Acta Cybernetica15(2001) 279–304.

Using Decision Trees to Infer Semantic Functions of Attribute Grammars *

Szilvia Zvada† and Tibor Gyimóthy†

Abstract

In this paper we present a learning method called LAG (Learning of Attribute Grammar) which infers semantic functions for simple classes of attribute grammars by means of examples and background knowledge. This method is an improvement on the AGLEARN approach as it generates the training examples on its own via the effective use of background knowledge. The background knowledge is given in the form of attribute grammars. In addition, the LAG method employs the decision tree learner C4.5 during the learning process. Treating the specification of an attribute grammar as a learning task gives rise to the application of attribute grammars to new sorts of problems such as the Part-of-Speech (PoS) tagging of Hungarian sentences.

Here we inferred context rules for selecting the correct annotations for ambiguous words with the help of a background attribute grammar. This attribute grammar detects structural correspondences of the sentences. The rules induced this way were found to be more precise than those rules learned without this information.

1 Introduction

Attribute grammars were introduced in [11] as a formalism for the specification of the semantics of program languages (see [1, 4]). They can be considered as an extension of context-free grammars in the sense that attributes and their semantic functions are related to the symbols of the grammar. An attribute is a named property with given values and a semantic function computes its value based on the values of other attributes. A semantic functions may be complex, therefore the specification of an attribute grammar may be a laborious task. Hence a tool which is able to complete a partially given attribute grammar by means of examples would be very useful. The term "partially given" here means that some of the attributes might lack semantic function. The task is to define these unknown semantic functions.

Based on the correspondence of the nonterminals of attribute grammars and the predicates of logic programs (see [5, 6, 17]), we can apply many of techniques to

*This work was supported by the grants OTKA T52721 and IKTA 8/99.

†Research Group on Artificial Intelligence, University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1, Hungary. e-mail: {zvada,gyimi}@sol.inf.u-szeged.hu

attribute grammars, which techniques were originally developed for logic programs. For instance, viewing the specification of an attribute grammar as a learning task, learning methods presented in the framework of inductive logic programming (ILP, see [12, 14]) can be used to solve this task.

ILP is an active research area of machine learning that studies the definitions of logic programs from examples and the presence of background knowledge. Since examples and background knowledge are expressed in first-order logic, ILP methods can be employed to learn relational and recursive definitions. This last property makes ILP methods very promising for the attribute grammars, because recursive rules often emerge in attribute grammars as well.

This fact suggested the use of a similar learning approach in the AGLEARN method ([8]) to that one employed in the ILP learning system called LINUS ([12]), but in a different representational framework. That is, the learning task and background knowledge is represented in the form of attribute grammars instead of logic programs. The task of AGLEARN is to complete the specifications of an S-attributed or an L-attributed grammar based on positive and negative examples. These examples contain strings derived from the target nonterminal, the attributes of this target nonterminal being evaluated in these strings. The main idea behind AGLEARN is converting the learning task into a propositional form then inferring the unknown semantic function with the help of a propositional learner.

In this paper we introduce the LAG approach which is based on the AGLEARN method, but it uses the given background knowledge more effectively and employs the C4.5 decision tree learner (see [19]) instead of a propositional learner. Doing this allows treating the learning task as a classification problem with multiple classes. The robustness of the C4.5 for classification problems has already been demonstrated. Another important difference between the AGLEARN and LAG methods can be seen in the handling of the training examples. In the case of the former, the user has to explicitly define each training example in advance. With the latter, the input of the LAG system consists of strings taken from the language generated by the partially given attribute grammar. The LAG system builds the decorated decision trees of these strings and evaluates the attribute instances during the tree traversals. Whenever an attribute instance with no semantic function is computed its value is defined by the user ("oracle"). Hence even a few strings can produce a large number of training examples.

The LAG method is applied to the Part-of-Speech tagging of Hungarian sentences. The task here is to distinguish the different morphologic classes of a word, as in the case of "múlt"[1], which might be annotated by a verbal, noun or adjectival tag. The tagging of Hungarian texts is very difficult due to the rich morphology of the language. Our method has been applied in order to infer the rules for selecting the contextually correct tags. The input data set, a corpus with about 100 000 pre-tagged words ([7, 16]), is employed for training and testing. The background attribute grammar determines some structural information of the parts of sentences

[1] *múlt (verb) -passed*
múlt (noun) - past
múlt (adjective) - past, last

like subject phrase and predicate phrase. Based on the latter the training data sets for the C4.5 system are generated. By using the training sets decision rules are inferred for ambiguous words. The experimental results show (see also [2, 9]) that the use of even simple attribute grammars as background knowledge yields more effective rules than a method which lacks this structural information.
In Section 2 the key definitions relating to the learning of attribute grammars are introduced, while in Section 3 the LAG method itself is discussed. Afterwards, in Section 4 a brief overview of the PoS tagging problem and the application of the LAG method is presented. The accuracy of the C4.5 and LAG approaches is compared in Section 5. In the final section, the conclusions are drawn and suggestions for future research are offered.

2 Preliminaries

In this section we introduce the terminology and notations used in this paper.

2.1 Attribute grammars

Attribute grammars were introduced in [11] as an extension of **context-free grammars** (**cfg** onwards) for specifying static semantics of programming languages, such as type-checking and name-analysis during syntax-directed parsing. This is achieved by attaching attributes (named properties with given values) to the symbols of the grammar. During the parsing a derivation tree based on the underlying *cfg* is constructed. In this tree nodes and leaves are labeled by nonterminals and terminals of the *cfg*, respectively. The **instances** of the attributes appear in this tree along with the grammar symbols which they are related to. This tree is called **decorated derivation tree** or simply **ddt**.
The value of an attribute instance is defined by its semantic function during the traversal of the ddt. The value of an attribute is determinable iff the values of all the attributes in the argument of the semantic function have already been computed. In this way the semantic functions define dependency relations among the attributes. The attributes transmit information within the ddt in two directions: from the root to the leaves, where they are named **inherited attributes**, or backwards, where they are called **synthesized attributes**.
Before we formally define the learning task for attribute grammars, let us first consider the definitions and notations of attribute grammars (cf. [1]).
An **attribute grammar** (briefly **ag**) is a four tuple $AG = (G, SD, AD, R)$ which consists of the following components:

- an underlying *cfg* $G = (V_N, V_T, P, S)$
- a semantic domain $SD = (\mathcal{T}, \mathcal{F})$ consisting of a set $\mathcal{T}$ of the domains of attributes and a set $\mathcal{F}$ of functions over the attributes: $type_1 \times \cdots \times type_m \rightarrow type_0$ for $type_i \in \mathcal{T}$ $(0 \leq i \leq m)$.
  (If $type_0 = \{true, false\}$ then we talk about *relations*.)

- an attribute description is a triple $AD = (Inh, Syn, \tau)$ where Inh and Syn are finite, disjoint sets of **inherited** and **synthesized attributes**, respectively. $Attr = Inh \cup Syn$ is the set of all attributes of AG. Let $X.a$ denote an attribute $a \in Attr$ attached to the grammar symbol $X \in V_N \cup V_T$. The set $Inh(X)$ and set $Syn(X)$ consist of the inherited and synthesized attributes of the symbol X, respectively. τ is a function mapping attributes to their types (domains) such that $\tau : Attr \to \mathcal{T}$.
- a set $R = \{R(p) \,|\, p \in P\}$ consisting of finite sets $R(p)$ of **semantic functions** which are associated with the production $p : X_0 \to X_1 \ldots X_{m_p}$. An **occurrence** of an attribute $X_k.a$ in the production p is denoted by $X_k^p.a$. The set

  $$DO(p) = \{X_0^p.s \in Syn(X_0)\} \cup \{X_k^p.i \in Inh(X_k) \text{ with } 1 \le k \le m_p\} \text{ and}$$

  $$UO(p) = \{X_0^p.i \in Inh(X_0)\} \cup \{X_k^p.s \in Syn(X_k) \text{ with } 1 \le k \le m_p\}$$

  of **defined attribute occurrences** and **used attribute occurrences** of p, respectively, are assigned to every production $p \in P$. For every $X_k^p.a \in DO(p)$ there is exactly one semantic function given in $R(p)$

  $$X_k^p.a = f\left(X_{k_1}^p.a_1, \ldots X_{k_s}^p.a_s\right)$$

  with $(f : \tau(a_1) \times \cdots \times \tau(a_s) \to \tau(a)) \in \mathcal{F}$ and $X_{k_i}^p.a_i \in UO(p)$ for $1 \le i \le s$. Then we say that $X_k^p.a$ depends on $X_{k_i}^p.a_i$, for $1 \le i \le s$. (Note that if $s = 0$ the function is a constant $c \in \tau(a)$.)

