Exploring Interactive Chart Parsing

Manuel Vilares Ferro Miguel Angel Alonso Pardo

Abstract

This work explores the problem of incremental analysis in the context of chart parsing, probably the most commonly used framework for the analysis of natural language. Incrementality here means that syntax correctness of a text is checked dynamically as the text is edited, changing the internal representation of the analysis rather than generating an entirely new one. This implies that the system then may interact with the user in order to resolve problems that occur. As a consequence, these kinds of techniques can be used to develop highly interactive and reactive natural language processors.

The kernel of the work is an incremental parsing algorithm that analyses arbitrary changes of a text, allowing competing analyses to be developed in parallel.

Key Words: *Chart Parsing, Dynamic Programming, Incremental Parsing, Interactive Processing, Push-Down Automata.*

1 Introduction

The notion of incrementality has been used in two differing senses in the literature on parsing. In the first sense, the purpose of an incremental parser is to construct the analysis of a text bit by bit in a single left-to-right pass, rather than in one go when the text has come to an end. This is typically the case of syntax-directed editors, where the user writes the text in a top-down fashion, guided by the system itself. The other sense for incrementality stresses the necessity of efficiently analyzing arbitrary changes within the set of input. We shall focus our attention on this last meaning that clearly includes the first one.

The reasons to achieve incremental parsing in natural language are well known. At the outbreak, interest in it was due to the necessity of efficiently handling arbitrary changes within current input during text composition in language-sensitive editors. Here, incremental parsing can be used to make the overall parsing process efficient, in a context where several consecutive corrections of the text are usually made. This means that preparing a text requires significantly less effort than developing it from scratch. Another application that can motivate incremental parsing is the growing importance of highly interactive, and real-time systems, where the analysis process must be prompted immediately at the onset of new input. Incrementality is also required in systems allowing incomplete parsing, as is the case of the speech recognition [1, 2], where the input language can only be approximately defined, and individual inputs can vary widely from the norm. Finally, the incremental facility can take an interest in parse systems capable of combining pieces of information from different knowledge sources. This is the case of systems involving multimodal communication, where different parts of an utterance can be expressed in different modalities, say, one part in natural language and another by gesture.

The incremental parser development has been grounded in a chart parsing framework [3] since this is a frequently adopted technique in natural language processing. Chart parsing is not a fixed parsing algorithm, but rather a data and control structure, based on the dynamic programming paradigm [4], which guarantees a polynomial complexity both in time and space, and is highly independent of particular control strategies and grammar formalisms[1] [5]. The kind of structure they produce to represent all parses of the analyzed sentence is an essential characteristic of these algorithms: The *chart*, whose edges representing partial analyses are also called *items*.

The aim of this work is computational. In

M. Vilares is with the Computer Science Department, University of Corunna, Campus de Elviña S/N, 15071 A Coruña, Spain. E. mail: vilares@dc.fi.udc.es.

M. A. Alonso is currently with the Ramón Piñeiro Linguistic Research Center, Estrada Santiago-Noya, Km. 3, A Barcia, 15896 Santiago de Compostela, Spain. E. mail: alonso@dc.fi.udc.es.

This work was partially supported by the Eureka Software Factory project, and by the Autonomous Government of Galicia under project XUGA10501A93.

[1] this characteristic can be used to compare experimentally parsing schemata from different points of view: parser size, parsing speed and size of shared forest.

consequence, we are not guided or motivated by data concerning psycholinguistic plausibility[2]. In this manner, we shall center our attention around context-free grammars. In effect, grammars of this type are well suited to computational use[3], as a backbone to build efficient natural language analyzers [6]. So, it seems feasible that by using them, systems of the future could have natural language components which are both computational efficient and linguistically elegant, in contrast to the heuristic approach of many older systems. In this context, a major goal is to investigate how natural languages parsers can be adapted to interactive applications.

1.1 Previous work

We present now our general framework for both chart and incremental parsing.

1.1.1 Chart parsing

The first chart parsing algorithms, also called *tabular methods*, were developed independently by researchers in compiler construction [7, 8, 9] and in natural language processing [3, 10]. Whichever the case was, these first techniques do not provide a simple manner to extract parse trees since they only associate non-terminal categories to segments of the analyzed sentence. This represents an important drawback since, in practice, parsing algorithms should produce a structure that explicitly relates the instances of non-terminals associated with sentence fragments to their constituents, possibly in several ways in case of ambiguity, sharing some common branches between the distinct ambiguous parses [5]. This is the case of *Earley-like* algorithms, also called *parallel methods*, in which context we choose to work. Essentially, we consider a simple variation of Earley's dynamic programming construction [11], where in order to solve the problems derived from grammatical constraints, the classic construction can be extended to push-down transducers (PDT's), separating the execution strategy from the implementation of the PDT interpreter [5, 6, 12]. In comparison with tabular methods, which have a better asymptotic behavior [13], parallel algorithms seem to be the most appropriate for practical applications since theoretical complexity bounds are rarely obtained.

1.1.2 Incremental parsing

Incremental parsing has not previously been addressed within natural language processing with the exception of Wirén [14], who reports preliminary work on adapting PATR-II[4] [15] to this purpose. This work has a theoretical interest, but the results are not considered practical by the author himself. As an additional restriction, Wirén only considers cycle-free grammars. There are probably two reasons to justify this lack of efficiency: First, the algorithm directly interprets the grammar, rather than first compiling it. Secondly, the algorithm makes use of dependencies between chart edges in order to keep track of affected parts of the analysis, something which requires a lot of assumptions about these connections since the point at which a modification is applied may be located arbitrarily far ahead in the text. In practice, extra time and space is needed for implementation, besides complicating the update of the syntactic structures during the incremental process. In order to avoid this, we look for an incremental parsing method capable of combining the following characteristics: First, the algorithm should may be extended to the family of chart parsers without restrictions, which ensures uniform incremental framework for parsing both the programming languages and the natural ones. The basic motivation for this approach is to benefit from the context-free parsing technology whose development over thirty years has lead to powerful and efficient parsers, some of which have been largely applied in the domain of natural language analysis, in special Earley-like algorithms. Secondly, the incremental algorithm should be closer to reason maintenance, in which an inference is recorded by a dependency between nodes representing the antecedent and consequent formula. That reduces the complexity in relation to classic dependencies between nodes in the chart.

At this point, the only practical reference to our work is that represented by interactive programming environments for general context-free languages. To the best of our knowledge, the problem has not

[2] in spite of this, psycholinguistic data are useful to work in natural language processing. So, such a data provide a complementary perspective, which may generate fruitful questions and serve as an important source of inspiration in order to find a parse which most closely matches the input.

[3] we are not suggesting that there is a context-free grammar for a given natural language. It is probably more appropriate to view the grammar as a convenient control structure for directing the analysis of the input string.

[4] a unification-based linguistic formalism of the tool type, is presently at the heart of much research in computational linguistics.

previously been addressed except by Van den Brand in [16], Rekers in [17] and the first author of this paper in [12], even though the approach is different. In effect, the incremental parsing routines used in [16] and [17] take the non-terminal as a parameter to which the text that is to be parsed should be reduced. Although in most cases the focused node in the original program will cover the alterations, for a few of them the reparsing will have to start in an ancestor node of the focus, which is found by a sequence of trial and error. At this point, it is not possible to prevent the system from doing unnecessary work during the search for this minimal node covering the complete syntactical effect of the modification, for example, when the text to be parsed contains an error. In the case of Vilares, the update of the parse forest is run parallel to the parsing process itself which ensures the earlier detection of parse errors, avoiding unnecessary work. To be more exact, when the parsing process for the introduced modification begins, the system has previously verified if this is viable in relation to the current syntactic context. This work may be easily extended to Earley-like algorithms, which leads us to conjecture that the technique described is at the heart of incremental parsing constructions in dynamic programming.

