ecLR-attributed Grammars: Attribute Grammars Suitable for LR Parsing

Masataka Sassa and Harushi Ishizuka
(Univ. of Tsukuba)

1. Introduction

Attribute grammars [Knu][Wai] are an extension of context-free grammars which unify syntax and semantics of programming languages. They are becoming widely used in compilers, interpreters and in other fields [Sas82].

This note concerns attribute grammars for which attributes can be evaluated during LR parsing without making a syntax tree. Such grammars make up a subset of the general attribute grammmars for which evaluation is made after making a syntax tree. But they are becoming attractive due to their efficiency and practicality and the fact that most modern programming languages are now designed around the easier one-pass processing technique.

A class of attribute grammars called LR-attributed grammars has been proposed by Jones and Madsen as a (practically) maximum class for which attributes (occurrences) can be evaluated during LR parsing [Jon]. It can deal with a sub set of inherited attributes as well as synthesized attributes. However, the original definition of LR-attributed grammars was rather theoretical, and cannot be applied directly to practical compilers: evaluation of attributes was assumed to be made in a way that all inherited attributes related to an LR-state are evaluated and stored separately causing space and time inefficiency; the definition lacked consideration of stack configuration; an algorithm for checking the "LR-attributed" property was not given.

We have developed an LR-attributed grammar class for which time and space of evaluation can be competitive with those of hand-written compilers, hopefully as in many one-pass Pascal compilers. It is based on certain equivalence classes of inherited attributes and hence named ecLR(equivalence class LR)-attributed grammars. Attributes belonging to the same equivalence class are evaluated and stored only once.

In defining the ecLR-attributed grammars, the definition of Jones et al. was re-formalized to remedy the problems mentioned above.

Two algorithms for checking the "ecLR-attributed" property are developed, and a practical compiler generator was made according to this ecLR-attributed grammar [Ish]. Preliminary results obtained by this compiler generator seem to be favorable. (This note is a development of [Sas83] and [Sas84]. A full version will appear as [Sas84b].)

2. Related works (omitted)

3. Attribute Evaluation during LR-Parsing

In this section, an analyzer for LR- and ecLR- attributed grammars is presented, which evaluates attributes during LR-parsing [Jon]. Before that, some preparation is made. For details of attribute grammars, see, for example , [Knu] [Wai].

In the following, the system is explained using an example attribute grammar G1 as shown in Fig. 1. Semantic rules are enclosed in { and }. Attribute occurrences of inherited attribute a and synthesized attribute b of symbol X are represented by X↓a and X↑b, respectively. The set of inherited and synthesized attributes of symbol X are represented by AI(X) and AS(X), respectively. We assume that the attribute grammar is well-defined [Wai].

Def 3.1 (omitted)

Def 3.2 For the production $X_0 \rightarrow X_1 \ldots X_n$, $AI(X_0)$ and $AS(X_j)$ (j=1, ... ,n) are called input (applied) attributes.

In the following, we assume L-attributed grammars in canonical form.

For a given grammar, LR states can be constructed as usual (Fig. 2(a)). As in [Pur], we subdivide an LR-state into (LR-) partial states according to look-ahead terminal symbols (Fig. 2(b)).

For a partial state PS, we define IN(PS) as follows.

Def 3.3 Given a partial state PS,

IN(PS) = { a | a is an inherited attribute of nonterminal B such that
A -> α . B β is an LR item of PS }