In several applications it is useful to attach a special, synthesized, boolean attribute *accept* to the start symbol S of the underlying *cfg*. Using the attribute *accept* we can define the language generated by an attribute grammar like so:

$$Lang(AG) = \{w \,|\, w \in Lang(G) \text{ and } S.accept = true \text{ in the ddt of } w\}.$$

Let $AG = (G, SD, AD, R)$ be an *ag* with an underlying *cfg* $G = (V_N, V_T, P, S)$, a semantic domain $SD = (\mathcal{T}, \mathcal{F})$ and an attribute description $AD = (Inh, Syn, \tau)$. Furthermore, let t be a *ddt* and n_0 be a node of t, which is labelled by $X_0 \in V_N \cup V_T$. The set $Inh(n_0) = \{n_0.i | X_0.i \in Inh(X_0)\}$ of **inherited attribute instances** and the set $Syn(n_0) = \{n_0.s | X_0.s \in Syn(X_0)\}$ of **synthesized attribute instances** are associated with the node n_0. Thus $Inst(n_0) = Inh(n_0) \cup Syn(n_0)$. (Note that $\tau(n_0.i) = \tau(X_0.i) = \tau(i)$ holds for any $n_0.i \in Inst(n_0)$.)
Further, let the production $p : X_0 \to X_1 \ldots X_{m_p}$ be applied at node n_0. Then $X_1, \ldots, X_{m_p}$ label the successors $n_1, \ldots, n_{m_p}$ of n_0, from left to right, respectively. Let

$$DI(n_0, p) = \{n_k.a \,|\, X_k^p.a \in DO(p) \text{ with } 0 \le k \le m_p\} \text{ and}$$

$$UI(n_0, p) = \{n_k.a \,|\, X_k^p.a \in UO(p) \text{ with } 0 \le k \le m_p\}$$

be the set of **defined attribute instances** and **used attribute instances** of n_0, respectively. Then an instance $n_l.a$ of the defined attribute occurrence $X_l^p.a$ is determined by $f(n_{k_1}.a_1, \ldots, n_{k_m}.a_m)$, where $n_{k_1}a_1 \ldots n_{k_m}a_m \in UI(n_0, p)$ are the

instances of the attribute occurrences $X^p_{k_1}.a_1, \ldots, X^p_{k_m}.a_m \in UO(p)$ and f is the interpretation of the semantic function

$$X^p_l.a = f\left(X^p_{k_1}.a_1, \ldots X^p_{k_m}.a_m\right).$$

An attribute instance $n_{k_0}.a$ depends on the attribute instances $n_{k_i}.a_i$, for $1 \leq i \leq m$. It is clear that an attribute instance $n_l.a$ can be computed if all attribute instances on which it depends have already been evaluated.
An *ag* is *circular* if it has a ddt such that there is a circular dependency among the attribute instances. Otherwise an *ag* is *non-circular*. Here we consider two subsets of the non-circular *ags*, namely the L-attributed and S-attributed grammars.
An *ag* is **L-attributed** if all attribute instances of an arbitrary ddt of this *ag* can be evaluated in one left-to-right tree traversal. The left-to-right traversal and the attribute evaluation are described by the following procedure:

```
proc tree_traversal(node : n0);
    begin
        for i := 1 to mp do
            begin
                eval(Inh(ni));
                tree_traversal(ni);
            end;
        eval(Syn(n0));
    end;
```

One can formulate conditions for the L-attributed property. Let $X^p_l.a$ be a defined attribute occurrence of the production p and $X^p_l.a = f\left(X^p_{k_1}.a_1, \ldots X^p_{k_s}.a_s\right)$. Then the *ag* is L-attributed if the following conditions hold for each defined attribute occurrence (see Figure 2.1):

- if $X^p_l.a$ is an inherited attribute occurrence then $X^p_{k_i}.a_i \in Inh(X^p_0)$ or $X^p_{k_i}.a_i \in Syn(X^p_{k_i})$, with $1 \leq i \leq s$ and $1 \leq k_i < l$. This means that an inherited attribute occurrence $X^p_l.a$ may depend on the synthesized attribute occurrences of the rhs symbols $X^p_{k_i}$, that have been defined before than X^p_l. It may also depend on the inherited attribute occurrences of the lhs symbol X^p_0 as shown in Figure 1. Here an inherited attribute occurrence is visualized by a white circle above the respective symbol, whereas a synthesized attribute occurrence is depicted as a black dot below it.

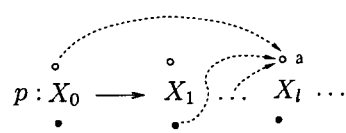


Figure 1: L-attributed dependencies of $X^p_l.a$

- if $X^p_0.a$ is a synthesized attribute occurrence then $X^p_{k_i}.a_i \in Inh(X^p_0)$ or $X^p_{k_i}.a_i \in Syn(X^p_{k_i})$, with $1 \leq i \leq s$ and $1 \leq k_i \leq m_p$. Namely, this

means that an lhs synthesized attribute occurrence $X_0^p.a$ of a rule may depend on synthesized attribute occurrences of rhs symbols and inherited attribute occurrences of the lhs symbol, itself. Figure 2 presents these relations.

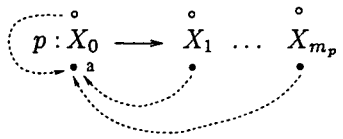


Figure 2: L-attributed dependencies of $X_0^p.a$

Let the set $UO_{L-attr}(X_l^p.a)$ denote the used attribute occurrences of p which fulfill these two conditions with respect to the attribute occurrence $X_l^p.a$.
The other subset of non-circular *ag*s investigated in this paper is the S-attributed grammar. An *ag* is **S-attributed** if solely synthesized attributes are related to the symbols of the grammar. It is clear that the set of S-attributed grammars is a subset of L-attributed grammars.
To help to make these definitions clearer, let us illustrate their use with a concrete example.

Example 1 *The S-attributed* ag $AG_{typ} = (G_{typ}, SD_{typ}, AD_{typ}, R_{typ})$ *defined below determines whether the type of an arithmetical expression is real or integer.*

nonterminals and terminals $V_N = \{Expr, Term, Factor, AddOp, MulOp, \}$
$V_T = \{Integer, Real, =, -, *, /, \lambda\}$

the semantic domain SD_{typ} $\mathcal{T} = \{type_{mode}, type_{op}\}$, *where*
$type_{mode} = \{int, real\}$, *and*
$type_{op} = \{add, sub, mul, div\}$
$\mathcal{F} = \{f_1 : type_{mode} \times type_{mode} \rightarrow type_{mode},$
$f_2 : type_{mode} \times type_{op} \times type_{mode} \rightarrow type_{mode}\}$
where $f_1(x, y) =$ ***if*** $(x = int) \wedge (y = int)$
then *int*
else *real*
$f_2(x, y, z) =$ ***if*** $(x = int) \wedge (y = mul) \wedge (z = int)$
then *int*
else *real*

the attribute descriptions AD_{typ} $Inh = \emptyset$
$Syn = \{mode, op\}$
$Syn(Expr) = Syn(Term) = Syn(Factor) = \{mode\}$
$Syn(AddOp) = Syn(MulOp) = \{op\}$
$\tau(mode) = \{int, real\}$
$\tau(op) = \{add, sub, mul, div\}$

the underlying cfg G_{typ} *and the set* R_{typ} *of semantic functions:*

1, $Expr_0 \rightarrow Expr_1\ AddOp\ Term$
$R(1) = \{\ Expr_0.mode := f_1(Expr_1.mode, Term.mode)\}$

$2,\ Expr \rightarrow Term$
$R(2) = \{ Expr.mode := Term.mode\}$
$3,\ Term_0 \rightarrow Term_1\ MulOp\ Factor$
$R(3) = \{Term_0.mode := f_2(Term_1.mode, MulOp.op, Factor.mode)\}$
$4,\ Term \rightarrow Factor$
$R(4) = \{Term.mode := Factor.mode\}$
$5,\ Factor \rightarrow Integer$
$R(5) = \{ Factor.mode := int\}$
$6,\ Factor \rightarrow Real$
$R(6) = \{ Factor.mode := real\}$
$7,\ Factor \rightarrow (\ Expr\ )$
$R(7) = \{ Factor.mode := Expr.mode\}$
$8,\ AddOp \rightarrow +$
$R(8) = \{ AddOp.op := add\}$
$9,\ AddOp \rightarrow -$
$R(9) = \{ AddOp.op := sub\}$
$10,\ MulOp \rightarrow \times$
$R(10) = \{ MulOp.op := mul\}$
$11,\ MulOp \rightarrow /$
$R(11) = \{ MulOp.op := div\}$

some of the defined and used attribute occurrences:

$DO(1) = \{Expr_0.mode\}$
$DO(2) = \{Expr.mode\}$
$DO(3) = \{Term_0.mode\}$

$UO(1) = \{Expr_1.mode, AddOp.op, Term.mode\}$
$UO(2) = \{Term.mode\}$
$UO(3) = \{Term_1.mode, MulOp.op, Factor.mode\}$

It is immediately apparent that for S-attributed grammars, all the used attribute occurrences satisfy the L-attributed property.
Nevertheless, the specification of semantic functions is not trivial even in the case of L-attributed and S-attributed grammars. The current paper introduces a method which learns the semantic functions of *ags* like these.

