Parsing incomplete sentences revisited *

M. Vilares[1], V.M. Darriba[1], and J. Vilares[2]

[1] Department of Computer Science, University of Vigo
Campus As Lagoas s/n, 32004 Ourense, Spain
{vilares,darriba}@uvigo.es

[2] Department of Computer Science, University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
jvilares@mail2.udc.es

Abstract. We describe a context-free parsing algorithm to deal with incomplete sentences, including unknown parts of unknown length. It produces a finite shared-forest compiling all parses, often infinite in number, that could account for both the error and the missing parts.
In contrast to previous works, we derive profit from a finer dynamic programming construction, leading to an improved computational behavior. We also introduce a deductive construction, which has the advantage of simplifying the task of description.

1 Introduction

An ongoing question in the design of dialogue systems is how to provide the maximal coverage and understanding of the language, finding the interpretations that have maximal thresholds, when the computational process must be prompted immediately at the onset of new input. This is largely due to the fact that the user often does not know the type of questions that the system answers. In this sense, it is often better to have a system that tries to guess a specific interpretation in case of ambiguity rather than ask the user for a clarification. As a consequence, analysis of the utterance should continuously anticipate the interaction with the user, based on the expectations of the system.

To comply with these requests, we need a parser which analyses the input simultaneously as it is entered, even when current data are only partially known. Two factors are at the origin of this behavior in natural language man-machine interfaces, whether text or speech-based. In the case of the former, the input language can only be approximately defined and individual inputs can vary widely from the norm [6] due to ungrammatical spontaneous phenomena. In the case of the latter [7], inputs can only often be considered as a distorted version of any of several possible patterns resulting from an erroneous recognition process.

In this context, our aim is computational. We restrict interaction types to only those necessary for immediate understanding using a predictive model

* Research partially supported by the Spanish Government under projects TIC2000-0370-C02-01 and HP2002-0081, and the Autonomous Government of Galicia under projects PGIDT01PXI10506PN, PGIDIT02PXIB30501PR and PGIDIT02SIN01E.

based on the parsing algorithm for unrestricted context-free grammars (CFG's) proposed by Vilares in [9]. In relation to previous works [8, 3], our proposal provides a formal definition framework and an improved computational behavior.

2 The standard parser

Our aim is to parse a sentence $w_{1\ldots n} = w_1 \ldots w_n$ according to an unrestricted CFG $\mathcal{G} = (N, \Sigma, P, S)$, where the empty string is represented by ε. We generate from $\mathcal{G}$ a *push-down transducer* (PDA) for the language $\mathcal{L}(\mathcal{G})$. In practice, we choose an LALR(1) device generated by ICE [9], although any shift-reduce strategy is adequate. A PDA is a 7-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \delta, q_0, Z_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input symbols, Δ the set of stack symbols, q_0 the initial state, Z_0 the initial stack symbol, $\mathcal{Q}_f$ the set of final states, and δ a finite set of transitions of the form $\delta(p, X, a) \ni (q, Y)$ with $p, q \in \mathcal{Q}$, $a \in \Sigma \cup \{\varepsilon\}$ and $X, Y \in \Delta \cup \{\varepsilon\}$. Let the PDA be in a configuration $(p, X\alpha, ax)$, where p is the current state, $X\alpha$ is the stack contents with X on the top, and ax is the remaining input where the symbol a is the next to be shifted, $x \in \Sigma^*$. The application of $\delta(p, X, a) \ni (q, Y)$ results in a new configuration $(q, Y\alpha, x)$ where a has been scanned, X has been popped, and Y has been pushed.

