Chart Generation

Martin Kay
Stanford University and
Xerox Palo Alto Research Center
kay@parc.xerox.com

Abstract

Charts constitute a natural uniform architecture for parsing and generation provided string position is replaced by a notion more appropriate to logical forms and that measures are taken to curtail generation paths containing semantically incomplete phrases.

1 Charts

Shieber (1988) showed that parsing charts can be also used in generation and raised the question, which we take up again here, of whether they constitute a natural uniform architecture for parsing and generation. In particular, we will be interested in the extent to which they bring to the generation process advantages comparable to those that make them attractive in parsing.

Chart parsing is not a well defined notion. The usual conception of it involves at least four related ideas:

Inactive edges. In context-free grammar, all phrases of a given category that cover a given part of the string are equivalent for the purposes of constructing larger phrases. Efficiency comes from collecting equivalent sets of phrases into (inactive) *edges* and constructing edges from edges rather than phrases from phrases.

Active edges. New phrases of whatever size can be built by considering existing edges pair-wise if provision is made for partial phrases. Partial phrases are collected into edges that are said to be *active* because they can be thought of as actively seeking material to complete them.

The algorithm schema. Newly created edges are placed on an *agenda*. Edges are moved from the agenda to the *chart* one by one until none remains to be moved. When an edge is moved, all interactions between it and edges already in the chart are considered and any new edges that they give rise to are added to the agenda.

Indexing. The positions in the string at which phrases begin and end can be used to index edges so that the algorithm schema need consider interactions only between adjacent pairs.

Chart parsing is attractive for the analysis of natural languages, as opposed to programming languages, for the way in which it treats ambiguity. Regardless of the number of alternative structures for a particular string that a given phrase participates in, it will be constructed once and only once. Although the number of structures of a string can grow exponentially with the length of the string, the number of edges that needs to be constructed grows only with the square of the string length and the whole parsing process can be accomplished in cubic time.

Innumerable variants of the basic chart parsing scheme are possible. For example, if there were languages with truly free word order, we might attempt to characterize them by rules like those of context-free grammar, but with a somewhat different interpretation. Instead of replacing nonterminal symbols in a derivation with strings from the right-hand side of corresponding rules, we would remove the nonterminal symbol and insert the symbols from the right-hand side of the rule at arbitrary places in the string.

A chart parser for languages with free word order would be a minor variant of the standard one. An edge would take the form X_v, where v is a vector with a bit for every word in the string and showing which of those words the edge covers. There is no longer any notion of adjacency so that there would be no indexing by string position. Interesting interactions occur between pairs of edges whose bit vectors have empty intersections, indicating that they cover disjoint sets of words. There can now be as many edges as bit-vectors and, not surprisingly, the computational complexity of the parsing process increases accordingly.

2 Generation

A parser is a transducer from strings to structures or logical forms. A generator, for our purposes, is the inverse. One way to think of it, therefore, is as a parser of structures or logical forms that delivers analyses in the form of strings. This view has the apparent disadvantage of putting insignificant differences in the syntax of a logical forms, such as the relative order of the arguments to symmetric operators, on the same footing as more significant facts about them. We know that it will not generally be possible to reduce

logical expressions to a canonical form but this does not mean that we should expect our generator to be compromised, or even greatly delayed, by trivial distinctions. Considerations of this kind were, in part, responsible for the recent resurgence of interest in "flat" representations of logical form (Copestake *et al.*,1996) and for the representations used for transfer in Shake-and-Bake translation (Whitelock, 1992). They have made semantic formalisms like those now usually associated with Davison (Davidson, 1980, Parsons, 1990) attractive in artificial intelligence for many years (Hobbs 1985, Kay, 1970). Operationally, the attraction is that the notations can be analyzed largely as free word-order languages in the manner outlined above. Consider the expression (1)

(1) r: run(r), past(r), fast(r), arg1(r, j), name(j, John)

which we will take as a representation of the logical form of the sentences *John ran fast* and *John ran quickly*. It consists of a distinguished index (r) and a list of predicates whose relative order is immaterial. The distinguished index identifies this as a sentence that makes a claim about a running event. "John" is the name of the entity that stands in the 'arg1' relation to the running which took place in the past and which was fast. Nothing turns on these details which will differ with differing ontologies, logics, and views of semantic structure. What concerns us here is a procedure for generating a sentence from a structure of this general kind.

