AWARE - DAG-TRANSFORMATIONS FOR SEMANTIC ANALYSIS

Aarno Lehtola and Timo Honkela
KIELIKONE-project, SITRA Foundation
P.O.Box 329, SF-00121 Helsinki
Finland
tel. intl + 358 0 641 877

AWARE is a knowledge representation language for specifying NLU inference rules. AWARE-system takes as its input the parse trees of NL utterances and further refines them by using DAG-transformations (Directed Acyclic Graph) and recursive descent translation techniques. AWARE has been used for semantic analysis in our Finnish language database interface. The input dependency tree is transformed first into a predication DAG and then reduced into a conceptual database query.

Keywords: semantics, graph grammars, inference tools

1 INTRODUCTION

In 1982 SITRA Foundation launched a major project (KIELIKONE) for the study of general computational models for the interpretation of written Finnish. The target application is a Finnish understanding portable database interface.

Currently our hierarchical model of language interpretation consists of six processes: word analysis, lexicalization and disambiguation, sentence parsing, logico-semantic analysis, inference and query adaptation (Jäppinen & al. 1988). All intermediate structures until the so called predication DAG represent more and more refined analysis results (Figure 1). The predication DAG is semantically the richest representation in the model. The following semantic processes simplify and modify the representations towards database queries.

In our model there is a clean separation between linguistic knowledge and processing mechanisms. The extensive use of specialized knowledge description languages characterizes the different components (Lehtola & al. 1987a). There is also a hierarchy of representations. When we work in the morphological stratum, the associated knowledge language deals with sets of features. In the syntactic stratum trees with feature sets in nodes are the dominating representation. In this paper we outline the computational methods used in our logico-semantic stratum which deals with directed acyclic graphs.

AWARE - DAG-transformations for Semantic Analysis
Aarno Lehtola, Timo Honkela
Proceedings of NODALIDA 1987, pages 58-68

DAG-transformations are practical for inferring NL meaning. For instance, they may be used to spesify which NL expressions are near by meaning. Graph rewrite rules may be used to map their syntactic representations into each other. Such rewrite rules would form a meaning-preserving rulebase for a canonizing process. Graph rewrite rules have proved to be convenient also for solving ellipses and anaphoras.

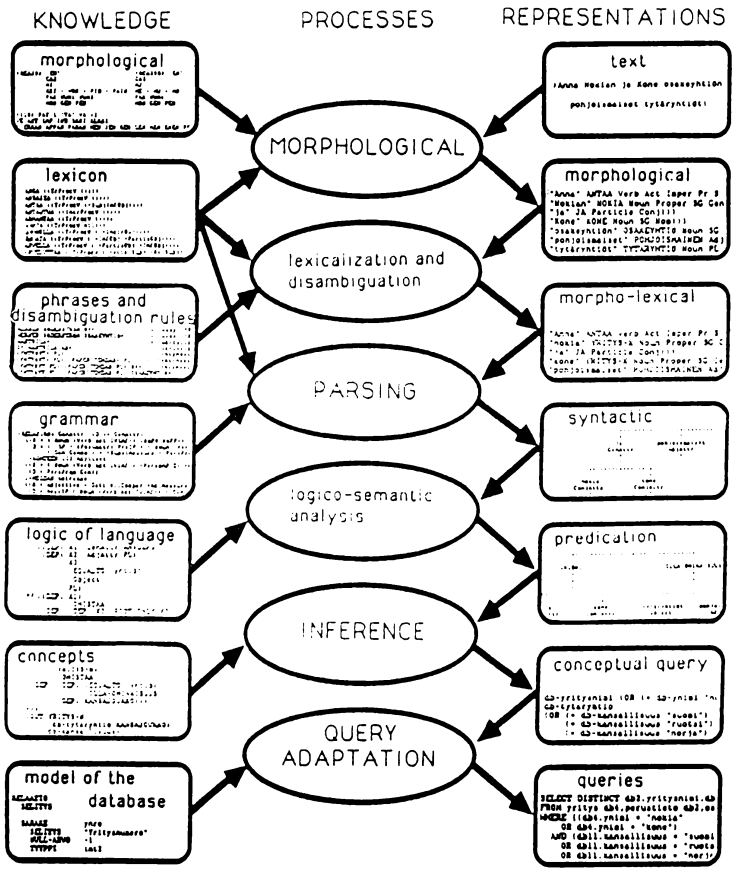


Figure 1. A stratified model of NL interpretation.