To get polynomial complexity, we avoid duplicating stack contents when ambiguity arises. We determine the information we need to trace in order to retrieve it [4]. This information is stored in a table $\mathcal{I}$ of *items*, $\mathcal{I} = \{[q, X, i, j],\ q \in \mathcal{Q},\ X \in \{\varepsilon\} \cup \{\nabla_{r,s}\},\ 0 \leq i \leq j\}$; where q is the current state, X is the top of the stack, and the positions i and j indicate the substring $w_{i+1} \ldots w_j$ spanned by the last terminal shifted to the stack or by the last production reduced. The symbol $\nabla_{r,s}$ indicates that the part $A_{r,s+1} \ldots A_{r,n_r}$ of a rule $A_{r,0} \rightarrow A_{r,1} \ldots A_{r,n_r}$ has been recognized.

We describe the parser using *parsing schemata* [5]; a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{I}$ the table of items previously defined, $\mathcal{H} = \{[a, i, i+1],\ a = w_i\}$ an initial set of triples called *hypotheses* that encodes the sentence to be parsed[3], and $\mathcal{D}$ a set of *deduction steps* that allow new items to be derived from already known items. Deduction steps are of the form $\{\eta_1, \ldots, \eta_k \vdash \xi \,/\, conds\}$, meaning that if all antecedents $\eta_i \in \mathcal{I}$ are present and the conditions *conds* are satisfied, then the consequent $\xi \in \mathcal{I}$ should be generated. In the case of ICE, $\mathcal{D} = \mathcal{D}^{\text{Init}} \cup \mathcal{D}^{\text{Shift}} \cup \mathcal{D}^{\text{Sel}} \cup \mathcal{D}^{\text{Red}} \cup \mathcal{D}^{\text{Head}}$, where:

$$\mathcal{D}^{\text{Shift}} = \{[q, X, i, j] \vdash [q', \varepsilon, j, j+1] \Big/ \begin{array}{l} \exists\, [a, j, j+1] \in \mathcal{H} \\ shift_{q'} \in action(q, a) \end{array} \}$$

$$\mathcal{D}^{\text{Sel}} = \{[q, \varepsilon, i, j] \vdash [q, \nabla_{r,n_r}, j, j] \Big/ \begin{array}{l} \exists\, [a, j, j+1] \in \mathcal{H} \\ reduce_r \in action(q, a) \end{array} \}$$

$$\mathcal{D}^{\text{Red}} = \{[q, \nabla_{r,s}, k, j][q, \varepsilon, i, k] \vdash [q', \nabla_{r,s-1}, i, j] \,/ q' \in reveal(q)\}$$

$$\mathcal{D}^{\text{Init}} = \{\vdash [q_0, \varepsilon, 0, 0]\} \qquad \mathcal{D}^{\text{Head}} = \{[q, \nabla_{r,0}, i, j] \vdash [q', \varepsilon, i, j] \,/ q' \in goto(q, A_{r,0})\}$$

---

[3] The empty string, ε, is represented by the empty set of hypothesis, $\emptyset$. An input string $w_{1\ldots n}$, $n \geq 1$ is represented by $\{[w_1, 0, 1],\ [w_2, 1, 2],\ \ldots,\ [w_n, n-1, n]\}$.

with $q_0 \in \mathcal{Q}$ the initial state, and *action* and *goto* entries in the PDA tables [1]. We say that $q' \in$ *reveal(q)* iff $\exists\, Y \in N \cup \Sigma$ such that $\mathit{shift}_q \in \mathit{action}(q', Y)$ or $q \in \mathit{goto}(q', Y)$, that is, when there exists a transition from q' to q in $\mathcal{A}$. This set is equivalent to the dynamic interpretation of non-deterministic PDA's:

- A deduction step *Init* is in charge of starting the parsing process.
- A deduction step *Shift* corresponds to pushing a terminal a onto the top of the stack when the action to be performed is a shift to state st'.
- A step *Sel* corresponds to pushing the ∇_{r,n_r} symbol onto the top of the stack in order to start the reduction of a rule r.
- The reduction of a rule of length $n_r > 0$ is performed by a set of n_r steps *Red*, each of them corresponding to a pop transition replacing the two elements $\nabla_{r,s}\, X_{r,s}$ placed on the top of the stack by the element $\nabla_{r,s-1}$.
- The reduction of a rule r is finished by a step *Head* corresponding to a swap transition that recognizes the top element $\nabla_{r,0}$ as equivalent to the left-hand side $A_{r,0}$ of that rule, and performs the corresponding change of state.

