The Hierarchy of LR-Attributed Grammars

Rieks op den Akker
Department of Computer Science, University of Twente
P.O. Box 217, NL-7500 AE Enschede, the Netherlands
e-mail: infrieks@cs.utwente.nl

Bořivoj Melichar
Department of Computers, Czech Technical University
Karlovo náměstí 13, 121 35 Prague, Czechoslovakia

Jorma Tarhio
Department of Computer Science, University of Helsinki
Teollisuuskatu 23, SF-00510 Helsinki, Finland
e-mail: tarhio@cs.Helsinki.fi

Abstract

The problem of attribute evaluation during LR parsing is considered. Several definitions of LR-attributed grammars are presented. Relations of corresponding attribute grammar classes are analysed. Also the relations between LR-attributed grammars and LL-attributed grammars and between LR-attributed grammars and a class of one-pass attributed grammars based on left-corner grammars are considered.

1. Introduction

In one-pass compilation, based on an attribute grammar, parsing and semantic analysis have been merged so that all attributes are evaluated in conjunction with the parsing process. The advantages of this approach become apparent in time and space efficiency, because the derivation tree need not be stored. Moreover, attribute values may be used to solve parse conflicts. As might be expected, the class of grammars suitable for one-pass compilation is restricted in semantical sense, but it is, however, large enough from the practical point of view.

In one-pass compilation, the evaluation strategy depends on the parsing method used, because the order in which the productions are recognized affects the evaluation order. For example, LL parsing allows a full top-down evaluation traversal (a depth-first, left-to-right walk through a derivation tree), but LR parsing allows only a bottom-up traversal, in which every node is visited once. This may suggest that it would be possible to evaluate more attribute grammars during LL parsing than during LR parsing, though LL grammars are syntactically a subclass of LR grammars. However, Brosgol [Bro74] shows how LL parsing can be simulated in LR parsing, and therefore it is possible to evaluate in conjunction with LR

parsing every attribute grammar that can be evaluated during LL parsing.

S-attributed grammars [LRS74] having only synthesized attributes can be evaluated in a natural way during LR parsing. In principle, it is possible to describe with mere synthesized attributes the same translations as with inherited and synthesized attributes [Knu68], but this can lead to unreadable and very complicated semantic rules and complicated data structures as attribute value domains. Thus in practice, it is necessary to have inherited attributes or some alternative formalism for inherited information.

Evaluation of inherited attributes is a problem during LR parsing because of insufficient information of the upper part of the derivation tree. Several strategies have been proposed to deal with this problem, starting from the pioneering work of Watt [Wat77]. So there are several algorithms for the attribute evaluator and its construction. Methods for one-pass (i.e. parse time) evaluation of attribute grammars based on LR grammars can be classified as follows: the class of attribute grammars covered, the construction of the evaluator and ease of implementation. Unfortunately, the variant methods are described in different formalisms and so it is sometimes difficult to compare them.

Section 3 of this paper presents a more precise definition of attribute evaluation during parsing. In section 4 attribute grammar classes are defined for all known evaluation strategies during LR parsing. In section 5 we will compare the classes of LR-attributed grammars covered by the definitions presented in the preceding sections. Section 6 considers the class of LC-attributed grammars, one-pass attributed grammars based on left-corner grammars. Basic concepts and notations are presented in the following section.

2. Basic concepts and notations

This section presents some basic concepts and notations concerning LR parsing, attribute grammars and evaluation of attributes during LR parsing.

2.1. Parsing

We introduce basic concepts of LR parsing following mainly Aho et al. [ASU86] and Aho and Ullman [AhU72]. We denote a *context-free grammar* by a four-tuple $G=(N,\Sigma,P,Z)$. The sets N of *nonterminals* and Σ of *terminals* form the *vocabulary* $V=N\cup\Sigma$. Elements of V are called *grammar symbols* and they are denoted by roman capitals at the end of the alphabet. A, B, ... denote elements of N. The letters a, b, ... denote elements of Σ, and u, v and w denote elements of Σ^*. Greek letters $\alpha,\beta,\cdots$ are used to denote the elements of V^*, the set of strings over V. The symbol ε denotes the empty string. $P\subseteq N\times V^*$ is the set of productions. A *production* $p\in P$ is written $X\to\alpha$, where $X\in N$ is called the *left-hand side* of p and $\alpha\in V^*$ is called the *right-hand side* of p. The symbol $Z\in N$ is the *start symbol* which has only one production and which does not appear on the right-hand side of any production. We assume that every grammar is *reduced*, i.e. V does not contain useless symbols and P does not contain useless productions.

The derivation relation $\Rightarrow$ is defined as follows. For any $\alpha,\beta\in V^*$, $\alpha\Rightarrow\beta$ if $\alpha=\gamma_1 A\gamma_2$, $\beta=\gamma_1\gamma_0\gamma_2$ and $A\to\gamma_0\in P$ where $A\in N$ and $\gamma_0,\gamma_1,\gamma_2\in V^*$. If $\gamma_2\in\Sigma^*$ we write $\alpha\Rightarrow_{rm}\beta$. If $\alpha\Rightarrow^*_{rm}\beta$, we say that β is obtained by a rightmost derivation from α ($\Rightarrow^*$ denotes the reflexive and transitive closure of the relation $\Rightarrow$). Strings in V^* obtained by a rightmost derivation from the start symbol Z are called *right sentential forms*. A sequence $p_1,p_2,...,p_k$ of productions is called a *right parse* of β to α in the grammar G, if β is obtained by a

rightmost derivation from α by applying the productions in the reverse order. The set of terminal strings derived from the start symbol Z is denoted by $L(G)$.
A nonterminal A is *left-recursive* if $A \Rightarrow_{rm}^{+} A\alpha$ for some $\alpha \in V^*$. If for some $A \in N$ there is a derivation $A \Rightarrow A_1\alpha_1 \Rightarrow \cdots \Rightarrow A_n\alpha_n \cdots \alpha_1, (n \geq 1)$ with $A_n = A$, then the productions $A_i \rightarrow A_{i+1}\alpha_{i+1}$ are called *left-recursive* productions of the grammar.
The *derivation tree* for a terminal string $w \in L(G)$ is a finite ordered tree in which every node is labeled by $X \in V$ or by ε. If a node n labeled as X has sons $n_1, n_2, ..., n_m$ labeled as $X_1, X_2, ..., X_m$, then $X \rightarrow X_1 \cdots X_m$ must be a production in P. The labels of the leaves of the tree for w, concatenated from left to right, form w.