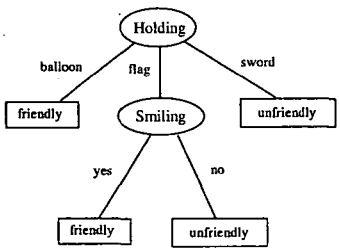
2.2 Inductive learning

The idea of using inductive learning methods to define semantic functions of an attribute grammar was motivated by the parallelism found between the nonterminals of attribute grammars and the predicates of logic programs (see [5, 6, 17]).
In general, an inductive learning method studies a set of positive and negative *training examples* and *background knowledge* in order to infer a hypothesis which approximates the *target concept.* The inferred hypothesis explains the training examples together with the background knowledge such that all positive examples can be 'proved' by it and no negative example can be 'derived' from it. Many inductive learning approaches use an attribute-value language to represent the

examples, background knowledge and the concept to be induced. The most popular of these attribute-value learners are *decision tree learners* used widely in solving classification problems ([13, 19]).
These methods construct decision trees for modelling the target hypotheses from the training examples expressed as attribute-value vectors. In a decision tree, every interior node is labelled with a test over an attribute which is expected to most efficiently classify the current subset of training examples. The possible outcomes of these attribute tests assign a name to the branches descending from the nodes. The leaves show a "class" to which the examples of the current training set belong. The decision trees can be also represented by a set of *decision rules* (see Example 2). The LAG method makes the use of the decision rules during the learning process. The decision trees can be constructed by a heuristically guided, hill climbing algorithm called ID3 ([12]). Its heuristic is based on an information-theoretic measure called *entropy*, which measures the length of the encoding of the current training set in bits. The most popular decision tree learner algorithm is the C4.5 system ([19]) which is widely used in academic and industrial spheres. There are many good textbooks available on decision tree learner methods ([12, 13, 19]). In the following, we represent a decision tree constructed for a learning task.

Example 2 *(A modified version of an example in [12].) The task is to find a concept which describes whether a robot is friendly or not, based on the properties Smiling, Holding, Has_tie, Head_Shape, Body_Shape, and an initial set of training examples.*

Smiling	*Holding*	*Has_tie*	*Head_Shape*	*Body_Shape*	*Class*
yes	*balloon*	*yes*	*square*	*square*	*friendly*
yes	*flag*	*yes*	*octagon*	*octagon*	*friendly*
yes	*balloon*	*no*	*round*	*round*	*friendly*
yes	*flag*	*no*	*octagon*	*octagon*	*friendly*
yes	*flag*	*no*	*octagon*	*octagon*	*friendly*
yes	*balloon*	*no*	*square*	*square*	*friendly*
yes	*sword*	*yes*	*round*	*octagon*	*unfriendly*
yes	*sword*	*no*	*square*	*octagon*	*unfriendly*
no	*sword*	*no*	*octagon*	*round*	*unfriendly*
no	*flag*	*no*	*round*	*square*	*unfriendly*



$Rule_1$: *Holding* = *balloon* → *class* ***friendly***
$Rule_2$: *Smiling* = *yes* ∧ *Holding* = *flag* → *class* ***friendly***
$Rule_3$: *Smiling* = *no* → *class* ***unfriendly***
$Rule_4$: *Holding* = *sword* → *class* ***unfriendly***

Default class: ***friendly***

Figure 3: The decision tree and decision rules constructed by the C4.5

These learning methods which generally yield robust, reliable results are even able to handle noisy input data and continuous attributes. However, they have some drawbacks as well. In the attribute-value-based representation, variables cannot be used, hence these learning methods cannot deal with complex relations. Another disadvantage is the inability of use of background knowledge.
The above problem was bridged by the introduction of inductive logic programming (ILP, [12, 14]). The learning methods developed in the ILP framework employ first-order logic to represent the learning task, the training examples and background knowledge. The latter is used intensively in the learning process.
The ILP learning system called LINUS ([12]) combines the advantages of attribute-value learners and first-order-logic-based representation. The learning approach of the LINUS system can be summarized in three steps:

- It transforms the learning task into a propositional form.
- The transformed learning task is solved by using an appropriate propositional learner.
- The results of this propositional learner are converted back into a first-order logic form.

A similar learning method (see Figure 4) is used in the AGLEARN algorithm for inducing attribute grammars. However, the AGLEARN describes the learning task and background knowledge used with the help of an attribute grammar instead of a logic program.

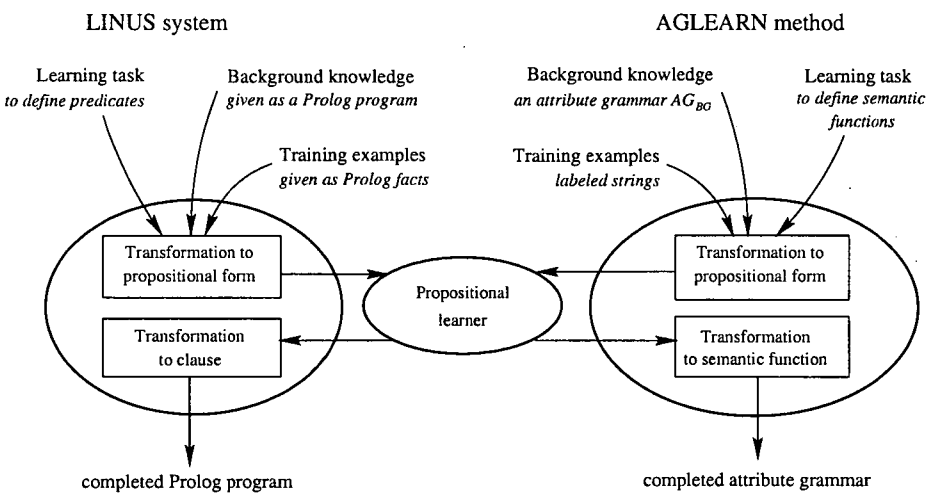


Figure 4: Similarities and differences between the LINUS system and the AGLEARN method

2.3 Description of the learning task

In this section, we formulate the learning task of *ags* in the following way:
The goal of the learning is to give a complete specification for the *ag* $AG = (G, SD, AD, R)$ from a *partially given L-attributed ag* $AG_{inp} = (G, SD, AD, R_{inp})$ and a set $\mathcal{W}_{inp}$ of *strings* taken from the language generated by AG_{inp}.
The term "partially given" here means that $R_{inp} \subseteq R$, namely some of the semantic functions of AG_{inp} are undefined. This AG_{inp} not only describes the background knowledge and the learning task, but is used to generate the training examples from the strings of $\mathcal{W}_{inp}$. The **background knowledge** is given as a fully defined *ag* $AG_{BG} = (G, SD_{BG}, AD_{BG}, R_{inp})$, where $SD_{BG} \subseteq SD$ and $AD_{BG} \subseteq AD$. The **learning task** is specified by the following items:

(1) The semantic domain $SD_{tar} = (\mathcal{T}_{tar}, \mathcal{F}_{tar})$ which consists the types of the target attributes ($\mathcal{T}_{tar}$) and initial functions ($\mathcal{F}_{tar}$) over these attributes. SD_{tar} is defined in advance, such that $SD_{tar} \cup SD_{BG} = SD$ holds. The LAG method constructs the unknown semantic functions from the elements of $\mathcal{F}_{tar}$.

(2) The description $AD_{tar} = (Inh_{tar}, Syn_{tar}, \tau)$ of the target attributes are related to the symbols of *cfg* G such that $AD_{tar} \cup AD_{BG} = AD$ holds.

(3) A set $TO(p)$ of the target attribute occurrences is assigned to production $p : X_0 \to X_1 \ldots X_{m_p}$ of G.
A defined attribute occurrence $X_l^p.a \in TO(p)\,(0 \leq l \leq m_p)$ if it has no semantic function in R_{inp}. In this case $X_l^p.a$ is called **target attribute occurrence**. $TO = \bigcup_p TO(p)$ denotes the set of all target attribute occurrences.

To be more exact, the learning method infers the unknown semantic functions of R_{tar} for the target attribute occurrences then completes the specification of AG_{inp} such that $R_{tar} \cup R_{inp} = R$ will hold.
The **training examples** for the target attribute occurrences are generated during the derivation of the input strings of $\mathcal{W}_{inp}$. Based on the AG_{inp}, a ddt_w is built for each $w \in \mathcal{W}_{inp}$ string. Let n_0 be a node of ddt_w labelled by X_0 and let $p : X_0 \to X_1 \ldots X_{m_p}$ be applied at this node. Moreover, let $X_1, \ldots, X_{m_p}$ each label the successor $n_1, \ldots, n_{m_p}$ of n_0, respectively. Then, during the traversal and evaluation of ddt_w for each instance $n_l.a$ of $X_l.a \in TO(p)$, $(0 \leq l \leq m_p)$, an example

$$e = (w, (u_1, v_1), \ldots (u_k, v_k), (n_l.a, v_0))$$

is added to the training set $\mathcal{E}_{X_l^p.a}$. The $v_1, \ldots, v_k$ denote the values of the instances of the used attribute occurrences $u_1, \ldots, u_k \in UO_{L-attr}(X_l^p.a)$ that have already been computed. With a knowledge of these values, the value v_0 of the target attribute instance $n_l.a$ is defined by the user.