These steps are applied until new items cannot be generated. The splitting of reductions into a set of *Red* steps allows us to share computations corresponding to partial reductions, attaining a worst case time (resp. space) complexity $\mathcal{O}(n^3)$ (resp. $\mathcal{O}(n^2)$) with respect to the length n of the input string [9]. The input string is recognized iff the final item $[q_f, \nabla_{0,0}, 0, n+1]$, $q_f \in \mathcal{Q}_f$, is generated.

When the sentence has several distinct parses, the set of all possible parse chains is represented in finite shared form by a CFG that generates that possibly infinite set, which is equivalent to using an AND-OR graph. In this graph, AND-nodes correspond to the usual parse-tree nodes, while OR-nodes correspond to ambiguities. Sharing of structures is represented by nodes accessed by more than one other node, and may correspond to sharing of a complete subtree, but also to sharing of a part of the descendants of a given node.

3 Parsing incomplete sentences

In order to handle incomplete sentences, we extend the input alphabet. Following Lang in [3], we introduce two new symbols. So, "?" stands for one unknown word symbol, and "$*$" stands for an unknown sequence of input word symbols.

3.1 The parsing algorithm

Once the parser detects that the next input symbol to be shifted is one of these two extra symbols, we apply the set of ded uction steps $\mathcal{D}_{\text{incomplete}}$, which includes the following two sets of deduction steps:

$$\mathcal{D}^{\text{Shift}}_{\text{incomplete}} = \{[q, \varepsilon, i, j] \vdash [q', \varepsilon, j, j+1] \;\Big/\; \begin{array}{l} \exists\, [?, j, j+1] \in \mathcal{H} \\ \mathit{shift}_{q'} \in \mathit{action}(q, a) \\ a \in \Sigma \end{array} \}$$

$$\mathcal{D}^{\text{Loop\_shift}}_{\text{incomplete}} = \{[q, \varepsilon, i, j] \vdash [q', \varepsilon, j, j] \;\Big/\; \begin{array}{l} \exists\, [*, j, j+1] \in \mathcal{H} \\ \mathit{shift}_{q'} \in \mathit{action}(q, X) \\ X \in N \cup \Sigma \end{array} \}$$

while we maintain the rest of the deduction steps in $\mathcal{D}^{\rm Init}$, $\mathcal{D}^{\rm Shift}$, $\mathcal{D}^{\rm Sel}$, $\mathcal{D}^{\rm Red}$, and $\mathcal{D}^{\rm Head}$. From an intuitive point of view, $\mathcal{D}^{\rm Shift}_{\rm incomplete}$ applies any shift transition independently of the current lookahead available, provided that this transition is applicable with respect to the PDA configuration and that the next input symbol is an unknown token. In relation to $\mathcal{D}^{\rm Loop\_shift}_{\rm incomplete}$, it applies any valid shift action on terminals or variables to items corresponding to PDA configurations for which the next input symbol denotes an unknown sequence of tokens. Given that in this latter case new items are created in the same starting itemset, shift transitions may be applied any number of times to the same computation thread, without scanning the input string.

All deduction steps in dealing with incomplete sentences are applied until a parse branch links up to the right-context by using a standard shift action, resuming the standard parse mode. In this process, when we deal with sequences of unknown tokens, we can generate nodes deriving only "$*$" symbols. This over-generation is of no interest in most practical applications and introduces additional computational work, which supposes an extra loss of parse efficiency. So, our goal is to replace these variables with the unknown subsequence terminal, "$*$". We solve this problem by extending the item structure in order to consider an insertion counter to tabulate the number of syntactic and lexical categories used to rebuild the incomplete sentence. When several items representing the same node are generated, only those with minimal number of insertions are saved, eliminating the rest, which are pruned from the output parse shared-forest.

