

ALTERNATIVE PARSERS FOR CONCEPTUAL DEPENDENCY:
GETTING THERE IS HALF THE FUN

Marc Eisenstadt
The Open University
Milton Keynes
ENGLAND, U.K.

This paper takes a look behind the scenes at two conceptually-based parsers in order to shed light on the true differences between them. The first one is Riesbeck's parser for Schank's conceptual dependency; the second is the 'LNR' parser for Norman and Rumelhart's active semantic networks. Both are described in terms of Kaplan's General Syntactic Processor formalism. This analysis shows that 'conceptual dependency' and 'active semantic networks' have little or nothing to do with the actual functioning of the parsers. Computationally, the two parsers differ only in terms of (a) effective use of interrupts and (b) reliance on selectional restrictions to guide parsing. A synthesis of the best features of both is suggested.

1. INTRODUCTION

An important trend in natural language processing which emerged in the early 1970's was the use of case-based language-independent notation to represent the conceptual content of segments of text (3,5,7). Although there is much overlap among the theories of different research teams working in this area, there appear to be fundamental differences in the ways in which these theories specify the actual processing of text. But which differences are deep theoretical ones, and which are merely artifacts of implementation? I address this question by looking at the processes underlying two parsers which were influential during the 1970's: Riesbeck's parser (4) for Schank's conceptual dependency formalism (5); and the 'LNR' parser for Norman and Rumelhart's active semantic network formalism (3). The actual internal representation of text is not the main focus of this paper. Rather, it is the parsing process itself which is under scrutiny.

To carry out this analysis, I have implemented versions of both parsers using Kaplan's General Syntactic Processor, GSP (1). I will first describe the GSP versions of each parser, and then the way in which their best features can be combined.

2. RIESBECK'S PARSER

2.1 Overview

The intent of Riesbeck's parser is to build a

conceptual dependency representation of a segment of text. In operation, the main verb of each segment (phrase) of text is located, and a representation of its 'meaning' (its decomposition) is constructed in memory. The surrounding text, primarily noun-phrases, is then systematically mapped onto vacant slots within the memory representation of the decomposed verb. The parser's 'expectations' are represented by active packets of production rules. Important phrase boundaries and slot-filler (case) information are signposted by words which may (a) satisfy existing expectations of the parser (by matching the condition side of a production rule) or (b) generate new expectations for the parser (by activating a new packet of production rules).

2.2 The GSP version

In GSP terms, Riesbeck's 'active' production rules are represented by standard ATN-style arcs in the grammar (6). The interrupts which activate new packets of rules are modelled in GSP by storing in the lexicon the code which causes control to shift to a new grammar state (since any change in the value of global variables, such as Riesbeck's 'request list', is formally equivalent to a new GSP-machine state).

For example, the lexical entry for the verb 'give' includes, among other things, the following:

INTERRUPT: (SETR CONCEPT (GETF DECOMPOSITION))
(NEWSTATE GIVEØ)

GSP's controlling executive knows to look at the 'interrupt' code stored in the lexicon. In this case, the code says to set the CONCEPT register to the structure found for this entry under the indicator DECOMPOSITION (i.e. its conceptual dependency representation), and to advance the grammar to the state labelled GIVEØ. State GIVEØ is the one which initiates the actions corresponding to Riesbeck's setting up of new expectations, and is basically an entire sub-grammar devoted to the verb 'give'. This sub-grammar asks specific questions about the attributes of incoming noun phrases, and assigns these phrases to case slots accordingly, e.g. the surface pattern HUMAN give HUMAN PHYSOBJ results in the second HUMAN getting placed in the RECIPIENT slot.

Although conceptual dependency emphasises the role of 'deep conceptual cases', the parser itself is necessarily a hybrid beast - it builds conceptual representation the instant a verb is encountered, but it fills vacant case slots on the basis of purely surface syntactic considerations, combined with relatively simple Katz & Fodor-type selectional restrictions (although Riesbeck does allow the restrictions to be relaxed if nothing else makes more sense). Riesbeck's parser could in fact build some other kind of internal representation, based on any one of several linguistic theories of lexical decomposition, without any effect on the course of parsing.

Riesbeck's parser discovers noun phrases when a signalling word, such as 'a', interrupts processing and passes control to a noun-phrase sub-grammar. Active production rules which were pending before the 'a' was discovered are saved on a special 'hold' list, and restored when a 'phrasebreak' (e.g. a verb or a period) is encountered. At that point, an EVAL-PHRASE routine (not unlike an ATN BUILDQ) constructs an internal representation of the just-parsed noun-phrase. In GSP terms, this activity is almost identical to the standard ATN 'PUSH NP', except that the potentially expensive aspects of a PUSH and POP (namely, saving return states and building possibly incomplete NP structures) are not carried out until it is virtually certain that they will succeed. This variation is handled in the GSP version by having the appearance of an item such as a determiner (which would normally be sought at the beginning of an NP sub-grammar) initiate a PUSH 'safely' after that key item is encountered in the text. This is done by having the 'interrupt' code for determiners pass control directly to state NP in the grammar. Then, instead of performing a tentative POP when the NP sub-grammar thinks it is finished, the

'interrupt' code of verbs (and other items which signal a 'phrasebreak' in various contexts) initiates the equivalent of a 'safe' POP directly when it is appropriate.