Ex 3.2 IN(PS) for the partial state PS of Fig. 2(b) is {E↓env,T↓env,P↓env}

The configuration of syntax and semantic analysis for LR- attributed grammars is as shown in Fig. 3. In addition to the usual parsing stack for LR parsing, attribute stacks which behave synchronously with the parsing stack are used. The stack 'ias' is for inherited attributes, and the stack 'sas' is for synthesized attributes. (The type of elements in these stacks is a union of

```
ASST    ->  V  :=  E^1,3
 { V  ↓env = ASST↓env
   E^1,3 ↓env = ASST↓env }
E^2,0  ->  E^2,1  +  T^2,3
 { E^2,1 ↓env = E^2,0 ↓env
   T^2,3 ↓env = E^2,0 ↓env }
E^3,0  ->  T^3,1
 { T^3,1 ↓env = E^3,0 ↓env }
T^4,0  ->  P^4,1  **  T^4,3
 { P^4,1 ↓env = T^4,0 ↓env
   T^4,3 ↓env = T^4,0 ↓env }
T^5,0  ->  P^5,1
 { P^5,1 ↓env = T^5,0 ↓env }
P^6,0  ->  name
 { condition name↑tag in P^6,0 ↓env }
P^7,0  ->  (  E^7,2  )
 { E^7,2 ↓env = P^7,0 ↓env }
```

(superscripts are only for explanation)

Fig. 1 Attribute grammar G1

(a)

```
ASST  ->  V  :=  .  E
E     ->  .  E  +  T
E     ->  .  T
T     ->  .  P  **  T
T     ->  .  P
P     ->  .  name
P     ->  .  (  E  )
```

(b)

(a) is an LR state (S)
(b) is an LR partial state(PS)
(state=S, look-ahead=name)

Fig.2 An LR state and partial state of G1

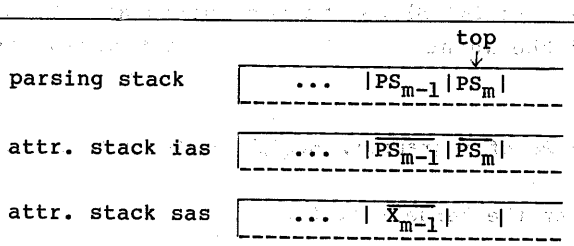
records, each record consisting of a set of attributes.)

More formally, let an attributed parse configuration (abbreviated as configuration) of the analyzer be

$(PS_0\ \overline{PS_0}\ X_0\ \overline{X_0}\ \cdots\ PS_{m-1}\ \overline{PS_{m-1}}\ X_{m-1}\ \overline{X_{m-1}}\ PS_m\ ,\ a_j\ \cdots\ a_n\$)$

where $PS_i (i=0,\ldots)$ is a partial state, $\overline{PS_i}$ is a set of all inherited attribute values in $IN(PS_i)$ (on the stack ias), X_i is a grammar symbol, $\overline{X_i}$ is a set of all synthesized attribute values of X_i (on the stack sas), and $a_j \cdots a_n$ is the remaining input. (Note that in practice, grammar symbols X_i's need not be stored as in Fig. 3.)

An attribute evaluator which evaluates attributes during LR-parsing is shown below. Note that attribute evaluation occurs in processing the grammar symbols of the RHS of a production. This is in contrast to conventional bottom-up syntax-directed translation where semantic analysis is made only at reduction time.



let PS_m be $\{X_0 \rightarrow \cdots X_{k-2}\ X_{k-1}\ .X_k \cdots\ ,\ \cdots\}$

$\overline{PS_m}$ contains values of inherited attributes in $IN(PS_m)$

$\overline{X_{m-1}}$ contains values of synthesized attributes of X_{k-1}

Fig. 3 Configuration for analysis of LR-attributed grammmars

```
proc An attribute evaluator during LR- parsing
begin
   configuration := (PS0 , a1...an$);
   loop
      let configuration be
         (PS0PS0X0X0 ... PSm-1PSm-1Xm-1Xm-1PSm , aj...an$);
      action := ACTION[PSm, aj]
                {ACTION in the parse table: ACTION[PSm, aj] is similar to
                 ACTION[Sm, aj] where partial state PSm in state Sm};
      if action = accept or action = error then exit;
      PSm := compute values of inherited attributes in IN(PSm) {see section 4};
      configutation := ( ... PSmPSm , aj...an$ );
      case action of
         shift PS :
            aj := compute values of synthesized attributes of aj
                  {from lexical analysis};
            configuration := ( ... PSmPSmajajPS , aj+1...an$);
         reduce by "X -> α " :
            k := | α |;
            PS := GOTO[PSm-k, X, aj]
```