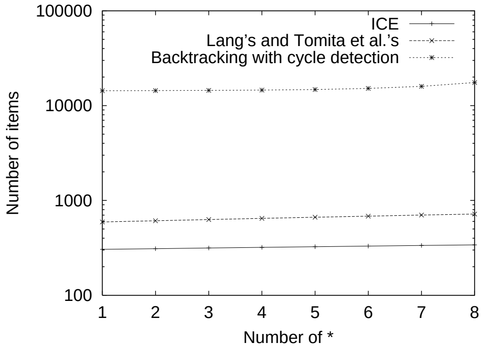

Fig. 1. Number of items for the *noun*'s example

Formally, items extended with counters, e, are of the form $[p, X, i, j, \mathrm{e}]$ and, to deal with them, we should redefine the set of deduction steps $\mathcal{D}_{\rm incomplete}$ as follows:

$$\mathcal{D}^{\text{Shift}}_{\text{incomplete}} = \{[q, \varepsilon, i, j, e] \vdash [q', \varepsilon, j, j+1, e + I(a)] \;\Big/\; \begin{array}{l} \exists\, [?, j, j+1] \in \mathcal{H} \\ \mathit{shift}_{q'} \in \mathit{action}(q, a) \\ a \in \Sigma \end{array} \}$$

$$\mathcal{D}^{\text{Loop\_shift}}_{\text{incomplete}} = \{[q, \varepsilon, i, j, e] \vdash [q', \varepsilon, j, j, e + I(X)] \;\Big/\; \begin{array}{l} \exists\, [*, j, j+1] \in \mathcal{H} \\ \mathit{shift}_{q'} \in \mathit{action}(q, X) \\ X \in N \cup \Sigma \end{array} \}$$

where $I(X)$ is the insertion cost for $X \in N \cup \Sigma$, and we have to adapt the previous deduction steps to deal with counters:

$$\mathcal{D}^{\text{Init}}_{\text{count}} = \{ \vdash [q_0, \varepsilon, 0, 0, 0] \}$$

$$\mathcal{D}^{\text{Shift}}_{\text{count}} = \{[q, X, i, j] \vdash [q', \varepsilon, j, j+1] \;\Big/\; \begin{array}{l} \exists\, [a, j, j+1] \in \mathcal{H} \\ \mathit{shift}_{q'} \in \mathit{action}(q, a) \end{array} \}$$

$$\mathcal{D}^{\text{Sel}}_{\text{count}} = \{[q, \varepsilon, i, j, e] \vdash [q, \nabla_{r,n_r}, j, j, e] \;\Big/\; \begin{array}{l} \exists\, [a, j, j+1] \in \mathcal{H} \\ \mathit{reduce}_r \in \mathit{action}(q, a) \end{array} \}$$

$$\mathcal{D}^{\text{Red}}_{\text{count}} = \{[q, \nabla_{r,s}, k, j, e][q', \varepsilon, i, k, e'] \vdash [q', \nabla_{r,s-1}, i, j, e + e'] \,/ q' \in \mathit{reveal}(q) \}$$

$$\mathcal{D}^{\text{Head}}_{\text{count}} = \{ [q, \nabla_{r,0}, i, j, e] \vdash [q', \varepsilon, i, j, e] \,/ q' \in \mathit{goto}(q, A_{r,0}) \}$$

As for the standard mode, these steps are applied until new items cannot be generated. The resulting complexity bounds are also, in the worst case, $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ for time and space, respectively, with respect to the length n of the input string. The parse is defined by the final item $[q_f, \nabla_{0,0}, 0, n+1, e]$, $q_f \in \mathcal{Q}_f$.