We define the set $First_k(\gamma)$ for $\gamma \in V^*$, as follows (the length of a string $\alpha \in V^*$ is denoted by $|\alpha|$): $First_k(\gamma) = \{ x \in \Sigma^* \mid \gamma \Rightarrow^* x\alpha$ and $|x| = k$ or $\gamma \Rightarrow^* x$ and $|x| < k \}$.

Definition 2.1 We say that G is an *LR (k) grammar*, $k \geq 0$, if the three conditions

(1) $Z \Rightarrow_{rm}^{*} \alpha Aw \Rightarrow_{rm} \alpha\beta w$,

(2) $Z \Rightarrow_{rm}^{*} \gamma Bx \Rightarrow_{rm} \alpha\beta y$, and

(3) $First_k(w) = First_k(y)$

imply that $\alpha Ay = \gamma Bx$.

A production $A \rightarrow \beta$ of G is said to *satisfy the LR (k) condition* if the conditions (1), (2) and (3) above always imply $\alpha Ay = \gamma Bx$.

An *LR parser* is an algorithm that produces for an input string its right parse to the start symbol or reports an error if the string is not in $L(G)$. An LR parser scans the input from left to right without any backtracking. We use the following model for an LR parser. The parser has an *input buffer*, a *parsing stack* and a *parsing table*. The input in the buffer consists of the string to be parsed followed by the endmarker \$. Initially, the stack is empty. For evaluation purposes we assume that the parsing stack is implemented as an array, i.e. we can access every element of the stack. At any moment of parsing, the sequence of grammar symbols α on the stack and the input w not yet consumed form a right sentential form αw of G. The pair (α, w) is called a *parsing situation*. Besides the grammar symbols, an LR parser places special *state symbols* on the stack. The next move of the parser is determined by the next k symbols of the current input and the state symbol on the top of the stack. Information concerning the move to be made is stored in the parsing table, which consists of two parts: an *action table* and a *goto table*. The move can be one of the four types:

1. **Shift**. The current input symbol is shifted and pushed on the stack and the state symbol determined by the goto table is placed on the top of the stack.
2. **Reduce** by a production $A \rightarrow \alpha$. First $2 \times |\alpha|$ symbols are popped off the stack. The goto table gives the next state symbol s according to the state symbol on the top of the stack and the nonterminal A. The nonterminal A and the symbol s are pushed on the stack.
3. **Accept**. Parsing has been completed successfully.
4. **Error**. The input string does not belong to $L(G)$. An error recovery routine is called.

We give the construction of the LR(k) parsing table for a given cfg G.

An *LR(k) item* of a context-free grammar G is $[A \rightarrow \alpha \bullet \beta, u]$ where $A \rightarrow \alpha\beta$ is a rule of G and u is a lookahead string. Its first component $A \rightarrow \alpha \bullet \beta$ is called the *core* of the item.

The *closure* of a set of LR(k) items I is a set of items $CLOSURE(I)$, defined as follows:

1. Every item in I is in $CLOSURE(I)$.

2. If $[A \rightarrow \alpha \bullet B\beta, u]$ is in *CLOSURE* (I), $B \rightarrow \gamma$ is a rule of G and v is in $First_k(\beta u)$, then add the item $[B \rightarrow \bullet\gamma, v]$ to I, if it is not already there. We say that the latter item is *directly derived from* the item $[A \rightarrow \alpha \bullet B\beta, u]$.

The set of items *GOTO* (I, X) for a set of items I and a grammar symbol X is defined to be the closure of the set of all items $[A \rightarrow \alpha X \bullet \beta, u]$ such that $[A \rightarrow \alpha \bullet X\beta, u]$ is in I. The set of all items $[A \rightarrow \alpha X \bullet \beta, u]$ is called *BASIS* of *GOTO* (I, X).

Using *CLOSURE* and *GOTO* operations, the collection of sets of LR(k) items, $\{I_0, I_1, ..., I_n\}$, is constructed starting from the initial set of LR(k) items, I_0=*CLOSURE*$([Z' \rightarrow \bullet Z, \varepsilon])$. These sets I_j correspond to the states of the LR(k)-automaton. The state symbols of the LR parser represent these states. Notice that every item in I=*CLOSURE* (*BASIS* (I)) corresponds to the last element of a sequence $A_0 \rightarrow \alpha \bullet A_1\beta_1, A_1 \rightarrow \bullet A_2\beta_2, ..., A_{n-1} \rightarrow \bullet A_n\beta_n$, where $A_0 \rightarrow \alpha \bullet A_1\beta_1$ is the core of an item in *BASIS* (I).

The action table f is computed as follows:

1. $f(I, x)$= **shift** if an item $[A \rightarrow \alpha \bullet a\beta, u]$ is in set I and $x \in First_k(a\beta u)$, $a \in \Sigma$,
2. $f(I, x)$= **reduce** $A \rightarrow \alpha$ if an item $[A \rightarrow \alpha \bullet, x]$ is in set I,
3. $f(I, \varepsilon)$= **accept** if item $[Z' \rightarrow Z \bullet, \varepsilon]$ is in set I,
4. $f(I, x)$= **error** in all other cases.

The goto table g is defined as follows:

1. $g(I, X)$=J if *GOTO* (I, X)=J,
2. $g(I, X)$= **error** if *GOTO* (I, X)=$\varnothing$.

Remark that, if *GOTO* $(I, X) = J$ and *GOTO* $(I', Y) = J$ then $X = Y$. Thus the grammar symbol leading to some state in the LR automaton is unique for that state. This implies that the grammar symbol need not be stored on the parse stack by the LR parser. This is done however in order to mark places for storing attribute values associated with these grammar symbols.

For an LR(k) grammar every entry in the parsing table constructed, is uniquely defined. Besides LR(k), there are two other LR parsing methods, SLR(k) and LALR(k), which can be implemented with the scheme presented above. The methods use the same parsing algorithm, but they employ different methods for the construction of the parsing tables. Further information can be found in Aho et al. [ASU86].

2.2. Attribute grammars

Our definition of attribute grammars is based on the works of Knuth [Knu68] and Filé [Fil83]. An attribute grammar G over a semantic domain D is a context-free grammar G_0 augmented with attributes and semantic rules. A semantic domain D is a pair (Ω, Φ), where Ω is a set of sets, the sets of attribute values, and Φ is a collection of mappings of the form $f: V_1 \times V_2 \times ... \times V_m \rightarrow V_0$, where $m \geq 0$ and $V_i \in \Omega$, $0 \leq i \leq m$.
The set of *attribute symbols* is denoted by A and partitioned into I_A (*inherited* attribute symbols) and S_A (*synthesized* attribute symbols). For each attribute symbol $b \in A$, a set $V(b) \in \Omega$ contains all possible values of the attributes corresponding to b. There is a fixed set of attribute symbols associated with every grammar symbol. An *attribute* is a pair $X.a$, where $X \in V$ and $a \in A$. We assume that no inherited attribute symbols are associated with terminals and the start symbol. Attribute sets of each grammar symbol are linearly ordered with the inherited attributes preceding the synthesized attributes.

