MONTAGOVIAN DEFINITE CLAUSE GRAMMAR

R. I. Bainbridge,
Dept. of Computer Science,
Teesside Polytechnic,
Middlesbrough, Cleveland, England.

Abstract

This paper reports a completed stage of ongoing research at the University of York. Landsbergen's advocacy of analytical inverses for compositional syntax rules encourages the application of Definite Clause Grammar techniques to the construction of a parser returning Montague analysis trees. A parser MDCG is presented which implements an augmented Friedman - Warren algorithm permitting post referencing, and interfaces with a language of intensional logic translator LILT so as to display the derivational history of corresponding reduced IL formulae. Some familiarity with Montague's PTG and the basic DCG mechanism is assumed.

Keywords Compositional Semantics, Definite Clause Grammar, Friedman Warren Algorithm, Intensional Logic, Montague Grammar, Natural Language Processing, PROLOG.

1 Introduction

Consideration is given by Landsbergen [20] to the global algorithmic structure of a top down parser demonstrably equivalent to a compositional grammar such as that of PTG [22]. The method is as follows:

1. Formulate the original grammar in terms of indexed compositional-M rules of form:

If syntax trees $\langle Sk..Sn \rangle$ satisfy condition C then combine $\langle Sk..Sn \rangle$ into S_j

such that the compositional history may be represented on a derivation tree (i.e. a skeletal analysis tree lacking node labels).

2. Subject to specified restraints evolve inverse analytical-M Rules of form:

If S_j conforms to condition C' then analyse S_j into $\langle Sk..Sn \rangle$.

3. Prove that the compositional and analytical M rules are equivalent.

4. Construct a two stage parser:

(i) Parse a sentence using a context free grammar (CFG) thus deriving a syntax tree.

(ii) Traverse the syntax tree in postorder [19] under the guidance of the analytical-M rules, constructing the derivation tree which reflects the reverse order application of the inverse rules.

An abstract algorithm describing the parser is given in the form of procedural pseudo code, however the problem of establishing that an implementation conforms to the algorithm is deferred, a problem perhaps aggravated by the absence of a formal notation for M rules which might otherwise have suggested appropriate data structures.

The postorder traverse in (ii) of a preorder creation involves a duplication which may be avoided by adopting the PROLOG Definite Clause Grammar (DCG) formalism, ([28] cf. [3], [4], [5], [21]), which, as has been observed [32] virtually forces the methodology of syntax directed translation coupled with compositional semantics. A DCG may be ingenuously characterised as a CFG having category symbols augmented by argument places, and containing supplementary goals not limited in function to input consumption. Logical variables in argument places permit synthesised and inherited attributes [18] to be handled with equal facility. The clauses of a DCG may be directly executed by a PROLOG interpreter, hence if combined CFG+analytical-M rules are presented in the form of Definite Clauses, the problem of mapping algorithm to implementation does not arise: the algorithm and program are one and the same.

The parsers of both Landsbergen [20] and Friedman & Warren [9] generate only skeletal trees, other details being recoverable from the leaves and operation indices: however the tedium of such recovery may properly devolve on the computer, and for pedagogical purposes at

least the production of full analysis trees would be advantageous. This paper outlines a DCG implementation of a version of the compositional syntax of PTG which returns full Montague analysis trees in the form of vine diagrams modified at most by additional feature marking on variables. Given an input sentence, MDCG returns sets of trees, optionally passing members to a language of intensional logic translator (LILT) which generates corresponding IL formulae. The indeterminacy of PROLOG implies that a DCG written with circumspection may also be used in reverse, but it remains to be investigated whether the model could be so modified as to achieve the recent objectives of Friedman [8]. To handle quantification MDCG employs a variation of the Friedman-Warren algorithm (FWA) [9].

The programs are implemented in University of Edinburgh DEC-10 PROLOG and run on the University of York DEC-10 computer.

