63:0 know which professor teaches every student
    #1:= know
    #24:18:3 which professor teaches every student
      #29:0 which professor
        #1:= which
        #1:= professor
      #21:0 ? he3 teaches every student
        #0:= ?
        #40:1 he3 teaches every student
          #1:= he3
          #41:0 teaches every student
            #0:= present
            #60:0 teach every student
              #1:= teach
              #2:0 every student
                #1:= every
                #1:= student

Logical Form
****

present(know(john,^lambda(p:exists(_4283:(professor(_4283)
    @('p&equals(p,^present(all(_2168:~(student(_2168)
      @~*teach(_4283,_2168)))))))))))

Parse No. 6
****

#14:2:2 john knows which professor teaches every student
  #1:= john
  #40:1 he2 knows which professor teaches every student
    #1:= he2
    #41:0 knows which professor teaches every student
      #0:= present
      #63:0 know which professor teaches every student
        #1:= know
        #24:18:3 which professor teaches every student
          #29:0 which professor
            #1:= which
            #1:= professor
          #21:0 ? he3 teaches every student
            #0:= ?
            #14:2:4 he3 teaches every student
              #2:0 every student
                #1:= every
                #1:= student
              #40:1 he3 teaches him4
                #1:= he3
                #41:0 teaches him4
                  #0:= present
                  #60:0 teach him4
                    #1:= teach
                    #1:= he4

Logical Form
****

present(know(john,^lambda(p:exists(_4300:(professor(_4300)
        @('p&equals(p,^all(_3273:~(student(_3273)
            @~present(*teach(_4300,_3273)))))))))))

Parse No. 7
****

#14:2:2 john knows which professor teaches every student
  #1:= john
  #40:1 he2 knows which professor teaches every student
    #1:= he2
#41:0 knows which professor teaches every student
  #0:= present
  #63:0 know which professor teaches every student
    #1:= know
    #26:19:4 which professor teaches every student
      #2:0 every student
        #1:= every
        #1:= student
      #24:18:3 which professor teaches him4
        #29:0 which professor
          #1:= which
          #1:= professor
        #21:0 ? he3 teaches him4
          #0:= ?
          #40:1 he3 teaches him4
            #1:= he3
#41:0 teaches him4
  #0:= present
  #60:0 teach him4
    #1:= teach
    #1:= he4

Logical Form
****

present(know(john,^lambda(p:~(^lambda(x4:~(p,lambda(p:exists(_3526:(professor(_3526)
    @('p&equals(p,^present(*teach(_3526,x4)))))))),lambda(q:all(_4551:~(student(_4551)
      @~'q(_4551))))))))

Parse No. 8
****

#14:2:2 john knows which professor teaches every student
  #1:= john
  #14:2:4 he2 knows which professor teaches every student
    #2:0 every student
  #1:= every
  #1:= student
#40:1 he2 knows which professor teaches him4
  #1:= he2
  #41:0 knows which professor teaches him4
    #0:= present
    #63:0 know which professor teaches him4
      #1:= know
      #24:18:3 which professor teaches him4
        #29:0 which professor
          #1:= which
          #1:= professor
        #21:0 ? he3 teaches him4
          #0:= ?
          #40:1 he3 teaches him4
            #1:= he3
            #41:0 teaches him4
              #0:= present
              #60:0 teach him4
                #1:= teach
                #1:= he4

Logical Form
****

all(_6103:~(student(_6103)
    @~present(know(john,^lambda(p:exists(_3559:(professor(_3559)
        @('p&equals(p,^present(*teach(_3559,_6103))))))))))

Parse No. 9
****

#14:2:4 john knows which professor teaches every student
#2:0 every student
#1:= every
#1:= student
#14:2:2 john knows which professor teaches him4
#1:= john
#40:1 he2 knows which professor teaches him4
#1:= he2
#41:0 knows which professor teaches him4
#0:= present
#63:0 know which professor teaches him4
#1:= know
#24:18:3 which professor teaches him4
#29:0 which professor
#1:= which
#1:= professor
#21:0 ? he3 teaches him4
#0:= ?
#40:1 he3 teaches him4
#1:= he3
#41:0 teaches him4
#0:= present
#60:0 teach him4
#1:= teach
#1:= he4

Logical Form
****

```
all(_6659:~(student(_6659)
        @~present(know(john,^lambda(p:exists(_3554:(professor(_3554)
            @('p&equals(p,^present(*teach(_3554,_6659))))))))))
```

no more parses

(H-31) A child who would be a king was born.