A production $p: X_0 \to X_1 X_2 \cdots X_n$ has an *attribute occurrence* $k.b$, $0 \le k \le n$, if $X_k.b$ is an attribute. An attribute occurrence $k.b$ of p is called an *input occurrence*, if either $b \in I_A$ and $k = 0$, or $b \in S_A$ and $k > 0$. Otherwise $k.b$ is said to be an *output occurrence*. For each output occurrence $k.b$ of p, there is exactly one semantic rule $k.b := f(j_1.a_1, ..., j_m.a_m)$, where every $j_i.a_i$ is an *input occurrence* of p and f is a function of the form $f: V_1 \times V_2 \times ... \times V_m \to V_0$ in Φ such that $V_0 = V(b)$ and $V_i = V(a_i)$ for $1 \le i \le m$. When f is the identity function, the rule is called a *copy rule*. When f is constant, the rule is called a *constant rule*. There are no semantic rules associated with the input occurrences. We employ also an alternative notation for semantic rules where attributes are used instead of attribute occurrences: $X_k.b := f(X_{j_1}.a_1, ..., X_{j_m}.a_m)$. Notice that attribute grammars are in Bochmann normal form.
An attributed tree for a terminal string w is a derivation tree for w, where every node n is attached with attribute instances which correspond to the attributes of the nonterminal X which is the label of n. An attribute instance is said to be evaluated when its value has been computed.
An attribute grammar is *L-attributed* [LRS74], if for every semantic rule $k.b := f(j_1.a_1, ..., j_m.a_m)$ such that $b \in I_A$, $j_i < k$ for each $i = 1, ..., m$.

3. Evaluation of attributes during LR parsing

What do we mean with evaluation of attributes *during* LR parsing? An LR parser/evaluator is a deterministic algorithm which is an extension of a shift/reduce LR parsing algorithm in such a way that it evaluates all synthesized attribute instances of a node in a (virtual) derivation tree as soon as it has recognized this node and has parsed the subtree rooted in this node (i.e. with a reduce action of the parser). Moreover, the LR parser/evaluator also computes all inherited attributes of this recognized node before it computes the instances of the synthesized attributes of this node. An attribute grammar allows attribute evaluation during LR parsing if parsing and evaluation is implementable by such an LR parser/evaluator. An attribute grammar which allows attribute evaluation during LR parsing in this sense (which is rather informal but can be formalized) is called an LR-attributed grammar.

Although an LR parser/evaluator can handle a restrictive form of non left-attributedness, we will only consider L-attributed grammars. In general, in any LR-attributed grammar semantic rules for inherited attributes of left-recursive nonterminals occurring at the first position in the right-hand side of a left-recursive production must be either copy-rules or constant rules. Moreover, if $B \to A\,\alpha$ and $C \to A\,\beta$ are left-recursive productions with constant rules for some inherited attribute of A, then these rules must be the same.

We will only consider in this paper the case with one symbol of look-ahead. Most definitions of attribute grammar classes are easy to generalize for k symbols of look-ahead. We start with a definition of a class of LR-attributed grammars that includes all classes that will be presented in the following sections. The definition uses the concept of a *d-expression* for the evaluation of an inherited attribute instance, related to some derivation. We first define this concept. If G is an unambiguous grammar there is a unique partial derivation tree associated with each right-sentential form αAw of G. This is the tree with yield αAw. Let $<\alpha A, a>$, with $a \in \Sigma$ denote the set of all partial trees of G corresponding to a right-sentential form αAw with a the leading symbol of w and let $<\alpha A, \varepsilon>$ denote the set of partial trees corresponding to a right-sentential form αA. Let $t \in <\alpha A, a>$ and let $A_0 \to \alpha_1 A_1 \gamma_1$, $A_1 \to \alpha_2 A_2 \gamma_2$, ... , $A_{n-1} \to \alpha_n A_n \gamma_n$, be the sequence of productions applied at the nodes of the path $<m_0, ..., m_n>$ from the root m_0 of t to the node m_n with the distinguished label A (i.e.

$Z = A_0$, $A = A_n$ and $\alpha_1 \cdots \alpha_n = \alpha$). Let $\alpha = Y_1 \cdots Y_q$ with $Y_i \in V$. Each occurrence of a symbol in α is identified by the index in this sequence.

The d-expression for the inherited attribute instance $A_i.y$ associated with node m_i of tree t is defined as follows. Let the semantic rule for inherited attribute occurrence $A_i.y$ associated with production $A_{i-1} \rightarrow \alpha_i A_i \gamma_i$ be $A_i.y := f(d_1, ..., d_j)$, where $d_1, ..., d_j$ are input attribute occurrences of A_{i-1} or of the symbols in α_i. The d-expression for $A_i.y$ is the expression $f(d'_1, ..., d'_j)$ where d'_k is the d-expression for $A_{i-1}.z$ if $d_k = A_{i-1}.z$ and where d'_k is $Y_{k'}.c$ when $d_k = Y.c$ and k' is the index of this occurrence of Y in α.
A d-expression for the inherited attribute y of $A = A_n$ in the right sentential form αAaw is a d-expression for the inherited attribute instance y of the node m labeled with this occurrence of A in some tree $t \in <\alpha A, a>$. In general an inherited attribute $A.y$ in a right sentential form αAaw may have several different d-expressions for different trees in the set $<\alpha A, a>$.

Definition 3.1 An attribute grammar G is *DLR-attributed*, if

a) G is an L-attributed $LR(1)$ grammar and

b) For all $A \in N$, for all inherited attributes y of A, the d-expression of y in a right sentential form αAw is uniquely determined.

The attribute grammar G, given in Figure 3.1 is not DLR-attributed, although it is an L-attributed $LR(1)$ grammar.

1.	Z	$\rightarrow$	CA	$A.x := C.x;$	$A.y := C.y$
2.	A	$\rightarrow$	Bc	$B.x := A.x;$	$B.y := A.y$
3.	B	$\rightarrow$	Ad	$A.x := B.y;$	$A.y := B.x$
4.	C	$\rightarrow$	c	$C.x := 1;$	$C.y := 2$
5.	B	$\rightarrow$	a		

Fig. 3.1 A grammar which is not DLR-attributed

The right sentential forms CBc and $CBcdc$ have partial derivation trees which both belong to the set $<CB, c>$. The d-expression for the inherited attribute of $B.x$ is clearly not unique.
Sometimes additional look-ahead for semantic disambiguation may be used. We will not consider this here. We assume that for semantic disambiguation the same amount of look-ahead is used as for parsing.