Of course 'safe' doesn't always mean 'guaranteed'. It is clear that there are all sorts of examples which would cause Riesbeck's parser to go astray. However, Riesbeck is concerned mainly about the fundamental operating principles, rather than performance on a specific case. The assumption is that these principles could be applied to more elaborate examples. This remains to be demonstrated for a so-called 'conceptually-based' parser, although a related theme underlying the workings of Marcus' parser(2) indicates that 'safe' PUSHes and POPs indeed make powerful parsing tools. The advantages of lookahead in parsing are significant only when such lookahead is relatively cheap and reasonably accurate. The GSP implementation of Riesbeck's parser allows one to automatically incorporate fairly simple lookahead within the framework of other, more well established parsing mechanisms (e.g. the PUSH NP of ATNs).

3. THE LNR PARSER

3.1 Overview

The LNR parser is designed to map sentences onto a surface case frame. This case frame is an n-ary predicate which contains a pointer to the appropriate code responsible for decomposing the predicate into a canonical 'meaning' representation. The case frame of each verb is stored in a memory network, and is referred to by the parser for guidance in picking up arguments to go with the main predicate. Aside from the subject and object slots, which typically surround the verb, other (arbitrary) case slots are signalled by the appearance of key prepositions.

3.2 The GSP version

The LNR parser is implemented as a fairly straightforward ATN, with the PUSH NP and PUSH PP arcs augmented with extra tests which must be passed for the transitions to be allowed. These tests are simply checks to see whether, say, the current prepositional phrase is consistent with the 'surface frame' stored in the lexical entry for the current verb, and whether the relevant case slot is still vacant. For example, the lexical entry for the verb 'give' includes the following:

```
SURFACE-FRAME: NP1: AGENT
                NP2: OBJ
                'TO': RECIPIENT
```

The slots 'NP1' and 'NP2' are signposts for the

parser, saying that the NPs preceding and following the verb should be placed into the AGENT and OBJ registers, respectively. The 'to' slot is a signpost saying that a prepositional phrase beginning with 'to' is acceptable, and that its (nested) noun phrase should be placed in the RECIP register.

The LNR parser only decomposes the main verb after the surface case frame is entirely filled in. Since the parser consults only the surface-frame for guidance during processing, the final internal representation has no effect on the actual course of parsing, and could have been based on any of several theories of lexical decomposition, including conceptual dependency.

Selectional restrictions are not included at all in the LNR parsing process - this must be done as part of the code which is executed during decomposition. The problem with this approach is that it is not clear how total failure of such selectional tests would influence the course of parsing itself. A method of incorporating Riesbeck-style tests is described below.

4. A SYNTHESIS

Many of the differences between the LNR and Riesbeck parsers are apparent rather than real. LNR's use of a surface frame as a signpost to guide a general procedure (e.g. PUSH NP) to work on a specific verb is isomorphic to Riesbeck's branching off to an entirely separate sub-grammar following the verb. Also, LNR's test for a 'vacant slot' is equivalent to Riesbeck's use of self-deleting production rules. Neither the final decomposed representation of the main verb nor the time at which decomposition occurs have any effect on the actual course of parsing for either Riesbeck or LNR.

The two parsers do, however, differ in terms of whether they use interrupts and selectional restrictions to guide parsing. Riesbeck's parser uses both, and is on these counts computationally superior to the LNR parser. The main strength of the LNR parser is its notational clarity, which derives from its description as an ATN. This makes it an ideal starting point for a hybrid parser made by augmenting it with Riesbeck's use of interrupts and selectional restrictions.

Beginning with an ATN specification of the LNR parser, Riesbeck-style interrupts are added by splicing into the lexicon the 'interrupt' code for items which can initiate 'safe' PUSHes (e.g. determiners) and items which can initiate 'safe' POPs (e.g. verbs). The old PUSHes and POPs then become redundant, and can be eliminated.

Selectional restrictions of the simplicity of Riesbeck's (i.e. looking for features such as HUMAN, PHYSOBJ, etc.) are then easily added. This simply involves augmenting LNR's surface-frame slots with selectional information, e.g.

'TO': RECIP/HUMAN,

and then conjoining a general test onto the PUSH PP arc so that it tests whether the noun phrase following the preposition is an instance of the class named after the '/'.

Preliminary results with my own hybrid LNR/Riesbeck Parser (implemented in my POP-2 version of GSP) indicate that the combination is cost-effective for simple sentences, where the occurrence of obviously ill-fated PUSHes (e.g. testing for non-existing prepositional phrases) can be totally avoided. The selectional restrictions provide an extra heuristic for case slot-filling purposes, but can be easily fooled by counterexamples, so are better treated as 'preferences' to be followed tentatively. The most promising line for future development is the use of 'safe' PUSHes and POPs to handle multi-constituent lookahead, as in (2). I am currently working on a GSP version of Marcus' parser. Insights gained from placing his parser in a common framework with that of Riesbeck and LNR will be described in a forthcoming paper.

REFERENCES

- (1) Kaplan, R.M. A general syntactic processor. In R. Rustin (ed.), Natural language processing. Englewood Cliffs, N.J.: Prentice-Hall, 1973.
- (2) Marcus, M. A theory of syntactic recognition for natural language. AI Memo, AI Laboratory, MIT, Cambridge, Mass., 1978.
- (3) Norman, D.A., Rumelhart, D.E., and the LNR Research Group. Explorations in cognition. San Francisco: W.H. Freeman, 1975.
- (4) Riesbeck, C.K. Computational understanding: analysis of sentences and context. Working paper 4, Istituto per gli Studi Semantici e Cognitivi, Castagnola, Switzerland, 1974.
- (5) Schank, R.C. Conceptual dependency: a theory of natural language understanding. Cognitive Psychology, 1972, 3, 552-631.
- (6) Woods, W.A. Transition network grammars for natural language analysis. Comm. ACM 1970, 13, 591-606.
- (7) Wilks, Y. An intelligent analyser and understander of English. Comm. ACM, 1975, 18, 264-274.