Interactive Incremental Chart Parsing

Mats Wirén
Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden
mgw@ida.liu.se

Abstract

This paper presents an algorithm for incremental chart parsing, outlines how this could be embedded in an interactive parsing system, and discusses why this might be useful. Incremental parsing here means that input is analysed in a piecemeal fashion, in particular allowing arbitrary changes of previous input without exhaustive reanalysis. Interactive parsing means that the analysis process is prompted immediately at the onset of new input, and possibly that the system then may interact with the user in order to resolve problems that occur. The combination of these techniques could be used as a parsing kernel for highly interactive and "reactive" natural-language processors, such as parsers for dialogue systems, interactive computer-aided translation systems, and language-sensitive text editors. An incremental chart parser embodying the ideas put forward in this paper has been implemented, and an embedding of this in an interactive parsing system is near completion.

1 Background and Introduction

1.1 The Problem

Ideally, a parser for an interactive natural-language system ought to analyse input in real time in such a way that the system produces an analysis of the input while this is being received. One aspect of this is that the system should be able to "keep up" with new input that, piece by piece, is entered from left to right. Another aspect is that it ought to be able to keep up also with piecemeal *changes* of previous input. For example, in changing one word in the beginning of some utterance(s), one would not want all the input (either from the beginning or from the change point) to be completely reanalysed. From the perspective of efficiency as well as of modelling intelligent behaviour, the amount of processing required to analyse an update ought to be somehow correlated with the difficulty of this update. Thus, a necessary (but not sufficient) condition for realizing a real-time parsing system as suggested above is an *interactive* and *incremental* parsing system. The goal of this paper is to develop a basic machinery for incremental chart parsing and to outline how this could be embedded in an interactive parsing system.

1.2 Incremental Parsing

The word "incremental" has been used in two differing senses in the (parsing) literature. The first sense stresses that input should be analysed in a *piecemeal* fashion, for example Bobrow and Webber (1980), Mellish (1985), Pulman (1985, 1987), Hirst (1987), Haddock (1987). According to this view, an incremental parser constructs the analysis of an utterance bit by bit (typically from left to right), rather than in one go when it has come to an end.

The other sense of "incremental" stresses the necessity of *efficiently handling arbitrary changes* within current input. Thus, according to this view, an incremental parser should be able to efficiently handle not only piecemeal additions to a sentence, but, more generally, arbitrary insertions and deletions in it. This view of incremental parsing is typical of research on interactive programming environments, e.g. Lindstrom (1970), Earley and Caizergues (1972), Ghezzi and Mandrioli (1979, 1980), Reps and Teitelbaum (1987).

As indicated above, we are here interested in the latter view, which we summarize in the following working definition.

This research has been supported by the National Swedish Board for Technical Development. The system is implemented on machines donated by the Xerox Corporation through their University Grants Program.

I would like to thank several people for fruitful discussions on the topics of this paper, in particular Lars Ahrenberg (also for commenting on drafts), Bernt Nilsson, and Peter Fritzson; furthermore Nils Dahlbäck, Arne Jönsson, Magnus Merkel, Henry Thompson, and an anonymous referee. In addition, I would like to thank Ulf Dahlén, Åsa Detterfelt, Mikael Karlsson, Per Larses, Jukka Nylund, and Michael Spicar for implementing (the interactive portion of) LIPS.

Incremental parser. A parser capable of handling changes of previous input while expending an amount of effort which is proportional to the complexity of the changes.[1]

It should be pointed out that we are here limiting ourselves to a machinery for incremental parsing as opposed to incremental interpretation. In other words, the derivation of an utterance here takes into account only "context-free" (lexical, syntactic, compositional-semantic) information obtained from grammar and dictionary. Nevertheless, I believe that this framework may be of some value also when approaching the more difficult problem of incremental interpretation.

1.3 Interactive Parsing

We adopt the following working definition.