Parse No. 1
****

```
#40:1 a child who would be a king
  #2:0 a child who would be a king
    #1:= a
    #31:3:1 child who would be a king
      #1:= child
      #40:1 he1 would be a king
        #1:= he1
        #42:0 would be a king
          #0:= past
          #50:0 will be a king
            #1:= will
            #64:0 be a king
              #1:= be
              #2:0 a king
                #1:= a
                #1:= king
#42:0 was born
          #0:= past
          #46:0 be born
            #1:= be
            #58:0 born
              #59:0 born
                #0:= passive
                #1:= bear
              #0:= agent
```

Logical Form
****

```
past(exists(_4998:((child(_4998)&exists(t:ab(t,past(future(king(_4998))))))
      @exists(_1891:*bear(_1891,_4998)))))
```

Parse No. 2
****

#40:1 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#40:1 he1 would be a king
#1:= he1
#42:0 would be a king
#0:= past
#50:0 will be a king
#1:= will
#64:0 be a king
#1:= be
#2:0 a king
#1:= a
#1:= king
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

past(exists(_5000:((child(_5000)&exists(t:ab(t,past(future(king(_5000))))))
@exists(_1893:*bear(_1893,_5000)))))

Parse No. 3
****

#14:2:2 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#40:1 he1 would be a king
#1:= he1
#42:0 would be a king
#0:= past
#50:0 will be a king
#1:= will
#64:0 be a king
#1:= be
#2:0 a king
#1:= a
#1:= king
#40:1 he2 was born
#1:= he2
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#58:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_5526:((child(_5526)&exists(t:ab(t,past(future(king(_5526))))))
@past(exists(_2139:*bear(_2139,_5526)))))

Parse No. 4
****

#14:2:2 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#40:1 he1 would be a king
#1:= he1
#42:0 would be a king
#0:= past
#50:0 will be a king
#1:= will
#64:0 be a king
#1:= be
#2:0 a king
#1:= a
#1:= king
#40:1 he2 was born
#1:= he2
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_5514:((child(_5514)&exists(t:ab(t,past(future(king(_5514))))))
@past(exists(_2127:*bear(_2127,_5514)))))

Parse No. 5
****

#40:1 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#14:2:3 he1 would be a king
#2:0 a king
#1:= a
#1:= king
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#58:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

past(exists(_5476:((child(_5476)&exists(t:ab(t,exists(_4730:(king(_4730)
@past(future(equals(_5476,_4730)))))))
@exists(_2119:*bear(_2119,_5476)))))

Parse No. 6
****

#40:1 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#14:2:3 he1 would be a king
#2:0 a king
#1:= a
#1:= king
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

past(exists(_5464:((child(_5464)&exists(t:ab(t,exists(_4718:(king(_4718)
@past(future(equals(_5464,_4718))))))))
@xists(_2107:*bear(_2107,_5464)))))

Parse No. 7
****

#14:2:4 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#14:2:3 he1 would be a king
#2:0 a king
#1:= a
#1:= king
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#40:1 he4 was born
#1:= he4
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#58:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_5990:((child(_5990)&exists(t:ab(t,exists(_5244:(king(_5244)
@past(future(equals(_5990,_5244)))))))
@past(exists(_2353:*bear(_2353,_5990)))))

Parse No. 8
****

#14:2:4 a child who would be a king was born
#2:0 a child who would be a king
#1:= a
#31:3:1 child who would be a king
#1:= child
#14:2:3 he1 would be a king
#2:0 a king
#1:= a
#1:= king
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#40:1 he4 was born
#1:= he4
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_5978:((child(_5978)&exists(t:ab(t,exists(_5232:(king(_5232)
@past(future(equals(_5978,_5232))))))))
@past(exists(_2341:*bear(_2341,_5978)))))

Parse No. 9
****

#14:2:3 a child who would be a king was born
#2:0 a king
#1:= a
#1:= king
#40:1 a child who would be he3 was born
#2:0 a child who would be he3
#1:= a
#31:3:1 child who would be he3
#1:= child
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#58:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_6051:(king(_6051)
@past(exists(_4830:((child(_4830)&exists(t:ab(t,past(future(equals(_4830,_6051))))))
@exists(_2171:*bear(_2171,_4830)))))))