Assume that the lexicon contains entries like those in (2) in which the italicized arguments to the semantic predicates are variables.

(2)

Words	Cat	Semantics
John	np(x)	x: name(x, John)
ran	vp(x, y)	x: run(x), arg1(x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

A *prima facie* argument for the utility of these particular words for expressing (1) can be made simply by noting that, *modulo* appropriate instantiation of the variables, the semantics of each of these words subsumes (1).

3 The Algorithm Schema

The entries in (2), with their variables suitably instantiated, become the initial entries of an agenda and we begin to move them to the chart in accordance with the algorithm

schema, say in the order given.

The variables in the 'Cat' and 'Semantics' columns of (2) provide the essential link between syntax and semantics. The predicates that represent the semantics of a phrase will simply be the union of those representing the constituents. The rules that sanction a phrase (e.g. (3) below) show which variables from the two parts are to be identified.

When the entry for *John* is moved, no interactions are possible because the chart is empty. When *run* is moved, the sequence *John ran* is considered as a possible phrase on the basis of rule (3).

(3) s(x) $\rightarrow$ np(y), vp(x, y).

With appropriate replacements for variables, this maps onto the subset (4) of the original semantic specification in (1).

(4) r: run(r), past(r), arg1(r, j), name(j, John)

Furthermore it is a complete sentence. However, it does not count as an output to the generation process as a whole because it subsumes some but not all of (1). It therefore simply becomes a new edge on the agenda.

The string *ran fast* constitutes a verb phrase by virtue of rule (5) giving the semantics (6), and the phrase *ran quickly* with the same semantics is put on the agenda when the *quickly* edge is move to the chart.

(5) vp(x) $\rightarrow$ vp(x) adv(x)

(6) r: run(r), past(r), fast(r), arg1(r, y)

The agenda now contains the entries in (7).

(7)

Words	Cat	Semantics
John ran	s(r)	r: run(r), past(r), arg1(r, j), name(j, John)
ran fast	vp(r, j)	r: run(r), past(r), fast(r), arg1(r, j)
ran quickly	vp(r, j)	r: run(r), past(r), fast(r), arg1(r, j)

Assuming that adverbs modify verb phrases and not sentences, there will be no interactions when the *John ran* edge is moved to the chart.

When the edge for *ran fast* is moved, the possibility arises of creating the phrase *ran fast quickly* as well as *ran fast fast*. Both are rejected, however, on the grounds that they would involve using a predicate from the original semantic specification more than once. This would be similar to allowing a given word to be covered by overlapping phrases in free word-order parsing. We proposed eliminating this by means of a bit vector and the same technique applies here. The fruitful interactions that occur here are between *ran fast* and *ran quickly* on the one hand, and *John*

on the other. Both give sentences whose semantics subsumes the entire input.

Several things are noteworthy about the process just outlined.

1. Nothing turns on the fact that it uses a primitive version of event semantics. A scheme in which the indices were handles referring to subexpressions in any variety of flat semantics could have been treated in the same way. Indeed, more conventional formalisms with richly recursive syntax could be converted to this form on the fly.

2. Because all our rules are binary, we make no use of active edges.

3. While it fits the conception of chart parsing given at the beginning of this paper, our generator does not involve string positions centrally in the chart representation. In this respect, it differs from the proposal of Shieber (1988) which starts with all word edges leaving and entering a single vertex. But there is essentially no information in such a representation. Neither the chart nor any other special data structure is required to capture the fact that a new phrase may be constructible out of any given pair, and in either order, if they meet certain syntactic and semantic criteria.

4. Interactions must be considered explicitly between new edges and all edges currently in the chart, because no indexing is used to identify the existing edges that could interact with a given new one.