In this paper we present the AWARE DAG-transformation system for modelling logico-semantic analysis. First we outline the architecture of AWARE. Then we demonstrate the power of AWARE-rules by examples which deal with canonization of sentence structures, recognition and marking of semantic predications and solution of certain ellipses. Furthermore we discuss how the resulting predication DAGs can be translated into linear expressions, in our case into queries in an universal relation query language. Next we wiev the knowledge acquisition and rule-base maintenance tools. In the end we evaluate the performance of AWARE.

2 THE ARCHITECTURE OF AWARE-SYSTEM

The AWARE-system consists of rule-base maintenance tools and a run-time system (Figure 2). The rule-base is divided into rule packets, which contain rules of

equal priority. Momentarily one or more packets are active. The activation order of packets is specified by a special control language. Each packet has a name and a type. Possible types are 'bottom-up-recursive-scan', 'top-down-recursive-scan', 'wait' and 'transfer'. The type label defines the way the search is carried out.

In AWARE-system the DAGs are usually formed from trees by introducing extra connections. They have one node as the ancestor of all the other nodes. This node we call the root node although in general graph terminology that term is not used. Those nodes which have no descendants we call leaves. The edges are directed out from root nodes towards leaves. The label 'bottom-up-recursive-scan' makes the system to start the search for possible transformations from the leaves and to proceed towards the root node. The recursion comes from the fact that after a succesful transformation the system restarts using the new structure. 'top-down-recursive-scan' works similarly but starts from the root. It is convenient sometimes to let a transformation rule evaluate partially. This is the case when we try to model distant dependencies eg. certain ellipses and anaphora. The rules in 'wait' packets are then used to finnish the incomplete transformations. The label 'transfer' marks those packets which contain attributed rules for recursive descent compilation. These rules are called transfer-rules.

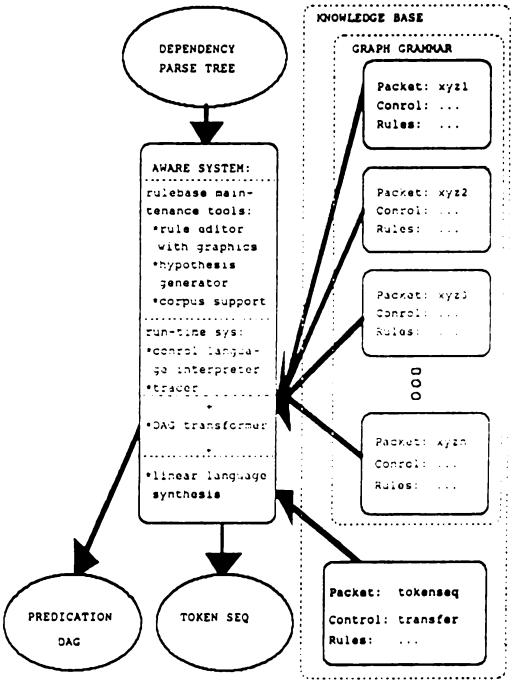


Figure 2. The architecture of the AWARE-system.

The rulebase maintenance tools include a rule editor, a rule hypothesis generator and an automatic book keeping system for corpora. The run-time system contains a control language interpreter, a rule tracer, a precompiler and an interpreter for the actual DAG-transformation rules and for the recursive descent compilation rules.

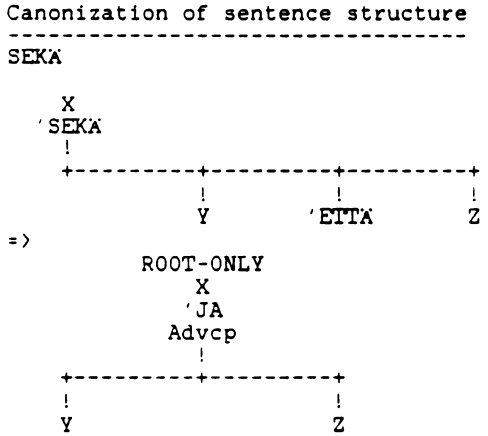
AWARE-rules may function locally without paying attention into larger contexts of the processed constructs. They may also cover whole utterances and on global grounds recognize semantic predications which have their parts syntactically distributed. Rules may amplify themselves by referring to other rules. Also recursive transformations are possible.