Parse No. 10
****

#14:2:3 a child who would be a king was born
#2:0 a king
#1:= a
#1:= king
#40:1 a child who would be he3 was born
#2:0 a child who would be he3
#1:= a
#31:3:1 child who would be he3
#1:= child
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_6039:(king(_6039)@
@past(exists(_4818:((child(_4818)&exists(t:ab(t,past(future(equals(_4818,_6039))))))
@exists(_2159:*bear(_2159,_4818)))))))

Parse No. 11
****

#14:2:3 a child who would be a king was born
#2:0 a king
#1:= a
#1:= king
#14:2:5 a child who would be he3 was born
#2:0 a child who would be he3
#1:= a
#31:3:1 child who would be he3
#1:= child
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#40:1 he5 was born
#1:= he5
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#58:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_6544:(king(_6544)
@exists(_5359:((child(_5359)&exists(t:ab(t,past(future(equals(_5359,_6544))))))
@past(exists(_2420:*bear(_2420,_5359)))))))

Parse No. 12
****

#14:2:3 a child who would be a king was born
#2:0 a king
#1:= a
#1:= king
#14:2:5 a child who would be he3 was born
#2:0 a child who would be he3
#1:= a
#31:3:1 child who would be he3
#1:= child
#40:1 he1 would be he3
#1:= he1
#42:0 would be he3
#0:= past
#50:0 will be he3
#1:= will
#64:0 be he3
#1:= be
#1:= he3
#40:1 he5 was born
#1:= he5
#42:0 was born
#0:= past
#46:0 be born
#1:= be
#57:0 born
#59:0 born
#0:= passive
#1:= bear
#0:= agent

Logical Form
****

exists(_6532:(king(_6532)
@exists(_5347:((child(_5347)&exists(t:ab(t,past(future(equals(_5347,_6532))))))
@past(exists(_2408:*bear(_2408,_5347)))))))

no more parses

BIBLIOGRAPHY

[A1] Ajdukiewicz K. (1935). Syntactic Connexion. In McCall S. (Ed.) *Polish Logic 1920-1939*. Clarendon, Oxford, 1967.

[A2] Aho A.V. & Ullman J.D. (1977). *Principles of Compiler Design*. Addison Wesley.

[A3] Aqvist L. & Guenthner F. (1978). Fundamentals of a Theory of Verb Aspect and Events within the context of an Improved Tense Logic. In Guenthner & Rohrer (1979).

[A4] Aqvist L., Guenthner F. & Rohrer C. (1978). Definability in ITL of some Subordinate Temporal Conjunctions in English. In Guenthner & Rohrer (1979).

[B1] Bach E.W. (1979). Control in Montague Grammar. *Linguistic Inquiry* Vol.10.

[B2] Bach E.W. (1979). In Defense of Passive. *Linguistics & Philosophy* Vol.3.

[B3] Bach E.W. (1980). Tenses and Aspects as Functions on Verb Phrases. In Rohrer (1980).

[B4] Bainbridge R.I. (1985). Montagovian Definite Clause Grammar. *Proceedings of the 2nd. European Conference of the Association for Computational Linguistics*, Geneva.

[B5] Bainbridge R.I. (1986). *On the Recursive Generation of Intransitive Verb Phrases and Subordinate Time Relativisation*. University of York Computer Science Report YCS84.

[B6] Barwise J. & Cooper R. (1981). Generalised Quantifiers for Natural Language. *Linguistics & Philosophy* Vol.4.

[B7] Bennett M. (1972). Accommodating the Plural in Montague's Fragment of English. In Rodman (1972).

[B8] Bennett M. (1976). A Variation and Extension of a Montague Fragment of English. In Partee (1976).

[B9] Bennett M. (1977). A Response to Karttunen on Questions. *Linguistics & Philosophy* Vol.1.

[B10] Bennett M. (1978). Demonstratives and Indexicals in Montague Grammar. *Synthese* Vol.39.

[B11] Bennett M. (1979). *Questions in Montague English*. Indiana University Linguistics Club.

[B12] Bowers J.S. & Reichenbach U.K.H. (1979). Montague Grammar & Transformational Grammar: A Review of Formal Philosophy, Selected Papers of Richard Montague. *Linguistic Analysis* 5.

[B13] Bratko I. (1986). *PROLOG Programming for Artificial Intelligence*. Addison Wesley.