5. The process is exponential in the worst case because, if a sentence contains a word with k modifiers, then a version it will be generated with each of the 2^k subsets of those modifiers, all but one of them being rejected when it is finally discovered that their semantics does not subsume the entire input. If the relative orders of the modifiers are unconstrained, matters only get worse.

Points 4 and 5 are serious flaws in our scheme for which we shall describe remedies. Point 2 will have some importance for us because it will turn out that the indexing scheme we propose will require the use of distinct active and inactive edges, even when the rules are all binary. We take up the complexity issue first, and then turn to how the efficiency of the generation chart might be enhanced through indexing.

4   Internal and External Indices

The exponential factor in the computational complexity of our generation algorithm is apparent in an example like (8).

(8)   Newspaper reports said the tall young Polish athlete ran fast

The same set of predicates that generate this sentence clearly also generate the same sentence with deletion of all subsets of the words *tall, young,* and *Polish* for a total of 8 strings. Each is generated in its entirety, though finally rejected because it fails to account for all of the semantic material. The words *newspaper* and *fast* can also be deleted independently giving a grand total of 32 strings.

We concentrate on the phrase *tall young Polish athlete* which we assumed would be combined with the verb phrase *ran fast* by the rule (3). The distinguished index of the noun phrase, call it p, is identified with the variable *y* in the rule, but this variable is not associated with the syntactic category, s, on the left-hand side of the rule. The grammar has access to indices only through the variables that annotate grammatical categories in its rules, so that rules that incorporate this sentence into larger phrases can have no further access to the index p. We therefore say that p is *internal* to the sentence *the tall young Polish athlete ran fast.*

The index p would, of course, also be internal to the sentences *the young Polish athlete ran fast, the tall Polish athlete ran fast,* etc. However, in these cases, the semantic material remaining to be expressed contains predicates that refer to this internal index, say 'tall(p)', and 'young(p)'. While the lexicon may have words to express these predicates, the grammar has no way of associating their referents with the above noun phrases because the variables corresponding to those referents are internal. We conclude that, as a matter of principle, no edge should be constructed if the result of doing so would be to make internal an index occurring in part of the input semantics that the new phrase does not subsume. In other words, the semantics of a phrase must contain all predicates from the input specification that refer to any indices internal to it. This strategy does not prevent the generation of an exponential number of variants of phrases containing modifiers. It limits proliferation of the ill effects, however, by allowing only the maximal one to be incorporated in larger phrases. In other words, if the final result has phrases with m and n modifiers respectively, then 2^n versions of the first and 2^m of the second will be created, but only one of each set will be incorporated into larger phrases and no factor of $2^{(n+m)}$ will be introduced into the cost of the process.

5   Indexing

String positions provide a natural way to index the strings input to the parsing process for the simple reason that there are as many of them as there are words but, for there to be any possibility of interaction between a pair of edges, they must come together at just one index. These are the natural points of articulation in the domain of strings. They cannot fill this role in generation because they are not natural properties of the semantic expressions that are the input to the process. The corresponding natural points of articulation in

202

flat semantic structures are the entities that we have already been referring to as *indices*.

In the modified version of the procedure, whenever a new inactive edge is created with label B(*b* ...), then for all rules of the form in (9), an active edge is also created with label A(...)/C(*c* ...).

(9) A(...) → B(*b* ...) C(*c* ...)

This represents a phrase of category A that requires a phrase of category C on the right for its completion. In these labels, *b* and *c* are (variables representing) the first, or *distinguished* indices associated with B and C. By analogy with parsing charts, an inactive edge labeled B(*b* ...) can be thought of as *incident from* vertex *b*, which means simply that it is efficiently accessible through the index *b*. An active edge A(...)/C(*c* ...) should be thought of as incident from, or accessible through, the index *c*. The key property of this scheme is that active and inactive edges interact by virtue of indices that they share and, by letting vertices correspond to indices, we collect together sets of edges that could interact.

We illustrate the modified procedure with the sentence (10) whose semantics we will take to be (11), the grammar rules (12)-(14), and the lexical entries in (15).