{GOTO in the parse table: GOTO[PS_{m-k}, X, a_j] is similar to GOTO[S_{m-k}, X] where partial state PS_{m-k} in state S_{m-k}, but with look-ahead a_j};

pop configuration down to (PS_0 ... $PS_{m-k}\overline{PS_{m-k}}$, $a_j...a_n\$$);

$\overline{X}$:= compute values of synthesized attributes of X;

configuration := (... $PS_{m-k}\overline{PS_{m-k}}X\ \overline{X}$ PS , $a_j...a_n\$$)

end case

end loop

end

4. LR-attributed Grammars

LR-attributed grammars are defined as follows, with some refinement to [Jon].

Each inherited attribute $\underline{a}$ in IN(PS) can be considered as a function of input (applied) attributes of the kernels of PS. Let the function be named $E_{PS}(\underline{a})$.

Def 4.1 For a partial state PS of a grammar, $E_{PS}(\underline{a})$ is defined recursively by the following.

(1) If $\underline{a}$ is an input attribute of the kernels of PS,

$E_{PS}(\underline{a}) = \{(\underline{a}, \underline{o}_a)\}$

where $\underline{o}_a$ is the offset (from top) of $\underline{a}$ in the attribute stack (cf. Fig. 3).

(2) If $\underline{a}$ is in IN(PS),

$E_{PS}(\underline{a}) = \{f(\underline{e}_1,\underline{e}_2,...)$

such that "$\underline{a}=f(\underline{a}_1,\underline{a}_2,...)$" is a semantic function for defining $\underline{a}$, and $\underline{e}_i$ in $E_{PS}(\underline{a}_i)$ for i=1,... }.

Ex 4.1 For the partial state PS of Fig. 2(b),

$E_{PS}(E\downarrow env) = \{(ASST\downarrow env,-2)\}$,

since $E_{PS}(E^{1,3}\downarrow env) = E_{PS}(ASST\downarrow env) = \{(ASST\downarrow env,-2)\}$, and

$E_{PS}(E^{2,1}\downarrow env) = E_{PS}(E^{2,0}\downarrow env) = ... = E_{PS}(E^{1,3}\downarrow env) = E_{PS}(ASST\downarrow env)$

$= \{(ASST\downarrow env,-2)\}$.

Similarly,

$E_{PS}(T\downarrow env) = \{(ASST\downarrow env,-2)\}$,

$E_{PS}(P\downarrow env) = \{(ASST\downarrow env,-2)\}$.

Ex 4.2 (omitted)

Def 4.2 A grammar G is <u>LR-attributed</u> if

(1) G is L-attributed, and

(2) For any partial state PS of G, $E_{PS}(\underline{a})$ contains just one expression for any inherited attribute $\underline{a}$ **in** IN(PS).

Ex 4.3 Grammar G1 is LR-attributed since

(1) G1 is L-attributed, and

(2) For the partial state PS of Fig. 2(b), $E_{PS}(E\downarrow env)$ contains exactly one expression {(ASST↓env, -2)} (cf. Ex 4.1). Similar reasoning holds for $E_{PS}(T\downarrow env)$, $E_{PS}(P\downarrow env)$ and other partial states.

In LR-attributed grammars, the value of each inherited attribute is unique in each partial state. Thus, each inherited attribute can be evaluated consistently and stored in the 'ias' stack.

Ex 4.5 For G1, and at the partial state PS of Fig. 2(b), E↓env, T↓env and P↓env can be copied from the ASST↓env field of ias[top-2]. A snapshot of the configuration in the analysis is, in Fig. 3,

PS_m = Fig. 2(b)
$\overline{PS_m}$ = **record** value of E↓env, value of T↓env, and
value of P↓env **end record**
$\overline{X_k}$ = $\overline{:=}$ = ϕ

5. ecLR-attributed Grammars

In LR-attributed grammars, inherited attributes are assumed to be evaluated and stored separately. But as pointed out in a remark in [Jon], the values of several attributes may often coincide. By taking this fact into account, we have defined ecLR-attributed grammars based on the equivalence class of E_{PS}. Let $EC=\{EC_1,EC_2,...\}$ be a (disjoint) partition of the set of all inherited attributes, namely, $U_{i=1}^{n} EC_i = AI$ (AI is the set of all inherited attributes) and $EC_i \cap EC_j = \phi$ $(i \neq j,\ 1 \leq i,j \leq n)$. We assume that $EC_i \neq \phi$ $(1 \leq i \leq n)$.