3.2 Previous works

Both, Tomita *et al.* [8] and Lang [3], apply dynamic programming techniques to deal with no determinism in order to reduce space complexity and improve computational efficiency. However, the approach is different in each case:

- From the point of view of the descriptive formalism, Lang's proposal generalizes Tomita *et al.*'s. In effect, in order to solve the problems derived from grammatical constraints, Earley's construction [2] is extended by Lang to PDA's, separating the execution strategy from the implementation of the interpreter. Tomita *et al.*'s work can be interpreted as simply a specification of Lang's for LR(0) PDA's.
- From the point of view of the operational formalism, Lang introduces items as fragments of the possible PDA computations that are independent of the initial content of the stack, except for its two top elements, allowing partial sharing of common fragments in the presence of ambiguities. This relies on the concept of *dynamic frame* for CFG's [9], for which the transitional mechanism is adapted to be applied over these items. Tomita *et al.* use a shared-graph based structure to represent the stack forest, which improves the computational efficiency at the expense of practical space cost.
- Neither Lang nor Tomita *et al.*, avoid over-generation in nodes deriving only "$*$" symbols. In relation with this, only Lang includes a complementary

simplification phase to eliminate these nodes from the output parse shared forest. In addition, these authors do not provide details about how to deal with these nodes when they are generated from more than one parse branch, which is usual in a non-deterministic frame.

Our proposal applies Lang's descriptive formalism to the particular case of an LALR(1) parsing scheme, which makes lookahead computation easier, whilst maintaining the state splitting phenomenon at reasonable levels. This ensures a good sharing of computation and parsing structures, leading to an increase in efficiency. In relation to Tomita *et al.*'s strategy, our deterministic domain is larger and, in consequence, the time complexity for the parser is linear on a larger number of grammars.

With regard to the operational formalism, we work in a dynamic frame S^1, which means that our items only represent the top of the stack. This implies a difference with Lang's proposal, or implicitly Tomita *et al.*'s, which used S^2. From a practical point of view, S^1 translates in a better sharing for both syntactic structures and computations, and improved performance.

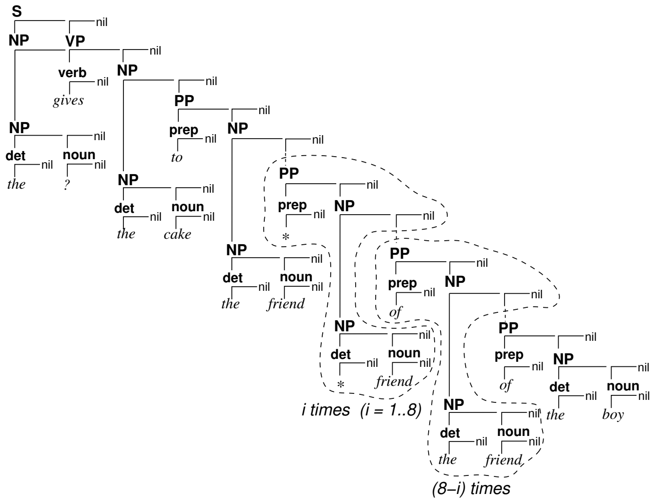

Fig. 2. Shared-forest for the *noun*'s example

Finally, we solve both the consideration of an extra simplification phase and the over-generation on unknown sequences by considering a simple subsumption criteria over items including error counters.

4 Experimental results

We consider the language of pico-English to illustrate our discussion, comparing our proposal on ICE [9], with Lang [3] and Tomita *et al.*'s algorithm [8]. As grammatical formalism, we take the following set of rules:

$$\begin{array}{lll} \text{S} \rightarrow \text{NP VP} & \text{NP} \rightarrow \text{det noun} & \text{VP} \rightarrow \text{verb NP} \\ \text{S} \rightarrow \text{S PP} & \text{NP} \rightarrow \text{NP PP} & \text{PP} \rightarrow \text{prep NP} \end{array}$$

generating the language. Tests have been applied on input strings of two types:

