

Feature Value Propagation Analysis for Natural Language Grammars

Ettore Merlo¹, Michel Gagnon¹, Giuliano Antoniol², and Dominic Letarte¹

¹Department of Computer Engineering, École Polytechnique de Montréal,
P.O. Box 6079, Downtown Station, Montreal, Quebec, H3C 3A7, Canada
e-mail: {ettore.merlo, michel.gagnon, dominic.letarte}@polymtl.ca

²Department of Engineering, University of Sannio,
Research Centre on Software Technology (RCOST),
Via Traiano - Palazzo ex Poste - I-82100 Benevento - ITALY
e-mail: antoniol@ieee.org

Abstract

Grammars used in parsers for natural language are usually based on feature values that are propagated by the rules. In this paper, we present a flow analysis that has been developed for these grammars and we show that it is useful to identify defects in a grammar.

1 Introduction

The construction of a large-scale grammar for natural language parsing is a task that requires many men-years of work. Not only must we build the rules to cover all the possible syntactic forms, but we must also achieve a fine tuning of these rules to take into account all the subtleties of the language and the idiosyncratic forms. The task becomes even worse if our objective is a grammar tolerant to mistakes. Generally, this results in a big grammar and a large lexicon, both with many intricated features.

When one is faced with a situation that requires a modification of the grammar or the lexicon, it is very difficult to identify the impact of this modification in the parsing process. With the tools available at this moment, typically, we would resort to a pre-analysed corpus of sentences and check that the analysis of these sentences remains the same after the modification. The problem with this approach is that it is time-consuming and does not point specifically at the problem in the grammar design. For example, it would be useful to know what are the other grammar rules or lexical entries that could potentially be affected by some change in a grammar rule.

To assist this kind of impact measures, we propose to adapt a flow analysis that is well known in software engi-

neering. We will argue that using this simple method, we can easily identify problems in the design of a grammar.

In the next section, we present briefly the grammatical formalism that has been used to test our method. The flow analysis itself is formally described in section 3. Finally, a small experiment on a grammar for Portuguese is presented and used to show the advantages of our method.

2 Unification grammar

Unification grammar is a constraint-based formalism, where every symbol in the grammar rules is paired with a feature structure. Based on an unification process originally proposed by Kay [5] which is an extension of the Prolog unification, it is now used in the majority of mainstream grammar formalisms, such as Head-Driven Phrase Structure Grammar [9] and Tree-adjoining Grammar [10]. It is also at the base of the general purpose tools that are available for designing grammars, such as ALE [3, 2].

A feature structure is essentially a list of feature-value pairs. The structure is recursive in the sense that a feature value may be another feature structure, but here we will simplify by assuming that the value of a feature may be either an atom or a variable. This simplification turns easier the understanding of the propagation mechanism described in the next section, without affecting its theoretical foundations.

For example, the following structure says that feature a has the value 1, and that the values for features b and c are unknown but must be the same, since they share the same variable:

$$(F_1) \quad \begin{bmatrix} a & 1 \\ b & X \\ c & X \end{bmatrix}$$

The mechanism used to obtain the derivation tree is the unification. The algorithm is well known (see for example [4] for a description of the algorithm) and an efficient implementation is proposed in [6]. Intuitively, the unification algorithm tries to find a value for all the variables in such a way that the feature structures become identical. In the derivation process, a rule may be activated when its head feature structure may be unified with some feature structure in the body of another rule. The terminal nodes in the derivation are feature structures associated to lexical items. For example, the grammar illustrated in Figure 1 may be used to produce the derivation of Figure 2. Note that variables and atoms are expressed by capital symbols and uncapitalized tokens, respectively. Note also that the value of a feature may be propagated into another feature structure, in the body or the head of the grammar, by simply reusing the same variable.

$$\begin{aligned}
S &\rightarrow NP \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right], VP \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right] \\
NP \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right] &\rightarrow DET \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right], N \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right] \\
VP \left[\begin{array}{c} \text{NUM} \\ \text{X} \end{array} \right] &\rightarrow V \left[\begin{array}{cc} \text{NUM} & \text{X} \\ \text{VAL} & \text{intr} \end{array} \right] \\
DET \left[\begin{array}{c} \text{NUM} \\ \text{sing} \end{array} \right] &\rightarrow \text{the} \\
N \left[\begin{array}{c} \text{NUM} \\ \text{sing} \end{array} \right] &\rightarrow \text{baby} \\
V \left[\begin{array}{cc} \text{NUM} & \text{sing} \\ \text{VAL} & \text{intr} \end{array} \right] &\rightarrow \text{slept}
\end{aligned}$$

Figure 1. Example of grammar

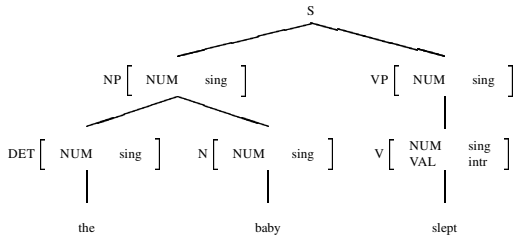


Figure 2. Example of derivation

3 Flow analysis

The feature value propagation can be fully described by giving the lattice of the problem, the partial order the lattice is defined on, the direction of analysis - forward or backward - and the flow equations. A general description of flow

analysis can be found in [1], while an approach for constant propagation analysis has been presented in [7].

