EFFICIENT PARSING USING PREFERENCES

Herman CAEYERS
Siemens CSL

Geert Adriaens
Siemens CSL and
University of Leuven

C/O METAL Project
M. Theresiastraat 21
B-3000 Leuven
Belgium

Abstract

In this paper we address the problem of choosing the best analysis of a sentence from the set of possible ones given a parsing mechanism that explores multiple alternatives. The role of such a system in an application environment where response time is critical is explained. We give a detailed description how this is done in the METAL automatic translation system.

0. Introduction

The aim of automatic translation is to give a correct (most 'useful' w.r.t. postediting) and unique translation for each sentence of a text. To be correct all the necessary rules and information applied in these rules must be there. However, correctness is a relative notion. In a restricted context an analysis of a sentence part can be correct, although further analysis might reject it. The current state of natural language processing still restricts the context in which the linguist can work with the consequence of overgeneration of analyses. Nevertheless, some of the results might have a higher likelihood than the others. The mechanism which compares the interpretations is called a preference mechanism. A question to ask when building such a mechanism is at which moment of the analysis the ordering of interpretations will be executed. A frequent approach is to order them gradually during the parse based on specific constraints. In this approach a numeric value, the score, is calculated, independently for each competing interpretation. This can be done by counting the number of constraints satisfied by the interpretation (Wilks 1973) where these constraints might be assigned relative weights by the linguist (Robinson 1982) or calculated automatically (Papegaaij 1986). Opponents to this approach argue that the score is often based on the way interpretations are built and that it is unnatural for a linguist to associate scores to particular structures (Petitpierre et al. 1987).

Their proposal is to keep the preference mechanism in a separate module and to execute it only when all the interpretations are available. The mechanism results then in the comparison of the interpretations using preference statements expressed by the user in the form of rules. Theoretically, we can agree with this viewpoint but we do not consider it as being feasible in an application environment where response time is critical. In the METAL automatic translation system, the grammars we use are fairly large and syntactic analysis can consume lots of CPU-time (rule applications are computationally expensive). Since there is only a limited amount of (real) time available for this process, it is not possible to pursue all alternatives during syntactic analysis exhaustively. Instead, we need to find ways to get the best possible analysis for a sentence given severe resource constraints on the number of rules we are allowed to consider. Therefore we will opt for a mechanism that makes the parsing efficient and which makes use of transparent, controlled decisions of the linguist.
The preference mechanism we will describe is a modification of the one J. Slocum originally wrote for METAL (Bennett and Slocum 1985). For clarity's sake we first briefly give the main characteristics of the METAL parser.

1. The parser

In the METAL Automatic Translation System a chart parser is used. Chart parsers are a popular type of parsers that have interested a lot of researchers over the last two decades (see e.g. Kay 1967, 1980; Kaplan 1973; Winograd 1983, Thompson & Ritchie 1984), and that have found their way into many systems because they belong to the most efficient parsing algorithms for natural languages. Their distinctive characteristic is that they avoid recomputation of syntactic substructures by storing them in a special data structure, the chart. As Winograd states it, they "make it possible to achieve the two principles of efficient parsing: only do what is relevant and don't do anything more than once" (1983,116). A chart consists of nodes and edges. Each node represents a position in the input sentence, whereas the edges represent partial or complete phrases that have been built across the nodes while trying to apply the grammar rules to the input. Partial phrases are called active edges, complete phrases are called inactive edges. The whole parsing process aims at turning active edges (i.e. partially applied rules) into inactive ones, looking for an inactive sentence edge spanning the complete input. In relation to top-down versus bottom-up parsing, a chart parser is neutral with respect to this dichotomy. Depending on whether the inactive edges or the active ones dominate the algorithm, it will be more of a bottom-up parser or more of a top-down one respectively. If the addition of a complete (inactive) edge causes proposing new rules to try out, the parser is bottom-up; if the addition of an active edge (itself a hypothesis about phrases to come) causes the proposing (more hypotheses), the parser works top-down.