2 Implied Modifications to PTG

The version of PTG grammar implemented in MDCG has both significant and cosmetic changes. As regards the first, Partee observes ([24], [25]) that a version of S12 which inserts labelled bracketing, and a version of S4 sensitive to such bracketing and generalised to add subject-agreement to the first verb in each conjunct of a conjoined verb phrase, is needed in order to distinguish (1) from (2).

(1) John tries to walk and talks.

(2) John tries to walk and talk.

Without labelled bracketing, PTG has diff-

and then constrains the predicate to be a conjunction of one or more verb phrases identifiable as commencing with concordant finite forms. Likewise the procedure which parses infinitival complements in accordance with SB accepts a conjunction of one or more verb phrases starting with infinitives. MDCG successfully generates the trees illustrated in fig 1, thus tacitly assuming compositional counterparts adopting modifications such as the bracketing of Partee ([24], [25]), or the headverb flagging convention of Bennett [2]. Bennett's simplified semantic typing, which results from treating IV and CN as primitive categories, is also exploited in LILT as illustrated in the appendix.

The MDCG post referencing facility requires the admission of alternative capitalised variables, and an amended f10 which undertakes the replacement by term T of the earlier of:

(a) the first uncapitalised variable with index n

or (b) the last occurring variable with index n.

Whether capitalised variables would prove popular with advocates of the "well formedness constraint" [27] is uncertain.

Feature matching, which is achieved by PROLOG's cross-goal variable instantiation conventions, plainly affords a simple mechanism, from the syntactic viewpoint, for handling number concord and selectional restrictions on the basis of feature marked lexical entries. Indeed since the alternative operations licenced by S2 are also identified in the lexicon, MDCG has the facility without amendment to produce analysis trees for plural sentences such

```
*****
*                                     *
*      (a)                           *
*      #4:4 john tries to walk and talks      *
*      #1:= john                               *
*      #12:8 try to walk and talk              *
*      #8:6 try to walk                        *
*      #1:= try                               *
*      #1:= walk                              *
*      #1:= talk                              *
*      (b)                           *
*      #4:4 john tries to walk and talk      *
*      #1:= john                               *
*      #8:6 try to walk and talk              *
*      #1:= try                               *
*      #12:8 walk and talk                    *
*      #1:= walk                              *
*      #1:= talk                              *
*
*                                     *
*                                     *
*      fig 1.                             *
*
*****
```

iculty identifying head verbs, but since a DCG works top down it encounters no such problems. The MDCG analogue of S4 first identifies the features of the subject,

as:

(3) The men have not eaten the fishes.
given a further determiner clause in the lexicon introducing a definite article

paired with an additional operation number and marked with the features [def,pl]. The principle of compositionality [10] demands that this syntactical facility remain officially untriggered pending the introduction of appropriate plural determiner interpretation clauses in LILT; however its introduction for experimental purposes allows MDCG and LILT to provide a testbed for the investigation of senses for additional quantifiers.

The cosmetic variation involves the introduction of further feature marking on variables, but since variables receive semantic interpretation only in leaf position where PTG and MDCG are equivalent, the change has no semantic significance. Variables as leaves are in the range heO..he_n, but whereas PTG introduces only accusative marking as a side effect of combination, MDCG adds markings for gender (and if needed number). Amendments to PTG to reflect these innovations would be purely decorative. S2 would mark its output with a number feature derived from the quantifier, while both S4 and S5 would, like S2, licence alternative operations such that f4.0 and f5.0 would be restricted to cases where the input T were not a variable, and f4.1..f4.4, f5.1..f5.4 would generate he_n IV .. they_n IV, TV him_n .. TV them_n respectively. Since the translation rules T4 and T5 refer to the value of the input term of a function in the f4, f5 series these would be unaffected. Rules in the range S3_n, S14_n .. S16_n would apply on condition that the input sentence did not include a variable with index n having discordant features. If plural forms became available, the subject agreement clause of S4 would need generalising, and S13 would, like S11 and S12, gain access to f8, marking its output with the number of its first argument in case the operation were f9, or with [+plural] otherwise.