[B14] Brough D. & Hogger C.J. (forthcoming). Grammar Related Transformations of Logic Programs. Research Report, Imperial College, University of London.

[B15] Bruce B.C. (1972). A Model for Temporal Reference and its Application in a Question Answering Program. *Artificial Intelligence* Vol.3.

[B16] Bull W.E. (1960). *Time Tense & The Verb*. University of California.

[C1] Carnap R. (1947). *Meaning & Necessity*. University of Chicago.

[C2] Church A. (1940). A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic* Vol.5.

[C3] Clocksin W.F. & Mellish C.S. (1981). *Programming in PROLOG*. Springer-Verlag, Berlin.

[C4] Colmerauer A. (1975). Metamorphosis Grammars. In Bolc L. (Ed.) *Natural Language Communication with Computers*. Springer-Verlag, Berlin, 1978.

[C5] Cooper R. & Parsons T. (1976). Montague Grammar, Generative Semantics, and Interpretive Semantics. In Partee (1976).

[C6] Cooper R. (1977). Review of Richard Montague's Formal Philosophy. *Language* 53.

[C7] Cooper R. (1978). Variable Binding and Relative Clauses. In Guenthner & Schmidt (1979).

[C8] Cooper R. (1978). Montague's Syntax. In Moravscik & Wirth (1980).

[C9] Copi I. (1971). *The Theory of Logical Types*. Routledge & Kegan Paul,London.

[C10] Cresswell M.J. (1973). *Logics & Languages*. Methuen, London.

[C11] Cresswell M.J. (1976). Review of Formal Philosophy, Selected Papers of Richard Montague. *Philosophia* Vol 6.

[C12] Cresswell M.J. (1977). Interval Semantics & Logical Words. In Rohrer (1977).

[D1] Dahl V. (1981). Translating Spanish into Logic through Logic. *American Journal of Computational Linguistics* Vol.7 No.3.

[D2] Dahl V. & McCord M.C. (1983). Treating Coordination in Logic Grammars. *American Journal of Computational Linguistics* Vol.9 No.2.

[D3] Davidson D. & Harman G. (Eds.) (1972). *Semantics of Natural Language.* Reidel, Dordrecht, Holland.

[D4] Davis S. & Mithun M. (Eds.) (1979). *Linguistics, Philosophy, and Montague Grammar.* University of Texas, Austin.

[D5] Dowty D.R. (1976). Montague Grammar and the Lexical Decomposition of Causative Verbs. In Partee (1976).

[D6] Dowty D.R. (1979). *Word Meaning & Montague Grammar.* Reidel, Dordrecht.

[D7] Dowty D.R. (1979). Dative Movement and Thomason's Extensions of Montague Grammar. In Davis & Mithun (1979).

[D8] Dowty D.R. (1982). Tenses, Time Adverbs and Compositional Semantic Theory. *Linguistics & Philosophy* Vol.5.

[D9] Dowty D.R., Wall R.E. & Peters S. (1981). *Introduction to Montague Semantics.* Reidel, Dordrecht, Holland.

[F1] Feigl H. & Sellars W. (Eds.) (1949). *Readings in Philosophical Analysis.* Appleton- Century- Crofts, New York.

[F2] Frege G. (1893). Uber Sinn und Bedeutung. Translated (as On Sense and Reference) in Geach P. & Black M. (Eds.) *Philosophical Writings of Gottlob Frege.* Blackwell, Oxford, 1966.

[F3] Friedman J. & Warren D.S. (1978). A Parsing Method for Montague Grammars. *Linguistics & Philosophy* Vol.2.

[F4] Friedman J., Moran D.B., & Warren D.S. (1978). Two Papers on Semantic Interpretation in Montague Grammar. *American Journal of Computational Linguistics,* Microfiche 74.

[F5] Friedman J. (1979). An Unlabeled Bracketing Solution to the Problem of Conjoined Phrases in Montague's PTQ. *Journal of Philosophical Logic* Vol.8.

[F6] Friedman J. (1981). Expressing Logical Formulas in Natural Language. In Groenendijk, Janssen, & Stokhof (1981).

[G1] Gabbay D.M. (1973). Representation of the Montague Semantics as a form of the Suppes Semantics

with Applications. In Hintikka, Moravcsik & Suppes (1973).