Interactive parser. (Synonym: *on-line parser.*) A parser which monitors a text-input process, starting to parse immediately at the onset of new input, thereby achieving enhanced efficiency as well as a potential for dynamic improvement of its performance, for example by promptly reporting errors, asking for clarifications, etc.[2]

Within the area of programming environments, (generators for) language-based editors have been developed that make use of interactive (and incremental) parsing and compilation to perform program analysis, to report errors, and to generate code while the program is being edited, for example Mentor, Gandalf, and the Synthesizer Generator (Reps and Teitelbaum 1987).

Within natural-language processing, Tomita (1985) and Yonezawa and Ohsawa (1988) have reported parsers which operate on-line, but, incidentally, not incrementally in the sense adopted here.[3]

[1]This definition is formed partly in analogy with a definition of "incremental compilation" by Earley and Caizergues (1972:1040). We use "complexity" instead of "size" because different updates of the same size may cause differing processing efforts depending on the degree of grammatical complexity (ambiguity, context-sensitiveness) constraining the updates in question.

[2]Incidentally, interactive parsing could be seen as one example of a general trend towards *immediate computation* (Reps and Teitelbaum 1987:31), also manifest in applications such as WYSIWYG word processing and spreadsheet programs, and sparked off by the availability of personal workstations with dedicated processors.

[3]The user may delete input from right to left, causing the systems to "unparse" this input. This means that if the user wants to update some small fragment in the beginning of a sentence, the system has to reparse exhaustively from this update and on. (Of course, in reality the user has to first backspace and then retype everything from the change.)

1.4 Outline of Paper

Section 2 presents an algorithm for incremental char parsing. Section 3 discusses some additional aspect and alternative strategies. Section 4 gives a brie outline of the combined interactive and incrementa parsing system, and section 5 summarizes the con clusions.

2 Incremental Chart Parsing

2.1 Chart Parsing

The incremental parser has been grounded in chart-parsing framework (Kay 1980, Thompso 1981, Thompson and Ritchie 1984) for the follow ing reasons:

- chart parsing is an efficient, open-ended, we understood, and frequently adopted techniqu in natural-language processing;
- chart parsing gives us a previously unexplore possibility of embedding incrementality at a lo cost.

2.2 Edge Dependencies

The idea of incremental chart parsing, as put fo ward here, is based on the following observatio The chart, while constituting a record of parti analyses (chart edges), may easily be provided wit information also about the *dependencies* betwee those analyses. This is just what we need in in cremental parsing since we want to propagate th effects of a change precisely to those parts of th previous analysis that, directly or indirectly, depen on the updated information.

In what ways could chart edges be said to depen on each other? Put simply, an edge depends upo another edge if it is formed using the latter edg Thus, an edge formed through a prediction step de pends on the (one) edge that triggered it.[4] Likewis an edge formed through a combination[5] depends o the active–inactive edge pair that generated it. scanned edge, on the other hand, does not depen upon any other edge, as scanning can be seen as kind of initialization of the chart.[6]

In order to account for edge dependencies we ass ciate with each edge the set of its immediate sourc

[4]In the case of an initial top-down prediction, the sour would be non-existent.

[5]The *completer* operation in Earley (1970); the *fundament rule* in Thompson (1981:2).

[6]It might be argued that a dependency should be esta lished also in the case of an edge being proposed but reject (owing to a redundancy test) because it already exists. Ho ever, as long as updates affect all preterminal edges extendi from a vertex, this appears not to be crucial.

edges ("back pointers"). This information could be used to derive the corresponding sets of dependent edges ("forward pointers") that we are interested in. For example, when a word in the previous input has been deleted, we want to remove all edges which depend on the preterminal (lexical) edge(s) corresponding to this word, as well as those preterminal edges themselves.

Formally, let $\mathcal{D}$ be a binary dependency relation such that $e \mathcal{D} e'$ if and only if e' is a dependant of e, i.e., e' has been formed (directly) using e. If $\mathcal{D}^*$ is the reflexive transitive closure of $\mathcal{D}$, all edges e'' should be removed for which $e \mathcal{D}^* e''$ holds, i.e., all edges which directly or indirectly depend on e, as well as e itself. In addition, we are going to make use of the transitive closure of $\mathcal{D}$, $\mathcal{D}^+$.