3 Tree Structures and Parsing Procedures

Nodes on an analysis tree are represented internally by analogues of the "syn" structures of McCord [21], having the form:

node(N, F, L, D)

where:

N = A rule number in the form #Syn:Fun, #Syn:(Fun,Inx), or #1:= such that Syn and Fun are Montague syntax rule and structural operation numbers, Inx is a variable subscript, and #1= indicates lexical insertion.

F = A list of features intrinsic to the node.

L = A node label in list form.

D = In the case of a non-terminal node a binary list of daughters both of

which are nodes, otherwise a structure of form:

sense(Item,Category)

used by LILT in the generation of IL formulae.

Procedures which parse grammatical categories normally have ten arguments the nature of which will where necessary be explained in subsequent sections. The general form is as follows:

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

where

N = A node structure as described.

F = The features of the category - in - context which may exceed the node features. For example case is not an intrinsic noun phrase leaf feature, but it constrains adoption to specified configurations.

E = The environment (preorder predecessors) of the category relative to which the parse is aborted if N is non unique.

L = The transmission label.

Ia, Iz = String buffers before and after parsing.

FVB = Free variables below list

SA = Substitutions above list.

SRa, SRz = Substituents, required lists before and after parsing.

4 Implementing FWA in PROLOG

The FWA handles the introduction and subsequent binding of indexed variables on n-ary substitutes for skeletal analysis trees by the manipulation of two lists, FVB (free variables below) and SA (substitutions above). In order to implement the algorithm in a PROLOG DCG directed towards the production of strictly Montagovian trees, each clause responsible for creating a node requires both FVB and SA argument places, the first to act as an output and the second as an input parameter, with the proviso that the top level "sentence" call set both to the empty list.

A clause charged with the construction of a T (=NP) node, provided that it does more than read a surface pronoun, must be given the option of returning a default node, or alternatively of binding the noun phrase discovered to the next available variable, adding the binding to the FVB set, and returning a variable node instead. In MDCG a binding takes the form not of a <variable, noun-phrase> pair but of a structure:

bind(Var, Inx, Node)

where:

Var = The indexed variable.

Inx = The subscript.

Node = The complete structure node(N, F, L, D) for a T or, in case the binding is performed under the S3

requirement that in the top level "sentence" call FVB must be []. The latter requirement constitutes a final filter as suggested, albeit with reservation, by Janssen [16] as a means of ensuring syntactic conformity to the "variable principle".

When a parsing procedure is called other than at top level, the SA is initialised as the union of the SA of the head goal and the FVB of any goal constructing an elder sister. A noun phrase parsing clause which reads a surface pronoun may reference any binding in the SA such that, where Node = node(N,F,L,D), the features in F conform with the pronoun in number and gender. A variable node having the indexed variable from the binding in its L field is returned, thus achieving an antecedent reference.

Neither LIFO nor FIFO lists suffice to generate all permitted quantifier scope variations. If FVB and SA are formed by simple concatenation then substitute must be capable of extracting members randomly. Alternatively substitute may safely select the next available item provided that the lists are formed in such a way that all permutations emerge in due course. MDCG adopts the latter choice, employing a predicate:

`mix(L1,L2,L3)`
which, given successive calls, simulates
the scattering of the members of L1 within
L2 in a random pattern on the assumption
that L2 is already random.

fig 2.

5 Augmenting FWA

Since the grammar of PTQ does not generate post referencing pronouns, FWA is not designed to accommodate them. In MDCG an augmented FWA is introduced to handle post referencing via capitalised variables which are always realised as surface pronouns. For example in response to the input:

28

It is now required that whenever a noun phrase making clause exercises its option to introduce a variable, a check must first be made of the SR list, and if possible a suitable dummy binding extracted and completed with no addition to the FVB list. The behaviour of PROLOG ensures that completion effects all existing occurrences of the dummy. A constraint on substitution must now prohibit the extraction from the FVB of any binding appearing in the SRz list returned to the head goal. In this way not only may no younger sister dominate quantification over a variable remaining free in the family of an elder sister (the original constraint), but the elder sister must extend the same courtesy to her sibling.