The METAL chart parser is a bottom-up parser: whenever a phrase is built that forms the left corner of one or more grammar rules (i.e. the leftmost right-hand-side element), all these rules will be turned into active chart edges looking for phrases to complete them. When all

right- hand-side phrases are available, the rule can be applied, i.e. the parser will attempt to build an inactive (complete) edge spanning the found phrases with as a category the left-hand-side of the rule. This attempt will succeed if all the tests of the rule in question are successful.
The continuous interplay of active and complete edges does not happen via a simple linear structure like a stack or queue, but it is governed by a more complex agenda structure. In order to make the parser efficient the agenda is partitioned into priority classes. Each task (i.e. each creation of an active edge or attempted creation of a complete edge) is evaluated as to its priority and it then gets written into the appropriate priority class. Before we can explain how this is done we have to introduce the notions of static leveling and dynamic preferencing.

2. Static Leveling

In the METAL grammar the rules are partitioned into 10 classes by the numeric LVL ("level") attribute that is assigned to each rule. A high LVL attribute represents the grammar writer's confidence that the rule (and the phrase it builds) will be part of a syntactic analysis for any given sentence once the rule has fired. Mathematically this can be expressed by the following formula which is based on the probability of a rule to appear in an interpretation, given that this rule was successful:

$$LVL = 10 \times \frac{\#\text{ productive calls}}{\#\text{ successful calls}}$$

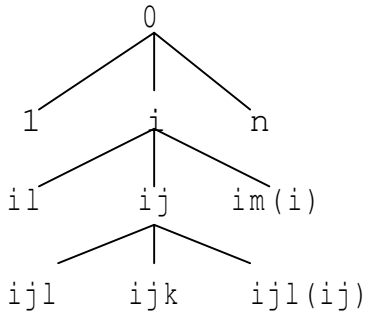
A successful rule is considered productive when the resulting subtree appears at least once in an interpretation of a well-formed sentence. Static leveling enables the linguist to make explicit certain linguistic observations in the grammar. Rules that generalize very frequent constructions will get higher levels than rules that involve the same categories in less frequent constructions (e.g. the rule NO —> ADJ that rewrites an adjective directly to a nominal constituent has a much lower level than the rule NO —> ADJ NO that reflects the (more frequent) adjectival use of an ADJ. A rule with a high level will result in a high priority for any task that is spawned by this rule. That way, rules with a high LVL are tried first; rules with a low LVL are tried only when the high LVL rules have been found to be inapplicable.

3. Dynamic Preferencing

Every phrase that gets built has a preference attribute, called score, the grammar writer's relative confidence as to whether the parse containing this phrase will be the preferred one among a set of several possible alternative parses. The score of a tree related to a parse will be computed from the average of the preference attributes on the

nodes of the tree. For nodes built by a grammar rule this attribute will be the level of that rule. For a lexical node the maximal level 10 is taken as a preference value by default. Following the definition of level the probability of a lexical item to occur in an interpretation is in most cases maximal. Based on the definition of LVL we can say that the meaning of the score of a parse tree is the probability of it to occur in the well-formed interpretation.

For the following general parse tree:



The score is the following:

$$SCORE(0) = \frac{LVL(0) + \sum_{i} LVL(i) + \sum_{i,j} LVL(ij) + \ldots}{1 + n + \sum_{i} m(i) + \sum_{i,j} l(ij) + \ldots}$$

The sum of all levels is divided by the total number of: nodes. From this we can derive the more practical recursive formula:

$$SCORE(0) = \frac{LVL(0) + \sum_{i} SCORE(i) \times z(i)}{1 + \sum_{i} z(i)}$$

with $z(i) = 1 + m(i) + l(ij) + \ldots$, representing the number of nodes in the subtree i.