Example 3 *We show what these definitions look like with the help of the type-checking example* AG_{typ} *(see Example 1). Let us suppose that the semantic functions in the production 1 and 3 are unknown:* $R(1) = R(3) = \emptyset$.

$$
\begin{aligned}
\textit{input strings } \mathcal{W}_{inp} = \{\ & ((3*2+6)-7)/(3*1.5-2.5/5),\\
& (3/2-1)*3+(0.7*(0.1+1))/(6*2+4.3)\}
\end{aligned}
$$

$$
\begin{aligned}
\textit{background knowledge } AG_{BG} &= (G_{typ}, SD_{typ}, AD_{typ}, R_{BG}),\ \textit{where } R_{BG} \subseteq R_{typ}\\
\textit{learning task } SD_{tar}\ \ \mathcal{T}_{tar} &= \{\{true, false\}\}\\
\mathcal{F}_{tar} &= \{=^2\},\ \textit{where } =^2 \textit{ is the identity relation}\\
AD_{tar}\ \ Syn &= \{mode\}\\
Syn(Expr) &= Syn(Term) = \{mode\}\\
\tau(mode) &= \{int, real\}\\
\textit{target attribute occurrences}\quad TO &= \{Expr_0^1.mode, Term_0^3.mode\}
\end{aligned}
$$

3 Learning of L-attributed grammars

In this section we introduce the *LAG* system which infers semantic functions for L-attributed grammars. It takes a partially given *ag* AG_{inp} and a set $\mathcal{W}_{inp}$ of strings of the language generated by AG_{inp} as input. The term 'partially given' here means that AG_{inp} has some attribute occurrences which have no semantic function. During the learning process the LAG method infers these unknown semantic functions and adds them to AG_{inp} to complete its specification.
AG_{inp} describes the learning tasks and the background *ag* AG_{BG}. In addition, it is used to generate the training examples from the strings of $\mathcal{W}_{inp}$. For each string a ddt is constructed by AG_{inp}, which also consists of instances of target attribute occurrences. During the evaluation of the ddt the values of these target instances are determined by the user with the knowledge of the values of other attribute instances. The latter have been computed automatically based on AG_{inp}. This is an important advantage of this system compared to other learning methods where a whole set of training examples have to be given in advance. After generating the training examples for the target attribute occurrences, the LAG system transforms the learning task and background knowledge into[2] an attribute-value representation.[3]
The learning tasks represented this way are solved by the decision tree learner, C4.5 ([19]). Finally, the hypotheses produced by the C4.5 in the form of decision rules are transformed back into *"if-then"* semantic functions (see Example 1).
The basic steps of the *LAG method* can be summarized as follows:

- *Generation of the training examples* from the input strings.
- *Transformation into attribute-value tuples*: a training table consisting of attribute-value tuples is constructed for each target attribute occurrence.
- *Decision tree learning*: solving the transformed learning tasks using the C4.5 system: the decision rules are built based on the training tables.
- *Formulating semantic functions*: The rules inferred by C4.5 are transformed back into the form of semantic functions.

[2] described by an attribute grammar
[3] expressed as attribute-value vectors

3.1 Generation of training examples

Using the input *ag* AG_{inp} we build a ddt_w for each input string w in $\mathcal{W}_{inp}$. In these *ddt*s the target attribute occurrences may have arbitrarily many instances.

Let n_0 be a node of ddt_w where the production $p : X_0 \to X_1 \ldots X_{m_p}$ is applied and let $n_l.a$, $(0 \leq l \leq m_p)$ be the instance of the target attribute occurrence $X_l^p.a \in TO(p)$. Further, let $UI_{L-attr}(n_l, a)$ denote the set of used attribute instances $n_{k_1}.u_1, \ldots, n_{k_s}.u_s \in UI(n_0, p)$, which fulfill the L-attributed conditions (see p. 283): they were computed before the evaluation of target attribute instance $n_l.a$.

During the evaluation of the ddt_w, the user is asked about the values of the target attribute instances by substituting the unknown semantic functions with a **question IQ**:

```
proc  IQ (set : UI_{L-attr}, inst : target);
      begin write('The used attribute instances have the following values:');
            write(UI_{L-attr}, p );
            read(target);
      end;
```

In addition, replacing using the procedure *new_eval*() instead of the procedure *eval*() in the *tree_traversal*() (see p. 283) process yields examples which are added to the training set $\mathcal{E}_{X_l^p.a}$ for each instance of the target attribute occurrence $X_l^p.a$.

```
proc new_eval (set : DI, node : n);
   begin for each a ∈ DI do
             if a ∉ TO then eval(n.a);
             else begin
                  a := IQ(UI_{L-attr}(n, a), a);
                  add_example(w, UI_{L-attr}(n, a), a);
                  end;
   end;
```

During the evaluation, one example is generated for each instance of each target attribute occurrence in the ddt_w. Hence examples can be produced for different training example sets. Since the training set $\mathcal{E}_{X_l^p.a}$ may contain an example more than once, even a small number of input strings can induce numerous training examples: $|\mathcal{W}_{inp}| \leq \left| \bigcup_{TO} \mathcal{E}_{X_l^p.a} \right|$.

Example 4 *(Continuing from Example 3) The training example set* $\mathcal{E}_{Expr_0^1.mode}$ *is generated for the target attribute occurrence* $Expr_0^1.mode$ *of production 1. (A similar training set can be constructed for the target attribute occurrence* $Term_0^3.mode$ *as well.)*

Let $w_1 = \big((3 * 2 + 6) - 7\big)/(3 * 1.5 - 2.5/5)$ *and*
$w_2 = (3/2 - 1) * 3 + \big(0.7 * (0.1 + 1)\big)/(6 * 2 + 4.3)$

denote the two input strings. The production 1 *is applied three times in* ddt_{w_1}*, hence three examples are generated for* $Expr_0^1.mode$ *during the traversal of* ddt_{w_1}*. Similarly, the traversal of* ddt_{w_2} *produces four examples. It is easy to check that* $\mathcal{E}_{Expr_0^1.mode}$ *is the following after the evaluation of* ddt_{w_1} *and* ddt_{w_2}*:*

	UO_{L-attr}			
$w \in \mathcal{W}_{inp}$	$Expr_1.mode$	$Addop.op$	$Term.mode$	$Expr_0.mode$
w_1	*int*	*add*	*int*	*INT*
w_1	*int*	*sub*	*int*	*INT*
w_1	*real*	*sub*	*real*	*REAL*
w_2	*real*	*sub*	*int*	*REAL*
w_2	*real*	*add*	*int*	*REAL*
w_2	*int*	*add*	*real*	*REAL*
w_2	*real*	*add*	*real*	*REAL*

3.2 Transformation into attribute-value tuples

Upon generating the training example set, the LAG method transforms the learning task into attribute-value tuples. One training table is generated for each target attribute occurrence (i.e. in the type-checking example two training tables are constructed: one for $Expr_0^1.mode$ and one for $Term_0^3.mode$).

There are two ways of formulating the training tables depending on the type of target attribute occurrences:

(1) Enumerated case: when the domain of the target attribute occurrence $X_l^p.a$ is defined by a finite list. In this case our aim is to infer a classification-like semantic function for it, where the classes are made up of the $c_1, \ldots, c_k$ elements of the domain. The training table $T_{X_l^p.a}$ consists of columns

$$\{string\} \cup UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}} \cup \{class\},$$

where columns *string*, *class* and UO_{L-attr} are constructed from the training example set $\mathcal{E}_{X_l^p.a}$. The column *class* contains the value of $X_l^p.a$ computed during the evaluation of ddt_w, where $w \in \{string\}$. The set $\mathcal{R}_{\mathcal{U}}$ consists of the *satisfiable interpretations* of each relation $r : \tau(x_1) \times \cdots \times \tau(x_m) \rightarrow \{true, false\}$ given in SD_{inp}. An interpretation $r(u_1, \ldots, u_m)$ is satisfiable iff $u_i \in UO_{L-attr}$ and $\tau(u_i) = \tau(x_i)$, for all $i = 1 \ldots m$.

Example 5 *(Continuing from Example 4) Since we have only one relation '=' in SD_{inp}, the set $\mathcal{R}_{\mathcal{U}}$ only consists of the column r_1 : ($Expr_1.mode = Term.mode$). The training table $T_{Expr_0^1.mode}$ generated is:*

string	UO_{L-attr}			$\mathcal{R}_{\mathcal{U}}$	*class*
$w \in \mathcal{W}_{inp}$	$Expr_1.mode$	$Addop.op$	$Term.mode$	r_1	$Expr_0.mode$
w_1	*int*	*add*	*int*	*true*	*INT*
w_1	*int*	*sub*	*int*	*true*	*INT*
w_1	*real*	*sub*	*real*	*true*	*REAL*
w_2	*real*	*sub*	*int*	*false*	*REAL*
w_2	*real*	*add*	*int*	*false*	*REAL*
w_2	*int*	*add*	*real*	*false*	*REAL*
w_2	*real*	*add*	*real*	*true*	*REAL*

Based on the training tables constructed in this way the semantic functions are produced in the following form:

$$X_l^p.a = \begin{array}{l} \textbf{if } Test_{1,1} \wedge \cdots \wedge Test_{1,i_1} \textbf{ then } c_{i_1} \\ \textbf{else if } Test_{2,1} \wedge \cdots \wedge Test_{2,i_2} \textbf{ then } c_{i_2} \\ \quad \vdots \\ \textbf{otherwise } c_{i_s} \end{array}$$

where $c_{i_1}, \ldots, c_{i_s} \in \tau(X_l^p.a)$, and $Test_{k,j} (k = 1 \ldots s, j = i_1 \ldots i_s)$ is given in the form $(Column_{k,j} = v_j)$ with $Column_{k,j} \in UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}}$ and $v_j \in \tau(Column_{k,j})$.