1.2 A simple road map

In section 2 of this paper, we give an overview of chart parsing by dynamic programming. In section 3, we describe the incremental algorithm, justifying tactical decisions, comparing it with other methods including complexity bounds both in time and space. In section 4, we give an extensive range of comparative tests between standard and incremental parsing. Section 5 is a conclusion about the work presented.

Finally, appendix A reviews from a practical point of view the results previously explained.

2 Standard Parsing

We assume that by using a standard technique we produce a recognizer for the context-free language $\mathcal{L}(\mathcal{G})$, based on a PDT, possibly non-deterministic. Our aim is to parse sentences in the language $\mathcal{L}(\mathcal{G})$ according to its syntax, where the notation is N for the set of non-terminals, Σ for the set of terminal symbols, P for the rules and S for the start symbol. The empty string will be represented by ε.

2.1 The descriptive model

An apparently major difference with other chart parsers is the kind of structure we use to represent the output shared forest, by using context-free grammars. When the sentence has several distinct parses, the set of all possible parse chains is represented in finite shared form by a context-free grammar that generates that possibly infinite set [5].

However, this difference is only apparent. In effect, context-free grammars can be represented by AND-OR graphs which in our case are in fact the shared forest graph. More exactly, OR-nodes are represented by the non-terminal categories, and AND-nodes are represented by the rules of the grammars. There are also leaf-nodes corresponding to the terminal categories. The OR-node corresponding to a non-terminal X has exiting arcs leading to each AND-node n representing a rule that defines X. If there is only one arc, this is represented by placing n immediately under X. The sons of an AND-node are the grammatical categories found on the right-hand-side of the rule, in that order. The convention for orienting the arcs is that they leave a node from below and reach a node from above.

A characteristic of the AND-OR graph representing a grammar is that all nodes have different labels. Conversely, any labeled AND-OR graph so that all the node labels are different may be translated into a context-free grammar so that AND-node labels are rule names, OR-node labels represent non-terminal categories, and leaf-node labels represent terminal categories.

2.2 The operational model

Formally, a PDT is represented by a 8-tuple $\mathcal{T}_{\mathcal{G}} = (\mathcal{Q}, \Sigma, \Delta, \Pi, \delta, q_0, Z_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input word symbols, Δ the set of stack symbols, Π the set of output symbols, q_0 the initial state, Z_0 the initial stack symbol, and $\mathcal{Q}_f$ the set of final states. In relation to δ, it is a finite set of transitions of the form $\tau = \delta(p, X, a) \ni (q, Y, u)$ with $p, q \in \mathcal{Q}$; $a \in \Sigma \cup \{\varepsilon\}$; $X, Y \in \Delta \cup \{\varepsilon\}$, and $u \in \Pi^*$.

To represent the state of a PDT in a moment of the parse process, we define a *configuration* as a 5-tuple $(p, X\alpha, ax, u)$, where p is the current state, $X\alpha$ the stack contents with X on the top, ax the remaining input where the symbol a is the next to be shifted, $x \in \Sigma^*$, and u is the already produced output. The application of a transition $\tau = \delta(p, X, a) \ni (q, Y, v)$ results in a new configuration $(q, Y\alpha, x, uv)$

where the terminal symbol a has been scanned, X has been popped, Y has been pushed, and v has been concatenated to the existing output u. So, for example, if the terminal symbol a is ε in the transition, no input symbol is scanned. If X is ε then no stack symbol is popped from the stack. In a similar manner, if Y is ε then no stack symbol is pushed on the stack.

2.2.1 The parsing algorithm

The algorithm proceeds by building a collection of *items*[5], essentially compact representations of the stack in the transducer in order to guarantee a good level of sharing of the computational process[6]. We associates a set of items S_i^w, habitually called *itemset*, for each word symbol w_i at the position i in the input string of length n, $w_{1..n}$. New items are produced by applying transitions to existing ones, until no new application is possible. In order to favour understanding, we shall denote an item $I \in S_i^w$ by $I_{w_i}^w$ or I_i^w when the context is clear.

Items are also used as non-terminals of the output grammar, for which rules are constructed together with their left-hand-side item. Each time a pop or a scan is applied, we generate a rule. In both cases, the left-hand-side of this rule is the new item describing the resulting configuration. In relation to the right-hand-side, it is composed of the token recognized in the case of a scan and, in the case of a pop, of the items popped from the stack in that action. The start symbol is the last item produced by a successful computation. At this point, items are not only elements of the computation process, but also non-terminals of the output grammar. That allows us to identify items with nodes in the resulting parse forest. We shall denote by $I_{w_i}^{w_i}.forest$ all the rules in the output grammar whose left-hand-side is the item $I_{w_i}^{w_i}$.

It is necessary to ensure that the representation of configurations by items is compatible with the formalism of transitions. To formalize this idea, we consider the concept of *dynamic frame*, establishing the conditions over which correctness and completeness of computations with items are verified in relation to the classic framework, that we call S^T.

2.2.2 The concept of dynamic frame

Given a transducer, we define a *dynamic frame* as a pair $(\Re, \mathrm{Op})$ where $\Re$ is an equivalence relation on the stacks, whose classes are named *items*, and Op is an operator that translates transitions in S^T to the new framework; and verifies the following conditions:

- *Compatibility*: every computation in S^T has its counterpart in the dynamic frame.
- *Completeness*: every final configuration in S^T has its counterpart in the dynamic frame.
- *Correctness*: every final configuration in the dynamic frame has its counterpart in S^T.

Dynamic frames were originally introduced by Villemonte de la Clergerie in [18] to formalize the notion of item in relation to the use of *logical push-down automata*[7].

3 Incremental Parsing

The aim of incremental chart parsing is to characterize the part of the previous analysis that is affected in answer to update operations, and to recover the stable parts of the chart. The new chart structure must show the same level of sharing as in standard parsing. New analysis that result from the update can then be generated by means of ordinary chart parsing.

In the context of the recovery process, we shall define *stable* items between the initial parsing of $w_{1..n}$ and the parsing of the modified input string $x_{1..n+k}$, $k \in [-n, \infty)$, as those items that represent a stable configuration of the transducer that would be reconstructed if we had redone an entire parse of the modified input string up to that point. We shall denote it as $I_i^w \equiv I_{w_i}^x$. In this case, items I_i^w and $I_{w_i}^x$ would represent equivalent trees of their shared forest. Henceforth, we shall also denote $\sqsubset$ the relation of inclusion induced by $\equiv$ in the sets of items.

Since we are interested in analyzing arbitrary changes of a text, we shall begin by investigating how update operations can affect the structure of the chart. So, we can consider four different cases of incremental recovery, as is shown in Fig. 1. These are:

- *Total recovery*. The idea is to detect when the parsing process becomes independent of the modification.

[5] which corresponds with the concept of edge in classic chart parsing terminology.

[6] the main purpose of chart parsing is to avoid any kind of duplicated information.

[7] essentially, automata that store atoms and substitutions on their stack, and use unification to apply transitions. They are due to Lang [19], which obtains an exponential reduction in complexity over the traditional resolution methods.