(10) The dog saw the cat.
(11) dog(d), def(d), saw(s), past(s), cat(c), def(c), arg1(s, d), arg2(s, c).
(12) s(x) → np(y) vp(x, y)
(13) vp(x, y) → v(x, y, z) np(z)
(14) np(x) → det(x) n(x)
(15)

Words	Cat	Semantics
cat	n(x)	x: cat(x)
saw	v(x, y, z)	x: see(x), past(x), arg1(x, y), arg2(x,z)
dog	n(x)	x: dog(x)
the	det(x)	x: def(x)

The procedure will be reminiscent of left-corner parsing. Arguments have been made in favor of a head-driven strategy which would, however, have been marginally more complex (e.g. in Kay (1989), Shieber, *et el.* (1989)) and the differences are, in any case, not germane to our current concerns.

The initial agenda, including active edges, and collecting edges by the vertices that they are incident from, is given in (16).

The grammar is consulted only for the purpose of creating active edges and all interactions in the chart are between active and inactive pairs of edges incident from the

same vertex.

(16)

Vert	Words	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	dog	n(d)	d: dog(d)
s	saw	v(s, d, c)	s: see(s), past(s), arg1(s, d), arg2(s, c)
c	saw	vp(s, d)/np(c)	r: see(s), past(s), arg1(r, j)
	the	det(c)	c: def(c)
	the	np(c)/n(c)	c: def(c)
	cat	n(c)	c: dog(c)

(17)

Vert	Words	Cat	Semantics
d	the dog	np(d)	d: dog(d), def(d)
	saw the cat	vp(s, d)/np(d)	s: see(s), past(s), arg1(s, d), arg2(s, c), cat(c), def(c)
c	the cat	np(c)	c: cat(c), def(c)
s	saw the cat	vp(s, d)	s: see(s), past(s), arg1(s, d), arg2(s, c), cat(c), def(c)

Among the edges in (16), there are two interactions, one at vertices c and d. They cause the first and third edges in (17) to be added to the agenda. The first interacts with the active edge originally introduced by the verb "saw" producing the fourth entry in (17). The label on this edge matches the first item on the right-hand side of rule (12) and the active edge that we show in the second entry is also introduced. The final interaction is between the first and second edges in (17) which give rise to the edge in (18).

This procedure confirms perfectly to the standard algorithm schema for chart parsing, especially in the version that makes predictions immediately following the recognition of the first constituent of a phrase, that is, in the version that is essentially a caching left-corner parser.

(18)

Vert	Words	Cat	Semantics
s	The dog saw the cat	s(s)	dog(d), def(d), see(s), past(s),arg1(s, d), arg2(s, c), cat(c), def(c).

6 Acknowledgments

References

Copestake, A., Dan Flickinger, Robert Malouf, and Susanne Riehemann, and Ivan Sag (1996). Translation Using Minimal Recursion Semantics. *Proceedings of The Sixth International Conference on Theoretical and Methodological Issues in Machine Translation*, Leuven (in press).

Davidson, D. (1980). *Essays on Actions and Events.* *Oxford*: The Clarendon Press.

Hobbs, J. R. (1985). Ontological Promiscuity. *23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, ACL.

Kay, M. (1970). From Semantics to Syntax. *Progress in Linguistics*. Bierwisch Manfred, and K. E. Heidolf. The Hague, Mouton: 114-126.

Kay, M. (1989). Head-driven Parsing. *Proceedings of Workshop on Parsing Technologies*, Pittsburgh, PA.

Parsons, T. (1990). *Events in the Semantics of English.* Cambridge, Mass.: MIT Press.

Shieber, S. (1988). A Uniform Architecture for Parsing and Generation. *COLING-88*, Budapest, John von Neumann Society for Computing Sciences.

Shieber, S. M. *et al.* (1989). A Semantic-Head-Driven Generation Algorithm for Unification Based Formalisms. *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver. B.C.

Whitelock, P. (1992). Shake and-Bake Translation. *COLING-92*, Nantes.