[G2] Gabbay D.M & Moravcsik J.M.E. (1974). Branching Quantifiers, English, & Montague Grammars. *Theoretical Linguistics* Vol.1.

[G3] Gabbay D.M. & Moravcsik J.M.E. (1980). Verbs, Events and the Flow of Time. In Rohrer (1980).

[G4] Gallin D. (1975). *Intensional and Higher Order Modal Logic*. North Holland, Amsterdam.

[G5] Gazdar G., Klein E., Pullum G.K., & Sag I. (1984). *Generalised Phrase Structure Grammar*. Blackwell, Oxford.

[G6] Geach P.T. (1962). *Reference & Generality*. Cornell University Press, Ithaca.

[G7] Groenendijk J.A.G., Janssen T.M.V., & Stokhof M.B.J (Eds.) (1981). *Formal Methods in the Study of Language 1 & 2*. Mathematisch Centrum, Amsterdam.

[G8] Groenendijk J.A.G. & Stokhof M.B.J. (1982). Semantic Analysis of Wh-Complements. *Linguistics & Philosophy* Vol.5.

[G9] Guenthner F. & Rohrer C. (Eds.) (1978). *Studies in Formal Semantics*. North Holland, Amsterdam.

[G10] Guenthner *F. & Schmidt S.J. (Eds.) (1979). Formal Semantics & Pragmatics of Natural Language*. Reidel, Dordrecht, Holland.

[H1] Halvorsen P-K. & Ladusaw W.D. (1979). Montague's Universal Grammar: an Introduction for the Linguist. *Linguistics & Philosophy* Vol.3.

[H2] Hamblin C.H. (1973). Questions in Montague English. In Partee (1976).

[H3] Hausser R. & Zaefferer D. (1978). Questions and Answers in a Context Dependent Montague Grammar. In Guenthner & Schmidt (1978).

[H4] Hayes P.J. (1977). In Defence of Logic. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge.

[H5] Hintikka K.J.J., Moravcsik J.M.E. & Suppes P. (Eds.) (1973). Approaches to Natural Language. Reidel, Dordrecht, Holland.

[H6] Hirst G. (1983). A Foundation for Semantic Interpretation. *Proceedings of the 21st. Annual Meeting of the Association for Computational Linguistics*.

[H7] Hobbs J.R. & Rosenschein S.J. (1978). Making Computational Sense of Montague's Intensional Logic. *Artificial Intelligence* Vol.9.

[H8] Hudson R.A. (1976). *Arguments for a Non-Transformational Grammar.* University of Chicago Press.

[H9] Hutchins W.J. (1986). *Machine Translation Past, Present, Future.* Ellis Horwood.

[J1] Janssen T.M.V. (1978). Simulation of a Montague Grammar. *Annals of Systems Research* 7.

[J2] Janssen T.M.V. (1980). Logical Investigations on PTQ arising from Programming Requirements. *Synthese* Vol.44.

[J3] Janssen T.M.V. (1980). On Problems concerning the Quantification Rules in Montague Grammar. In Rohrer (1980).

[J4] Janssen T.M.V. (1981). Compositional Semantics and Relative Clause Formation in Montague Grammar. In Groenendijk, Janssen & Stokhof (1981).

[J5] Jones M. & Warren D.S. (1982). Conceptual Dependency and Montague Grammar - A Step towards Conciliation. *Proceedings of the National Conference on Artificial Intelligence.* Pittsburg.

[K1] Kamp J. (1971) Formal Properties of "Now". *Theoria* 37.

[K2] Kanger S. (1957). New Foundations for Ethical Theory. In Halpinen R. (Ed.) *Deontic Logic: Introductory & Systematic Readings.* Reidel, Dordrecht, Holland, 1970.

[K3] Kaplan R.M. (1973). A General Syntactic Processor. In Rustin (1973).

[K4] Karttunen L. (1977). Syntax and Semantics of Questions. *Linguistics & Philosophy* Vol.1.

[K5] Karttunen L. & Peters S. (1980). Interogative Quantifiers. In Rohrer (1980).

[K6] Keenan E.L. & Faltz L.M. (1978). *Logical Types for Natuaral Language.* UCLA Occasional Papers in Linguistics No.3.

[K7] Keenan E.L. & Faltz L.M. (1980). A New Approach to Quantification in Natural Language. In Rohrer (1980).