In the calculation of the score of a parse tree, each node is treated on equal terms. This means that e.g. a node built by morphosyntactic rules (which are also part of the METAL grammar handled by the chart parser) has the same contribution to the final score as a node built by purely syntactic ones. If we compare syntactic structures, then we can remark that big structures (many nodes) have a bigger contribution to the score than small structures (few nodes), which seems to be sensible. The problem now is that some lexical nodes, like idioms, are structures. Therefore we have to give such a node the same weight as the compositional analysis of it. This will be done automatically by

considering the idiom as an analysis tree built of an amount of nodes equal to:

$$3 \times (\ \#\ \text{words in the idiom})$$

This formula gives the estimated maximal amount of "virtual" nodes building the idiom. The factor 3 is the result of a statistical investigation. Each of these virtual nodes has the maximal preference attribute 10.

4. Combining static leveling and dynamic preferencing

Both static leveling and dynamic preferencing are used when the parser has to decide which tasks to execute. The basic idea behind this is the following:

- Prefer tasks involving rules with a high level (because they are more likely to yield a parse)
- Prefer tasks involving phrases with high preference factors (because they are more likely to yield a preferred parse)

An important part of this scheme is how to combine these two heuristics. We have currently implemented a scheme where the static level of a rule serves as the major factor taken into account when selecting tasks from the agenda. Within a set of tasks where all the rules belong to the same level, we use the dynamic preferences of the phrases that are involved in the various tasks as an additional factor in ordering these tasks. Dynamic preferencing is a refinement of the static leveling which is needed for the linguist to keep the ordering of the grammar rules under control.
In the actual implementation this works by partitioning the agenda into 111 priority classes. The priority of a task is a numerical factor that is the combination of the static level of the rule and the preferences of the phrase:

$$\text{Priority class} = 10 \times \text{LVL}(0) + \frac{\sum_i \text{SCORE}(i) \times z(i)}{\sum_i z(i)}$$

with z(i) as defined before.

The parser loops through the agenda vector, starting from the highest priority class. As soon as it hits an agenda slot containing tasks to be executed, it retrieves them from the agenda, zeroes the slot, and executes all the tasks (creating new ones in the process). It then starts again from the highest priority class. The process either stops successfully when one or more complete (S-buiding) parses have been found, or it stops unsuccessfully when its agenda is empty and no S has been built. (In practice, we stop the parser after a preset maximum number of attempted rule applications.) When parsing is successful, the unique parse or the one with the highest preference (if there are several) is handed to the translation component.

5. Influencing dynamic preferencing

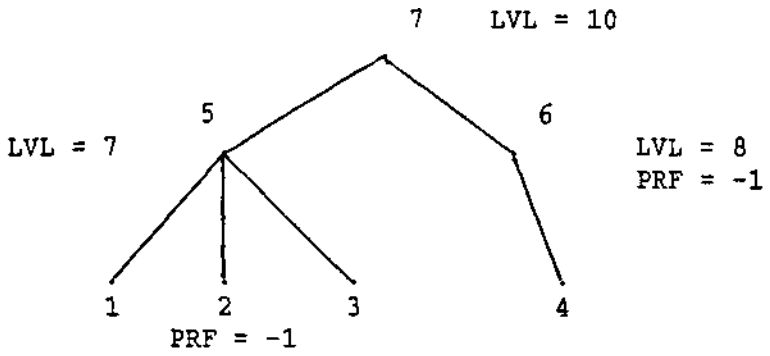
Sometimes it is useful to modify slightly the preference attribute of a rule to influence the score of a parse tree with the intention to promote or suppress a specific interpretation. Changing the level of the rule is not appropriate here, because this will directly affect the major index of the priority class and by consequence also the rule ordering. It is possible that a rule always has to be executed before another one, even if the result of the execution has a lower probability of occurrence than that of this other one. Therefore we introduced an operator in the grammar (called PRF), which can be used within a rule to modify its level when calculating the score. More specifically, the modification of the level especially affects the calculation of the score. Setting a PRF in a rule must have an immediate and predictable effect on the score of the sentence. Therefore the effect must not be influenced either by the place in the tree, or by the size of the tree. The following modification of the score by a PRF fulfils these requirements:

$$SCORE(0) = \frac{LVL(0) + PRF(0) + \sum_{i} SCORE(i) \times z(i)}{1 + \sum_{i} z(i)}$$