A grammar G is defined as follows:

$$\begin{aligned}
G &= (N, T, RULES, S, FEAT) \\
propagSymbol &: RULES \rightarrow N \\
propagSet &: RULES \rightarrow \\
&\rightarrow FEAT \times \{VARS \cup ATOMS\} \\
unifySets &: RULES \rightarrow \\
&\rightarrow (FEAT \times \{VARS \cup ATOMS\})^n, (n \in \mathcal{N}^+) \\
vars &: RULES \rightarrow VARS
\end{aligned} \tag{1}$$

where N is the set of non-terminals, T is the set of tokens or lexical items, $RULES$ is the set of production rules used in the derivation process, S is the start symbol, and $FEAT$ is the set of features propagated by the grammar rules.

Some functions are defined on rules. $propagSymbol$ returns the non-terminal on the left hand side of a production rule, $propagSet$ returns the set of features and their values propagated by a rule, $unifySets$ returns the n-tuple of sets of features and their values or variables used by the right hand side of a production rule for unification purposes, and $vars$ returns the name space of variables in a rule.

The feature propagation flow analysis problem can be defined as follows: at any rule r in the grammar and for any feature a , determine the set V of values a may have.

A solution lattice for the flow analysis problem can be build by considering the partial order existing between sets of feature values. \perp represents the lattice node for which all features do not have any propagated value. \top represents the lattice node in which all the features in the grammar can have all the possible values in their domains. A generic node S_i in the lattice represents a particular configuration of information about the values of all the features in the grammar.

Suppose there are n distinct features named a_1 to a_n in a grammar G . Formally, a node S_i is denoted by the flow information associated to it as follows:

$$S_i = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$$

where $V_{i,k} \subseteq \mathcal{P}(ATOMS)$ is the set of values associated with feature a_k .

The top and bottom of the lattice are respectively:

$$\begin{aligned}
\top &= (V_{\top}, \dots, V_{\top}, \dots, V_{\top}) \\
\perp &= (V_{\perp}, \dots, V_{\perp}, \dots, V_{\perp})
\end{aligned} \tag{2}$$

where V_{\top} is a special symbol indicating that feature a_k has an associated set of values which is equal to the total set of atoms and V_{\perp} is a special symbol indicating that feature a_k has an empty set of associated values.

For any node S_i in the lattice the partial order is defined as follows:

$$\begin{aligned} \perp &\preceq S_i \\ S_i &\preceq \top \end{aligned} \quad (3)$$

For any two nodes S_i and S_j in the lattice which are neither \top nor \perp the partial order between nodes is defined on the set inclusion between the sets of values corresponding to all the features in the grammar:

$$S_i \preceq S_j \iff V_{i,k} \subseteq V_{j,k}, 1 \leq k \leq n \quad (4)$$

Let's define $DG = (V_{DG}, E_{DG})$ to be the grammar derivation graph such that:

$$\begin{aligned} (V_{DG} = N \cup T) \\ (v_1, v_2 \in V_{DG}), (v_1, v_2) \in E_{DG} \iff \\ (\exists r_1, r_2 \in RULES \mid \\ (v_1 \in LHS(r_1)) \wedge (v_2 = RHS(r_2))) \end{aligned} \quad (5)$$

Flow analysis can be computed on the derivation graph. The direction of analysis is forward since we compute the current set of values based on previous values in the grammar derivation graph.

Feature propagation equations for any given grammar rule r are described in terms of the flow information coming in and out rule r . Sets $IN(r)$ and $OUT(r)$ denote such information and correspond also to nodes in the flow problem lattice.

Initially the flow analysis starts with with empty sets for all features, that is $\forall r \in RULES, IN(r) = OUT(r) = (V_{\perp}, \dots, V_{\perp}, \dots, V_{\perp}) = (\emptyset, \dots, \emptyset, \dots, \emptyset)$.

Let the flow information coming into rule r be $IN(r) = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$ and the flow information coming out from r be $OUT(r) = (V_{j,1}, \dots, V_{j,k}, \dots, V_{j,n})$.