4. Classes of LR-attributed grammars

In this section we present several definitions of LR-attributed grammars. The first two definitions are based on methods presented by Watt, in [Wat77], and by Tarhio (see [Tar88]). In both methods values of inherited attributes are not computed since they are copied from particular synthesized attributes which are kept on a stack. This copying is done with the reduce action of the LR parser. In Watt's method, originally presented in the formalism of affix-grammars, the stack position of the synthesized attribute value that has to be copied to obtain the value of an inherited attribute, must always be the same.

Definition 4.1 An attribute grammar G is *WLR-attributed*, if

a) G is an *L*-attributed $LR(1)$ grammar,

b) semantic rules for inherited attributes are copy-rules.

c) for any right-most derivation $Z \Rightarrow_{rm}^{*} \alpha A w$ and for the k^{th} inherited attribute d of A such that A has s inherited attributes, the value of the attribute instance corresponding to $A.d$ is the same as the $s+k-1^{th}$ synthesized attribute instance of α in the reversed order.

In Watt's method attributes have to be copied over and over again to put them in the right order on top of the semantic stack. This is not necessary in the method for uncle-attributed grammars of Tarhio. Here, each inherited attribute has a set of uncle-attributes from which its value may be copied. We define a relation $\underline{C}$, on the set of attribute symbols of the grammar as follows: $b \underline{C} c$, if there is a copy-rule $i.c := j.b$ associated with a production $X_0 \rightarrow X_1 \cdots X_n$, where c is an inherited attribute symbol. Let b and d be inherited and s be a synthesized attribute. If $s \underline{C} d \underline{C}^{*} b$ and $i.d := j.s$ is a copy-rule associated with the production $X_0 \rightarrow X_1 \cdots X_n$, we say that the grammar symbol X_j is an *uncle symbol* of b and that the attribute $X_j.s$ is an *uncle-attribute* of b ($\underline{C}^{*}$ denotes the reflexive and transitive closure of $\underline{C}$). Conditions on the attribute dependencies of the attribute grammar should assure that in each parse situation the value of an inherited attribute d of a symbol to which a reduction takes place, can be copied from the topmost uncle-attribute of d on the stack.

Definition 4.2 An attribute grammar G is ULR-attributed, if
a) G is an L-attributed $LR(1)$ grammar,
b) semantic rules for inherited attributes are copy rules;
c) if $X_0 \rightarrow X_1..X_n$ is a production with a copy rule $j.b := i.c$, where $0 \le i < j \le n$, then for every inherited attribute symbol d, such that $b \underline{C}^{*} d$, none of the grammar symbols X_{i+1}, ..., X_{j+1} is an uncle symbol of d.
d) An inherited attribute symbol is copy-dependent on only one of the synthesized attribute symbols associated with its uncle symbol.

Watt and Tarhio both have introduced transformations into the class of WLR- and ULR-attributed grammars, respectively. These transformations make the methods more practical than the rather restrictive definitions of these classes might suggest. We do not treat these transformations here. The following definition is from [Poh83].

Definition 4.3 An attribute grammar G is *ALR-attributed*, if

a) G is DLR-attributed,

b) semantic rules for inherited attributes are copy rules or constant rules.

The following definitions of particular LR-attributed grammar classes are based on methods in which inherited attribute values do not necessarily have to be either copied from other values or assigned a constant value. These inherited attributes are also computed before reduction take place. In fact some inherited attributes are computed if a state symbol is pushed on the stack. The methods are all based on an idea presented by Madsen in [Mad80] (see also [JoM80]).

If $[A \rightarrow \alpha \bullet B \beta; u]$ is an item in state S, then the inherited attributes of B are associated with state S of the LR-automaton. Formally, the set $IN(S)$ of *inherited attributes of state S* is defined as
$IN(S) = \{ a \mid a \in I_A(B)$ for some B such that $[A \rightarrow \alpha \bullet B \beta; u]$ in $S \}$. The inherited attributes in $IN(S)$ are evaluated when a state S is pushed on the parse stack.

If $[A \rightarrow \alpha \bullet \beta; u] \in BASIS(S)$ then every attribute of every symbol in α is considered different, also in case the same symbol occurs more than once in α. It follows from the construction of the LR-states that in an L-attributed grammar all attributes in $IN(S)$ are some function of the input attributes of S. During parsing, these attributes are stored in the (attribute) parse stack. The problem how to refer to the right attribute occurrences in the attribute stack is not solved

satisfactory by Jones and Madsen in [JoM80] and [Mad80]. The problem is solved by Sassa and others; cf. [SIN85] and [SIN87]. We follow their exposition with some minor modifications. We distinguish occurrences of an attribute of a nonterminal symbol at different positions in the attribute stack. An occurrence of attribute $A.a$ in the stack is a pair $(A.a, \mathit{offset}(A.a))$ where the second element indicates the position in the attribute stack relative to the top of this stack.

Let the *attributed parse configuration* be
$(S_0\, IN(S_0) X_1 \bar{X}_1 \cdots X_{m-k} \bar{X}_{m-k} S_{m-k}\, IN(S_{m-k}) \cdots X_m \bar{X}_m S_m ; a_j \cdots a_n \perp)$, where S_i is a state, $IN(S_i)$ is a record containing the inherited attributes of state S_i, X_i is a grammar symbol and $\bar{X}_i$ is a record containing the synthesized attributes of X_i.

The *offset* of an attribute in the stack is defined as follows. If a is a synthesized attribute of X_{m-k} or if a is an inherited attribute of state S_{m-k} then $\mathit{offset}(a) = k$.
Let $[A \rightarrow X_{m-p} .. X_{m-i} .. X_m \bullet B\, \beta ; u\,]$ be an item in state S_m on top of the stack. It follows from the nature of *LR* parsing that if a is an inherited attribute of A then $\mathit{offset}(a)$ is $p+1$. If a is a synthesized attribute of X_{m-i} then $\mathit{offset}(a)$ is i.

The set $INP(S)$ of *input attribute occurrences* of state S is defined as follows.

$INP(S) =$
$\{(A.a, k) \mid a \in S_A(A)$ for some A s.t. $[B \rightarrow \alpha_1 A\, \alpha_2 \bullet \beta ; u\,]$ in S and $k = \mathit{offset}(A.a)\,\} \cup$
$\{(A.a, k) \mid a \in I_A(A)$ for some A s.t. $[A \rightarrow \alpha \bullet \beta ; u\,]$ in $BASIS(S)$ and $k = \mathit{offset}(A.a)\,\}$.