6.1 Handling Left Recursion

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

```
* common(Node, Ft, E, L, Ia, Iz, FVB, SA, SRa, SRz) :-
```

```
*   wfst(common(CN, Ft, E, La, Ia, Ib, FVBA, SA, SRa, SRb)),
```

```
*   scan([such, that], Ib, Ic),
```

```
*   gensym(he, He, Inx, Suffix),
```

```
*   join([bind(He, Inx, CN)|FVBA], SA, SAa),
```

```
*   join(E, CN, E1),
```

```
*   sentence(S, [dc1], E1, Lb, Ic, Iz, FVBb, SAa, SRb, SRz),
```

```
*   eligible(bind(He, Inx, CN), FVBb, [], []),
```

```
*   dominated(He, S),
```

```
*   makevars(Nom, _, Acc, _, Suffix, Subj, Obj, Ft),
```

```
*   editline(Nom, Acc, Subj, Obj, Lb, Lc),
```

```
*   join(La, [such, that|Lc], Ld),
```

```
*   mix(FVBA, FVBb, FVBc),
```

```
*   substitute(cn, node(#3:(3:Inx), Ft, Ld, [CN, S]),
```

```
*           Node, Ld, L, [], [], FVBc, FVB, [], SRz),
```

```
*   recordz(wfst(common(Node,
```

```
*               Ft, E, L, Ia, Iz, FVB, SA, SRa, SRz))).
```

```
* *
```

```
* 
```

```
* Fig 3.
```

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

Since the WFST must record the relative position of entries, grammar rule notation (GRN) which insulates the programmer from lexical decomposition must be proscribed: accordingly MDCQ is written in raw PROLOG, pairs of variables in the range Ia..Iz representing string buffers before and after parsing.

(6) a man loves herk.
Before this may be prefixed with the
common noun plus "such that" to become the

29

6.3 Node and Transmission Labels

6.4 Calls to "substitute" and "option"

where:

Node = The default node constructed.

T1, T11 = Default and replacement trans-

ustrates the MDCG equivalent of S10, the node label L1 must contain the bare infinitive of the head verb while L_a contains a finite form. Having processed the adverb, a pair of new labels must

mission labels (T11=T1 if no substitution made).

N1,N11 = Default and replacement node labels (N11=N1 if no substitution made, and N1,NL1=[] if T=s or T=cn

accordingly be constructed, one for the default node and one for its transmission label. Should a substitution then be made, twin labels for the introduced higher node must likewise be maintained by the substitute procedure.

since the new: node label is taken
to be T11).

FVB, FVB1 = The free variable below lists
before and after any extraction.

Sk = Those bindings bypassed in ancestor calls to substitute (At top level Sk=[]).

30

```
option(Node, F, Node1, T1, T11, FVB, FVB1,
        SR, SR1)
```

SR, SR1 = The substituents required lists
before and after any subtraction.

Warren [32] suggests two possibilities for encoding lambda terms in PROLOG given the desire to represent a full typed lambda calculus, the first portraying lambda variables as PROLOG structures and the second equating them with PROLOG vari-

Whereas Janssen ([14], [15]) accomplishes reduction by a process of essentially localised tree transformations, the simplification algorithm of LILT takes advantage of PROLOG's list processing capabilities to undertake global list transformations whenever necessary. MDCQ - LILT exemplifies the reorganised directed process approach discussed by Warren and Friedman [33], i.e. LILT is (optionally) called after each parse. The present objective of display-

```
*****  
*  
*      translate(node(N,F,L,[sense(R,T)]),S) :-  
*          !, sense(R,T,S), message(O,[L,S]).  
*  
*      translate(Tree,IL) :-  
*          structure(Tree,node(N,F,L,_),Lsub,Rsub),  
*          translate(Rsub,Rnew), translate(Lsub,Lnew),  
*          construct(node(N,F,L,_),Lnew,Rnew,Tree1),  
*          formulate(Tree1,IL1),  
*          message(N,IL1),  
*          simplify(IL1,IL).  
*  
*  
*                                     fig 7.  
*****
```