Elements of $OUT(r)$ can be computed as follows:

$$V_{j,k} = \begin{cases} V_{\perp} & \text{if } \nexists (a_k, v) \in propagSet(r) \\ \{v\} & \text{if } (a_k, v) \in propagSet(r) \wedge \\ & (v \in ATOMS) \\ \bigcup V_h & \text{if } (a_k, x) \in propagSet(r) \wedge \\ & (x \in VARS(r)) \\ & V_h \in IN(r) \mid \\ & (\exists unifySet_i \in unifySets(r) \mid \\ & (a_h, x) \in unifySet_i) \end{cases} \quad (6)$$

When several arcs of the grammar derivation graph merge in a rule, it is necessary to merge the flow information coming out from r_i and r_j , for example, to obtain the flow information coming into r_m . Let us assume that $OUT(r_i) = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$ and $OUT(r_j) = (V_{j,1}, \dots, V_{j,k}, \dots, V_{j,n})$.

The merged information is:

$$\begin{aligned} IN(r_m) = OUT(r_i) \bowtie OUT(r_j) = \\ = (V_{m,1}, \dots, V_{m,k}, \dots, V_{m,n}) \end{aligned} \quad (7)$$

where \bowtie is the merge operator and

$$\forall k, (1 \leq k \leq n), V_{m,k} = V_{i,k} \cup V_{j,k} \quad (8)$$

Since the presented flow equations preserve the partial order defined by equations 3 and 4, fix-point solution is guaranteed to converge.

4 Experimentation and results

The grammar used in our experimentation is adapted from a Portuguese grammar that has been built for another project [8], where the objective was to test the sensibility of some parsers for phrase structure grammar. The grammar, which contains about 84 rules and uses a lexicon of about 7250 words, recognizes basic sentences. An important characteristic of this grammar is that it has been designed with the objective of making it insensible to common mistakes that could appear in texts written by Brazilians.

The flow analysis has been implemented in Perl on an AMD Athlon XP1700+ processor with 1477 MHz speed. The grammar derivation graph is obtained in 0.9 sec. and contains 100 nodes (84 for the rules and 16 for the terminal symbols). The grammar uses 24 features, 308 variables and 81 possible values for the features.

Table 1 gives a summary of the results. It gives the number of features in the grammar whose set of values has one of the following cardinality: maximal (all values are possible), any cardinality greater than one and less than maximal, singleton, and empty set.

Theses results show that in almost all cases, the method is not very informative about the possible values of the features. In 76% of the cases the domain value is unconstrained. This is not a surprise, considering the very conservative choice of union for the merging operator, which does not take into account the strong constraining effect of unification on the possible values. Even so, four singletons have been identified, and each one points to an actual problem in the grammar or a peculiar characteristic that is worth mentioning. In two cases, it happens that the value propagated to some feature in a rule is always the same, making this propagation useless. In this case, we can either remove the feature in every rule that propagates it, and replace the variable by the propagated value in the rule where the singleton has been detected. It may be also the case that some rule is missing that would propagated another value. In both cases, some decision must be made to fix the grammar, or at least document the idiosyncrasy.

In the other two cases of singleton, the feature has a unique possible value because all the entries in the lexicon instantiate this feature with the same value. The lexicon could contain other entries that would give another value to this feature, but at this moment it does not have such an entry.

Cardinality	Feature	%
Maximal	235	76
$1 < \text{card} < \text{Maximal}$	69	23
Singleton	4	1
Empty set	none	0

Table 1. Summary of results

5 Conclusion and future work

In this paper, we proposed a simple flow analysis that can be used to identify potential problems in the design of a grammar for natural language processing. As far as we know, this kind of technique, widely known in software engineering, has not been used in computational linguistic researches. We showed that even with a very conservative approach regarding the propagation, we can identify real problems in a grammar. Every instance of singleton or empty set points to a potential error in the grammar design.

We conclude that this approach is very promising, and should give more convincing results when applied to a very large grammar. Also, a refinement of the formalism, to make it reflect more precisely the effect of unification in the derivation process, should turn the analysis more expressive in terms of problems identified in the grammar design.

References

- [1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers—Principles, Techniques, and Tools*. Addison-Wesley Publishing Co., 1986.
- [2] B. Carpenter and G. Penn. Compiling types attribute-value logic grammars. In *Recent advances in Parsing Technology*. Kluwer, 1996.
- [3] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine - User's guide*. Philosophy Department, Carnegie-Mellon University, 1994.
- [4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.
- [5] Martin Kay. Functional grammar. In *Proceedings of the Fifth Meeting of the Berkeley Linguistics Society*, pages 142–158, Berkeley, 1979.
- [6] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1992.
- [7] E. Merlo, J.F. Girard, L. Hendren, and R. De Mori. Multi-valued constant propagation analysis for user interface reengineering. *International Journal of Software Engineering and Knowledge Engineering*, 5(1), 1995.
- [8] Mariza Miola. Construção de gramática to português para um estudo comparativo da robustez de alguns algoritmos de análise grammatical. Technical report, Master Dissertation, Universidade Federal do Paraná, 2002.
- [9] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994.
- [10] K. Vijay-Shanker and A. Joshi. Feature structure based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 714–719, 1988.