Given a partition, we define $E'_{PS}(\underline{a})$ similarly to $E_{PS}(\underline{a})$.

Def 5.1 For a partial state PS and a partition EC of a grammar, $E'_{PS}(\underline{a})$ is defined recursively by the following.

(1) If $\underline{a}$ is an input attribute of the kernels of PS,
$E'_{PS}(\underline{a}) = (EC_{\#a}, o_a)$
where $EC_{\#a}$ is a set in EC to which $\underline{a}$ belongs, and o_a is the offset (from top) of $\underline{a}$ in the attribute stack.

(2) Same as in Def 4.1.

Ex 5.1 For the partial state PS of Fig. 2(b) and the partition $EC=\{EC_1\}$ where EC_1 = {ASST↓env, E↓env, T↓env, P↓env},

$E'_{PS}(E\downarrow env) = \{(EC_{\#ASST.env}, o_{ASST.env})\} = \{(EC_1,-2)\}$.

Similarly,

$E'_{PS}(T\downarrow env) = E'_{PS}(P\downarrow env) = ... = \{EC_{\#ASST.env}, o_{ASST.env})\} = \{(EC_1, -2)\}$.

Def 5.2 A grammar G is ecLR-attributed wrt. a partition $EC=\{EC_1,EC_2,...\}$, if

(1) G is L-attributed, and

(2) For each EC_i (i=1,...) and for each partial state PS, $E'_{PS}(\underline{a})$'s are the same and contain just one expression for all inherited attributes $\underline{a}$ **in** $EC_i \cap IN(PS)$.

Ex 5.2 Grammar G1 is ecLR-attributed wrt. $EC=\{EC_1\}$ where EC_1 = {ASST↓env, E↓env, T↓env, P↓env}, since

(1) G is L-attributed, and

(2) for a partial state PS of Fig. 2(b), $E'_{PS}(\underline{a})$ contains exactly one expression $\{(EC_1, -2)\}$ for all inherited attributes $\underline{a}$ **in** $EC_1 \cap IN(PS)$ = {E↓env, ..., P↓env} (cf. Ex 5.1). Similar logic holds for other partial states.

Note that in (2) of Def 5.2, no condition is imposed on a which are in EC_i, but do not belong to IN(PS). Thus, EC_i is an extension of the equivalence class in the sense that an EC_i can include mutually "independent" inherited attributes

Ex 5.3 (omitted)

Our method of dealing with ecLR-attributed grammars is to evaluate the value of inherited attributes belonging to the same EC_i only once and to store it in a single location in the attribute stack. The configuration for analysis of ecLR-attributed grammars is shown in Fig. 4. Inherited attribute stacks ias_1, ..., ias_n, each corresponding to EC_i, are used. (The type of the elements of a stack ias_i is in general the union of the types of attributes in EC_i.)

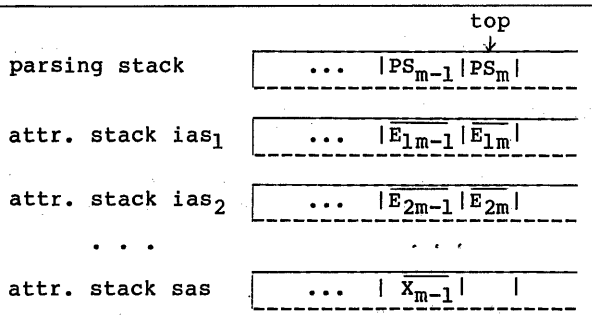
Ex 5.4 For G1 and the partition EC of Ex 5.1, a snapshot of the configuration of Fig. 4 is