The essence of this scheme may be extracted from fig 6 which illustrates the clause correlating singular definite article with its sense. The top level translation clauses are illustrated in fig 7. These constitute a recursive procedure which generates reduced IL formulae with

[2] Bennett M. (1976) A variation and extension of a Montague fragment of

- English. in Partee (1976).
- [3] Clocksin W.F. & Mellish C.S. (1981) Programming in PROLOG. Springer-Verlag, Berlin.
- [4] Colmerauer A. (1975) Metamorphosis grammars. in Bolc L. (Ed.) Natural Language Communication with Computers. Springer-Verlag, Berlin, 1978.
- [5] Dahl V. (1981) Translating spanish into logic through logic. American Journal of Computational Linguistics Vol. 7 No. 3.
- [6] Davis S. & Mithun M. (Eds.) (1979) Linguistics, Philosophy, and Montague Grammar. University of Texas, Austin.
- [7] Dowty D.R., Wall R.E. & Peters S. (1981) Introduction to Montague Semantics. Reidel, Dordrecht: Holland.
- [8] Friedman J. (1981) Expressing logical formulas in natural language. in Groenendijk, Janssen, & Stokhof (1981).
- [9] Friedman J. & Warren D.S. (1978) A parsing method for Montague grammars. Linguistics & Philosophy 2.
- [10] Frege G. (1893) On sense and reference. in Geach P. & Black M. (Eds.) Philosophical Writings of Gottlob Frege. Blackwell, Oxford, 1966.
- [11] 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.
- [12] Hintikka K. J. J., Moravcsik J. M. E. & Suppes P. (Eds.) (1973) Approaches to Natural Language. Reidel, Dordrecht: Holland.
- [13] Hobbs J.R. & Rosenschein S.J. (1978) Making computational sense of Montague's intensional logic. Artificial Intelligence 9.
- [14] Janssen T.M.V. (1978) Simulation of a Montague grammar. Annals of Systems Research 7.
- [15] Janssen T.M.V. (1980) Logical investigations on PTG arising from programming requirements. Synthese 44.
- [16] Janssen T.M.V. (1981) Compositional semantics and relative clause formation in Montague grammar. in Groenendijk, Janssen & Stokhof (1981).
- [17] Kaplan R.M. (1973) A general syntactic processor. in Rustin (1973).
- [18] Knuth D.E. (1968) Semantics of context free languages. Mathematical Systems Theory Vol. 2 No. 2.
- [19] Knuth D.E. (1975) The Art of Computer Programming Vol. 1: Fundamental Algorithms. Addison - Wesley, Reading, Mass.
- [20] Landsbergen J. (1981) Adaptation of Montague grammar to the requirements of parsing. in Groenendijk, Janssen & Stokhof (1981).
- [21] McCord M. (1982) Using slots and modifiers in logic grammars for natural language. Artificial Intelligence 18.
- [22] Montague R.M. (1972) The proper treatment of quantification in ordinary English. in Hintikka et al (1973) and Thomason (1974).
- [23] Partee B.H. (1972) Comments on Montague's paper. in Hintikka et al (1973).
- [24] Partee B.H. (1973) Some transformational extensions of Montague grammar. in Partee (1976).
- [25] Partee B.H. (1975) Montague grammar and transformational grammar. Linguistic Inquiry 6.
- [26] Partee B.H. (Ed.) (1976) Montague Grammar. Academic Press, N.Y.
- [27] Partee B.H. (1977) Constraining transformational Montague grammar: a framework and a fragment. in Davis & Mithun (1981).
- [28] Pereira F.C.N. & Warren D.H.D. (1980) Definite clause grammars for language analysis. Artificial Intelligence 13.
- [29] Rustin R. (Ed.) (1973) Natural Language Processing. Algorithmics Press, N.Y.
- [30] Thomason R.H. (1974) (Ed.) Formal Philosophy - Selected Papers of Richard Montague. Yale, New Haven.
- [31] Thompson H. (1981) Chart parsing and rule schemata in PSG. Proceedings of the 19th. annual meeting of the Association for Computational Linguistics 167-172.
- [32] 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
- [33] Warren D.S. & Friedman J. (1982) Using semantics in non context free parsing of Montague grammar. American Journal of Computational Linguistics 8.
- [34] Winograd T. (1983) Language as a Cognitive Process. Addison-Wesley, Reading, Mass.
- [35] Woods W.A. (1970) An experimental parsing system for transition network grammars. in Rustin (1973).

