Syntax Directed Translation with LR Parsing

Bořivoj Melichar

Department of Computers, Czech Technical University
Prague, Czechoslovakia

Abstract. A simple extension of the usual LR parser construction is made in order to build a translator. The LR parsing algorithm is extended by a facility to do output operations within the action shift and reduce. A class of translation grammars, called R-translation grammars, is introduced as an extension of the class of postfix translation grammars. Transformations called shaking-down and postponing of output symbols are used for transformation of some non-R-translation to R-translation ones.

1 Introduction

There is a class of formal translations that can be described by (context-free) translation grammars. An implementation of such formal translations directed by LR parsing is simple for postfix translation grammars [1], [2], [3], [6]. In this case, the output of output symbols is only performed when the right end of a rule is found, i.e., as a part of the reduce operation.

A class of translation grammars, called the R-translation grammars, has been introduced in [4] as an extension of the class of postfix translation grammars. This extension is based on a consideration that output symbols can be emitted also within a shift operation. Moreover, some non-R-translation grammars may be transformed to R-translation ones using, for instance, transformations called in [5] shaking-down and postponing of output symbols. The class of LR translation grammars consists of those translation grammars that may be transformed to R-translation grammars with LR input grammars.

2 Basic Notions and Notations

We refer to [1] for basic notions and notation concerning formal languages and context-free grammars.

By T^{*k} we shall denote the set $T^{*k} = \{x : x \in T^*, |x| \leq k, k > 0\}$, where the *length of string* $x \in T^*$ is denoted by $|x|$. Let $G = (N, T, P, S)$ is a context-free grammar. We define the sets $\mathrm{FIRST}_k(\alpha)$ for $\alpha \in (N \cup T)^*$, and $\mathrm{FOLLOW}_k(A)$ for $A \in N$ as follows.

$\mathrm{FIRST}_k(\alpha) = \{x \in T^* : \alpha \Rightarrow^* x\beta \text{ and } |x| = k, \text{ or } \alpha \Rightarrow^* x \text{ and } |x| < k\}$,

$\mathrm{FOLLOW}_k(A) = \{x \in T^* : S \Rightarrow^* \alpha A \beta \text{ and } x \in \mathrm{FIRST}_k(\beta)\}$.

A *formal translation* Z is a relation $Z \subset A \times B$, where A and B are sets of input and output strings, respectively.

A *context-free translation grammar* is a context-free grammar in which the set of terminal symbols is divided into two disjoint subsets, the set of input symbols and the set of output symbols, respectively.

A context-free translation grammar is a 5-tuple $TG = (N, T, D, R, S)$, where N is the set of nonterminal symbols, T is the set of input symbols, D is the set of output symbols, R is the set of rules of the form $A \to \alpha$, where $A \in N$, $\alpha \in (N \cup T \cup D)^*$, and S is the start symbol.

The *input homomorphism* h_i^{TG} and the *output homomorphism* h_o^{TG} from $(N \cup T \cup D)^*$ to $(N \cup T \cup D)^*$ are defined as follows:

$$h_i^{TG}(a) = \begin{cases} a \text{ for } a \in T \cup N \\ \epsilon \text{ for } a \in D \end{cases} \qquad h_o^{TG}(a) = \begin{cases} \epsilon \text{ for } a \in T \\ a \text{ for } a \in D \cup N \end{cases}$$

The derivation in a translation grammar TG is denoted by $\Rightarrow$, and called the *translation derivation*. The *formal translation* defined by a translation grammar TG is the set

$$Z(TG) = \{(h_i^{TG}(w), h_o^{TG}(w)) : S \Rightarrow^* w, w \in (T \cup D)^*\}.$$

The *input grammar* of a translation grammar TG is the context-free grammar

$$G_i = (N, T, R_i, S), \text{ where } R_i = \{A \to h_i^{TG}(\alpha) : A \to \alpha \in R\}.$$

Note: The superscript TG is omitted when no confusion arises.

A translation grammar TG is called a *postfix translation grammar*, if the strings of output symbols only appear at the ends of right-hand sides of the rules.