We use numbers as superscripts of nonterminal symbols to distinguish occurrences of a nonterminal symbol after the dot in different items in a particular state S. (e.g. $A^1, A^2, \cdots$.) In the same way we distinguish occurrences of an inherited attribute $A.a$ of these occurrences of A by $A^1.a, A^2.a, \cdots$.
We define $F_S(A^t.a)$, the set of *semantic expressions* for the occurrence $A^t.a$ of $A.a \in IN(S)$. It is defined in terms of attribute occurrences in $INP(S)$.

For each state S and for each $A.a \in IN(S)$, let $E_S(A.a)$ denote the set of semantic expressions for $A.a$.

$$E_S(A.a) = \bigcup_{1 \le t \le p} F_S(A^t.a),$$

where p is the number of items in state S in which A occurs at the position after the dot. $F_S(A^t.a)$ is defined for all occurrences of inherited attributes in S simultaneously as follows:

$F_S(A^t.a)$ is the smallest set such that:

a) if $[B \rightarrow \alpha \bullet A^t \beta ; u\,] \in BASIS(S)$, the semantic rule for $A.a$ associated with production $B \rightarrow \alpha A\, \beta$ is $A.a \leftarrow \mathit{expr}(a_1, \ldots, a_n)$, and $\underline{a_i} = (a_i, k)$ with k is the *offset* of the occurrence of a_i in this item, then
$\mathit{expr}(\underline{a}_1, \ldots, \underline{a}_n) \in F_S(A^t.a)$.

b) if $[B \rightarrow \bullet A^t \beta ; u\,] \in S$, this item is directly derived from item $[C \rightarrow \alpha \bullet B^v \gamma ; v\,] \in S$ and the semantic rule for $A.a$ associated with production $B \rightarrow A\, \beta$ is $A.a \leftarrow \mathit{expr}(a_1, \ldots, a_n)$, then $\mathit{expr}(e_1, \ldots, e_n) \in F_S(A^t.a)$, for all $e_i \in F_S(B^v.a_i)$ $(1 \le i \le n)$.

For the meaning of "item I is directly derived from item J" see the definition of the LR-automaton in section 2.1.

Definition 4.4 An attribute grammar G is *MLR-attributed* if

a) G is an L-attributed *LR* (1) grammar,

b) For all states S of the LR-automaton for G, for all attributes a in $IN(S)$ the set $E_S(a)$ of semantic expressions for a contains one element.

If an attribute grammar is MLR-attributed then the attribute a in $IN(S)$ of a state S can be evaluated when this state is pushed on the parse stack using the semantic expression in $E_s(a)$. Synthesized attributes are computed with reduce actions of the parser.

Example 4.5. The attribute grammar G is given by the following productions and semantic rules.

Z	$\rightarrow$	BA	$A.x$	:=	$B.s$
			$B.y$	:=	1
A_0	$\rightarrow$	CA_1B	$B.y$	:=	$A_0.x$
			$A_1.x$	:=	$C.s$
A_0	$\rightarrow$	A_1Bd	$B.y$	:=	$A_0.x$
			$A_1.x$	:=	$A_0.x$
C	$\rightarrow$	c	$C.s$	:=	2
A	$\rightarrow$	a			
B	$\rightarrow$	b	$B.s$	:=	1

G is not MLR-attributed. The problem concerns the offset of the inherited attribute of A. The LR-automaton for G has a state which contains items $[A \rightarrow CA \bullet B\,]$ and $[A \rightarrow A \bullet Bd\,]$. If this state is pushed on the parse stack it cannot be decided how far from the top we find the inherited attribute of A from which we have to copy the inherited attribute value of B. This depends on whether the B is in the right-hand side of the second or of the third production given in the table. The problem can be solved by splitting the production $A \rightarrow CAB$ in two productions, $A \rightarrow CH$ and $H \rightarrow AB$. The conflicting productions are then in different item sets. □

Nakata and Sassa [NaS86] have defined a class of $LR(1)$-attributed grammars which properly includes the class of L-attributed $LL(1)$ grammars. For this definition of $LR(1)$-attributed grammars, the states of the $LR(1)$-automaton are divided into *partial states* with respect to a set of look-ahead symbols.

Let S be a state (i.e. a set of $LR(1)$-items). The partial state of state S with look-ahead symbol a, $PS(S, a)$ is defined: $PS(S, a)=\{[A \rightarrow \alpha \bullet \beta,\ b\,] \in S \mid a \in First_1(\beta b)\ \}$.

The set of inherited attributes $IN(PS)$, which should be evaluated at partial state PS is defined as follows. $IN(PS)=\{B.b \mid$ *there is an item* $[A \rightarrow \alpha \bullet B\,\beta,\ a\,] \in PS$ *such that* $B.b \in I_A(B)\ \}$.

An interesting extension of Madsen's method is obtained by the introduction of equivalence classes of inherited attributes. These were proposed by Pohlmann in [Poh83] as a method for saving space for the attributes. It is further developed by Sassa and others [SIN87]. In most attribute grammars a lot of semantic rules are copy rules. Thus the values of a number of distinct attributes in a parse tree are often the same. These equivalence classes are, informally, defined as follows. If two inherited attributes of one and the same state of the LR-automaton always have the same value, then they can share the same storage location. These attributes belong to an equivalence class. Instead of a set $E_s(A.a)$ of semantic expressions for an attribute, we now have a set of expressions for an equivalence class of inherited attributes. For details we refer to [SIN87].

Definition 4.6 An attribute grammar G is *ECLR-attributed* if

a) G is an L-attributed $LR(1)$ grammar,

b) there is an partition of attributes such that for any partial state PS of the LR-automaton for G and for any inherited attribute $A.a$ in $IN(PS)$ the evaluation rule of $A.a$ can be uniquely determined.

Definition 4.7 An attribute grammar G is *SALR-attributed*, if G is ECLR-attributed with the strong partition on attributes (i.e. each block of the partition contains one attribute).

The difference between MLR-attributed and SALR-attributed grammars is that for the latter class we use partial states (i.e. lookahead) instead of the complete states as in MLR-attributed grammars. The class of ECLR-attributed grammars is a proper superset of the class of SALR-attributed grammars (see also the next section). Attribute grammar G from Example 4.5 is not ECLR-attributed. In general, using partial states instead of states doesn't solve problems with the offset of attribute occurrences in the stack. Also the introduction of equivalence classes is no remedy for offset problems.