(2) Non-enumerated case: if the domain of a target attribute occurrence $X_l^p.a$ is non-enumerated then the LAG system is going to infer a semantic function for it by employing the elements of $\mathcal{F}_{tar}$. If so, a slightly extended training table $T_{X_l^p.a}$ is produced:

$$\{string, target\} \cup UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}} \cup \mathcal{R}_{\mathcal{F}} \cup \{class\}.$$

The columns of the $string, UO_{L-attr}$, and $\mathcal{R}_{\mathcal{U}}$ are the same as those of the enumerated-typed target attribute occurrences. The main differences between the two cases surface in the columns of $\mathcal{R}_{\mathcal{F}}$, *target* and *class*.

The elements of the set $\mathcal{R}_{\mathcal{F}}$ are defined as a relation $(X_l^p.a = q)$, where q might be an attribute occurrence $u_k \in UO_{L-attr}$ or a satisfiable interpretation $f(u_1, \ldots, u_m)$ of $f : \tau(x_1) \times \cdots \times \tau(x_m) \rightarrow \tau(X_l^p.a)$. The values of $X_l^p.a$ computed during the parsing of the input strings make up the elements of the column *target*.

In addition, the elements of the column *class* $= \{+, -\}$ denote positive and negative examples. The positive examples of the training table will be elements of the set $\mathcal{E}_{X_l^p.a}$. The negative examples are generated from the positive ones by changing the elements of the column *target* with some randomly selected values of $\tau(X_l^p.a)$.

Example 6 *Let us consider an S-attributed ag AG_{ab}, which counts the number of letters in a string of the language a^*b^*.*

1, $S \rightarrow A\,B$	$S_0.n = A_1.n + B_2.n$	2, $A \rightarrow a\,A$	$A_0.n = inc(A_1.n)$
3, $A \rightarrow \lambda$	$A_0.n = 0$	4, $B \rightarrow b\,B$	$B_0.n = inc(B_1.n)$
5, $B \rightarrow \lambda$	$B_0.n = 0$		

Let us suppose that all of the semantic functions are unknown. The learning task will then defined as follows:

$$\begin{array}{ll} SD_{tar}: & \mathcal{T}_{tar} = \{\mathbb{N}\} \\ & \mathcal{F}_{tar} = \{inc^1, dec^1, +^2, -^2\}, \textit{ where} \\ & \quad inc: \mathbb{N} \rightarrow \mathbb{N} \qquad + : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ & \quad dec: \mathbb{N} \rightarrow \mathbb{N} \qquad - : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ AD_{tar}: & Syn = (n) \\ & Syn(S) = Syn(A) = Syn(B) = \{n\} \\ & \tau(n) = \mathbb{N} \\ \mathcal{W}_{inp} = & \{ab, aab, abb\} \\ TO = & \{S_0^1.n,\ A_0^2.n,\ A_0^3.n,\ B_0^4.n,\ B_0^5.n\} \end{array}$$

The training table $T_{A_0^2.n}$ for the target attribute occurrence $A_0^2.n$ consists of the following columns:

				$\mathcal{R}_\mathcal{F}$			
	$w \in \mathcal{W}_{inp}$	$A_0^2.n$	$A_1^2.n$	r_1	r_2	r_3	*class*
$UO_{L-attr} = \{A_1^2.n\}$	*ab*	1	0	*false*	*true*	*false*	+
$\mathcal{R}_\mathcal{U} = \emptyset$	*aab*	1	0	*false*	*true*	*false*	+
	aab	2	1	*false*	*true*	*false*	+
$r_1 : (A_0^2.n = A_1^2.n)$	*abb*	1	0	*false*	*true*	*false*	+
$r_2 : (A_0^2.n = inc(A_1^2.n))$	*ab*	2	0	*false*	*false*	*false*	−
$r_3 : (A_0^2.n = dec(A_1^2.n))$	*aab*	0	0	*true*	*false*	*false*	−
	aab	0	1	*false*	*false*	*true*	−
	abb	3	0	*false*	*false*	*false*	−

Similar training tables are generated for the target attribute occurrences $S_0^1.n$, $A_0^3.n$, $B_0^4.n$, $B_0^5.n$ as well.

Using the training tables structured in this way, the LAG system infers semantic functions which have the following form:

$$\begin{aligned} X_l^p.a = \; & \textbf{\textit{if}} \;\; Test_{1,1} \wedge \cdots \wedge Test_{1,i_1} \;\; \textbf{\textit{then}} \;\; q_{i_1} \\ & \textbf{\textit{else if}} \;\; Test_{2,1} \wedge \cdots \wedge Test_{2,i_2} \;\; \textbf{\textit{then}} \;\; q_{i_2} \\ & \quad \vdots \\ & \textbf{\textit{then}} \;\; q_{i_n} \end{aligned}$$

where $Test_{k,j}$ denotes the test $(Column_{k,j} = v_j)$ with $v_j \in \tau(Column_{k,j})$ and $Column_{k,j} \in UO_{L-attr} \cup \mathcal{R}_\mathcal{U}$. Here, q_{i_k} might be a function $f \in \mathcal{F}_{tar}$ or a used attribute occurrence $u \in UO_{L-attr}$.

3.3 Learning with the C4.5 system

The C4.5 system views the learning task described by the training table as a classification problem. The possible values of the target attribute occurrence make up the set of possible classes. The system constructs a classification model in the form of a decision tree or a set of decision rules. The LAG system formulates the semantic functions based on the decision rules.
The decision rules produced by the C4.5 system are represented as follows:

$$Rules_{X_l^p.a} = \left\{ \begin{array}{lll} Rule_1: & Column_{1,1} = v_1 & Column_{1,1} \in UO_{L-attr} \cup \mathcal{R}_\mathcal{U} \\ & \vdots & \\ & Column_{n_1} = v_{n_1} & Column_{n_1} \in UO_{L-attr} \cup \mathcal{R}_\mathcal{U} \\ & \rightarrow class \quad c_1 & c_1 \in \tau(X_l^p.a) \\ Rule_2: & \ldots & \\ \text{Default class:} & c_{default} & c_{default} \in \tau(X_l^p.a) \end{array} \right.$$

Example 7 *Based on the training table $T_{Expr_1^0.mode}$ given in the Example 5, the C4.5 system infers the following decision rules:*

$$Rules_{Expr_1^0.mode} = \left\{ \begin{array}{l} Rule_1: \;\; Expr_1.mode = real \rightarrow \;\; class\ REAL \\ Rule_2: \;\; Expr_1.mode = int \wedge \\ \qquad\quad Term.mode = int \rightarrow \;\; class\ INT \\ \\ Default\ class:\ REAL \end{array} \right.$$

Similar decision rules are inferred from the training table $T_{A_0^2.n}$ of the non-enumerated target attribute occurrence:

$$Rules_{A_0^2.n} = \begin{cases} Rule_1: & \big(A_0.n = inc(A_1.n)\big) = true & \rightarrow & class + \\ Rule_2: & \big(A_0.n = inc(A_1.n)\big) = false & \rightarrow & class - \\ Default\ class{:}+ & & & \end{cases}$$

3.4 Formulating semantic functions

First we simplify the set of rules learned by the C4.5 system then transform them into semantic functions.

(1) Enumerated case: The set of rules is reduced as follows:

$$Simplified\_Rules_{X_l^p.a} = \{r \in Rules_{X_l^p.a} \mid c_i \neq c_{default}\}.$$

This set is transformed to a semantic function of the form:

$$\begin{aligned} X_l^p.a = \ & \textbf{if}\ (Column_{1,1} = v_{i_1}) \wedge \cdots \wedge (Column_{1,n_1} = v_{1,n_1}) \\ & \textbf{then}\ c_1 \\ & \textbf{else}\ \textbf{if}\ (Column_{2,1} = v_{2,1}) \wedge \ldots \\ & \qquad \vdots \\ & \qquad \textbf{otherwise}\ c_{default} \end{aligned}$$

where $(Column_{i,j} = v_{i,j})$ occurs in the tests of $Simplified\_Rules_{X_l^p.a}$.