- *Partial recovery.* In this case, recovery applies to all trees of the shared forest corresponding to a part of the remaining input. To be more exact, we distinguish three cases in relation to this, shown in Fig. 1: First, when recovery is possible for all contexts on a finite interval of itemsets. This is the *grouped recovery.* Secondly, when recovery is possible for all the context, but only for some branches in the shared forest. This is the *isolated recovery.* Finally, recovery can be possible for all the context in some branches, in a finite interval of itemsets. In this case, we use the term *selective recovery.*

3.1 An informal description

Because of cumulativity of chart parsing, analysis of new input cannot invalidate previous analysis. Thus, the kinds of changes that we have to consider are those that require removal of information from the chart. So, the only update operations to be taken into account are the removal of a token and the splitting of the chart[8]. To begin with, we consider a simplified text-editing scenario, with a single modification and where the parse algorithm does not consider lookaheads. So, we assume $x_{1..n+k}$, $k \in [-n, \infty)$ is a modified input string from $w_{1..n}$, where x is of the form: $x_{1..n+k} = w_1 \cdots w_\ell \, u_1 \cdots u_j \, w_{\ell+\hbar+1} \cdots w_n$ with

$$\left\{ \begin{array}{l} u_{1..j} \in \Sigma^* \\ \hbar = \mid u \mid - k \end{array} \right. \quad \text{and} \quad \mid u \mid = \left\{ \begin{array}{l} k, \text{ if } k \geq 0 \\ 0, \text{ otherwise} \end{array} \right.$$

where we call S^w_ℓ, a *point of modification relative to w and x*. That is, we are assuming that our modification is the substitution[9] of $w_{\ell+1..\ell+\hbar}$ by $u_{1..j}$.

By dynamic programming construction, items in $S^w_{1..\ell}$ correspond to the stable part of the parsing process, while items in $S^w_\ell S^u_1 \cdots S^u_j$ correspond to the part of the parsing process which is new. Finally, items in $S^w_{\ell+\hbar+1..n}$ correspond to the part of the parsing process that will eventually be recovered.

Although all cases of incremental recovery have been studied, our experience has shown that the incremental treatment is not interesting from a practical point of view when only a part of an itemset is stable, as is usually the case when the input grammar has a lot of ambiguities generating

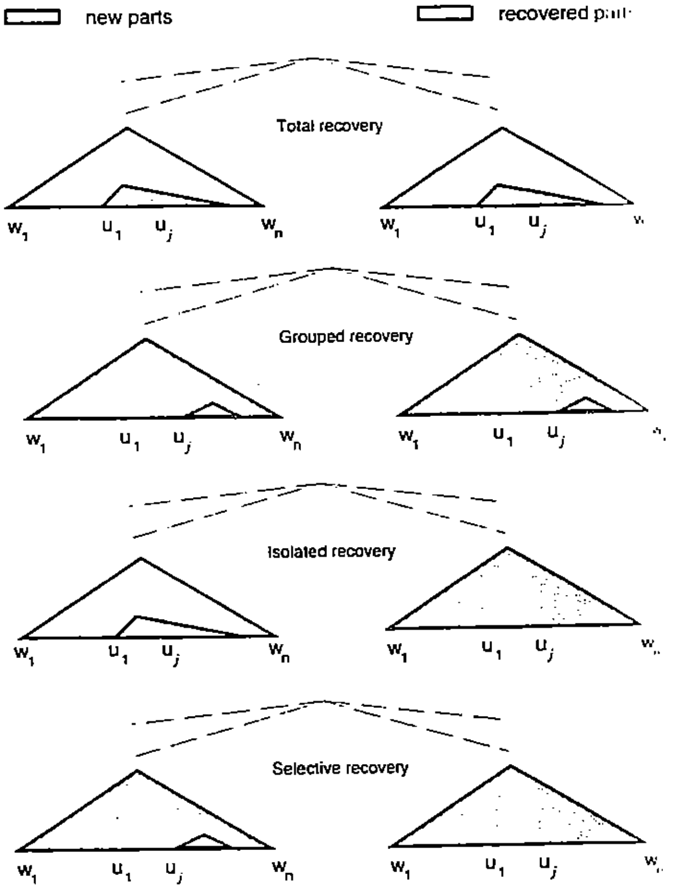

Figure 1: Types of incremental recovery

crossed forests. As a consequence, we shall o consider the case of recovery on the basis of compl itemsets[10], which reduce the problem to total a grouped recovery. Even if this does not guaran that all superfluous computations will be avoided. allows the comparison between chart configurati corresponding to the original and the modified inp string to be notably reduced.

3.2 Total recovery

We now have to establish a condition under wh total recovery is possible from a given input positi i. To do that, it is sufficient to find a condition capa to ensure that all future pop transitions do not dep on the modification, such as is shown in Fig. 2. Thi because pop transitions are the only ones depend on the past of the parsing process.

Therefore a sufficient condition for S^w_i so that allows total recovery from that itemset is that, each item $I \in S^w_{i-1}$ such that:

1. I is the argument in a pop transition from a v

[8] this is needed as a preparation for inserting a new token within the text, which affected the order of the previously analyzed tokens.

[9] for deletion we take $u = \varepsilon$ and $k < 0$, for insertion we assume $\mid u \mid = k$ with $k > 0$ and for substitution we consider $k = 0$.

[10] that is, scan actions are checkpoints in the parsing proc

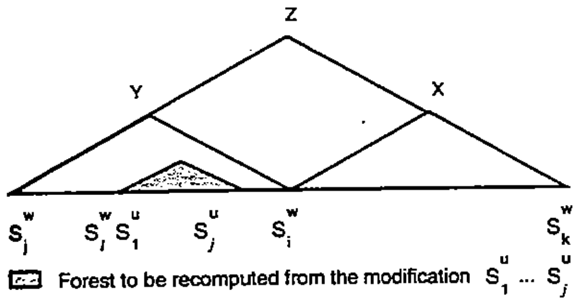
Z
Y
X
S_j^w
S_l^w S_1^u
S_j^u
S_i^w
S_k^w
Forest to be recomputed from the modification S_1^u ... S_j^u

Figure 2: A stable pop reducing Z from Y and X

suffix of $w_{1..i}$.

2. All pop transition taking I as argument, implies a return to an itemset S_t^w, $t < i - 1$.

I is stable, and $t < \ell$ the point of modification relative to w and x. More formally, we call items verifying conditions 1 and 2, $gaps_{w_i}^w$ or $gaps_i^w$ when the context is clear.

Observe that the concept of $gaps_i^w$ does not include those items in the same itemset S_i^w representing trivial nodes[11] from which the only possible actions to perform are empty reductions followed by a shift. Even if this does not guarantee that all superfluous computations will be avoided, it allows the comparison tests between stack configurations corresponding to the original and the modified input string to be notably reduced.

In practice, the sets $gaps_i^w$ are computed by considering the basis of S_i^w, that is, those items that were introduced into S_i^w by a push transition corresponding to a shift action of the transducer. From each one of those items, we put into $gaps_i^w$ the first descendant in the parse forest in S_{i-1}^w representing a non-trivial node, as is shown in Fig. 3.

3.2.1 The case of the recognizer

Once we have computed the gaps, we can give a sufficient condition for total recovery, in the case of the recognizer: Given $x_{1..n+k}$, $k \in [-n, \infty)$ a modified input string from $w_{1..n}$ and S_ℓ^w a point of modification relative to w and x, verifying

$\exists\, l \in [\ell + \hbar + 1, n)$, such that

1. $gaps_l^w \sqsubset gaps_{w_l}^x$ (*resp.* $gaps_l^w \equiv gaps_{w_l}^x$)

2. $\forall I \in gaps_{w_l}^x$, $I.back \leq \ell$

then, from the point of view of the recognizer $S_t^w \sqsubseteq S_{w_t}^x$, $\forall t \in [l, n)$ (resp. $S_t^w \equiv S_{w_t}^x$, $\forall t \in [l, n)$).

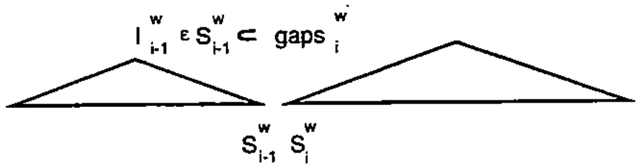

Figure 3: Computing gaps from the parse

Effectively, if the set of possible pop transitions is the same in two parsing processes and the not yet parsed substring is also the same, then the future is identical.