5. Relations between classes of LR-attributed grammars

In this paper we do not consider the different aspects of the implementations of the several classes of LR-attributed grammars presented. Therefore we refer to [MOT90]. Here, we will only compare the extensions of these classes.
The class of XYZ-attributed grammars is denoted by XYZ. In particular: DLR will denote the class of DLR-attributed grammars as defined in Definition 3.1. We will also consider the relations between these classes and LL, the class of *L-attributed LL*(1) grammars.
We say that two grammar classes A and B are *incomparable* if $A-B \neq \emptyset$ and $B-A \neq \emptyset$. By the definition of the classes DLR, ALR, MLR, ECLR and SALR, the following inclusions hold:

a) $ALR \subset DLR$

b) $MLR \subseteq SALR \subseteq ECLR \subseteq DLR$.

The grammar in Figure 5.1 shows that the class MLR is a proper subclass of SALR.

Z	$\rightarrow$	Ab	$A.x := 1$
Z	$\rightarrow$	Ac	$A.x := 2$
A	$\rightarrow$	ε	

Fig. 5.1 Grammar in SALR–MLR.

The grammar in Figure 5.2 is in ECLR, but it is not in SALR. The reason is that in the state with items $[A \rightarrow a \bullet Bb]$, $[C \rightarrow a \bullet Bc]$ and $[B \rightarrow \bullet d]$ there are two expressions $A.x$ and $C.x$ (with their offsets) for $B.x$. This conflict disappears when $A.x$ and $C.x$ are put in the same equivalence class.

Z	$\rightarrow$	A	$A.x := 1$
A	$\rightarrow$	aBb	$B.x := A.x$
A	$\rightarrow$	C	$C.x := A.x$
C	$\rightarrow$	aBc	$B.x := C.x$
B	$\rightarrow$	d	

Fig. 5.2 Grammar in ECLR–SALR.

The grammar G in Example 4.5 is in LR and hence (see the discussion in section 4) in

LR–ECLR. Thus we have also the *proper* inclusions: MLR $\subset$ SALR $\subset$ ECLR $\subset$ DLR.

We consider the relations with the class LL. By the definitions we have that:

a) LL $\subset$ DLR

b) LL and ALR are incomparable.

In [NaS86] it is shown that LL $\subset$ SALR. LL is not included in the class MLR since the grammar in Figure 5.1 is in LL but not in MLR, and so LL and MLR are incomparable classes. The fact that A in this grammar only generates the empty string is quite essential. A grammar that doesn't contain nonterminals that only generate ε, is called *p-reduced*. If an LL-grammar is p-reduced, then it is also in the class MLR. Thus partial states are not necessary to include the p-reduced $LL(1)$ grammars. Since p-reduced $LL(1)$ grammars are $LALR(1)$ [Bea82], we can also use the method for $LALR(1)$ grammars in the bottom-up parser and still handle all L-attributed p-reduced $LL(1)$ grammars. Notice that the insertion of marker nonterminal symbols in the right-hand side of production rules, often used in transformations, (cf. [Watt77] and [Tar88]) has the effect of the introduction of partial states. When we insert two distinct marker symbols in the first two productions of the AG in Figure 5.1, then the conflict is solved.

We now turn to the classes ALR, WLR and ULR. It is clear from the definitions that

a) WLR $\subseteq$ ALR

b) ULR $\subseteq$ ALR

The above inclusions are proper and there exist attribute grammars in ALR, which are neither in ULR nor in WLR.

The classes WLR and ULR are incomparable. To see this, first consider a grammar with productions and semantic rules as in Figure 5.3. Such a grammar cannot be in ULR but it might be in WLR.

Z	$\rightarrow$	AB	$B.x := A.y$
B	$\rightarrow$	AC	$C.z_1 := B.x$
			$C.z_2 := A.y$
A	$\rightarrow$	a	$A.y := 1$
B	$\rightarrow$	b	
C	$\rightarrow$	c	

Fig. 5.3 Grammar in WLR–ULR.

Consider a grammar with a production $S \rightarrow ABC$ and a semantic rule $C.x := A.y$. Suppose that B has a synthesized attribute. An attribute grammar which contains this construction cannot be in WLR, though it can be a member of ULR.

We finally compare the classes WLR and ULR with the class ECLR. Since the grammar G in Example 4.5 is in ULR and not in ECLR, these classes are incomparable. The grammar in Figure 5.4 is in WLR but not in ECLR, because there is an offset conflict. Hence also the classes WLR and ECLR are incomparable. It follows that ULR and WLR are incomparable with all subclasses of ECLR.

The classes ALR and ECLR are incomparable. The grammar in Figure 5.5 is in ALR–ECLR. The grammar in Figure 5.6 is in ECLR–ALR.

Z	$\rightarrow$	BA	$A.x := B.s$
			$B.x := 1$
A	$\rightarrow$	$aABb$	$B.x := A_0.x$
			$A_1.x := A_0.x$
A	$\rightarrow$	ABc	$A_1.x := A_0.x$
			$B.x := A_0.x$
A	$\rightarrow$	a	
B	$\rightarrow$	d	$B.s := 1$

Fig. 5.4 Grammar in WLR–ECLR.

Z	$\rightarrow$	AB	$A.x := 1$
Z	$\rightarrow$	Ac	$A.x := 2$
A	$\rightarrow$	a	

Fig. 5.5 Grammar in ALR–ECLR.

Z	$\rightarrow$	B	$B.x := 1$
B	$\rightarrow$	bA	$A.x := f_1(B.x)$
B	$\rightarrow$	cA	$A.x := f_2(B.x)$
A	$\rightarrow$	a	

Fig. 5.6 Grammar in ECLR–ALR.

6. LC-attributed grammars

In this section we define a class of one-pass attribute grammars based on the class of left-corner grammars (LC(k) grammars). LC(k) grammars form a class of context-free grammars between the class of LL(k) and the class of LR(k) grammars.

Definition 6.1 Let k be a nonnegative integer. A grammar G is said to be $LC(k)$ if each ε-production satisfies the $LR(k)$-condition (see Definition 2.1) and if for each production $A \rightarrow X\beta$, the conditions

(1) $Z \Rightarrow_r^* \alpha A z_1 \Rightarrow_r \alpha X \beta z_1 \Rightarrow_r^* \alpha X y_1 z_1$

(2) $Z \Rightarrow_r^* \alpha' B z_2 \Rightarrow_r \alpha' \alpha'' X \gamma z_2 \Rightarrow_r^* \alpha' \alpha'' X y_2 z_2$

(3) $\alpha'\alpha'' = \alpha$ and $First_k(y_1 z_1) = First_k(y_2 z_2)$,

always imply that $\alpha A = \alpha' B$ and $\beta = \gamma$.