Example 8 *The semantic function formulated for the target attribute occurrence $Expr_0^1.mode$ is the following:*

$$\begin{aligned} Expr_0.mode = \ & \textbf{if}\ \ (Expr_1.mode = int) \wedge (Term.mode = int) \\ & \textbf{then}\ INT \\ & \textbf{else}\ REAL \end{aligned}$$

(2) Non-enumerated case: here, the rules inferred by C4.5 classify the examples into one of two classes: $+, -$. A rule is accepted iff it tests exactly one column of $\mathcal{R}_{\mathcal{F}}$.

The set $Simplified\_Rules_{X_l^p.a}$ is constructed in the following way:

$$Simplified\_Rules_{X_l^p.a} = \left\{ r_i \in Rules_{X_l^p.a} \;\middle|\; \begin{array}{c} (c_i = +)\,,\ and\ for\ exactly\ one\ k: \\ Column_{i,k} = (X_l^p.a = q_i) \in \mathcal{R}_{\mathcal{F}} \\ with\ (Column_{i,k} = true) \end{array} \right\}$$

This set is transformed to a semantic function in the form:

$$\begin{aligned} X_l^p.a = \ & \textbf{if}\ (Column_{1,1} = v_{i_1}) \wedge \ldots (Column_{1,n_1} = v_{1,n_1}) \\ & \textbf{then}\ q_1 \\ & \textbf{else}\ \textbf{if}\ (Column_{2,1} = v_{2,1}) \wedge \ldots \\ & \qquad \vdots \\ & \qquad \textbf{then}\ q_n \\ & \qquad \textbf{otherwise}\ \text{WARNING} \end{aligned}$$

where $Column_{i,j}$ are the tests of $Simplified\_Rules_{X_l^p.a}$, such that $Column_{i,j} \in UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}}$, while q_i is a function and $(X_l^p.a = q_i)$ is

among the tests of $Simplified\_Rules_{X_l^p.a}$. (Note: if during the execution of the generated *ag* for a given input none of the conditions in the above semantic function are fulfilled, a warning message is induced for the user. This message indicates that the inferred semantic function is not applicable for that input. If the $Simplified\_Rules_{X_l^p.a} = \emptyset$, then it then means that the LAG system was not able to learn semantic function for $X_l^p.a$.)

Example 9 *The decision rules for the target attribute occurrence $A_0^2.n$ are simplified in the following way:*

$$Simplified\_Rules_{A_0^2.n} = \{Rule_1 : \ (A_0.n = inc(A_1.n)) = true \rightarrow \ class +\}$$

Since the simplified set of rules consists of a single rule not containing any tests over the elements of columns in $UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}}$ and the test of this rule is an element of $\mathcal{R}_{\mathcal{F}}$, the generated semantic function of $A_0^2.n$ is

$$A_0.n = inc(A_1.n)$$

which is the correct solution.

Within the non-enumerated learning there is a special case where a constant value should be assigned to the target attribute occurrence. In this case a semantic function

$$X_l^p.a = c, \text{ where } c \in \tau(X_l^p.a)$$

is generated automatically based on a preliminary check of positive examples.

4 Application of the LAG method in NLP

4.1 Part-of-Speech Tagging Problem

Research into both text and spoken language understanding is significantly helped by investigating those phenomena that occur in actual language use.
The first stage of the investigation is to assign part of speech (PoS) tags to every word representing its syntactic category and morphological properties based on large corpora. The **corpus** is an archive of annotated words including their morphological properties as codes called **tag**. Annotating a given text is a far from trivial task since the words often belong to several syntactic categories or morphological classes in different contexts (e.g. the Hungarian word "*múlt*"[4] might be annotated by a verbal, noun or adjectival tag).
The task of a PoS tagger (morphological disambiguater) is to automatically select the appropriate PoS annotation in a given context where possible. In principle there are two main approach for automatic part-of-speech tagging:

- the probabilistic one which normally uses Hidden Markov Models and
- the rule-based one which normally uses linguistic rules.

[4] *múlt (verb) – passed (Perfect 'pass')*
múlt (noun) – past
múlt (adjective) – past, last

In this section we infer rules for a rule-based tagger with the aid of the LAG method. We specify an *ag* which detects correspondences among the parts of the sentences such as predicate phrase and subject phrase. Using this structural information during the learning process, the LAG system produces disambiguater rules for each ambiguous class.

4.2 The initial data set

Our Hungarian corpus is the morphologically annotated translation of George Orwell's novel *1984*. The first tagged version of this corpus was produced by the MULTEXT-East project ([7]). The corpus includes approximately 100 000 words including punctuation characters. The novel consists of four chapters where the first two served as training data for the learning process while chapters 3 and 4 were used as test data.
The most widely used encoding is the Morpho-Syntactical Description (MSD, [7]). Unfortunately it associates too many different classes with the Hungarian language. E.g. based on its stems, a *noun* could be annotated with 1324 different MSD codes. In order to reduce the number of MSD classes the CTAG encoding scheme (Corpus Tagging,[16]) was employed, which has just 120 word tags, 4 punctuation tags and 1 tag for *unknown* words. Table 2 lists the distribution of the ambiguous classes whose instances occur over 100 times in the training and test data.

Table 2: The most frequently ambiguous classes and their cardinalities

Classes	Occurrence		Classes	Occurrence	
	Training data	Test data		Training data	Test data
asn,vmis3s	490	182	nsn,rgn,rp	112	46
cp,rg	880	294	psn,rp	142	57
cp,rg,vmip3s	247	125	psn,t	1867	620
cp,rp	334	149	pso,rg	217	85
cp,vmis3s	113	38	rg,rp	150	59
ms,t	751	222	rg,st	285	100
nsn,psn	111	52			

For instance, a word which belongs to the ambiguous class $[asn, vmis3s]$ could be annotated as a *nominative, singular adjective* or as a *verb in past tense, 3rd person singular*. In another ambiguous case, the $[psn, t]$ stands for the word '*az*', which could be annotated as a *singular pronoun, nominative case*[5] or as an *article*[6]. (A brief description of the corpus tags is given in the Appendix B.)
Besides the tags there is an identifier associated with every sentence which shows the location of a sentence in the original text, namely *Orwell, Hungarian translation, 1st chapter, 2nd section, 1st paragraph, 1st sentence* is

[5] 'az' – the
[6] 'az' – that

'0hu.1.2.1.1',"*Derült*, hideg áprilisi nap volt,
az órák éppen tizenhármat ütöttek." [7]

This sentence is annotated as follows:

'0hu.1.2.1.1',(*asn*, [*asn*, *vmis3s*]), wpunct, asn, asn, nsn, vmis3s, wpunct,
(*t*, [*psn*, *t*]), npn, rg, msa, vmis3p, spunct

In the sequence of corpus tags an ambiguous case is denoted by a pair given in round brackets. The second component is the set of possible tags of the word, while the first component shows its correct tag in the given sentence. Using the sequences of corpus tags during the learning process we can infer context rules which describe general regularities among the morpho-syntactical categories of the language.
Each ambiguous class is dealt with as an independent learning task so we generate an initial input set for each one, based on sequences of the corpus tags. Each element of these input sets is structured as follows:

Sentence_ID, before$_1$, ..., before$_7$, after$_1$, ..., after$_7$, *correct_ctag*

where *correct_ctag* denotes the observed morpho-syntactical category of the word in the given sentence. In addition, we consider 7 corpus tags *before* and *after* the ambiguous case. (Here: we denote the blanks with *xxx* when this 7-sized window extends over the beginning and the end of a sentence).
Continuing our example, the following tuples are added to the input set $\mathcal{W}_{asn,vmis3s}$ and $\mathcal{W}_{psn,t}$ of the ambiguous class $[asn, vmis3s]$ and $[psn, t]$, respectively:

'0hu.1.2.1.1', xxx, xxx, xxx, xxx, xxx, xxx, xxx,
wpunct, asn, asn, nsn, vmis3s, wpunct, t, *asn*

'0hu.1.2.1.1', wpunct, vmis3s, nsn, asn, asn, wpunct, asn,
npn, rg, msa, vmis3p, spunct , xxx, xxx, *t*

Using these sets of sequences the C4.5 system can infer disambiguater rules for each ambiguous class, i.e. produce a set of decision rules for the class $[asn, vmis3s]$ such that:

Rule1: $before_1 = t \rightarrow$ *class* ***asn***
Rule 2: $after_1 = npn \rightarrow$ *class* ***asn***

⋮

Rule 36: $after_1 = spunct \rightarrow$ *class* ***vmis3s***
Rule 37: $before_1 = nsa \rightarrow$ *class* ***vmis3s***

Default class: ***vmis3s***

In order to generate more effective rules the LAG method has been designed to recognize structural coherences in the sentences via an *ag* and extend the input of C4.5 with them.

4.3 Description of the learning task

The *ag* AG_{ctag} introduced here, detects parts of sentences and phrases in ambiguous cases.
The parts of sentences can be derived from the corpus tags, which refer to the suffixes of the words as well. The suffix determines the role of a word in a sentence.

[7] It was a bright, cold day in April and the clocks were striking thirteen.

We separate the corpus tags into *groups* based on the role they play in a sentence such as *predicate, subject, object, attribute, dative adverb, other adverb.* The rest of the sentence elements are denoted with the value *other*. Furthermore, the value *none* is generated for the case of *xxx* tags.