PS_m = Fig. 2(b)
$\overline{E_{1m}}$ = value of $EC_1 \cap IN(PS_m)$
= value of E↓env = value of T↓env = value of P↓env of PS_m (= ias_i[top])
PS_{m-1} = { ASST -> V . := E }
$\overline{E_{1m-1}}$ = ϕ (= ias_i[top-1])
$\overline{X_{m-1}}$ = $\overline{:=}$ = ϕ
PS_{m-2} = { ... -> . ASST, ASST -> . V := E, V -> ... }
$\overline{E_{1m-2}}$ = value of ASST↓env = value of V↓env (= ias_1[top-2])

By Ex 5.1, the set of all semantic rules $E^{1,3}$↓env = ASST↓env, $E^{2,1}$↓env = $E^{2,0}$↓env, $T^{3,1}$↓env = $E^{3,0}$↓env, $P^{4,1}$↓env = $T^{4,0}$↓env, $P^{5,1}$↓env = $T^{5,0}$↓env in the partial state of Fig. 2(b) is converted into a single assignment ias_1[top] := ias_1[top-2].

For the class of ecLR-attributed grammars, the following property is clear.

Property 5.1 If a partition EC is such that each EC_i corresponds to a single inherited attribute, ecLR-attributed grammars coincide with LR-attributed grammars.



let PS_m be $\{X_0 \text{ -> } \ldots X_{k-2}\ X_{k-1}\ .X_k \ldots , \ldots \}$
$\overline{E_{ij}}$ contains values of inherited attributes in $EC_i \cap IN(PS_j)$
$\overline{X_{m-1}}$ contains values of synthesized attributes of X_{k-1}

Fig. 4 Configuration for analysis of ecLR-attributed grammmars

6. Algorithms for checking the "ecLR-attributed" property

We give two algorithms for checking the "ecLR-attributed" property. The first is based on the recursive definition of $E'_{PS}(\underline{a})$. The second checks the "ecLR-attributed" property during the making of the closure of LR items. The latter was adopted in the compiler generator we developed [Ish].

The first algorithm borrows the concept of PSPG (partial state positon graph) from [Pur]. A PSPG is a directed graph of a partial state whose nodes represent LR items and edges represent "direct derivation" in the closure. Special nodes I, SHIFT, and REDUCE represent an initial node, a shift operation (in parsing), and a reduce operation (in parsing), respectively. (For details, see [Pur].) Purdom et al. uses a labeling method to check the LR-attributed property. But the method of [Pur] can no longer be adopted in ecLR-attributed grammars since the ecLR-attributed grammars involve equivalence classes. Therefore, a new algorithm was developed.

Ex 6.1 The PSPG for the partial state of Fig. 2(b) is shown in Fig. 5 ((1), (2), etc. will be explained soon).

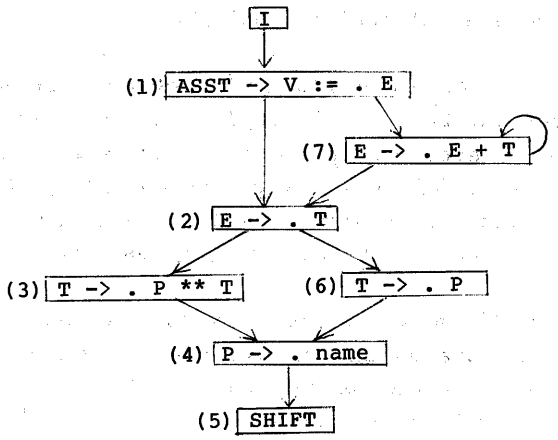


Fig. 5 PSPG of LR partial state of Fig. 2 (b)

The first algorithm checks the "ecLR-attributed" property, by recursively traversing the PSPG like a list marking algorithm.