An AWARE-description inherits its type definitions from the formal grammar description of the preceding dependency parser (see Lehtola & al. 1985 and Valkonen & al. 1987). A user may also define extra types to be used only in transformations and thereafter (Lehtola & al. 1987b). AWARE is aware of all information that has been derived by the preceding morphological analysis, the lexicalization and disambiguation process and the dependency parser.

3 HOW TO USE TRANSFORMATION RULES

In the following examples we demonstrate the use of transformation rules for different semantic recognition tasks. In many cases the graph transformation reduces into a tree transformation. The first example is very simple, later on we will present more complicated ones.

Canonization of sentence structure



In the rule above one defines a trivial situation where the conjunction phrase SEKÄ-ETTÄ (ie. BOTH-AND) is turned to an AND phrase. The rule is composed of a name and two patterns combined with the rewrite operator '=>'. The first pattern

(ie. left-hand-side of the rule, later abbreviated by lhs) shows the topological and feature conditions which will make the rule to perform a transformation.

This rule will be fired if it recognizes a node with the lexeme SEKÄ and with three subordinates on the right. The second subordinate must have the lexeme ETTÄ. X and Y are called glue variables and they are used in the construction of a new structure. The right hand side of the rule forms a new tree to substitute the matched one. When the lhs of the previous rule is matched the glue variable X will have as its value the structure headed by SEKÄ. The glue variable Y is bound to the first subordinate and the variable Z to the third subordinate. The root node of the new structure is the previous SEKÄ node (ie. root only of the structure bound to X) with its lexeme changed to JA and with a role label Advcp. The root will have two subordinates. The first is the same as the first subordinate in the matched tree and the second is the same as the third subordinate in the matched tree.

In the previous rule we provided the lhs-nodes with restrictive feature conditions (eg. 'SEKÄ and 'ETTÄ) and glue variables. In addition it is possible to provide them with the following directives: ANYNUMBER, ANYORDER, ANYOPTS, ANYDEPS. ANYNUMBER states that a node may have unrestrictedly many subordinates of the specified type. ANYORDER lets the subordinates to be located in any mutual order. ANYOPTS states that a node may have unlimited number of optional subordinates. ANYDEPS is a 'wildcard' for totally relaxing the subordinating structures. Finally the nodes may have references to other rules. By inserting a name of a rule into a node one amplifies his definition. In order to satisfy such rule the substructure starting from the marked node must satisfy the named rule.

The rhs-nodes may be provided with features to be over-written (eg. 'JA and Advcp) and with references to glue variables. The directive ROOT-ONLY is used to cut out the connections to the subordinates. In the example rule it is used to cut out the previous connections of 'SEKÄ node (referred by X) so that the new connections introduced in the rhs of the rule would not be overlapping.

```
Recognizing predications
------------------------
KUULUA
          'KUULUA
          PersForm
             3P
          ANYORDER
           ANYOPTS
            REST
              !
      +-------+-------+
      !               !
      X               Y
  'Subject        Adverbial
    Noun             Noun
   concrete       legal_person
  (OR Nom Part)       All
   ANYDEPS          ANYDEPS
=>
         'POSSESSION
         Predication
              !
      +-------+-------+-------+
      !               !       !
      Y               X     REST
    Actor
```

The preceding rule demonstrates how transformations can be used to define word valencies and to detect semantic predications. The example word is the Finnish verb KUULUA (to belong, to be part of, to be audible etc.). The rule describes one instance of the use of verb KUULUA. Here it is stated that the verb KUULUA expresses possession, when it is in a third person personal form and when it has the following two subordinates:

(1) a subject, which is a noun in nominative or partitive case and means a concrete thing