The phrases, called *syntagmas*, describe relations among the parts of sentences like the *predicate syntagma*, where the predicate and subject are related, or the *accusative syntagma*, where the predicate and object are related. It is clear that the identification of a *syntagma* depends on the attribute *group*.

Furthermore, our experiments show that in most cases the choice of the correct corpus tag of a word is influenced only by its neighboring tags. Hence, we use a simplified *ag* AG_{ctag} as background knowledge which deals only with tags next to the ambiguous case (size of window = 1) and it detects a syntagma among the tags after it. (A part of the ag can be found in the Appendix C.)

G_{ctag} 1 : $Ctag\_Sentence \rightarrow$
Sentence_ID "," BeforeCtags "," AfterCtags Ctag_Sentence
2 : $Ctag\_Sentence \rightarrow \lambda$
⋮

SD_{ctag} $\mathcal{T}_{ctag} = \left\{ \begin{array}{l} \text{CTAG} = \{asn, asnx, \ldots, wmis3s, spunct, wpunct \ldots\} \\ \text{GROUP} = \{Pred, Subj, Acc, AdvDat, AdvOth, Att, Other, None\} \\ \text{SYNTAGMA} = \{PredSynt, SubjSynt, AccSynt, AdvDatSynt, \\ \qquad AdvOthSynt, AttSynt, OtherSynt, NoneSynt\} \end{array} \right\}$
$\mathcal{F}_{ctag} = \{=^2\}$ where, $=$ is the identity relation

AD_{ctag} $Inh = \emptyset$
$Syn = \{ctag_1, group_1, syntagma\}$
$Syn(BeforeCtags) = \{ctag_1, group_1\}$
$Syn(AfterCtags) = \{ctag_1, group_1, syntagma\}$

$\tau(ctag_1) = \text{CTAG}$
$\tau(group_1) = \text{GROUP}$
$\tau(syntagma) = \text{SYNTAGMA}$

In order to choose the contextually correct tag in an ambiguous case, a synthesized attribute *correct_ctag* is associated with the start symbol *Ctag_Sentence*. Its semantic function is unknown, so the learning task is described as follows:

the semantic domain SD_{tar}	$\mathcal{T}_{tar} = \{\text{CTAG}\}$
	$\mathcal{F}_{tar} = \emptyset$
the attribute description AD_{tar}	$Syn = \{correct\_ctag\}$
	$Syn(Ctag\_Sentence) = \{correct\_ctag\}$
	$\tau(correct\_ctag) = \text{CTAG}$
R_{tar}	$R(1) = \emptyset$
target attribute occurrence	$TO(1) = \{Ctag\_Sentence_0^1.correct\_ctag\}$
input strings i.e.	$\mathcal{W}_{inp} = \mathcal{W}_{asn,vmis3s}$

The learning concept is inferred by the LAG method introduced in the Section 3.

4.4 Generation of the training examples

We build the ddt_s for every given sequence s of corpus tags for an ambiguous class. Recalling that the values of the target attribute occurrence *correct_ctag* are defined in advance in the training corpus, the question IQ is not used during the tree traversals.
For instance, in the case of the ambiguous class $(asn, vmis3s)$ given the set of input sequences of $\mathcal{W}_{asn,vmis3s}$:

```
Ohu.1.2.1.1,   xxx, xxx, xxx, xxx, xxx, xxx, xxx
               wpunct, asn, asn, nsn, vmis3s, wpunct, t, asn
Ohu.1.2.5.5,   t, cp, wpunct, vmn, vmis3s, rg, i, nso
               rg, vmip3p, rq, t, nsa, spunct, asn
...
Ohu.2.11.40.5, rg, rg, spunct, rp, vmcp3s, nsax, cp
               pso, ms, nsa, spunct, xxx, xxx, xxx, vmis3s
Ohu.2.11.40.5, nsa, asn, asn, xxx, xxx, xxx, xxx
               nso, wpunct, cp, nsax, vmcp3s, rp, spunct, vmis3s
```

The training example set $\mathcal{E}_{asn,vmis3s}$ generated in this case is:

Sentence_ID	UO_{L-attr}					*class*
	u_1	u_2	u_3	u_4	u_5	
Ohu.1.2.1.1	xxx	none	wpunct	Oth	AttSynt	*asn*
Ohu.1.2.5.5	t	Oth	nso	AdvOth	AdvOthSynt	*asn*
...						
Ohu.2.11.40.5	rg	Oth	pso	AdvOth	noneSynt	*vmis3s*
Ohu.2.11.40.5	nsa	Acc	nso	AdvOth	AdvOthSynt	*vmis3s*

$u_1 : BeforeCtags.ctag_1$ $u_2 : AfterCtags.ctag_1$
$u_3 : BeforeCtags.group_1$ $u_4 : AfterCtags.group_1$
$u_5 : AfterCtags.syntagma$

$class : Ctag\_Sentences.correct\_tag$.

4.5 Preparation of the training tables

Since the target attribute occurrence *Ctag_Sentences.correct_tag* is enumerated-typed, the training table consists of the columns

$\{Sentence\_ID\} \cup UO_{L-attr} \cup \mathcal{R}_{\mathcal{U}} \cup \{correct\_ctag\}$

where $\mathcal{R}_{\mathcal{U}}$ contains the relations

r_1 : *(BeforeCtags.ctag*$_1$ = *AfterCtags.ctag*$_1$*)*
r_2 : *(BeforeCtags.group*$_1$ = *AfterCtags.group*$_1$*)*

Hence, the training table $T_{asn,wmis3s}$ is constructed as follows:

Sentence_ID	UO_{L-attr}					$\mathcal{R}_{\mathcal{U}}$		*class*
	u_1	u_2	u_3	u_4	u_5	r_1	r_2	
Ohu.1.2.1.1	xxx	None	wpunct	Oth	AttSynt	false	false	*asn*
Ohu.1.2.5.5	t	Oth	nso	AdvOth	AdvOthSynt	false	false	*asn*
...								
Ohu.2.11.40.5	rg	Oth	pso	AdvOth	noneSynt	false	false	*vmis3s*
Ohu.2.11.40.5	nsa	Acc	nso	AdvOth	AdvOthSynt	false	false	*vmis3s*

4.6 Inferred context rules

The sets of decision rules are inferred based on the training tables, i.e.:

$$Rules_{asn,vmis3s}\begin{cases} Rule_1: & BeforeCtags.group_1 = Acc & \rightarrow class\ vmis3s \\ Rule_2: & AfterCtags.group_1 = Oth & \rightarrow class\ vmis3s \\ Rule_3: & AfterCtags.ctag_1 = pso & \\ & AfterCtags.syntagma = AttSynt & \rightarrow class\ asn \\ \vdots & & \\ Rule_{42}: & BeforeCtags.ctag_1 = wpunct & \\ & AfterCtags.syntagma = AttSynt & \rightarrow class\ asn \\ Default\ class:\ vmis3s & & \end{cases}$$

The rule sets are reduced and converted to the form of semantic functions. Let us take for instance the case of the ambiguous class *asn, vmis3s*:

$$\begin{aligned} Ctags\_Sentences.correct\_tag = \textbf{if}\ & (BeforeCtags.ctag_1 = wpunct)\ and \\ & (AfterCtags.syntagma = AttSynt) \\ \textbf{then}\ & asn \\ \textbf{else}\ & \textbf{if}\ (AfterCtags.ctag_1 = pso)\ and \\ & (AfterCtags.syntagma = AttSynt) \\ & \textbf{then} \\ & \ldots \\ & \textbf{else}\ vmis3s \end{aligned}$$

Since disambiguater rules for any ambiguity can be inferred this way the above method is a useful tool for a PoS tagger system.

5 Comparison of the results of C4.5 and LAG

In the following table we compare the accuracy of the disambiguater rules achieved by C4.5 and LAG based on the corpus of Orwell's novel. The accuracy of the rules is tested using the chapters 3 and 4 of the novel, these chapters not being used during training process.

Table 3 shows the error numbers and error percentages of the decision rule sets generated for the most frequent ambiguous classes. The rules inferred by the C4.5 system are based on the sequences of corpus tags (see p. 297). The LAG system, however, creates its results by the means of the training sequences which are augmented with structural information detected by the *ag* given in Section 4.3 In the column *Mark*, the sign

"+" denotes those classes where the use of LAG yields only minor improvements, and

"++" means significant improvements produced by employing the LAG method compared to C4.5.

The results show the accuracy of the inferred rules is improved if an *ag* as background knowledge is utilised during the learning process.