Algorithm 1 (using PSPG)
Input: Grammar G and partition $EC=\{EC_1,\ldots,EC_n\}$.
Output: Whether the given G is ecLR-attributed or not, and
semantic expressions $E'_{PS}(\underline{a})$ at each partial state.

```
{main}
for each LR state S of G do
   for each partial state PS of S do
   begin
      make the PSPG;
      eps[j]:=empty (j=1,...,n);
      for each kernel k of PS do trav(k);
      write out eps[j] (j=1,...,n) as semantic expressions at PS
   end;
```

```
proc trav(i:LR item);
   if (i=REDUCE) or (i=SHIFT) then return;
   let i be A -> α . Bβ ;
   for each inherited attribute a of B do
      begin
         let "a=f(a1,a2,...)" be the rule defining a;
         let #a, #a1 ,... be the number of the equivalence
             class to which a, a1,... belong;
         if i is a kernel then  e:=f((EC#a1,oa1),(EC#a2,oa2),...)
         else  e:=f(eps[#a1],eps[#a2],...);
         if eps[#a]=empty then eps[#a]:=e
         else if e<>eps[#a] then  "ecLR-attributed property is violated";
         if not marked(i) then
            begin  mark(i);  for each item ni derived from i do trav(ni)  end
         end;
```

Characteristic features of this algorithm are that the check is made by stepwise traversing of the "direct derivation" relation (in the closure), and that semantic expressions are obtained in eps[j]'s.

Ex 6.2 For the partial state PS of Fig. 2(b), PSPG is already shown in Fig.5. The algorithm traverses the PSPG, for example, in the order (1), (2), ..., (7) shown in Fig. 5. At (1), $\underline{e}$ becomes (EC_1, -2) and since eps[1] is empty, eps[1] becomes (EC_1, -2). At (2) and so on, $\underline{e}$ always becomes (EC_1, -2) and is equal to eps[1], thus the ecLR-attributed property is not violated in this partial state and eps[1]=(EC_1,-2) is obtained as the semantic expression.

Usually, the use of LR states instead of partial states will suffice. The second algorithm uses LR states and checks the "ecLR-attributed" property simultaneously with the construction of closures of LR items. This algorithm is non-recursive and does not require the PSPG.

Algorithm 2 (using closure)

```
Input, Output : same as Algorithm 1
{main}
for each LR state S of G do
   begin
      closure := empty;
      eps[j]:=empty (j=1,...,n);
      for each kernel k of S do
         begin check(k); closure := closure U k end;
      repeat
         select item i from closure;
         let i be A -> α . Bβ ;
         for each production p with B as LHS do
            begin
               let p be  B -> γ ;
               check( B -> . γ );
               closure := closure U { B -> .γ }
```

```
            end
        until (all LR items in closure checked);
        write out eps[j] (j=1,...,n) as semantic expressions at S
    end;
proc check(i:LR item);
    let i be A -> α . B β;
    for each inherited attribute a of B do
        begin
            let "a=f(a1,a2,...)" be the rule defining a;
            let #a, #a1 ,... be the number of the equivalence
                class to which a, a1,... belong;
            if i is a kernel then
                e:=f((EC#a1,oa1),(EC#a2,oa2),...)
            else  e:=f(eps[#a1],eps[#a2],...);
            if eps[#a]=empty then eps[#a]:=e
            else if e<>eps[#a] then
                "ecLR-attributed property is violated"
        end;
```

7. Preliminary Results

Preliminary experience with ecLR-attributed grammars seems favorable. The statistical data in Table 1 and 2 were obtained by describing a subset of Pascal using our compiler generator based on ecLR-attributed grammars [Ina] [Ish].

As seen from Table 2, the necessary space for inherited attributes is 13.2 k bytes when the ecLR-attributed grammar method is used. On the other hand, the space will be 95.4 k bytes if the LR-attributed grammar method is used. Thus, in this example, the space for inhented attributes in ecLR-attributed grammars shows a reduction by a factor of 7 compared to that of LR-attributed grammars.