(2) an adverbial, which is a noun in allative case and means a legal person

Both of the subordinates may have any subordinates. The root verb KUULUA may have those subordinates in any order and it may also have unrestricted number of optional subordinates that are to be bound to the set variable REST.

```
Solving ellipses
----------------
JA
                                    X3
                                   'JA
                                    !
                          +---------+---------+
                          !                   !
                         X2                  X4
                      (AKO: X5)           (AKO: X5)
                        Noun                Noun
                          !
      +-------------------+
      !
     X1
    'JA
 ANYNUMBER
      !
      +---------+
                !
              Gen
        (AKO: legal_person)
              Noun
=>
 ROOT-ONLY
     X3
      !
      +-------------------+-------------------+
                          !                   !
                     ROOT-ONLY               X4
                        X2                    .
                         !                    !
                         +---------+----------+
                                   !
                                  X1
```

The rule above is an example of how ellipses inside sentences can be solved by DAG-transformations. The rule is semantically restricted to the case of the form "the entity1 and entity2 of leganpersonA, legalpersonB, ... ,and legalpersonX". The dependency tree given by the parser has as its root the conjuntion phrase containing the entities. The two (AKO: X5) expressions test that the nouns X2 and X4 coordinate semantically. The conjunction phrase made of the legal persons is syntactically subordinated only to the first entity. The parser does not recognize the ellipsis that also the second entity is in relation with the same legal persons, The meaning of the rule is that when the described situation is recognized the structure X1 is made to be shared by both of the entities.

Wait-rules for distant bindings

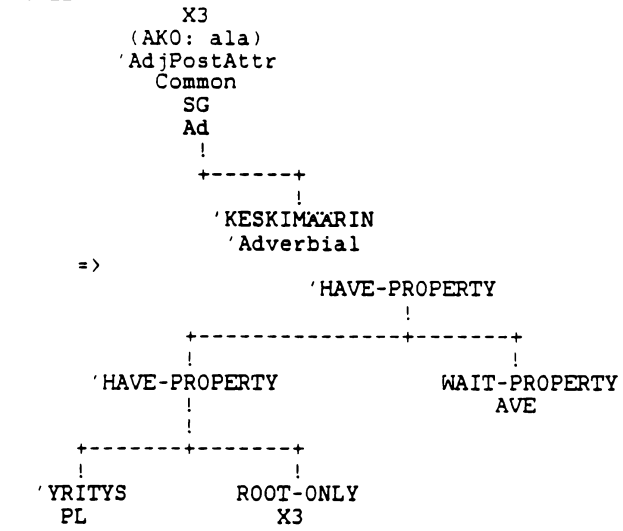
One may leave the filling part (rhs) of a rule partially unspecified. Part of a tree structure is replaced with a call of a wait-rule. Wait-rules are activated afterwards and they look for the matching element from the whole tree structure. Here we demonstrate the use of wait-rules in case of ellipsis. Lets consider the following sentences:

(1) "Anna yritykset, joiden liikevaihto on suurempi kuin metsäalalla keskimäärin!" (Give the companies the turnover of which is greater than the averagenal in forestry)
(2) "Anna yritykset, joiden liikevoiton suhde liikevaihtoon on suurempi kuin metsäalalla keskimäärin!" (Give the companies the ratio of profit and turnover of which is greater than the averagenal in forestry)

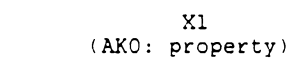
Both sentences have an elliptical expression 'the averagenal (turnover/ratio ..) in forestry'. The system cannot locally decide what is the property referred to. By applying wait-rules the decision can be delayed and the larger context is taken into account.

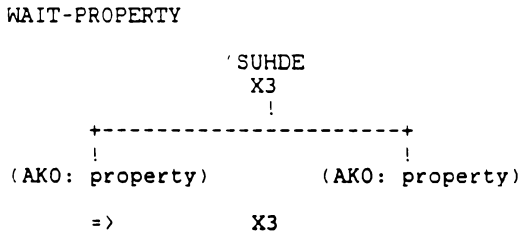
The following rule matches with the expression 'metsäalalla keskimäärin' (the

averagenal in forestry). The rhs of the rule contains a call of a wait-rule. Rule(s) WAIT-PROPERTY specifies different ways of expressing a property of something. See also the label 'AVE' for average which will be attach to the property found.



The following WAIT-rules match with our examples (1) and (2).





WAIT-rules are contained in a packet of their own. This packet is activated after the packet of the calling transformations has been analyzed. Each call of WAIT-rules may cause only one WAIT-rule to fire. If there is a firing WAIT-rule for each call, the whole dependency structure has been satisfied.