$$\text{det ? verb det noun prep det noun } \{* \text{ noun}\}^i \ \{\text{prep det noun}\}^{8-i} \text{ prep det noun} \quad (1)$$

$$\text{det ? verb det noun prep } \{* \text{ prep}\}^i \ \{\text{det noun prep}\}^{8-i} \text{ det noun} \quad (2)$$

where i represents the number of tokens "$*$", that is, the number of unknown sentences in the corresponding input string. This could correspond, for example, to concrete input strings of the form:

$$\text{The ? gives the cake to the friend } \{* \text{ friend}\}^i \ \{\text{of the friend}\}^{8-i} \text{ of the boy} \quad (3)$$

$$\text{The ? gives the cake to } \{* \text{ of}\}^i \ \{\text{the friend of}\}^{8-i} \text{ the boy} \quad (4)$$

respectively. As our running grammar contains rules "NP → NP PP" and "PP → prep NP", these incomplete sentences have a number of cyclic parses which grows exponentially with i. This number is:

$$C_0 = C_1 = 1 \quad \text{and} \quad C_i = \binom{2i}{i} \frac{1}{i+1}, \text{ if } i > 1$$

In effect, the parser must simulate the analysis of an arbitrary number of tokens and, in consequence, it is no longer limited by the input string. At this point, the parser may apply repeatedly the same reductions over the same grammar rules. So, although the running grammar is not cyclic, the situation generated is close to this kind of framework. More exactly, in dealing with unknown sentences, we can derive a non-terminal from itself without extra scan actions on the input string. This allows us to evaluate our proposal in a strongly ambiguous context with cycles, in spite of the simplicity of the grammar.

The essential experimental results are shown in Fig. 1 (resp. Fig. 3) in relation to running example 1 (resp. example 2), for which the output shared-forests are shown in Fig. 2 (resp. Fig. 4). Since the number of possible tree combinations in these forests is exponential, these figures focus only on particular examples. In all cases our reference for measuring efficiency is the number of items generated by the system during the parsing process, rather than of pure temporal criteria, which are more dependent on the implementation. The shared-forests represented clearly show the existence of a cyclic behavior and ambiguous analyses.

At this point, we are comparing three dynamic frames. The classic one, S^T, is comparable to parse methods based on backtracking and including some kind of mechanism to detect cycles. In this case, no sharing of computations and

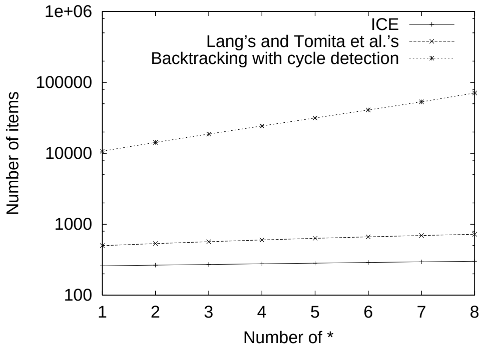

Fig. 3. Number of items for the *prep*'s example

structures is possible, and it is of only theoretical interest. The other two dynamic frames, S^1 and S^2, are of real practical interest. The first one is considered by ICE, while S^2 can be identified in these tests with Lang's and Tomita *et al.*'s results.