Table 3: *The comparison of the C4.5 and LAG system*

Ambiguity classes	Results by C4.5				Results by LAG				Mark
	training data		test data		training data		test data		
	#err	err %	#err	err %	#err	err %	#err	err %	
asn-vmis3s	39	8.0 %	15	8.2 %	34	6.9 %	11	6.0 %	+
cp-rg	142	16.1 %	72	24.5%	136	15.5 %	69	23.5%	+
cp-rg-vmip3s	14	5.7 %	31	24.8%	11	4.5 %	23	18.4%	+
cp-rp	41	12.3 %	16	10.7%	10	3.0 %	11	7.4 %	++
cp-vmis3s	2	1.8 %	0	0.0 %	0	0.0 %	0	0.0 %	+
nsn-psn	24	21.6 %	16	30.8%	4	3.6 %	6	11.5%	++
psn-rp	9	6.3 %	3	5.3 %	6	4.2 %	3	5.3 %	+
psn-t	28	1.5 %	17	2.7 %	25	1.3 %	15	2.4 %	+
pso-rg	73	33.6 %	34	40.0%	25	11.5 %	11	12.9%	++
rg-rp	57	38.0 %	15	25.4%	31	20.7 %	8	13.6%	++
rg-st	104	36.5 %	44	44.0%	62	21.8 %	35	35.0%	++

6 Summary

In this paper we investigated the specification of *ags* from the viewpoint of inductive learning. We described a learning task for inferring semantic functions of a partially defined *ag* and introduced an inductive learning method for solving this task. In the learning approach of the LAG system a number of similarities exist between it and ILP methods. These similarities arise from the close connection between logic programs and *ags*. The LAG method infers semantic functions for enumerated and non-enumerated attribute occurrences of an L-attributed or S-attributed grammar. During the learning process it derives the training examples from input strings with the help of background knowledge. The background knowledge given as an *ag* is employed in the preparation the training tables for the target attribute occurrences. Using the training tables the C4.5 system produces decision rules which are then converted to the form of semantic functions.

We plan to increase the efficiency of the LAG method by reducing the restrictions related to background knowledge, i.e. extend the the algorithm to more complex *ags* than the S-attributed and L-attributed ones. Moreover, we would like to develop a more precise algorithm for the non-enumerated cases.

As regards to the PoS tagging application we would also like improve the background attribute grammar to better describe the features of the Hungarian language.

References

[1] ALBLAS, H.: Introduction to Attribute Grammars. Springer Verlag LNCS 545, p.1–16, 1991.

[2] ALEXIN, Z., ZVADA, SZ., GYIMÓTHY, T.: Application of AGLEARN on Hungarian Part-of-Speech Tagging. In Proceedings of the Second Workshop on Attribute Grammars and their Applications (WAGA'99), Amsterdam, The Netherlands. p.133–152, INRIA Rocquencourt, 1999.

[3] CUSSENS, J.: Part-of-Speech Tagging Using Progol. In Proceedings of the Seventh International Workshop on Inductive Logic Programming (ILP97), Prague, Czech Republic, Springer Verlag LNAI 1297, p.37–44, 1997.

[4] DERANSART, P., JOURDAN, M., LORHO, B.: Attribute Grammars - Definitions, Systems and Bibliography. Springer Verlag LNCS 323, 1988.

[5] DERANSART, P., MALUSZYŃSKI, J.: Relating Logic Programs and Attribute Grammars. Journal of Logic Programming 2, p.119–156, 1985.

[6] DERANSART, P., MALUSZYŃSKI, J. : A Grammatical View of Logic Programming. MIT Press, 1993.

[7] ERJAVEC, T., MONACHINI, M.: Specification and Notation for Lexicon Encoding. Copernicus Project 106 "MULTEXT-EAST", Deliverable D1.1 F (Final Report). 1997

[8] GYIMÓTHY, T., HORVÁTH, T.: Learning Semantic Functions of Attribute Grammars. Nordic Journal of Computing 4, p.287–302, 1997.

[9] HORVÁTH, T., ALEXIN, Z., GYIMÓTHY, T., WROBEL, S.: Application of Different Learning Methods to Hungarian Part-of-speech Tagging. In Proceedings of Ninth Workshop on Inductive Logic Programming (ILP99), Bled, Slovenia, Springer Verlag LNAI 1634, p.128–139, 1999.

[10] KASTENS, U.: Ordered Attribute Grammars. Acta Informatica 13, p.229–256, 1980.

[11] KNUTH, D.E.: Semantics of Context-Free Languages. Mathematical Systems Theory 2(2), p.127–145, 1968. Correction: Mathematical Systems Theory 5(1), p.95–96, 1971.

[12] LAVRAČ, N., DŽEROSKI, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York, 1994.

[13] MITCHELL, T. Machine Learning. McGraw-Hill, 1997.

[14] MUGGLETON, S.: Inductive Logic Programming. Academic Press, London, 1992.

[15] MUGGLETON, S., DE RAEDT, L.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 19/20, p.629–679, 1994.

[16] ORAVECZ, CS.: Part-of-Speech Tagging in the Hungarian National Corpus – a Case Study. Research Institute for Linguistics of the Hungarian Academy of Sciences, 1998.

[17] PAAKKI, J.: A Logic-Based Modification of Attribute Grammars for Practical Compiler Writing. In Proceedings of the Seventh International Conference on Logic Programming (D.H.D. Warren, P. Szeredi, eds.), Jerusalem, p.203–217. MIT Press, 1990.

[18] QUINLAN, J.R. Induction of decision trees. Machine Learning, 1(1), p.81–106, 1986.

[19] QUINLAN, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

A Appendix

The corpus tags used in the Hungarian translation of the Orwell novel '1984':

ASN, ASNX, ASNY, ASA, ASAX, ASAY, ASD, ASDX, ASDY, ASO, ASOX, ASOY, APN, APNX, APNY, APA, APAX, APAY, APD, APDX, APDY, APO, APOX, APOY, MP, MPX, MPY, MS, MSX, MSY, MD, I, CP, NSN, NSNX, NSNY, NSA, NSAX, NSAY, NSD, NSDX, NSDY, NSO, NSOX, NSOY, NPN, NPNX, NPNY, NPA, NPAX, NPAY, NPD, NPDX, NPDY, NPO, NPOX, NPOY, PSN, PSNX, PSNY, PSA, PSAX, PSAY, PSD, PSDX, PSDY, PSO, PSOX, PSOY, PN, PPNX,PPNY, PPA, PPAX, PPAY, PPD, PPDX, PPDY, PPO, PPOX, PPOY, RG, RO, RP,RQ, RV, ST, T, VA, VMCP1S, VMCP1P, VMCP2, VMCP2S, VMCP2P, VMCP3S, VMCP3P, VMIP1S, VMIP1P, VMIP2, VMIP2P, VMIP2S, VMIP3S, VMIP3P, VMIS1S, VMIS1P, VMIS2, VMIS2P, VMIS2S, VMIS3S, VMIS3P, VMMP1S, VMMP1P, VMMP2, VMMP2P,VMMP2S, VMMP3S, VMMP3P, VMN, CPUNCT, SPUNCT, WPUNCT, UNKNOWN, X,Y

B Appendix

Here we briefly describe the above mentioned corpus tags. The first letter of each *ctag* stands for the *category* of the related words:

Ctag	Category	Ctag	Category
A	Adjective	R	Adverb
CP	Conjuntion	ST	Postposition
I	Interjection	T	Article
M	Numeral	V	Verb
N	Noun	X	Residual
P	Pronoun	Y	Abbreviation
SPUNCT	sent. punct.	CPUNCT	closing punct.
WPUNCT	wordpunct.		

Then the tags are constructed in the following way:

After A, N, M and P : The second letter after A, N, M and P denotes the *cardinality* while the third one is related to the *cases*, and the fourth letter refers to the *possessive* cases:

Position 2	Position 3	Position 4
	N nominative	
S singular	A accusative	X/M.X
P plural	D dative	Y/M.Y
	O other	

After V : in the case of verbs the situation is the following:

Position 2	Position 3 modes	Position 4 tenses	Position 5 person	Position 6
	I indicative			
M main	M imperative	P present	1	S single
A auxiliary	C conditional	S past	2	P prural
	N infinitive		3	

Other combination : MD numeral digit
RG general adverb
RP verbal participle
RV present partiple
RQ interrogative clitic

C Appendix

A part of the background ag AG_{ctag} defined for PoS tagging problem is:

```
Ctags_Sentences → Sentence_ID "," BeforeCtags "," AfterCtags Sentences
Ctags_Sentences → λ
...
AfterCtags → Acc_Group "," Synt_Acc          syntagma = Synt_Acc.syntagma
                                             ctag = Acc_Group.ctag
                                             group = Acc

AfterCtags → Pred_Group "," Synt_Pred        syntagma = Synt_Pred.syntagma
                                             ctag = Pred_Group.ctag
                                             group = Pred
...
Synt_Acc → Pred_Group "," Ctags              syntagma= AccSynt
Synt_Acc → NonPred_Group "," Synt_Acc        syntagma= Synt_Acc.syntagma

Synt_AdvDat → Pred_Group "," Ctags           syntagma= AdvDatSynt
Synt_AdvDat → NonPred_Group "," Synt_AdvDat  syntagma= Synt_AdvDat.syntagma

Synt_Subj → Pred_Group "," Ctags             syntagma= SubjSynt
Synt_Subj → NonPred_Group "," Synt_Subj      syntagma= Synt_Subj.syntagma
...
Ctags → 'asn'
Ctags → 'asnx'
...
```