The resulting style of incremental parsing resembles truth (or reason) maintenance, in particular ATMS (de Kleer 1986). A chart edge here corresponds to an ATMS *node*, a preterminal edge corresponds to an *assumption node*, the immediate source information of an edge corresponds to a *justification*, the dependency relation $\mathcal{D}^*$ provides information corresponding to ATMS *labels*, etc.

2.3 Technical Preliminaries

2.3.1 The Chart

The *chart* is a directed graph. The nodes, or *vertices*, $v_1, \ldots, v_{n+1}$ correspond to the positions surrounding the words of an n-word sentence $w_1 \cdots w_n$. A pair of vertices v_i, v_j may be connected by arcs, or *edges*, bearing information about (partially) analysed constituents between v_i and v_j. We will take an edge to be a tuple

$$\langle s, t, X_0 \rightarrow \alpha.\beta, D, E \rangle$$

starting from vertex v_s and ending at vertex v_t with dotted rule $X_0 \rightarrow \alpha.\beta$,[7] a dag D (cf. section 2.3.3), and the set of immediately dependent edges, E.[8]

In order to lay the ground for easy splitting and joining of chart fragments, we will take a vertex to consist of three parts, $\langle L, A_{loop}, R \rangle$, left, middle, and right. L and R will have internal structure, so that the full vertex structure will come out like

$$\langle \langle A_{in}, I_{in} \rangle, A_{loop}, \langle A_{out}, I_{out} \rangle \rangle$$

The left part, $\langle A_{in}, I_{in} \rangle$, consists of the incoming active and inactive edges which will remain with the left portion of the chart when it is split due to some internal sentence-editing operation. Correspondingly, the right part, $\langle A_{out}, I_{out} \rangle$, consists of the outgoing active and inactive edges which will remain with the right portion of the chart. The middle part, A_{loop}, consists of the active looping edges which, depending on the rule-invocation strategy, should remain either with the left or the right portion of the chart (cf. section 3.1).

We will make use of dots for qualifying within elements of tuples. For example, $e.s$ will stand for the starting vertex of edge e. Likewise, $v_i.L$ will stand for the set of edges belonging to the left half of vertex number i, and $v_i.A_{in}$ will denote the set of its active incoming edges. In addition, we will use $v_i.P_{out}$ as a shorthand for the set of inactive outgoing edges at v_i which are also preterminal (lexical).

2.3.2 Editing Operations

In general, parsing could be seen as a mapping from a sentence to a structure representing the analysis of the sentence — in this case a chart. Incremental parsing requires a more complex mapping

$$F(\eta, \kappa, \tau, c_0) \mapsto c_1$$

from an edit operation η, a pair of cursor positions κ, a sequence of words τ (empty in the case of deletion), and an initial chart c_0 to a new chart c_1 (and using a grammar and dictionary as usual).

We are going to assume three kinds of editing operation, *insert*, *delete*, and *replace*. Furthermore, we assume that every operation applies to a continuous sequence of words $w_l \cdots w_r$, each of which maps to one or several preterminal edges extending from vertices $v_l, \ldots, v_r$, respectively.[9]

Thus, η may here take the values *insert*, *delete*, or *replace*; κ is a pair of positions l, r such that the sequence of positions $l, \ldots, r$ map directly to vertices $v_l, \ldots, v_r$, and τ is the corresponding sequence of words $w_l \cdots w_r$.

In addition, we will make use of the constant $\delta = r - l + 1$, denoting the number of words affected by the editing operation.

2.3.3 Grammatical Formalism

In the algorithm below, as well as in the actual implementation, we have adopted a unification-based grammatical formalism with a context-free base, PATR (Shieber et al. 1983, Shieber 1986), because this seems to be the best candidate for a *lingua franca* in current natural-language processing. However, this formalism here shows up only within the edges, where we have an extra dag element (D), and when referring to rules, each of which consists of a

[7] A dotted rule $X_0 \rightarrow \alpha.\beta$ corresponds to an (active) X_0 edge containing an analysis of constituent(s) α, requiring constituent(s) β in order to yield an inactive edge.