4 RECURSIVE DESCENT TRANSLATION

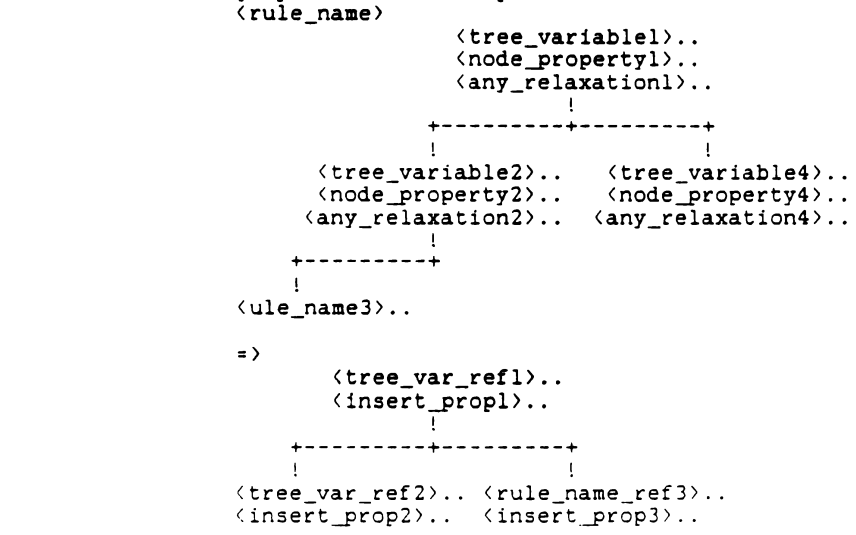
For the production of linear expressions there is an attribute grammar facility. Special translation rules specify the way how different DAG constructs are translated into linear expressions and how a collection of such expressions is mapped into a larger one. The idea is that the transfer rules are seen analogously to the cfg-rules in Knuth's attribute grammars (Knuth 1968). In this

case we are not recognizing a linear language, but rather a graph. The names of AWARE rules correspond to the nonterminals of lhs-parts of cfg-rules. The amplifying rule name references in the match patterns of AWARE rules correspond to the nonterminals in rhs-parts of cfg-rules. In attribute grammar there may be attribute values associated to nonterminals and passed up and down in the constituent tree. In AWARE similar bidirectional attribute value propagation is possible between rules which together cover the recognized graph. In Knuth's grammar attributes are properties of nonterminals, in AWARE they are properties of rule references.

The attribute values may be referred in the equations associated to transfer-rules. These equations consist of tests, assignments, semantic functions and structure building functions. Once a graph is succesfully covered with match-patterns of transfer-rules the equations are instantiated and solved. If there exists solutions the active transfer rules are satisfied, otherwise the search proceeds. The solving process brings the wanted token sequences.

5 RULEBASE MAINTENANCE TOOLS

The rule editor lets the user to manipulate his rules in graphical form. There are two user modifiable representations of the rules, a graphic one and a list expression. The following abstract example demonstrates the parallel use of graphic and list representations:



Is the same as:

```
(⟨rule_name⟩
((DEP: (DEP: ⟨rule_name3⟩)
       ⟨tree_variable2⟩..
       ⟨node_property2⟩..
       ⟨any_relaxation2⟩..)
 ⟨tree_variable1⟩..
 ⟨node_property1⟩..
 ⟨any_relaxation1⟩..
 (DEP: ⟨tree_variable4⟩.. ⟨node_property4⟩.. ⟨any_relaxation4⟩..))
=⟩
((DEP: ⟨tree_var_ref2⟩.. ⟨insert_prop2⟩..)
 ⟨tree_var_ref1⟩..
 ⟨insert_prop1⟩..
 (DEP: ⟨rule_name_ref3⟩.. ⟨insert_prop3⟩..)))
```

The list representations are isomorphic with the graphic representations and the user may choose which ones to edit. The graphical editor supports insertion, modification and deletion of rules. The strength of it becomes apparent when one is doing topological changes into rules. Also nodes are easy to manipulate in graphical form.