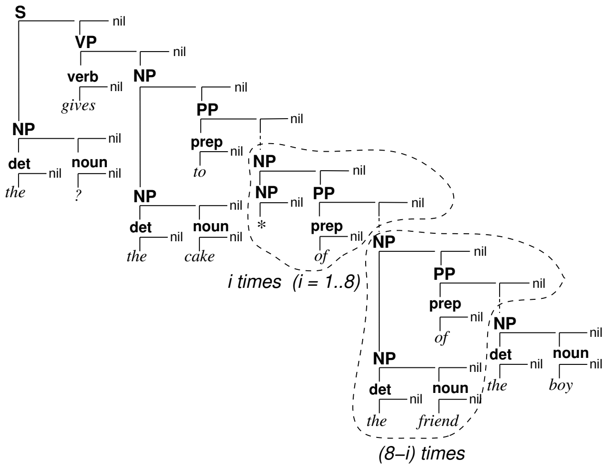

Fig. 4. Shared-forest for the *prep*'s example

In order to allow an objective comparison to be made between all proposals considered, we have made the parsing schema used uniform. So, although Lang's algorithm can be applied to any parse strategy, and Tomita *et al.*'s was originally intended for LR(0) PDA'S, we have adapted both of them to deal with an LALR(1) scheme, as used by ICE. In all cases, these experimental results illustrate the superior performance of our proposal, ICE, in relation to previous strategies. This is due to the following causes:

– We do not need a supplementary simplification phase in order to eliminate nodes deriving only sequences of unknown sequences, "$*$", from the output structure.
– The choice of S^1 instead of S^2 as dynamic frame provides a better sharing efficiency for both structures and computations. As a consequence, the number of items generated is smaller.

In order to illustrate the cost of the previously mentioned simplification phase used by Lang and Tomita *et al.*, Fig. 5 shows the number of items to be eliminated in this process for both examples, noun's and prep's. We include this estimation for S^2, the original dynamic frame for these proposals, and S^1. In this last case, we have previously adapted the original methods of Lang's and Tomita *et al.*.

5 Conclusions

Dialogue systems should provide total understanding of the input. However, in practice, this is not always possible with current technology, even when we restrict ourselves to the treatment of a limited domain of knowledge. In consequence, robustness becomes crucial in order to find a suitable interpretation for the utterance, and we are forced to compute hypotheses to guarantee the interactivity in this kind of frames. So, parsing of incomplete sentences is a fundamental task in a variety of man-machine interfaces, as part of the more general and complex robust parsing activity. This is the case of speech-based systems, where the language often appears to contain noise derived from human causes such as a stutter or a cough; or even mechanical ones due to an imperfect signal recognition.

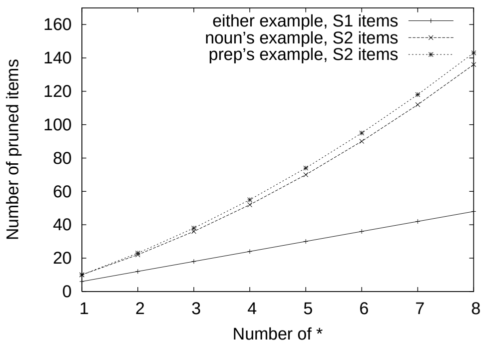

Fig. 5. Items to be pruned in the simplification phase

In this context, our proposal provides an improved treatment of the computation, avoiding extra simplification phases used in previous proposals and profiting from the concept of dynamic frame. In particular, this allows the sharing of computations and structures, reducing the amount of data to be taken into account as well as the work necessary to manipulate them.

References

1. A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, U.S.A., 1986.
2. J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.
3. B. Lang. Parsing incomplete sentences. In D. Vargha (ed.), editor, *COLING'88*, pages 365–371, Budapest, Hungary, 1988. vol. 1.
4. Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx, editor, *Automata, Languages and Programming*, number 14 in Lecture Notes in Computer Science, pages 255–269. Springer, Berlin, DE, 1974.
5. K. Sikkel. *Parsing Schemata.* PhD thesis, Univ. of Twente, The Netherlands, 1993.
6. Robert S. Stainton. The meaning of 'sentences'. *Noûs*, 34(3):441–454, 2000.
7. Andreas Stolcke. Linguistic knowledge and empirical methods in speech recognition. *The AI Magazine*, 18(4):25–31, 1998.
8. M. Tomita and H. Saito. Parsing noisy sentences. In *COLING'88*, pages 561–566, Budapest, Hungary, 1988.
9. M. Vilares. *Efficient Incremental Parsing for Context-Free Languages.* PhD thesis, University of Nice. ISBN 2-7261-0768-0, France, 1992.