[8] In other words, the set E of an edge e consists of all edges e_i for which $e \mathcal{D} e_i$ holds.

[9] Character editing is processed by the scanner; cf. section 3.3.

pair $\langle X_0 \rightarrow \gamma, D\rangle$ of a production and a dag. In the dag representation of the rule, we will store the context-free base under *cat* features as usual. We assume that the grammar is cycle-free.

2.4 An Algorithm for Incremental Chart Parsing

2.4.1 Introduction

This section states an algorithm for incremental chart parsing, divided into update routines, subroutines, and an underlying chart parser. It handles update of the chart according to *one* edit operation; hence, it should be repeated for each such operation. The underlying chart parser specified in the end of section 2.4.2 makes use of a bottom-up rule-invocation strategy. Top-down rule invocation will be discussed in section 3.1.

2.4.2 Incremental Chart-Parsing Algorithm

Input: An edit operation η, a pair of vertex numbers l, r, a sequence of words $w_l \cdots w_r$, and a chart c_0. We assume that chart c_0 consists of vertices $v_1, \ldots, v_{last}$, where $last \geq 1$. We furthermore assume the constant $\delta = r - l + 1$ to be available.

Output: A chart c_1.

Method: On the basis of the input, select and execute the appropriate update routine below.

Update Routines

Insert1: Insertion at right end of c_0

```
for $i$ := $l$, ..., $r$ do Scan($w_i$);
$last$ := $last + \delta$;
RunChart.
```

This case occurs when δ words $w_l \cdots w_r$ have been inserted at the right end of previous input (i.e., $l = last$). This is the special case corresponding to ordinary left-to-right chart parsing, causing the original chart c_0 to be extended δ steps to the right.

Delete1: Deletion at right end of c_0

```
for $i$ := $l$, ..., $r$ do
    $\forall e$: $e \in v_i.P_{out}$ RemoveEdgesInD*($e$);
$last$ := $last - \delta$.
```

This case occurs when δ words $w_l \cdots w_r$ have been deleted up to and including the right end of previous input (i.e., $r = last - 1$). It is handled by removing the preterminal edges corresponding to the deleted words along with all their dependent edges.

Delete2: Deletion before right end of c_0

```
for $i$ := $l$, ..., $r$ do
    $\forall e$: $e \in v_i.P_{out}$ RemoveEdgesInD*($e$);
MoveVertex/RightHalf($r + 1, l, -\delta$);
for $i$ := $l + 1$ to $last - \delta$ do
    MoveVertex($i + \delta, i, -\delta$);
$last$ := $last - \delta$;
RunChart.
```

This case occurs when δ words $w_l \cdots w_r$ have been deleted in an interval within or at the left end of previous input (i.e., $r < last - 1$). It is handled by removing the preterminal edges corresponding to the deleted words along with all their dependent edges, and then collapsing the chart, moving all edges from vertex v_{r+1} and on δ steps to the left.

Insert2: Insertion before right end of c_0

```
RemoveCrossingEdges($l$);
for $i$ := $last$ downto $l + 1$ do
    MoveVertex($i, i + \delta, \delta$);
MoveVertex/RightHalf($l, r + 1, \delta$);
for $i$ := $l$, ..., $r$ do Scan($w_i$);
$last$ := $last + \delta$;
RunChart.
```

This case occurs when δ words $w_l \cdots w_r$ have been inserted at a position within or at the left end of previous input (i.e., $l < last$). It is handled by first removing all edges that "cross" vertex v_l (the vertex at which the new insertion is about to start). Secondly, the chart is split at vertex v_l by moving all edges extending from this vertex or some vertex to the right of it δ steps to the right. Finally, the new input is scanned and the resulting edges inserted into the chart.

Replace: Replacement within c_0

```
for $i$ := $l$, ..., $r$ do
    $\forall e$: $e \in v_i.P_{out}$ RemoveEdgesInD*($e$);
for $i$ := $l$, ..., $r$ do Scan($w_i$);
RunChart.
```

This case occurs when δ words $w_l \cdots w_r$ have been replaced by δ other words at the corresponding positions within previous input (i.e., $1 \leq l$ and $r \leq last$; typically $l = r$). It is handled by first removing the preterminal edges corresponding to the replaced words along with all their dependent edges, and then scan the new words and insert the resulting edges into the chart.