[K8] Kowalski R.A. (1979). *Logic for Problem Solving.* North Holland, Amsterdam.

[K9] Knuth D.E. (1968). Semantics of Context Free Languages. *Mathematical Systems Theory* Vol.2 No.2.

[K10] Knuth D.E. (1975). *The Art of Computer Programming Vol.1: Fundamental Algorithms.* Addison-Wesley, Reading, Mass.

[K11] Kripke S.A. (1963). Semantical Considerations on Modal Logic. *Acta Philosophica Fennica* 16. Reprinted in Linsky (1971).

[K12] Kripke S.A. (1970). Naming and Necessity. In Davidson & Harman (1972).

[L1] Landsbergen J. (1981). Adaptation of Montague Grammar to the Requirements of Parsing. In Groenendijk, Janssen & Stokhof (1981).

[L2] Landsbergen J. (1982). Machine Translation based on Logically Isomorphic Montague Grammars. In Horecky J. (Ed.) *COLING 82* . North Holland, Amsterdam.

[L3] Leibniz G.W. (1679). Specimen of Universal Calculus. Translated in Loemker (1956).

[L4] Leibniz G.W. (1714). The Monadology. Translated in Loemker (1956).

[L5] Lewis C.I. & Langford C. (1932). *Symbolic Logic.* Dover, New York.

[L6] Lewis D.K. (1972). General Semantics. In Davidson & Harman (1972).

[L7] Linsky L. (Ed.) (1971). *Reference & Modality.* Oxford University Press.

[L8] Loemker L.E. (Ed.) (1956). *Gottfried Wilhelm Leibniz, Philosophical Papers & Letters.* Reidel, Dordrecht, Holland.

[M1] McCord M. (1982). Using Slots and Modifiers in Logic Grammars for Natural Language. *Artificial Intelligence* Vol.18.

[M2] Montague R.M. (1968). Pragmatics. In Thomason (1974).

[M3] Montague R.M. (1970a). Pragmatics and Intensional Logic. In Davidson & Harman (1972) and Thomason (1974).

[M4] Montague R.M. (1970b). English as a Formal Language. In Thomason (1974).

[M5] Montague R.M. (1970c). Universal Grammar. In Thomason (1974).

[M6] Montague R.M. (1972). The Proper Treatment of Quantification in Ordinary English. In Hintikka et al (1973) and Thomason (1974).

[M7] Moore R.C. (1981). Problems in Logical Form. *Proceedings of the 19th. Annual Meeting of the Association for Computational Linguistics.*

[M8] Moravscik E.A. & Wirth J.R. (Eds.) (1980). *Syntax & Semantics.* Academic Press, New York.

[N1] Nishida T. & Doshita S. (1983). An Application of Montague Grammar to English - Japanese Machine Translation. *Proceedings of the Conference on Applied Natural Language Processing.* Santa Monica.

[P1] Parsons T. (1977). Type Theory and Ordinary Language. In Davis & Mithun (1979).

[P2] Partee B.H. (1972). Comments on Montague's Paper. In Hintikka et al (1973).

[P3] Partee B.H. (1973). Some Transformational Extensions of Montague Grammar. In Partee (1976).

[P4] Partee B.H. (1975). Montague Grammar and Transformational Grammar. *Linguistic Inquiry* Vol.6.

[P5] Partee B.H. (Ed.) (1976). *Montague Grammar.* Academic Press, N.Y.

[P6] Partee B.H. (1977). Constraining Transformational Montague Grammar: a Framework and a Fragment. In Davis & Mithun (1979).

[P7] Pereira F.C.N. & Warren D.H.D. (1980). Definite Clause Grammars for Language Analysis. *Artificial Intelligence* Vol.13.

[P8] Prior A.N. (1957). *Time & Modality.* Clarendon Press, Oxford.

[P9] Prior A.N. (1967). *Past, Present & Future.* Oxford University Press.

[Q1] Quine W.V.O (1953a). *From a Logical Point of View.* Harvard University Press.

[Q2] Quine W.V.O. (1953b). Reference & Modality. In Quine (1953a) and Linsky (1971).

[Q3] Quine W.V.O. (1956). Quantifiers & Propositional Attitudes. *Journal of Philosophy* 53. Reprinted in Linsky (1971).

[Q4] Quine W.V.O. (1960). *Word & Object.* MIT, Cambridge,Mass.

