Schank/Riesbeck vs. Norman/Rumelhart: What's the Difference?

Marc Eisenstadt
The Open University
Milton Keynes, ENGLAND

This paper explores the fundamental differences between two sentence-parsers developed in the early 1970's: Riesbeck's parser for Schank's 'conceptual dependency' theory (4, 5), and the 'LNR' parser for Norman and Rumelhart's 'active semantic network' theory (3). The Riesbeck parser and the LNR parser share a common goal - that of transforming an input sentence into a canonical form for later use by memory/inference/paraphrase processes. For both parsers, this transformation <u>is</u> the act of 'comprehension', although they appear to go about it in very different ways. Are these differences real or apparent?

Riesbeck's parser is implemented as a production system, in which input text can either satisfy the condition side of any production rule within a packet of currently-active rules, or else interrupt processing by disabling the current packet of rules and enabling ('triggering') a new packet of rules. In operation, the main verb of each segment of text is located, and a pointer to its lexical decomposition (canonical form) is established in memory. The surrounding text, primarily noun phrases, is then systematically mapped onto vacant case frame slots within the memory representation of the decomposed verb. Case information is signposted by a verb-triggered packet of production rules which expects certain classes of entity (e.g. <u>animate</u> recipient) to be encountered in the text. Phrase boundaries are handled by keyword-triggered packets of rules which initiate and terminate the parsing of phrases.

In contrast to this, the LNR parser is implemented as an augmented transition network, in which input text can either satisfy a current expectation or cause backtracking to a point at which an alternative expectation can be satisfied. In operation, input text is mapped onto a surface case frame, which is an n-ary predicate containing a pointer to the appropriate code responsible for decomposing the predicate into canonical form. Case information is signposted by property-list indicators stored in the lexical entry for verbs. These indicators act as signals or flags which are inspected by augmented tests on PUSH NP and PUSH PP arcs in order to decide whether such transitions are to be allowed. Phrase boundaries are handled by the standard ATN PUSH and POP mechanisms, with provision for backtracking if an initially-fulfilled expectation later turns out to have been incorrect.

In order to determine which differences are due to notational conventions, I have implemented versions of both parsers in Kaplan's General Syntactic Processor (GSP) formalism (2), a simple but elegant generalization of ATNs. In GSP terms, Riesbeck's active packets of production rules are grammar <u>states</u>, and each rule is represented as a grammar <u>arc</u>. Rule-packet triggering is handled by storing in the <u>lexicon</u> the GSP code which transfers control to a new grammar state when an interrupt is called for. Each packet is in effect a sub-grammar of the type handled normally by an ATN PUSH and POP. The important difference is that the expensive actions normally associated with PUSH and POP (e.g. saving registers, building structures) only occur after it is <u>safe</u> to perform them. That is, bottom-up interrupts and very cheap 'lookahead' ensure that wasteful backtracking is largely avoided.

Riesbeck's verb-triggered packet of rules (i.e. the entire sub-grammar which is entered after the verb is encountered) is isomorphic to the LNR-style use of lexical flags, which are in effect 'raised' and 'lowered' solely for the benefit of augmented tests on verb-independent arcs. Where Riesbeck depicts a 'satisfied expectation' by deleting the relevant production rule from the currently-active packet, LNR achieves the same effect by using augmented tests on PUSH NP and PUSH PP arcs to determine whether a particular case frame slot has <u>already</u> been filled. Both approaches are handled with equal ease by GSP.

In actual practice, Riesbeck's case frame expectations are typically tests for simple selectional restrictions, whereas LNR's case frame expectations are typically tests for the order in which noun phrases are encountered. Prepositions, naturally, are used by both parsers as important case frame clues: Riesbeck has a verb-triggered action alter the interrupt code associated with prepositions so that they 'behave' in precisely the right way; this is isomorphic to LNR's flags which are stored in the lexical entry for a verb and examined by augmented tests on verb-independent prepositional phrase arcs in the grammar.

The behaviour of Riesbeck's verb-triggered packets (verb-dependent sub-grammars) is actually <u>independent</u> of <u>when</u> a pointer to the lexical decomposition of the verb is established (i.e. whether a pointer is added as soon as the verb is encountered or whether it is added after the end of the sentence has been reached). Thus, any claims about the possible advantages of 'early' or 'instantaneous' decomposition are moot. Since Riesbeck's cases are filled primarily on the basis of fairly simple selectional restrictions, there is no obvious reason why his parser couldn't have built some other kind of internal representation, based on any one of several linguistic theories of lexical decomposition. Although Riesbeck's decomposition <u>could</u> occur after the entire sentence has been parsed, LNR's decomposition <u>must</u> occur at this point, because it uses a network-matching algorithm to find already-present structures in memory, and relies upon the arguments of the main n-ary predicate of the sentence being as fully specified as possible.

Computationally, the major difference between the two parsers is that Riesbeck's parser uses interrupts to initiate 'safe' PUSHes and POPs to and from sub-grammars, whereas the LNR parser performs 'risky' PUSHes and POPs like any purely top-down parser. Riesbeck's mechanism is potentially very powerful, and the performance of the LNR parser can be improved by allowing this mechanism to be added automatically by the compiler which transforms an LNR augmented transition network into GSP machine code. Each parser can thus be mapped fairly cleanly onto the other, with the only irreconcilable difference between them being the degree to which they rely on verb-dependent selectional restrictions to guide the process of filling in case frames. This characterization of the differences between them, based on implementing them within a common GSP framework, is somewhat surprising, since (a) the differences have nothing to do with 'conceptual dependency' or 'active semantic networks' and (b) the computational difference between them immediately suggests a way to automatically incorporate bottom-up processing into the LNR parser to improve not only its efficiency, but also its psychological plausibility. A GSP implementation of a 'hybrid' version of the two parsers is outlined in (1).

REFERENCES

(1) Eisenstadt, M. Alternative parsers for conceptual dependency: getting there is half the fun. Proceedings of the sixth international joint conference on artificial intelligence, Tokyo, 1979.

(2) Kaplan, R.M. A general syntactic processor. In R. Rustin (Ed.) Natural language processing. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

(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, vol. 3, no. 4, 1972.