Alternatively, we could of course realize replace through delete and insert, but having a dedicated replace operation is more efficient.

Subroutines

RemoveEdgesInD*(e)**:**

$\forall e'$: $e\ \mathcal{D}^*\ e'$ remove e'.

This routine removes all edges that are in the reflexive transitive dependency closure of a given edge e.[10]

MoveVertex$(from, to, \delta)$**:**

```
$v_{to} := v_{from}$;
$v_{from} := \emptyset$;
$\forall e$: $e \in v_{to}.A_{loop} \cup v_{to}.R$
    $e.s := e.s + \delta$;
    $e.t := e.t + \delta$.
```

This routine moves the contents of a vertex from v_{from} to v_{to} and assigns new connectivity information to the affected (outgoing) edges.

MoveVertex/RightHalf$(from, to, \delta)$**:**

```
$v_{to}.R := v_{from}.R$;
$v_{to}.A_{loop} := v_{from}.A_{loop}$;
$v_{from}.R := \emptyset$;
$v_{from}.A_{loop} := \emptyset$;
$\forall e$: $e \in v_{to}.A_{loop} \cup v_{to}.R$
    $e.s := e.s + \delta$;
    $e.t := e.t + \delta$.
```

This routine moves the contents of the right half (including active looping edges) of a vertex from v_{from} to v_{to} and assigns new connectivity information to the affected (outgoing) edges.

RemoveCrossingEdges(e)**:**

```
$\forall e \forall f \forall g$:
    $f \in v_{l-1}.P_{out}$
    $g \in v_l.P_{out}$
    $e \in \{f\mathcal{D}^+e\} \cap \{g\mathcal{D}^+e\}$
        remove $e$.
```

The purpose of this routine, which is called from **Insert2**, is to remove all edges that "cross" vertex v_l where the new insertion is about to start. This can be done in different ways. The solution above makes use of dependency information, removing every edge which is a dependant of *both* some preterminal edge incident to the change vertex and some preterminal edge extending from it.[11] Alternatively, one could simply remove every edge e whose left connection $e.s < l$ and whose right connection $e.t > l$.

[10]It may sometimes be the case that not all edges in the dependency closure need to be removed because, in the course of updating, some edge receives the same value as previously. This happens for example if a word is replaced by itself, or, given a grammar with atomic categories, if (say) a noun is replaced by another noun. One could reformulate the routines in such a way that they check for this before removing an edge.

[11]For simplicity, we presuppose that preterminal edges only extend between adjacent vertices.

Chart Parser

Scan(w_i)**:**

If $w_i = a$, then, for all lexical entries of the form $\langle X_0 \rightarrow a, D\rangle$, add the edge $\langle i, i+1, X_0 \rightarrow a., D, \emptyset\rangle$.

Informally, this means adding an inactive, preterminal edge for each word sense of the word.

RunChart:

For each vertex v_i, do the following two steps until no more edges can be added to the chart.

1. **Predict/BottomUp:** For each edge e starting at v_i of the form $\langle i, j, X_0 \rightarrow \alpha., D, E\rangle$ and each rule of the form $\langle Y_0 \rightarrow Y_1\beta, D'\rangle$ such that $D'(\langle Y_1\ cat\rangle) = D(\langle X_0\ cat\rangle)$, add an edge of the form $\langle i, i, Y_0 \rightarrow .Y_1\beta, D', \{e\}\rangle$ if this edge is not subsumed[12] by another edge.

   Informally, this means predicting an edge according to each rule whose first right-hand-side category matches the category of the inactive edge under consideration.

2. **Combine:** For each edge e of the form $\langle i, j, X_0 \rightarrow \alpha.X_m\beta, D, E\rangle$ and each edge e' of the form $\langle j, k, Y_0 \rightarrow \gamma., D', E'\rangle$, add the edge $\langle i, k, X_0 \rightarrow \alpha X_m.\beta, D \sqcup [X_m\colon D'(Y_0)], \{e, e'\}\rangle$ if the unification succeeds and this edge is not subsumed by another edge.