[Q5] Quirk R., Greenbaum S., Leech G. & Svartvik J. (1972). A Grammar of Contemporary English.

Longmans, London.

[R1] Reichenbach H. (1966). *Elements of Symbolic Logic*. New York.

[R2] Ritchie G & Thompson H. (1984) Natural Language Processing. In O'Shea T. & Eisenstadt M. (Eds.) *Artificial Intelligence*. Harper & Row, New York, 1984.

[R3] Rodman R. (Ed.) (1972). *Papers in Montague Grammar: Occasional Papers in Linguistics 2*. UCLA.

[R4] Rodman R. (1976). Scope Phenomena, "Movement Transformations" and Relative Clauses. In Partee (1976).

[R5] Rohrer C. (Ed.) (1977). *On the Logical Analysis of Tense and Aspect*. TBL Verlag Gunter Narr, Tubingen.

[R6] Rohrer C. (Ed.) (1980). *Time, Tense & Quantifiers*. Max Niemeyer Verlag, Tubingen.

[R7] Rosenschein S.J. & Scheiber S.M. (1982). Translating English into Logical Form. *Proceedings of the 20th. Annual Meeting of the Association for Computational Linguistics.*

[R8] Ross J.R. (1967). Excerpts from Constraints on Variables in Syntax. In Harman G. (ed). *On Noam Chomsky*. Anchor.

[R9] Russell B. (1903). *Principles of Mathematics*. George Allen & Unwin, London.

[R10] Russell B. (1905). On Denoting. *Mind* 14. Reprinted in Feigl & Sellars 1949.

[R11] Russell B. (1918). The Philosophy of Logical Atomism. In Marsh R.C. (Ed.) *Logic and Knowledge*. George Allen & Unwin, 1956.

[R12] Rustin R. (Ed.) (1973). *Natural Language Processing*. Algorithmics Press, N.Y.

[S1] Schubert L.K. & Pelletier F.J. (1982). From English to Logic: Context Free Computation of 'Conventional' Logical Translation. *American Journal of Computational Linguistics* Vol.8.

[S2] Scott D. (1970). Advice on Modal Logic. In Lambert K. (Ed.) *Philosophical Problems in Logic: Some Recent Developments*. Reidel, Dordrecht, Holland, 1970.

[S3] Strawson P.F. (1950). On Referring. *Mind* 59.

[T1] Tarski A. (1931). The Concept of Truth in Formalised Languages. In Tarski (1969).

[T2] Tarski A. (1969). *Logic Semantics & Metamathematics*. (Translated by Woodger J.H.). Oxford.

[T3] Thomason R.H. (Ed.) (1974). *Formal Philosophy, Selected Papers of Richard Montague*. Yale, New Haven.

[T4] Thomason R.H. (1976). Some Extensions of Montague Grammar. In Partee (1976).

[T5] Thompson H. (1983). Natural Language Processing: a Critical Analysis of the Field with some Implications for Parsing. In Sparck-Jones K. & Wilks Y. (Eds.) *Automatic Natural Language Parsing*. Ellis Horwood, Chichester, 1983.

[V1] Van Emden M.H. (1975). Programming with Resolution Logic. In Elcock E.W & Michie D. (Eds.) *Machine Intelligence* 8. Ellis Horwood, Chichester, 1976.

[W1] Warren D.S. (1983). Using Lambda Calculus to represent Meanings in Logic Grammars. *Proceedings of the 21st. Annual Meeting of the Association for Computational Linguistics.*

[W2] Warren D.S. & Friedman J. (1982). Using Semantics in Non Context Free Parsing of Montague Grammar. *American Journal of Computational Linguistics* Vol.8.

[W3] Wilks Y. (1976). Philosophy of Language. In Charniak E. & Wilks Y. (Eds.) *Computational Semantics*. North Holland, Amsterdam, 1976.

[W4] Winograd T. (1983). *Language as a Cognitive Process*. Addison-Wesley, Reading, Mass.

[W5] Wittgenstein L. (1918). *Tractatus Logico Philosophicus*. (Translated by Pears D.F. & McGuiness B.F.) Routledge & Kegan Paul, London, 1961.

[W6] Wittgenstein L. (1953). *Philosophical Investigations*. (Edited by Anscombe G. E. M.). Blackwell, Oxford.

[W7] Woods W.A. (1970). An Experimental Parsing System for Transition Network Grammars. In Rustin (1973).