The rule hypothesis generator is integrated with the graphical editor. The idea is that by menus the user chooses one of the already created intermediate results to be the lhs of his new rule. The hypothesis generator generalizes the match pattern according to certain heuristic rules and automatically forms a rhs pattern. This rhs pattern is a reduced version of the lhs pattern and the knowledge engineer reforms it by the rule editor. The newly created rule is precompiled and ready for use as the user exists the hypothesis generator. The knowledge acquisition has been very fast by using these tools.

There is an automatic book-keeping facility that records the input sentences and their analysis results into a corpus file. This recording may be done automatically for all input or it may be invoked by the user. The idea is to collect test material to ensure monotonic improvement of knowledge descriptions. After a non trivial change is done in the rulebase, the system runs all test sentences and the results are automatically compared to the previous ones.

6 PERFORMANCE

The AWARE-system has proved to be practical in logico-semantic analysis of Finnish and in query synthesis. It is in daily use in our database interface prototype for a Finnish business database. Total processing of a one line long question takes between 5 and 50 seconds of CPU-time on VAX-11/780. The DAG transformations and the conceptual query synthesis consume about 50 percent of this. The size of the rule base is currently almost 400 rules.

At the first glance the figures may depress. Taken into account that the complexity of the transformational analysis is very high the time consumption is not surprising. At the moment the rules are precompiled into effective data structures, an inverted index is created out of the match conditions and structure sharing is used to minimize memory consumption. Internal data structures are only partly dynamic for the reason of fast information fetch. In spite of the

preceding measures there are still many ways to improve the performance. The current implementation is in FranzLisp.

7 CONCLUSIONS

Compared to certain well known transformation systems (eg. Periphrase of ALPS, MITRE) the AWARE-system offers the following extra properties:

(1) rich type system,
(2) processing generalized for directed acyclic graphs,
(3) orientation towards dependency structures,
(4) powerful tools for knowledge base maintenance,
(5) extensive use of graphics to illustrate the operation,
(6) attribute grammar facility for translation
(7) separate control language
(8) lazy evaluation possible using 'wait' rules

One of the design objectives in AWARE has been to make it so general that it could be used also in machine translation. Dependency structures have been found a good syntactic representation for machine translation purposes. Our dependency parser (Lehtola & al. 1985 and Valkonen & al. 1987) together with AWARE gives interesting prospects for MT.

References

ALPS (1986):
Periphrase Introduction. Report of A.L.P. Systems, Provo, 25 p.

Hobbs, J., Grishman, R. (1976):
The Automatic Transformational Analysis of English Sentences: An Implementation. Intern. J. Computer Math. 1976, Section A, Vol. 5, pp. 267-283.

Jäppinen, H., Honkela, T., Lehtola, A. and Valkonen, K. (1988):
Hierarchical Multilevel Processing Model for Natural Language Database Interface. Proceedings of the 4th IEEE Conference on Artificial Intelligence Applications, San Diego, California, 6 p. (in print).

Knuth, D. E. (1968):
Semantics of Context-Free Languages. Mathematical Systems Theory, vol. 2, no. 2, Springer-Verlag, New York, pp. 127-145.

Lehtola, A., Jäppinen, H. and Nelimarkka, E. (1985):
Language-based Environment for Natural Language Parsing. Proceedings of the 2nd European Conference of ACL, Geneve, pp. 98-106.

Lehtola, A. and Valkonen, K. (1987a):
Knowledge Representation Formalisms and Metadescriptions for the Interpretation of Finnish. Proceedings of the Third Finnish Symposium on Theoretical Computer Science, pp. 64-87.

Lehtola, A. and Honkela, T. (1987b):
AWARE - A DAG Production System with Attribute Grammar Facility *revised report*. Publications of the Kielikone-project, Series B, report 4, Helsinki, 36 p.

Valkonen, K., Jäppinen, H. and Lehtola, A. (1987):
Blackboard-based Dependency Parsing. 10th International Joint Conference of Artificial Intelligence, Milano, pp. 700-702.

Winograd, T. (1983):
Language as a Cognitive Process. Volume I: Syntax. Addison-Wesley Publishing Company, Reading, 640 p.

Zwicky, A., Friedman, J., Hall, B., Walker, D. (1965):
The MITRE Syntactic Analysis Procedures for Transformational Grammars. Proceedings of the Fall Joint Computer Conference 1965, pp. 317-326.