This result can be extended to the case of several simultaneous modifications of the input string, in a single recovery process. To do that, we define the notion of a *totally recovered point of modification relative to w and x*, as a point in the recovery process such that the corresponding modification has been totally recovered. Then the condition becomes: Given $x_{1..n+k}$, $k \in [-n, \infty)$ a modified input string from $w_{1..n}$ and $S_{\ell_{1..m}}^w$ m contiguous points of modification relative to w and x, verifying

$\exists\ i \in [1, m]$, $l \in [\ell_i + \hbar_i + 1, \ell_{i+1})$, such that

1. $gaps_l^w \sqsubset gaps_{w_l}^x$ (*resp.* $gaps_l^w \equiv gaps_{w_l}^x$)

2. $\forall I \in gaps_{w_l}^x$, $I.back \leq \ell_i$

3. $\forall j \in [1, i-1], S_{\ell_j}^w$ has been totally recovered

then, from the point of view of the recognizer $S_t^w \sqsubseteq S_{w_t}^x$, $\forall t \in [l, \ell_{i+1})$ (resp. $S_t^w \equiv S_{w_t}^x$, $\forall t \in [l, \ell_{i+1})$).

3.2.2 The case of the parser

To get incrementality for the parser, we must find in addition the scope of the modifications in the original forest, a simple task, given that the nodes affected by changes in its structure are those common with the new parse forest which have at least a changed descendant in relation to the new development, that is, those common items capable of being accessed in the continuation of the parsing process in the case no incremental treatment was applied. To update one of these nodes it will be sufficient to find its stable descendants in the parse forest which have been effectively recomputed and to replace them with the original corresponding structure in the recovered parse forest.

Taking into account that we only recover complete itemsets, this phenomenon is limited to those items

[11] for which $t = i - 1$ since they have not syntactic descendence.

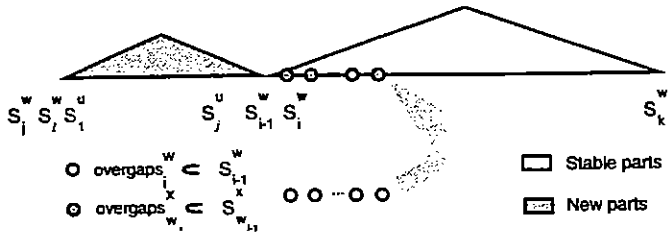

Figure 4: Extending incremental recovery to the parser

representing stable trivial nodes for which their ancestors in the parse forest are not computed in the same itemset S^w_l, where they are included. We call these items $overgaps^w_{w_l}$ or more simply $overgaps^w_l$ when the context is clear[12].

At this point, to extend total recovery to the parser, we must perform for all $I^w \in overgaps^w_l$ the assignments $I^w.forest := I^x.forest$, where I^w and I^x represent in each case the same node in the parse forest, that is $I^w \equiv I^x$. This process is illustrated in Fig. 4.

3.3 Grouped recovery

We start by considering the case of a single modification, and we now have to establish a condition under which grouped recovery is possible from itemset S^w_i to itemset S^w_j. To do that, it is sufficient to find a condition under which pop transitions would not depend on the modification for itemset $S^x_{w_i}$ to itemset $S^x_{w_j}$, as it is shown in Fig. 5. We assume no additional modification in the substring $w_{i..j}$.

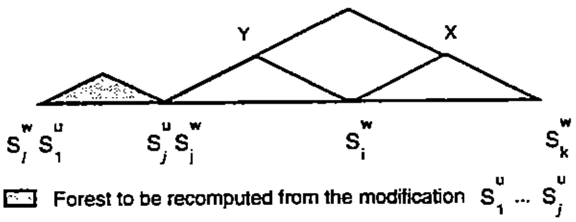
Z
Y
X
S^w_l S^u_1
S^u_j S^w_j
S^w_i
S^w_k
Forest to be recomputed from the modification S^u_1 ... S^u_j

Figure 5: A stable pop reducing Z from Y and X

Therefore a sufficient condition for S^w_i to allow partial recovery from that itemset to some itemset S^w_j is that for each item $I \in S^w_{i-1}$ such that:

1. I is the argument in a pop transition from a valid suffix of $w_{1..i}$.
2. All pop transition taking I as argument, imply a return to an itemset S^w_t, $t < i - 1$.

there does not exist a pop transition in $S^w_{i..j}$ taking I as argument[13].

It is important to remark that the above conditio does not imply the stability of the item I. This because it does not take into account the past o the parsing process represented by the back pointer as was the case in total recovery. So, we say tha these items are *weakly stables*. We shall denote it a $I^w_i \cong I^x_{w_i}$. In this case, items I^w_i and $I^x_{w_i}$ would no necessarily represent equivalent trees of their share forest. We shall denote $\prec$ the relation of inclusio induced by $\cong$ in the sets of items.

Taking into account the definition of $gaps^w_i$, w conclude that there is no pop transition in $S^w_{i..j}$ takin $I \in gaps^w_i$ as argument.

More formally, we can ensure that give $x_{1..n+k}$, $k \in [-n, \infty)$ a modified input string fro $w_{1..n}$ and $S^w_{\ell_{1..m}}$ m contiguous points of modificatio relative to w and x, verifying that

$$\exists\, i \in [1, m],\ l, j \in [\ell_i + \hbar_i + 1, \ell_{i+1}),\ \text{such that}$$

1. $gaps^w_l \preceq gaps^x_{w_l}$ (*resp.* $gaps^w_l \cong gaps^x_{w_l}$)
2. S^w_j is the first itemset applying a pop on *gaps*

then, from the point of view of the recognizer, S^w_t ⪯ $S^x_{w_t}$, $\forall t \in [l, j-1]$ (resp. $S^w_t \equiv S^x_{w_t}$, $\forall t \in [l, j-1]$). To extend this result to the parse forest, it is sufficien to perform for all $I^w \in overgaps^w_l$ the assignment $I^w.forest := I^x.forest$, where $I^w \cong I^x$.

3.4 Comparing with other approaches

Here, we focus our attention in the full increment parsing algorithm suggested by Van den Brand in [16 The author proposes a sequence of four steps to g full incrementality for an isolated modification:

1. The system focuses on the node to be updated
2. Prune this node in order to replace it later, possible, by the new one resulting from the par of the modification.
3. Reparse the substring representing t modification.
4. If this reparse is successful, we recover t resulting tree. If we can translate it into t place of the old pruned node, the increment

[12] the reason for which we have chosen the name *overgaps* is that these items are built from the gaps of the corresponding itemset.

[13] in practice, the realization of this test just necessitates store the minimum value of j when applying pop transitio after S^w_i, from S^w_j.

process is finished. Otherwise, we must focus on an ancestor of the old node to restart the process from the first step.

In comparison with our method, this approach seems to be less general. In effect:

- The concept of total recovery, the most advantageous case of incrementality, cannot be considered.
- If the reparse of the modification does not succeed, the complete program is reparsed.
- If the reparse of the modification succeeds, but the label of the node does not agree with the old one, the system can make some unnecessary work searching for the minimal node which covers the complete syntactical effect of the modification. At worst making a complete reparse of the program. In practice, to reduce the impact of this problem Van den Brand proposes a set of heuristic rules to be applied, but results are not guaranteed.

A similar idea is applied in the case of the SDF [20] environment, such as described by Rekers and Koorn in [17]. In this case, only the extent of the node focused is reparsed, which represents an additional constraint in relation to the algorithm presented by Van den Brand. Here, if the parse of the modification finds an error, the user of the system has to move the focus explicitly. Consequently, incremental parsing consists of a sequence of deriving, pruning and grafting operations interleaved with cursor movements that shift the focus of attention in the forest.