Informally, if a grammar is LC(k) then we can recognize the production applied at a node in a derivation tree of that grammar after we have recognized the first symbol of the right-hand side of that production. This symbol is called the *left-corner* of the production. If the production is an ε-production, the left-corner of the production is ε. This form of the definition of $LC(k)$ grammars is from Soisalon-Soininen and Ukkonen [SoU76]. Other characterizations of the left-corner grammars can be found in [Akk88]. It is shown in [SoS77] that LL(k) grammars are LC(k) and that LC(k) grammars are LR(k). These inclusions are proper. LC(k) grammars may be left-recursive.

In order to define the LC-attributed grammars we need to define a few notions.

Definition 6.2 Let $G=(N, \Sigma, P, S)$ be a grammar. For each symbol $X \in V=N \cup \Sigma$, we define the *set of chains* $CH(X)$ of X (with respect to G) as follows:

a) If $X \in \Sigma$ then $CH(X)=\{<X>\}$.

b) If $X \in N$ then $<X> \in CH(X)$ and if $X \rightarrow \varepsilon$ in P then $<X, \varepsilon> \in CH(X)$.

c) For all $X \in N$, $<X, \rho> \in CH(X)$ if $<\rho> \in CH(Y)$ for some $Y \in V$, such that $X \rightarrow Y\gamma$ in P.

Hence, a chain in $CH(X)$ is a sequence of symbols, starting with X. Moreover, Y may follow Z in a chain, if there is a production $Z \rightarrow Y\alpha$ for some $\alpha \in V^*$.
Elements of $CH(X)$ are called *chains of* X and denoted by σ. The last element of a chain σ is denoted by $l(\sigma)$. Notice that $CH(X)$ is an infinite set iff there is a derivation $X \Rightarrow_r^+ Xz$, $z \in \Sigma^*$, in G.

Let chain $\sigma=<X_0, X_1, ..., X_n> \in CH(X_0)$, with $n \geq 1$. It follows from the definition of a chain that there is a derivation

$$X_0 \Rightarrow_l^{p_1} X_1\gamma_1 \Rightarrow_l^{p_2} \cdots \Rightarrow_l^{p_n} X_n\gamma_n, \quad \gamma_i \in V^*, \ 1 \leq i \leq n \qquad (*)$$

in G. The sequence of productions $p_1p_2...p_n$ used in derivation (*) is called *a production sequence associated with the chain* σ. The production sequence associated with chains of the form $<X>$ is the empty sequence. The string γ_n is called an *r-string* of chain σ, or the *r-string* of the sequence $\pi=p_1p_2...p_n$ of productions. Notice that a chain may have more than one r-string. The length of a sequence of productions π is denoted by $|\pi|$. $l(\pi)$ denotes the last production of the sequence π.

Let $G=(N, \Sigma, P, S)$ be a grammar.
For all $A \in N$ and $X \in V$, let $CH(A, X) = \{\sigma \in CH(A) \mid l(\sigma)=X\}$ and
$PS(A, X) = \{\pi \mid \pi$ is a production sequence of a chain $\sigma \in CH(A, X)$ and $|\pi| \geq 1\}$.

Definition 6.3 For all $A \in N$, $X \in V$ and $u \in \Sigma$, the *partial set of production sequences* compatible with look-ahead symbol u, $PPS(A, X, u)$, is defined as follows: $\pi \in PPS(A, X, u)$, iff

i) $\pi \in PS(A, X)$

ii) There is production $B \rightarrow \alpha A \delta$ with $\alpha \neq \varepsilon$ and $u \in First_1(\gamma\delta.Follow_1(B))$, where γ is an r-string of π.

We now define the set of inherited attributes of a partial set of production sequences PPS as follows

$IN(PPS)=\{a \mid a \in I_A(B)$, B is left-hand side of $l(\pi)$, for some $\pi \in PPS\}$.

Let G be an $LC(1)$ grammar. Then for any two production sequences π_1 and π_2 of a set $PPS(A, X, u)$ associated with G, $l(\pi_1)=l(\pi_2)$. Thus, the inherited attributes in $IN(PPS)$ are inherited attributes of a nonterminal symbol of the grammar.

Consider a sequence π in $PPS(A, X, u)$, for some A, X and u. If π has length one, then $l(\pi)$ is A and the inherited attributes of $PPS(A, X, u)$ are the inherited attributes of A. Let $\pi=p_1p_2...p_n$ with $n>1$, $p_i: X_{i-1} \rightarrow X_i\gamma_i$ $(1 \leq i \leq n)$, $X_0=A$ and $X_n=X$. Let $a \in IN(PPS(A, X, u))$. This means that a is an inherited attribute of symbol X_{n-1}.
Suppose that G is the underlying context-free grammar of an L-attributed grammar. Then the inherited attributes of symbols X_i only depend on inherited attributes of the symbol X_{i-1}. Thus inherited attribute a depends –via a sequence of semantic functions associated with the productions that occur in π without the last production– on the inherited attributes of X_0 (A). The composite semantic function associated with the production sequence π for computing the value of a will be denoted by $csf_{\pi, a}$. If we apply $csf_{\pi, a}$ to the appropriate attribute of A

then we obtain the value of a. In case the length of π equals 1, $csf_{\pi,a}$ denotes the identity function.

Definition 6.4. An attribute grammar G is *LC-attributed*, if

a) G is L-attributed, and

b) G is an $LC(1)$ grammar, and

c) for all partial sets of production sequences PPS, if $\pi_1, \pi_2 \in PPS$, then for all $a \in IN(PPS)$, $csf_{\pi_1,a} = csf_{\pi_2,a}$.

For the construction of a parser-evaluator for LC-attributed grammars we refer to [Akk88].

Example 6.5.

Consider the following attribute grammar.

p_0:	$Z' \rightarrow E$	$E.left := true$
p_1:	$E \rightarrow E + T$	$E.left := true$
		$T.left := false$
p_2:	$E \rightarrow T$	$T.left := E.left$
p_3:	$T \rightarrow T \times F$	$T.left := true$
		$F.left := false$
p_4:	$T \rightarrow F$	$F.left := T.left$
p_5:	$F \rightarrow i$	

The underlying context-free grammar is LC(1). The attribute *left* indicates whether an operand is the leftmost operand of the corresponding operator.

$p_1^* p_2 p_3^+ \in PPS(E, T, \times) = PPS_1$. $p_1^* p_2 \in PPS(E, T, +) = PPS_2$. $p_3^+ p_4 p_5 \in PPS(T, i, \times) = PPS_3$. $p_4 p_5 \in PPS(T, i, +) = PPS_4$.

$IN(PPS_1) = \{T.left\}$, $IN(PPS_2) = \{E.left\}$, $IN(PPS_3) = \{F.left\}$ and $IN(PPS_4) = \{F.left\}$.