The time of overall analysis in ecLR-attributed grammars is estimated to be about 10% less compared to that of LR-attributed grammars.

Table 1 Results in a Description of a Subset Pascal

no. of input lines	2963
no. of productions	165
no. of nonterminal symbols	64
no. of terminal symbols	66
no. of synthesized attributes	128
no. of inherited attributes (maximum no. of fields in the stack for inh. attr. in LR-attributed grammars)	145
no. of equivalence classes (maximum no. of stacks for inh. attr. in ecLR-attributed grammars)	14

Table 2 Space Comparison of LR- and ecLR-attributed grammars

	LR-	ecLR-
no. of stacks for inh. attr.	145	14
space for stacks for inh. attr.	95.4 (kbyte)	13.2 (kbyte)

8. Concluding Remarks

We have presented a practical class of attribute grammars called ecLR-attributed grammars suitable for evaluation during LR parsing. Our method is based on certain equivalence classes of inherited attributes and overcomes space and time inefficiency of LR-attributed grammars. Two algorithms for checking the "ecLR-attributed" property of a given attribute grammar were presented. A compiler generator has been made based on this class.

Automatic partition of inherited attributes into equivalence classes is a future problem.

Acknowledgements

The authors wish to thank Ikuo Nakata for helpful discussions, and Rie Inada for the description of a Pascal subset in our compiler generator system.

This work is supported by the Grant in Aid form the Ministy of Education and Culture, No. 59780016.

References

[Jon] Jones,N.D.and Madsen,M., Attribute-influenced LR Parsing, Lecture Notes in Comp. Sci. 94,393-407(1980).

[Sas83] Sassa,M., One-pass Attribute Grammars Suitable for LR Parsing, 27th Conv. IPSJ, 7E-7 (Oct. 1983) (in Japanese).

[Sas84] Sassa,M. and Ishizuka,H., ecLR-attributed Grammars : Attribute Grammars Suitable for LR Parsing, 29th Conv. IPSJ, 4D-11 (Sep. 1984).

[Sas84b] Sassa,M. and Ishizuka,H., ecLR-attributed Grammars...,Tech. Rep., Univ. of Tsukuba, Inst. of Inf. Sci. and Ele. (1984).

[Ish] Ishizuka,H. and Sassa,M., A Compiler Generator Based on "ecLR-attributed" Grammars, 29th Conv. IPSJ, 4D-12 (Sep. 1984).

[Ish] Ishizuka, H., A Compiler Generator based on An Attribute Grammar Suitable for LR Parsing, Bachelor Thesis, College of Inf. Sci., Univ. of Tsukuba (1984) (in Japanese).

[Pur] Purdom, P. and Brown, C.A., Semantic Routines and LR(k) Parsers, Acta Informatica, Vol. 14, pp. 299-315 (1980).

[Isb] Ishibashi, H., Studies on Parsing Techniques Considering Semantic Analysis, Master Thesis, Dept. of Inf. Sci., Tokyo Inst. of Tech. (1980) (in Japanese).

[Wai] Waite,W.M. and Goos,G., Compiler Construction, Chap. 8, 1984 (Springer).

[Ina] Inada, R., Design of A Semantic Description Language Based on Attribute Grammars and its Application to Pascal, Bachelor Thesis, College of Inf. Sci., Univ. of Tsukuba (1984) (in Japanese).

[Knu] Knuth, D. E., Semantics of context-free languages, Math. Syst. Theory, Vol. 2, No. 2, pp. 127-145 (1968) and (correction) ibid, Vol. 5, No.1, pp. 95-96 (1971).

[Sas82] Sassa, M., Compiler Generators, Johoshori, Vol. 23, No. 9, pp. 802-817 (Sep. 1982) (in Japanese).

[Wat] Watt, D. A., The Parsing Problem for Affix Grammars, Acta Informatica, Vol. 8, pp. 1-20 (1977).

[Tar] Tarhio, J., Attribute Evaluation during LR Parsing, Rep. A-1982-4, Dept. of Comp. Sci., Univ. of Helsinki (1982).