   Informally, this means forming a new edge whenever the category of the first needed constituent of an active edge matches the category of an inactive edge,[13] and the dag of the inactive edge can be unified in with the dag of the needed constituent.

3 Discussion

3.1 Top-Down Parsing

The algorithm given in section 2.4.2 could be modified to top-down parsing by changing the predictor (see e.g. Wirén 1988) and by having **Move-Vertex/RightHalf** not move active looping edges ($v_l.A_{loop}$) since, in top-down, these "belong" to the left portion of the chart where the predictions of them were generated.

In general, the algorithm works better bottom-up than top-down because bottom-up predictions are

[12]One edge subsumes another edge if and only if the first three elements of the edges are identical and the fourth element of the first edge subsumes that of the second edge. For a definition of subsumption, see Shieber (1986:14).

[13]Note that this condition is tested by the unification which specifically ensures that $D(\langle X_m\ cat\rangle) = E(\langle Y_0\ cat\rangle)$.

made "locally" at the starting vertex of the triggering (inactive) edge in question. Therefore, a changed preterminal edge will typically have its dependants locally, and, as a consequence, the whole update can be kept local. In top-down parsing, on the other hand, predictions are "forward-directed", being made at the ending vertex of the triggering (active) edge. As a result of this, an update will, in particular, cause all predicted and combined edges after the change to be removed. The reason for this is that we have forward-directed predictions having generated active and inactive edges, the former of which in turn have generated forward-directed predictions, and so on through the chart.

On the one hand, one might accept this, arguing that this is simply the way top-down works: It generates forward-directed hypotheses based on the preceding context, and if we change the preceding context, the forward hypotheses should change as well. Also, it is still slightly more well-behaved than exhaustive reanalysis from the change.

On the other hand, the point of incremental parsing is to keep updates local, and if we want to take this seriously, it seems like a waste to destroy possibly usable structure to the right of the change. For example, in changing the sentence "Sarah gave Kim a green apple" to "Sarah gave a green apple to Kim", there is no need for the phrase "a green apple" to be reanalysed.

One approach to this problem would be for the edge-removal process to introduce a "cut" whenever a top-down prediction having some dependant edge is encountered, mark it as "uncertain", and repeatedly, at some later points in time, try to find a new source for it. Eventually, if such a source cannot be found, the edge (along with dependants) should be "garbage-collected" because there is no way for the normal update machinery to remove an edge without a source (except for preterminal edges).

In sum, it would be desirable if we were able to retain the open-endedness of chart parsing also with respect to rule invocation while still providing for efficient incremental update. However, the precise strategy for best achieving this remains to be worked out (also in the light of a fully testable interactive system).

3.2 Alternative Ways of Determining Affected Edges

3.2.1 Maintain Sources Only

Henry Thompson (personal communication 1988) has pointed out that, instead of computing sets of dependants from source edges, it might suffice to simply record the latter, provided that the frequency of updates is small and the total number of edges is not too large. The idea is to sweep the whole edge space each time there is an update, repeatedly deleting anything with a non-existent source edge, and iterating until one gets through a whole pass with no new deletions.

3.2.2 Maintain Neither Sources Nor Dependencies

If we confine ourselves to bottom-up parsing, and if we accept that an update will unconditionally cause all edges in the dependency closure to be removed (not allowing the kind of refinements discussed in footnote 10, it is in fact not necessary to record sources or dependencies at all. The reason for this is that, in effect, removing all dependants of all preterminal edges extending between vertices $v_l, \ldots, v_{r+1}$ in the bottom-up case amounts to removing all edges that extend somewhere within this interval (except for bottom-up predictions at vertex v_{r+1} which are triggered by edges outside of the interval). Given a suitable matrix representation for the chart (where edges are simultaneously indexed with respect to starting and ending vertices), this may provide for a very efficient solution.