where the value of PRF is an integer ranging between (10 - LVL) and (0 - LVL). LVL is the level of the rule where the PRF occurs. Using a preference within a rule means using another level in the calculation of the score which is the average of the levels in the parse tree. There is also the possibility to influence the reading of a lexical item by setting a PRF on its node. If a lexical entry has a PRF the default value 10 of the preference attribute will be decreased by this preference. Also here the lexical preference will mainly affect the score of the parse tree. An example is the way how ah idiom reading of some words can be promoted or suppressed with respect to the compositional one by using a PRF.
In practice, the result of introducing a PRF does not affect the interpretations of a sentence. They will only be ordered differently as a consequence of giving more or less weight to some substructures.

6. Example calculation

The following figure represents a parse tree with on the nodes the LVL of the rule that built the node and the possible PRFs figuring in the rule. For each node the score has been calculated and also the priority class of the tasks dealing with the rule.



NODE	SCORE	PRIORITY CLASS
1	10	--
2	9	--
3	10	--
4	10	--
5	9	80
6	8.5	90
7	9	109

7. Further development

The heuristics presented in the preceding section have proved to work well in the current production release of Metal. The speed of translation is manifestly increased and the quality is improved. However, current work on developing different language pairs has pointed out that there is room for improvement. An important consequence of the organisation of the agenda into priority classes is the fact that the parser does not operate in strictly left-to-right fashion: the tasks with the highest priority that are processed first might apply to phrases anywhere in the tree. This means that the left context of a rule application is not guaranteed to. be completely specified in the chart. Therefore the top-down filtering schemes which are typically proposed for bottom-up chart parsers (see e.g. Kay 1980) cannot be used here. We cannot discard potential rule applications on the grounds that they are incompatible with the left context because this context may not have been exhaustively analysed yet. A possible alternative would be to use the left context (insofar as it is specified) as an additional parameter when computing the priority class of a task. This way an active edge with a given priority that needs a constituent of a certain category at a given vertex in the chart would be used to influence the priority of a task that could produce a phrase of such a category at the appropriate location in the chart.

8. Conclusion

We have described the ordering heuristics within the current version of METAL. These are used to arrive at a desired syntactic analysis given the complexity of the linguistic database and the resource constraints in a production system. With this preferencing mechanism the quality and the speed of the translation have been drastically increased. Special attention was given to the transparency of the heuristic information which has to be introduced by the linguist. Finally, we have also suggested how the current scheme can be further improved.

Acknowledgements

We would like to thank Manfred Immler and Rudi Gebruers for their fruitful suggestions.

References

BENNETT, W. S. and SLOCUM J. (1985)
- The LRC machine Translation System. In Computational linguistics 11.

KAPLAN, R. M. (1973)
- A general Syntactic Processor. Algorithmics, New York.

KAY, M. (1967)
- Experiments with a powerful Parser. In Proceedings of the Second COLING Conference, Grenoble.

KAY, M. (1980)
- Algorithm Schemata and Data Structures in Syntactic Processing. CSL-80-12, XEROX PARC, October.

PAPEGAAIJ, B. ; SADLER, V. and WITKAM, T. (1986)
- Word Expert Semantics: an Interlingual Knowledge Based Approach. Foris, Dordrecht, Holland.

PETITPIERRE, D. et al. (1987)
- A model for preference. In Proceedings of the 3rd Conference of European Chapter of the ACL, Copenhagen.

ROBINSON, J.J. (1982)
- DIAGRAM: A Grammar for Dialogues. Communications of the ACM 25(1).

THOMPSON, H. and G. RITCHIE (1984)
- Implementing Natural Language Parsers. Chapter 9 in T. O'Shea and M. Eisenstadt, Artificial Intelligence. Tools, Techniques, and Applications. Harper&Row, New York.

WILKS, Y. (1973)
- An Artificial Intelligence Approach to Machine Translation. In: Schank, R. and Colby, M. Eds., Computer Models of Thought and Languages. W.H. Freeman and Co, San Francisco, California.

WINOGRAD, T. (1983)
- Language as a Cognitive Process. Volume 1: Syntax. Addison-Wesley, Reading Mass.