3.5 Complexity bounds

We assume $x_{1..n+k}$, $k \in [-n, \infty)$ is a modified input string from $w_{1..n}$, and $S^w_{\ell_{1..m}}$ m contiguous points of modification relative to w and x. In this context, the application of our incremental test takes a time $\mathcal{O}(n^3)$ and a space $\mathcal{O}(n^2)$, in the worst case. The reasons for this are:

1. The number of items in S^w_l is $\mathcal{O}(l)$, as is proved in [12]. Therefore, the number of items in $gaps^w_l$ is also $\mathcal{O}(l)$, in the worst case. The result is the same for the number of items in $overgaps^w_l$.
2. As a consequence, time complexity for the test $gaps^w_l \sqsubset gaps^x_{w_l}$ (resp. $gaps^w_l \prec gaps^x_{w_l}$) is $\mathcal{O}(l^2)$. On the other hand, we need a space $\mathcal{O}(l)$ to store $gaps^w_l$.
3. From that, we have that in the worst case, the consideration of the incremental mode takes a time $\mathcal{O}(\Sigma^{\ell_i-1}_{l=\ell_i+\hbar_i+1} l^2) \leq \mathcal{O}(\Sigma^n_{l=0} l^2) = \mathcal{O}(n^3)$ and a space $\mathcal{O}(\Sigma^{\ell_i-1}_{l=\ell_i+\hbar_i+1} l) \leq \mathcal{O}(\Sigma^n_{l=0} l) = \mathcal{O}(n^2)$

We can characterize the class of grammars which the algorithm do in time $\mathcal{O}(n)$. For some grammars, called *bounded item grammars*, the number of items in a given itemset cannot grow indefinitely. In this case, the number of items is $\mathcal{O}(k)$ with k constant, whichever it is the considered itemset. As a consequence, the test for incrementality takes a time $\mathcal{O}(\Sigma^{\ell_i-1}_{l=\ell_i+\hbar_i+1} k^2) \leq \mathcal{O}(\Sigma^n_{l=0} k^2) = \mathcal{O}(n)$, and a space $\mathcal{O}(\Sigma^{\ell_i-1}_{l=\ell_i+\hbar_i+1} k) \leq \mathcal{O}(\Sigma^n_{l=0} k) = \mathcal{O}(n)$.

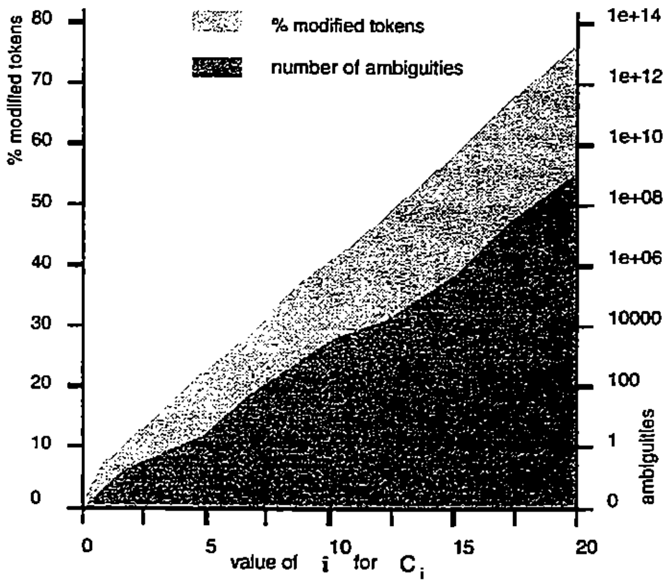

Figure 6: Testing incrementality

4 Experimental Results

Once the incremental parsing algorithm is introduced, our goal is now to prove the practical validity of our approach. To do it, we are interested in a scenario that cannot be qualified as favourable, whichever the point of view considered. At this point, we search for grammars with the following characteristics: The number of rules, and the size of the sentences used in the tests, should be as small as possible, in order to favour understanding. For the same reason, the language generated should be universally known. The language must also provide sentences with a high density of ambiguities, to prove the adaptation of the algorithm to this feature. The testing of sentences must assume an also high density of changes, in order to show the degree of interactivity in the system. Finally, the grammar must include the possibility to generate an also high number of crossed forests, the most unfavourable condition to apply the algorithm previously described.

At this point, pure natural language grammars seem not to be the most appropriate, and we turn our attention to simple context-free grammars. In this way, we shall use the syntax of ambiguous arithmetic expressions to show the efficiency of the incremental parsing process. Formally, our grammar considered is given by the following set of productions:

$$\begin{array}{llll} (0) & S \rightarrow S + S & (1) & S \rightarrow S * S \\ (2) & S \rightarrow (\ S\ ) & (3) & S \rightarrow \mathit{number} \end{array}$$

In effect, in relation to the preceding requirements, this grammar has a small size, and it is easy to write small sentences with a high level of ambiguities and crossed forests. In order to provide tests where the number of changes from the initial input text is important, we analize programs of the form:

$$(b+b)+b\{+(b+b)+b\}^i$$

to obtain

$$b+b\{+b+b\}^i$$

by the substitution of expressions $(b+b)$ by b, where $i \geq 0$ represents the number of times we repeat the corresponding expression. Results are given in Fig. 6, in relation to the number of tokens modified in the original program, and in Fig. 7 in relation to the number of items needed to reparse them. Given that the programs are of the form:

$$b\{+b\}^i$$

they contain a number of ambiguous parses which grows exponentially with i. This number is:

$$C_i = \begin{cases} 1 & \text{if} \quad i = 0,1 \\ \begin{pmatrix} 2i \\ i \end{pmatrix} \frac{1}{\mathrm{i+1}} & \text{if} \quad i > 1 \end{cases}$$

which provides good tests to test incrementality in ambiguous parsing.

5 Summary and Conclusions

Chart parsing has been largely applied in the domain of natural language analysis and speech processing, domains for which incremental treatment has a practical sense. Given that these kinds of parsers combine information in a piecemeal, accumulative fashion, a classic dependency relation on the set of chart edges seems to be unadequate, in practice, for determining the potentially affected parts of the analysis resulting from an update.

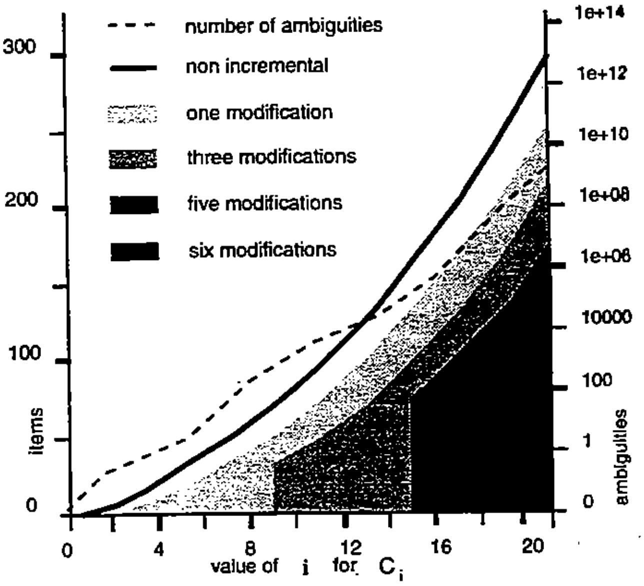

Figure 7: Testing incrementality

In this work, we have shown how chart parsing can be extended to allow an incremental treatment using reason maintenance, in connection with natural language processing. Although efficient incremental parsing may have seemed a difficult problem, we were able to keep the complexity of the algorithm low. The validity of this approach has been proved on examples where the number of ambiguities stays reasonably small, as is the case in practice.

To express incrementality we have only used generic concepts taken from parallel parsing theory. This leads us to conjecture that the technique described is at the heart of incremental constructions in dynamic programming.

References

[1] H. Höge and E. Marschall, "Statistical analysis of left-to-right parser for word-hypothesing", *NATO ASI Series*, pp. 297–303, 1988.