3.2.3 Maintain Dependencies between Features

There is a trade-off between updating as local a unit as possible and the complexity of the algorithm for doing so. Given a complex-feature-based formalism like PATR, one extreme would be to maintain dependencies between feature instances of the chart instead of between chart edges. In principle, this is the approach of the Synthesizer Generator (Reps and Teitelbaum 1987), which adopts attribute grammar for the language specification and maintains dependencies between the attribute instances of the derivation tree.

3.3 Lexical Component

An approach to the lexical component which seems particularly suitable with respect to this type of parser, and which is adopted in the actual implementation, is the letter-tree format.[14] This approach takes advantage of the fact that words normally are entered from left to right, and supports the idea of a dynamic pointer which follows branches of the tree as a word is entered, immediately calling for reaction when an illegal string is detected. In particular, this allows you to distinguish an incomplete word from a (definitely) illegal word. Another advantage of this

[14] *Trie* according to the terminology of Aho, Hopcroft, and Ullman (1987:163).

approach is that one may easily add two-level morphology (Koskenniemi 1983) as an additional filter.

A radical approach, not pursued here, would be to employ the same type of incremental chart-parsing machinery at the lexical level as we do at the sentence level.

3.4 Dependencies across Sentences

Incremental parsing would be even more beneficial if it were extended to handle dependencies across multiple sentences, for example with respect to noun-phrases. Considering a language-sensitive text editor, the purpose of which would be to keep track of an input text, to detect (and maybe correct) certain linguistic errors, a change in one sentence often requires changes also in the surrounding text as in the following examples:

> The house is full of mould. It has been judged insanitary by the public health committee. They say it has to be torn down.
>
> The salmon jumped. It likes to play.

In the first example, changing the number of "house" forces several grammatical changes in the subsequent sentences, requiring reanalysis. In the second example, changing "it (likes)" to "they (like)" constrains the noun-phrase of the previous sentence to be interpreted as plural, which could be reflected for example by putting the edges of the singular analysis to sleep.

Cross-sentence dependencies require a level of incremental interpretation and a database with non-monotonic reasoning capabilities. For a recent approach in this direction, see Zernik and Brown (1988).

4 Interactive Parsing

This section outlines how the incremental parser is embedded in an interactive parsing system, called LIPS.[15]

Figure 1 shows the main components of the system. The user types a sentence into the editor (a Xerox TEDIT text editor). The words are analysed on-line by the scanner and handed over to the parser proper which keeps the chart consistent with the input sentence. Unknown words are marked as illegal in the edit window. The system displays the chart incrementally, drawing and erasing individual edges in tandem with the parsing process.

[15]Linköping Interactive Parsing System.

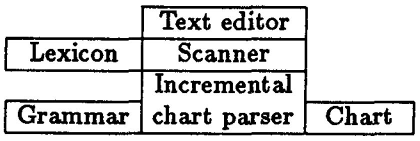

Figure 1. Main components of the LIPS system

It is planned to maintain a dynamic agenda of update tasks (either at the level of update functions or, preferably, at the level of individual edges), removing tasks which are no longer needed because the user has made them obsolete (for example by immediately deleting an inserted text).

In the long run, an interactive parsing system probably has to have some built-in notion of time, for example through time-stamped editing operations and (adjustable) strategies for timing of update operations.

5 Conclusion

This paper has demonstrated how a chart parser by simple means could be augmented to perform incremental parsing, and has suggested how this system in turn could be embedded in an interactive parsing system. Incrementality and interactivity are two independent properties, but, in practice, an incremental system that is not interactive would be pointless, and an interactive system that is not incremental would at least be less efficient than it could be. Although exhaustive recomputation can be fast enough for small problems, incrementality is ultimately needed in order to cope with longer and more complex texts. In addition, incremental parsing brings to the system a certain "naturalness" — analyses are put together piece by piece, and there is a built-in correlation between the amount of processing required for a task and its difficulty.

"Easy things should be easy..." (Alan Kay).

References

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman (1987). *Data Structures and Algorithms.* Addison-Wesley, Reading, Massachusetts.

Bobrow, Robert J. and Bonnie Lynn Webber (1980). Knowledge Representation for Syntactic/Semantic Processing. *Proc. First Annual National Conference on Artificial Intelligence*, Stanford, California: 316–323.

de Kleer, Johan (1986). An Assumption-based TMS. *Artificial Intelligence* 28(2):127–162.