Appendix : Sample Output

|: mary believes that john is a man.

Parse No. 1

```
#4:4 mary believes that john is a man
#1: = mary
#7:6 believe that john is a man
#1: = believe
#4:4 john is a man
#1: = john
#5:5 be a man
#1: = be
#2:2 a man
#1: = a
#1: = man
```

1? yes.

Composition & Simplification

```
[0] from Lexicon: Basic expression [man] =>
    man
[1] from Lexicon: Basic expression [a] =>
    lambda(p:lambda(q:exists(_3423:(`p(_3423)& `q(_3423))))
[2] from [0,1]: Construction by T2 =>
    eval(lambda(p:lambda(q:exists(_3423:(`p(_3423)&
    `q(_3423))))),`man)
[3] from [2]: Instantiate variable
    eval(` `man,_3423)
[4] from [3]: Relational notation
    ` `man(_3423)
[5] from [4]: Down-up conversion
    man(_3423)
[6] from [2]: Lambda conversion
    lambda(q:exists(_3423:(man(_3423)& `q(_3423))))
[7] from Lexicon: Basic expression [be] =>
    lambda(sub:lambda(_4607: `sub(`lambda(_4608:
    equals(_4607,_4608))))
[8] from [6,7]: Construction by T5 =>
    eval(lambda(sub:lambda(_4607: `sub(`lambda(_4608:
    equals(_4607,_4608))))),`lambda(q:exists(_3423:
    man(_3423)& `q(_3423))))
[9] from [8]: Instantiate variable
    eval(` `lambda(q:exists(_3423:(man(_3423)& `q(_3423))))),
    `lambda(_4608:equals(_4607,_4608)))
[10] from [9]: Down-up conversion
    eval(lambda(q:exists(_3423:(man(_3423)& `q(_3423))))),
    `lambda(_4608:equals(_4607,_4608)))
```

```

[11] from [10]: Instantiate variable
      eval(` ^lambda(_4608:equals(_4607,_4608)),_3423)
[12] from [11]: Down-up conversion
      eval(lambda(_4608:equals(_4607,_4608)),_3423)
[13] from [12]: Lambda conversion
      equals(_4607,_3423)
[14] from [10]: Substitute identicals
      man(_4607)
[15] from [10]: Lambda conversion
      man(_4607)
[16] from [8]: Lambda conversion
      lambda(_4607:man(_4607))
[17] from Lexicon: Basic expression [john] =>
      lambda(p: `p(john))
[18] from [16,17]: Construction by T4 =>
      eval(lambda(p: `p(john),^lambda(_4607:man(_4607)))
[19] from [18]: Instantiate variable
      eval(` ^lambda(_4607:man(_4607)),john)
[20] from [19]: Down-up conversion
      eval(lambda(_4607:man(_4607)),john)
[21] from [20]: Lambda conversion
      man(john)
[22] from [18]: Lambda conversion
      man(john)
[23] from Lexicon: Basic expression [believe] =>
      believe
[24] from [22,23]: Construction by T7 =>
      eval(believe,^man(john))
[25] from [24]: Relational notation
      believe(^man(john))
[26] from Lexicon: Basic expression [mary] =>
      lambda(p: `p(mary))
[27] from [25,26]: Construction by T4 =>
      eval(lambda(p: `p(mary),^believe(^man(john)))
[28] from [27]: Instantiate variable
      eval(` ^believe(^man(john)),mary)
[29] from [28]: Relational notation
      ` ^believe(mary,^man(john))
[30] from [29]: Down-up conversion
      believe(mary,^man(john))
[31] from [27]: Lambda conversion
      believe(mary,^man(john))

```

Logical Form

believe(mary,^man(john))