[2] A. Paeseler, "Modification of Earley's algorithm for speech recognition", *NATO ASI Series*, pp. 466–472, 1988.

[3] M. Kay, "Algorithm schemata and data structures in syntactic processing", Tech. Rep., XEROX Palo Alto Research Center, Palo Alto, California, U.S.A., 1980.

[4] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, U.S.A., 1957.

[5] B. Lang, "Deterministic techniques for efficient non-deterministic parsers", Tech. Rep. 72, INRIA, Rocquencourt, France, 1974.

[6] M. Tomita, "An efficient augmented-context-free parsing algorithm", *Computational Linguistics*, , no. 1–2, pp. 31–36, 1987.

[7] D.G. Hays, "Automatic language-data processing", in *Computer Applications in the Behavioral Sciences*, H. Borko ed., Ed., pp. 394–423. Prentice-Hall, 1962.

[8] J. Kasami, "An efficient recognition and syntax analysis algorithm for context-free languages", Tech. Rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, U.S.A., 1965.

[9] D.H. Younger, "Recognition and parsing of context-free languages in time n^3", *Information and Control*, , no. 2, pp. 189–208, 1967.

[10] R. M. Kaplan, "A general syntactic processor", in *Natural Language Processing*, R. Rustin, Ed., pp. 193–241. Algorithmics Press, New York, New York, U.S.A., 1973.

[11] J. Earley, "An efficient context-free parsing algorithm", *Communications of the ACM*, , no. 2, pp. 94–102, 1970.

[12] M. Vilares Ferro, *Efficient Incremental Parsing for Context-Free Languages*, PhD thesis, University of Nice, France, 1992.

[13] L.G. Valiant, "General context-free recognition in less than cubic time", *Journal of Computer and System Sciences*, pp. 308–315, 1975.

[14] M. Wirén, *Studies in Incremental Natural-Language Analysis*, PhD thesis, Linköping University, S-581 83 Linköping, Sweden, 1992, ISBN 91-7870-027-8.

[15] S.M. Shieber, H. Uszkoreit, F.C.N. Pereira, J.J. Robinson, and M. Tyson, "The formalism and implementation of PATR-II", Research on Interactive Acquisition and Use of Knowledge SRI Final Report 1894, SRI International, Menlo Park, California, U.S.A., 1983, Barbara Grosz and Mark Stickel, eds.

[16] M.G.J. van den Brand, *A Generator for Incremental Programming Environments*, PhD thesis, Katholieke Universiteit Nijmegen, Nijmegen, Nederlands, 1992.

[17] J. Rekers, *Parser Generation for Interactive Environments*, PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[18] E. Villemonte de la Clergerie, *Automates à Piles et Programmation Dynamique*, PhD thesis, University of Paris VII, France, 1993.

[19] B. Lang, "Complete evaluation of horn clauses, an automata theoretic approach", Tech. Rep. 913, INRIA, Rocquencourt, France, 1988.

[20] J. Heering, P.R.H. Hendriks, P. Klint, and J. Rekers, "The syntax definition formalism sdf - reference manual", *SIGPLAN Notices*, , no. 11, pp. 43–75, 1989.

A A practical example

To illustrate the following discussion, we shall assume the pico-grammar of English, taken from [6], and given by the productions:

(0) $\Phi \rightarrow S \dashv$ (1) $S \rightarrow SN\ SV$
(2) $S \rightarrow S\ PP$ (3) NP → noun
(4) NP → pronoun (5) NP → determiner noun
(6) NP → NP PP (7) PP → preposition NP
(8) VP → verb NP

whose representation as AND-OR graph is shown in Fig. 8.

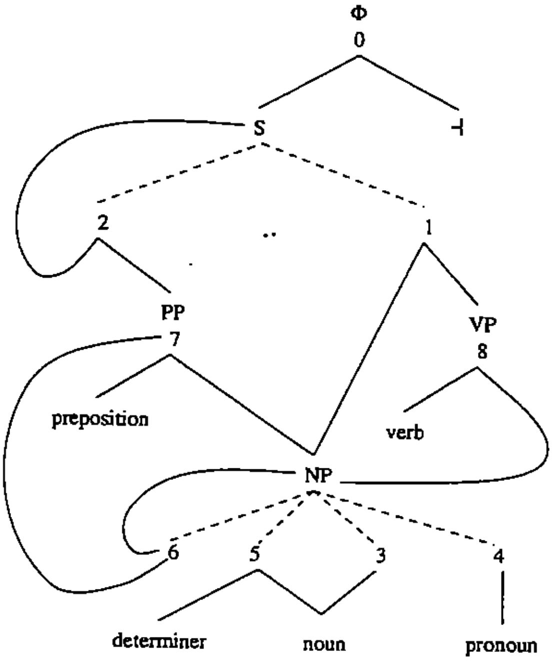

Figure 8: The pico-grammar of English using a graph

A.1 Standard parsing

The first thing to do is to choose a parallel parsing method. We shall consider an extended LALR(1) algorithm, for which the characteristic finite state machine associated to the pico-grammar of English is shown in Fig. 9.

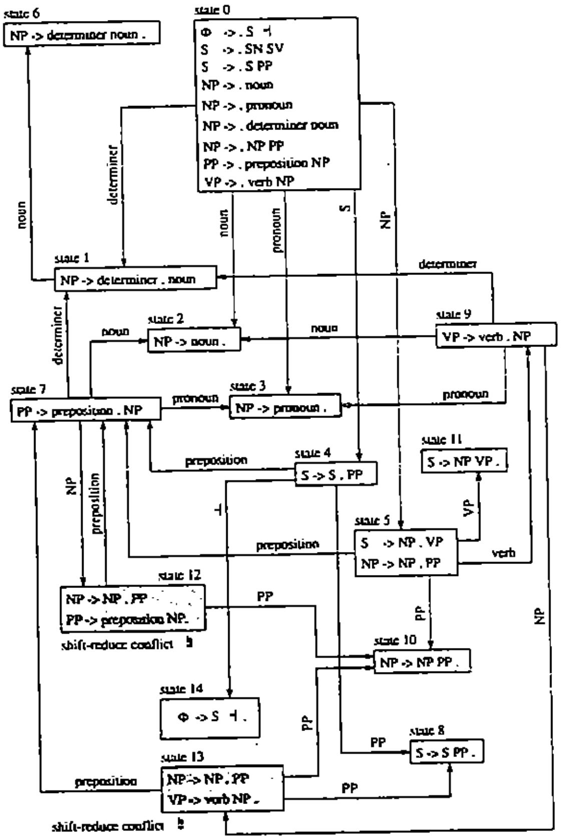

Figure 9: The LR(0) machine for the $\mathcal{G}$ grammar

Now, we must define our dynamic frame. So, we shall consider items of the form $[p, X, S_j^w, S_i^w]$, where p is a state, X is a stack symbol, S_j^w is the *back pointer* to the itemset associated to the input symbol w_i at which we began to look for that configuration of the transducer, and S_i^w is the current itemset. Given a transition $\tau = \delta(p, X, a) \ni (q, Y, u)$ defined on a configuration in the automata, we translate it into a new one capable to directly work on items in the form:

1. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \;\ni\; ([q, \varepsilon, S_i^w, S_i^w], \varepsilon)$
2. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \;\ni\; ([p, Y, S_i^w, S_{i+1}^w], a)$
3. $\tilde{\delta}([p, X, S_j^w, S_i^w], a) \;\ni\; (I_1^w, I_1^w \rightarrow I_2^w)$
4. $\tilde{\delta}([p, \varepsilon, S_j^w, S_i^w], a) \;\ni\; \tilde{\tau}_d$

if an only if

1. $Y = X$
2. $Y = a$
3. $Y \in N$
4. $Y = \varepsilon$, $\forall q \in Q$ such that $\exists\, \delta(q, X, \varepsilon) \ni (p, X, \varepsilon)$

respectively. Where we have considered:

$$\tilde{\tau}_d = \tilde{\delta}_d([q, \varepsilon, S_l^w, S_i^w], a) \ni ([q, \varepsilon, S_l^w, S_j^w], I_3^w \rightarrow I_4^w I_5^w)$$

$$I_1^w = [p, Y, S_i^w, S_i^w], \quad I_2^w = [p, X, S_j^w, S_i^w]$$
$$I_3^w = [q, \varepsilon, S_l^w, S_i^w], \quad I_4^w = [q, X, S_l^w, S_j^w]$$
$$I_5^w = [p, \varepsilon, S_j^w, S_i^w]$$

and

$$\begin{cases} \tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\} \times \Pi^* \\ \tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It} \end{cases}$$

where *It* is the set of all items developed in the parsin process, Π is given by a set of context-free rul directly built from items, and $\tilde{\delta}_d$ is called the set o *dynamic transitions.* Succinctly, we can describe th preceding cases as follows:

1. Corresponds to a goto action from the state p to state q under transition X.
2. Corresponds to a push of terminal a from stat p. The new item belongs to the itemset S_{i+1}^w.
3. Corresponds to a push of non-terminal Y fro state p.
4. Corresponds to a pop action from state p, whe q is an ancestor of state p under transition X in the transducer. In this case, we do not genera a new item, but a *dynamic transition* $\tilde{\tau}_d$ to de with the absence of information about the rest o the stack.

It is important to comment the behavior o the algorithm face to a pop action, the last cas represented. In effect, given that our compa representation of the stack is its top, we must conside a protocol to deal with the absence of informatio about the rest of the stack. The solution relies to th concept of *dynamic transition.* Briefly, it consists i generating a new transition from that implying th pop action. This new transition must be built i such a manner that it is applicable not only to th configuration resulting of the first one, but also o those to be generated and sharing the same syntacti structure.

Following with our example, table 1 shows th itemsets corresponding to the parsing process for tw different input strings:

w = John saw a man with a telescope ⊣
x = John in the room saw a man with a telescope

We only include in those tables, items correspondi to the recognition of a syntactic category in th original grammar.

Parse forests corresponding to input strings w ar x can be respectively seen in Fig. 10 and Fig. 11. order to facilitate understanding, we have included

S_0^w $	S_0^x $
S_1^w $(w_1 = John)$	S_1^x $(x_1 = John)$
$I_1^{w,0} \equiv [0, w_1, S_0^w, S_1^w]$	$I_1^{x,0} \equiv [0, x_1, S_0^x, S_1^x]$
$I_1^{w,1} \equiv [0, NP, S_0^w, S_1^w]$	$I_1^{x,1} \equiv [0, NP, S_0^x, S_1^x]$
	S_2^x $(x_2 = in)$
	$I_2^{y,0} \equiv [5, x_2, S_1^x, S_2^x]$
	S_3^x $(x_3 = the)$
	$I_3^{x,0} \equiv [7, x_3, S_2^x, S_3^x]$
	S_4^x $(x_4 = room)$
	$I_4x, 0 \equiv [1, x_4, S_3^x, S_4^x]$
	$I_4^{x,1} \equiv [7, NP, S_2^x, S_4^x]$
	$I_4^{x,2} \equiv [5, PP, S_1^x, S_4^x]$
	$I_4^{x,3} \equiv [0, NP, S_0^x, S_4^x]$
S_2^w $(w_2 = saw)$	S_5^x $(x_5 = saw)$
$I_2^{w,0} \equiv [5, w_2, S_1^w, S_2^w]$	$I_5^{x,0} \equiv [5, x_5, S_4^x, S_5^x]$
S_3^w $(w_3 = a)$	S_6^x $(x_6 = a)$
$I_3^{w,0} \equiv [9, w_3, S_2^w, S_3^w]$	$I_6^{x,0} \equiv [9, x_6, S_5^x, S_6^x]$
S_4^w $(w_4 = man)$	S_7^x $(x_7 = man)$
$I_4^{w,0} \equiv [1, w_4, S_3^w, S_4^w]$	$I_7^{x,0} \equiv [1, x_7, S_6^x, S_7^x]$
$I_4^{w,1} \equiv [9, NP, S_2^w, S_4^w]$	$I_7^{x,1} \equiv [9, NP, S_5^x, S_7^x]$
$I_4^{w,2} \equiv [5, VP, S_1^w, S_4^w]$	$I_7^{x,2} \equiv [5, VP, S_4^x, S_7^x]$
$I_4^{w,3} \equiv [0, S, S_0^w, S_4^w]$	$I_7^{x,3} \equiv [0, S, S_0^x, S_7^x]$
S_5^w $(w_5 = with)$	S_8^x $(x_8 = with)$
$I_5^{w,0} \equiv [13, w_5, S_4^w, S_5^w]$	$I_8^{x,0} \equiv [13, x_8, S_7^x, S_8^x]$
$I_5^{w,1} \equiv [4, w_5, S_4^w, S_5^w]$	$I_8^{y,1} \equiv [4, x_8, S_7^x, S_8^x]$
S_6^w $(w_6 = a)$	S_9^x $(x_9 = a)$
$I_6^{w,0} \equiv [7, w_6, S_5^w, S_6^w]$	$I_9^{x,0} \equiv [7, x_9, S_8^x, S_9^x]$
S_7^w $(w_7 = telescope)$	S_{10}^x $(x_{10} = telescope)$
$I_7^{w,0} \equiv [1, w_7, S_6^w, S_7^w]$	$I_{10}^{x,0} \equiv [1, x_{10}, S_9^x, S_{10}^x]$
$I_7^{w,1} \equiv [7, NP, S_5^w, S_7^w]$	$I_{10}^{x,0} \equiv [7, NP, S_8^x, S_{10}^x]$
$I_7^{w,2} \equiv [4, PP, S_4^w, S_7^w]$	$I_{10}^{x,2} \equiv [4, PP, S_7^x, S_{10}^x]$
$I_7^{w,3} \equiv [0, S, S_0^w, S_7^w]$	$I_{10}^{x,3} \equiv [0, S, S_0^x, S_{10}^x]$
$I_7^{w,4} \equiv [13, PP, S_4^w, S_7^w]$	$I_{10}^{x,4} \equiv [7, PP, S_7^x, S_{10}^x]$
$I_7^{w,5} \equiv [9, NP, S_2^w, S_7^w]$	$I_{10}^{x,5} \equiv [9, NP, S_5^x, S_{10}^x]$
$I_7^{w,6} \equiv [5, VP, S_1^w, S_7^w]$	$I_{10}^{x,6} \equiv [5, VP, S_4^x, S_{10}^x]$
S_8^w $(w_8 = \dashv)$	S_{11}^x $(x_{11} = \dashv)$
$I_8^{w,0} \equiv [4, w_8, S_7^w, S_8^w]$	$I_{11}^{x,0} \equiv [4x_{11}, S_{10}^x, S_{11}^x]$

Table 1: Itemsets for w and x

each node corresponding to the reduction of a non-terminal in the original input grammar, the number of the considered rule.

A.2 Incremental parsing

First of all, we must translate the concept of stability to our current framework. So, in our case, an item $I_i^w = [p, X, S_j^w, S_i^w]$ is stable if and only if there exists an item $I_{w_i}^x = [p, X, S_j^w, S_{w_i}^x]$, which implies that both items represent the same configuration in the automaton.

We shall present the incremental algorithm from the input strings w and x whose gaps in relation to the pico-grammar of English are shown in table 2.