Earley, Jay (1970). An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13(2):94–102.

Earley, Jay and Paul Caizergues (1972). A Method for Incrementally Compiling Languages with Nested Statement Structure. *Communications of the ACM* 15(12):1040–1044.

Ghezzi, Carlo and Dino Mandrioli (1979). Incremental Parsing. *ACM Transactions on Programming Languages and Systems* 1(1):58–70.

Ghezzi, Carlo and Dino Mandrioli (1980). Augmenting Parsers to Support Incrementality. *Journal of the Association for Computing Machinery* 27(3):564–579.

Haddock, Nicholas J. (1987). Incremental Interpretation and Combinatory Categorial Grammar. *Proc. Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy: 661–663.

Hirst, Graeme (1987). *Semantic Interpretation and the Resolution of Ambiguity.* Cambridge University Press, Cambridge, England.

Kay, Martin (1980). Algorithm Schemata and Data Structures in Syntactic Processing. Report CSL-80-12, Xerox PARC, Palo Alto, California. Also in: Sture Allén, ed. (1982), *Text Processing. Proceedings of Nobel Symposium 51.* Almqvist & Wiksell International, Stockholm, Sweden: 327–358.

Koskenniemi, Kimmo (1983). *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production.* Publication No. 11, Department of General Linguistics, University of Helsinki, Helsinki, Finland.

Lindstrom, G. (1970). The Design of Parsers for Incremental Language Processors. *Proc. 2nd ACM Symposium on Theory of Computing*, Northampton, Massachusetts: 81–91.

Mellish, Christopher S. (1985). *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood, Chichester, England.

Pulman, Steven G. (1985). A Parser That Doesn't. *Proc. Second Conference of the European Chapter of the Association for Computational Linguistics*, Geneva, Switzerland: 128–135.

Pulman, Steven G. (1987). The Syntax–Semantics Interface. In: Pete Whitelock, Mary McGee Wood, Harold Somers, Rod Johnson, and Paul Bennett, ed., *Linguistic Theory and Computer Applications.* Academic Press, London, England: 189–224.

Reps, Thomas and Tim Teitelbaum (1987). Language Processing in Program Editors. *Computer* 20(11):29–40.

Shieber, Stuart M. (1986). *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes No. 4. University of Chicago Press, Chicago, Illinois.

Shieber, Stuart M., Hans Uszkoreit, Fernando C. N. Pereira, Jane J. Robinson, and Mabry Tyson (1983). The Formalism and Implementation of PATR-II. In: Barbara Grosz and Mark Stickel, eds., *Research on Interactive Acquisition and Use of Knowledge.* SRI Final Report 1894, SRI International, Menlo Park, California.

Thompson, Henry (1981). Chart Parsing and Rule Schemata in GPSG. Research Paper No. 165, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland. Also in: *Proc. 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California: 167–172.

Thompson, Henry and Graeme Ritchie (1984). Implementing Natural Language Parsers. In: Tim O'Shea and Marc Eisenstadt, *Artificial Intelligence: Tools, Techniques, and Applications.* Harper & Row, New York, New York: 245–300.

Tomita, Masaru (1985). An Efficient Context-Free Parsing Algorithm for Natural Languages. *Proc. Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California: 756–764.

Yonezawa, Akinori and Ichiro Ohsawa (1988). Object-Oriented Parallel Parsing for Context-Free Grammars. *Proc. 12th International Conference on Computational Linguistics*, Budapest, Hungary: 773–778.

Wirén, Mats (1988). A Control-Strategy-Independent Parser for PATR. *Proc. First Scandinavian Conference on Artificial Intelligence*, Tromsø, Norway: 161–172. Also research report LiTH-IDA-R-88-10, Department of Computer and Information Science, Linköping University, Linköping, Sweden.

Zernik, Uri and Allen Brown (1988). Default Reasoning in Natural Language Processing. *Proc. 12th International Conference on Computational Linguistics*, Budapest, Hungary: 801–805.