Let $\pi(n, m)$ $(n \geq 0, m > 0)$ denote the production sequence $p_1^n p_2 p_3^m \in PPS_1$. The *composite semantic function* for computing the attribute *left* of T, $csf_{\pi(n,m),T.left}$ equals the function $(true)^{m-1} \circ id \circ (true)^n$, where *id* denotes the identity function. Since ($n = 0$ and $m = 1$) implies $T.left = true$, $T.left$ has the value *true* for all $n \geq 0, m > 0$. Notice that the value of $E.left$ is always *true*. □

We know that the $LL(k)$ grammars are a proper subset of the $LC(k)$ grammars. Does this proper inclusion also hold for the classes of L-attributed one-pass attribute grammars based on these context-free grammars?

If G is an $LL(1)$ grammar then for all $A, X \in N$ and $u \in \Sigma$, the set $PPS(A, X, u)$ contains at most one element. From this we may conclude that LL-attributed grammars are LC-attributed.

LC is a proper subclass of DLR, but grammar G in Example 6.5 is not ECLR-attributed although it is LC-attributed. Hence, the classes LC and ECLR are incomparable. The problem with the offset of attributes in the stack doesn't occur for LC grammars, because the production is recognized as soon as its left-corner symbol is recognized.

7. Concluding remarks

One-pass compilation based on attribute grammars has several advantages over more general methods. We have considered here a number of definitions of LR-attributed grammars for one-pass compilation. We also presented a class of one-pass left-corner attributed grammars and we have compared the extensions of the classes of attributed grammars defined. The extensions of the classes is only one of the relevant aspects. In [MOT90] also aspects of the various implementations of LR-attributed grammars are presented and compared. To get a clear picture of the relations between the different definitions and methods we did not include the transformations into the several classes in our comparison. For these transformations we also refer to [MOT90].

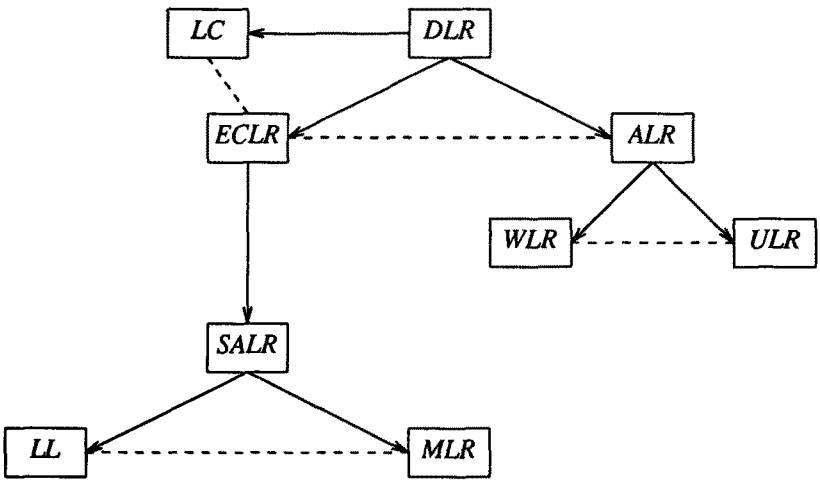

Fig. 7.1 Classes of LR-attributed grammars

Figure 7.1 shows the inclusions of the grammar classes that we considered in the paper. A dashed line between classes denotes that they are incomparable. An arrow from *A* to *B* means proper inclusion of *B* in class *A*.

References

[ASU86] Aho, A.V., Sethi, R. and Ullman, J.D. *Compilers, Principles, Techniques and Tools.* Addison-Wesley Publ., Reading, Mass., 1985.

[AhU72] Aho, A.V. and Ullman, J.D. *The Theory of Parsing, Translation and Compiling.* Vol.1 and Vol.2, Prentice Hall, Englewood Cliffs, N.J., 1972.

[Akk88] op den Akker, R. Parsing attribute grammars. Ph.D. Thesis, Department of Computer Science, University of Twente, The Netherlands, 1988.

[Bea82] Beatty, J.C. On the relationship between the LL(1) and LR(1) grammars, *Journal of the ACM* **29** (1982) 1007-1022.

[Bro74] Brosgol, B.M. Deterministic translation grammars. Ph.D. Thesis, TR-74, Harvard University, Cambridge, Mass., 1974.

[Fil83] Filé, G. The theory of attribute grammars. Doct. Diss., Twente University of Technology, Enschede, The Netherlands, 1983.

[JoM80] Jones, N.D. and Madsen, C.M. Attribute-influenced LR parsing. In: Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation, N.D. Jones (ed.) Springer-Verlag,Berlin, 1980, 393-407.

[Knu68] Knuth, D.E. Semantics of context-free languages, *Mathematical Systems Theory* **2** (1968) 127-145. Correction in: *Mathematical Systems Theory* **5** (1971) p.95.

[LRS74] Lewis, P.M., Rosenkrantz, D.J. and Stearns, R.E. Attributed translations, *J. Comput. System Sci.* **9** (1974) 279-307.

[Mad80] Madsen, C.M. Parsing attribute grammars. M. Sc. Thesis. Dept. of Computer Science, University of Aarhus, Aarhus, 1980.

[MOT90] Melichar, B., R. op den Akker and J. Tarhio. Evaluation of attributes during LR parsing. In preparation.

[NaS86] Nakata, I and Sassa, M. L-attributed LL(1)-grammars are LR(1)-attributed, *Information Processing Letters* **23** (1986) 325-328.

[Poh83] Pohlmann, W. LR Parsing of affix grammars, *Acta Informatica* **20** (1983) 283-300.

[SIN85] Sassa, M., Ishizuka, H. and Nakata, I. A contribution to LR-attributed grammars, *Journal of Information Processing* **8** (1985) 196-206.

[SIN87] Sassa, M., Ishizuka, H. and Nakata, I. ECLR-attributed grammars: a practical class of LR-attributed grammars, *Information Processing Letters* **24** (1987) 31-41.

[SoS77] Soisalon-Soininen, E. Characterization of LL(k) languages by restricted LR(k) grammars, Report A-1977-3. Department of Computer Science, University of Helsinki, Finland, 1977.

[SoU76] Soisalon-Soininen, E. and Ukkonen, E. A characterization of LL(k) languages. In: *Automata, Languages and Programming*, S. Michaelson and R. Milner (eds.), Edinburgh University Press, Edinburgh, 1976, 20-30.

[Tar88] Tarhio, J. Attribute grammars for one-pass compilation. Report A-1988-11, Ph.D Thesis, Department of Computer Science, University of Helsinki, Finland, 1988.

[Wat77] Watt, D.A. The parsing problem of affix grammars, *Acta Informatica* **8** (1977) 1-20.