A.2.1 Total recovery

To illustrate total recovery, we shall first consider the original input string x and the modified one w. That is, we shall here assume that:

- The only point of modification relative to x and w is S_2^x.
- The modification consists in the delete of the *"in the room"* words. Thus, $u_{1..j} = \varepsilon$
- $\hbar =\mid \varepsilon \mid -k = -3$.

The itemsets S_0^w and S_1^w are obtained without changes from S_0^x, S_1^x, but we must recompute the itemset S_2^w. From $gaps_2^x$ and $gaps_{x_5}^w = gaps_2^w$, we obtain that $gaps_5^x \sqsubset gaps_2^w$. We conclude that $S_t^x \sqsubseteq S_{x_t}^w$, $\forall t \in [5, 11]$. From the point of view of the parse, we recover all the AND-OR graph represented in Fig. 11, once the system has reduced *"John in the room"* by a nominal phrase to give $I_4^{x,3}$. The resulting graph is shown in Fig. 10.

$gaps_1^w = \emptyset$	$gaps_1^x = \emptyset$
	$gaps_2^x = \{I_1^{x,1}\}$
	$gaps_3^x = \{I_2^{x,0}\}$
	$gaps_4^x = \{I_3^{x,0}\}$
$gaps_2^w = \{I_1^{w,1}\}$	$gaps_5^x = \{I_4^{x,3}\}$
$gaps_3^w = \{I_2^{w,0}\}$	$gaps_6^x = \{I_5^{x,0}\}$
$gaps_4^w = \{I_3^{w,0}\}$	$gaps_7^x = \{I_6^{x,0}\}$
$gaps_5^w = \{I_4^{w,1}, I_4^{w,3}\}$	$gaps_8^x = \{I_7^{x,1}, I_7^{x,3}\}$
$gaps_6^w = \{I_5^{w,0}, I_5^{w,1}\}$	$gaps_9^x = \{I_8^{x,0}, I_8^{x,1}\}$
$gaps_7^w = \{I_6^{w,0}\}$	$gaps_{10}^x = \{I_9^{x,0}\}$
$gaps_8^w = \{I_7^{w,3}\}$	$gaps_{11}^x = \{I_{10}^{x,3}\}$

Table 2: Gaps for w and x

A.2.2 Grouped recovery

Here, the concept at stake is the weak stability. In our case, an item $I_i^w = [p, X, S_j^w, S_i^w]$ is weakly stable if and only if there exists an item $I_{w_i}^x = [p, X, S_{w_j}^x, S_{w_i}^x]$.

To illustrate grouped recovery, we now consider that x is a modified input string from w whose only point of modification is S_2^x. That is, the modification consists in the insertion of *"in the room"* words. In this case, $\hbar =\mid u \mid -k = 0$. As in the total recovery case, the first two itemsets remains unchanged. In this case $S_0^x \equiv S_0^w$ and $S_1^x \equiv S_0^w$.

Given that $\ell + \hbar + 1 = 2$, we shall firstly compare the gaps from S_2^w and $S_{w_2}^x = S_5^x$, to obtain that $gaps_2^w \not\leq gaps_5^x$, reason for which we shall continue to compare $gaps_3^w$ with $gaps_6^x$. In this case, we shall have $gaps_3^w \cong gaps_6^x$, and therefore we can ensure that $S_3^w \equiv S_6^x$, given that the first itemset applying a pop transition on $gaps_2^w$ is S_4^w. Intuitively, we have recovered all the development corresponding to nodes labeled by $I_2^{w,0}$ and $I_3^{w,0}$ in Fig. 10, which can be seen as nodes labeled by $I_5^{x,0}$ $I_6^{x,0}$in Fig. 11.

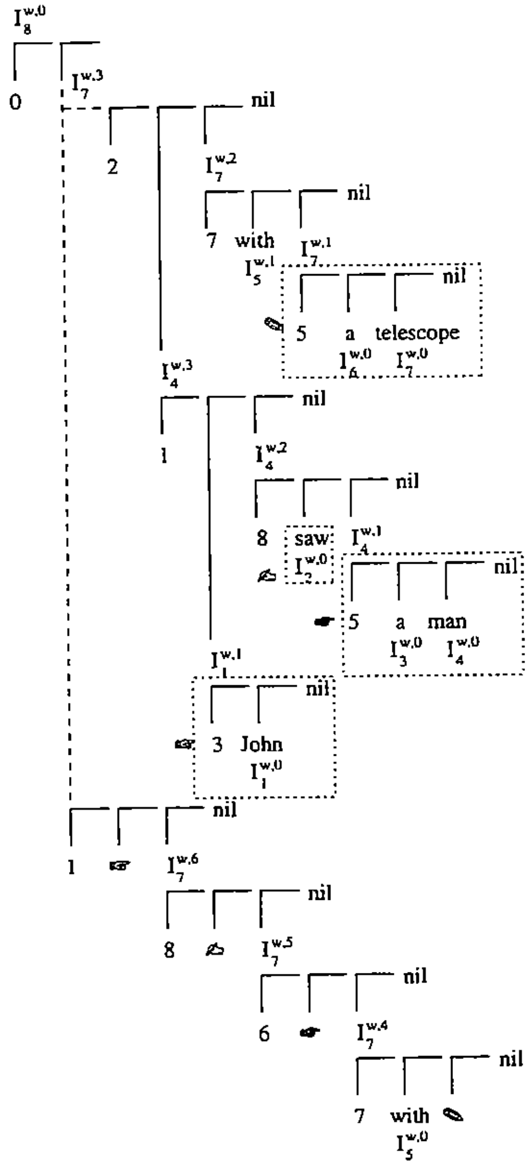

Figure 10: AND-OR graph for the input w

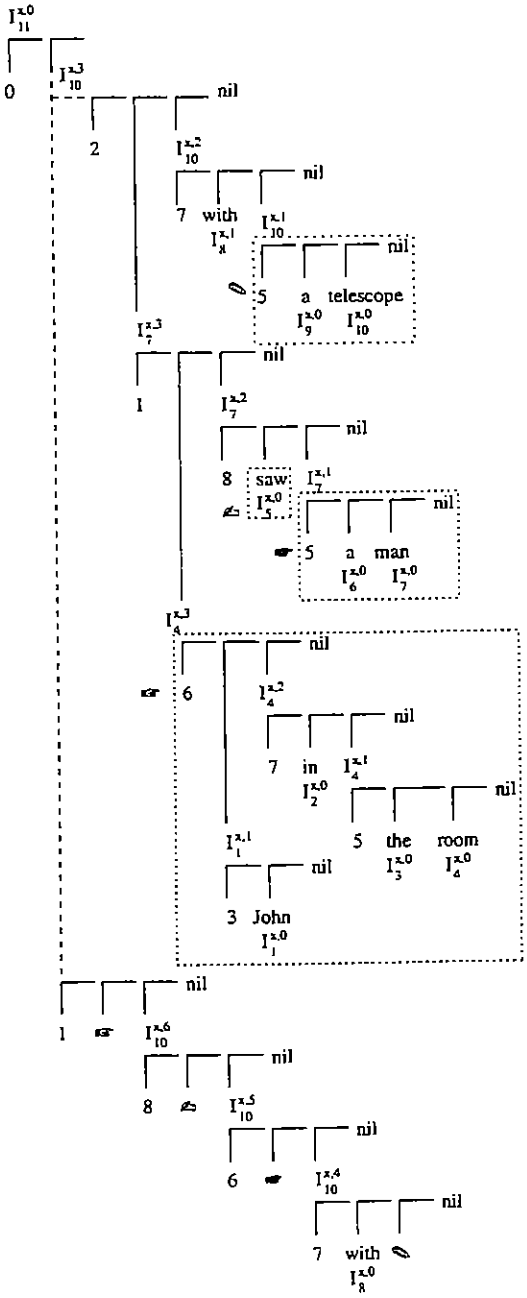

Figure 11: AND-OR graph for the input x