A translation grammar TG is called an *R-translation grammar*, if the strings of output symbols appear at the ends of right-hand sides of the rules and/or immediately in front of input symbols.

Postponing is the following transformation:

Let $TG = (N, T, D, R, S)$ be a translation grammar, and let R contain a rule $A \to \alpha x C \beta$, where $\alpha, \beta \in (N \cup T \cup D)^*$, C is either a terminal symbol or a nonterminal symbol generating only strings of input symbols, and $x \in D^+$. Then translation grammar $TG' = (N, T, D, R', S)$, in which

$$R' = (R - \{A \to \alpha x C \beta\}) \cup \{A \to \alpha C x \beta\},$$

is equivalent to grammar TG. String x in TG' is the postponed string.

Shaking-down is the following transformation:

Let $TG = (N, T, D, R, S)$ be a translation grammar, where R contains a rule $A \to \alpha x B \beta$, $x \in D^+$, $\alpha, \beta \in (N \cup T \cup D)^*$, $B \in N$, and $B \to \gamma_1 | \gamma_2 | \cdots | \gamma_n$ are all rules in R with nonterminal symbol B on the left-hand side.

Let $TG' = (N \cup \{[xB]\}, T, D, R', S)$, where

$$R' = (R - \{A \to \alpha x B \beta\}) \cup \{A \to \alpha [xB] \beta, [xB] \to x\gamma_1 | x\gamma_2 | \cdots | x\gamma_n\}.$$

Then $Z(TG) = Z(TG')$. String x in TG' is the shaken-down string.

3 Translation $LR(k)$ Items

The algorithm of formal translation described below is directed by an LR parser. The conventional LR parser is extended by adding some operations to perform a translation. Similarly to the LR parser, the algorithm of the formal translation is table-driven, and the construction of necessary tables is similar to the construction of LR tables. Hence, we shall use the notion of a translation $LR(k)$ item which is an extension of the notion of the conventional $LR(k)$ item.

Definition 1. A *translation LR(k) item* for the translation grammar $TG = (N, T, D, R, S)$ is an object of the form

$$[A \to \alpha \cdot \beta, x, w],$$

where $A \to \alpha\beta$ is a rule in the input grammar of translation grammar TG, $x \in D^*, w \in T^{*k}$, $k \geq 0$.
For $k = 0$, an $LR(0)$ translation item will be written in the form $[A \to \alpha \cdot \beta, x]$.

Let us now formulate two basic transformations of postponing and shaking-down in terms of translation items.
Let $TG = (N, T, D, R, S)$ be a translation grammar with a rule $A \to \alpha x C \beta$ in R, where $\alpha, \beta \in (N \cup T \cup D)^*$, $x \in D^*$, C is either a terminal symbol or a nonterminal symbol generating only strings of input symbols, and α does not end with an output symbol. Let us have a set of translation items M that contains item $[A \to h_i(\alpha) \cdot Ch_i(\beta), y, u]$. The set GOTO$(M, C)$ contains item
$[A \to h_i(\alpha)C \cdot h_i(\beta), x, u]$. This item corresponds to the rule $A \to \alpha C x \beta$ which may be obtained by postponing string x in the rule $A \to \alpha x C \beta$.

Similarly, if there is an item $[B \to h_i(\alpha) \cdot Ch_i(\beta), x, u]$ in set M and
$C \to y_1\gamma_1 | y_2\gamma_2 | \cdots | y_n\gamma_n \in R$, where for $1 \leq i \leq n$, $y_i \in D^*$, $\gamma_i \in (N \cup T)(N \cup T \cup D)^*$, then translation items
$[C \to \cdot h_i(\gamma_1), xy_1, v]$, $[C \to \cdot h_i(\gamma_2), xy_2, v]$, $\cdots$ $[C \to \cdot h_i(\gamma_n), xy_n, v]$,
where $v \in \text{FIRST}_k(\beta u)$, correspond to the following rules obtained by shaking-down string x in the rule $B \to \alpha x C \beta$:

$$C \to xy_1\gamma_1 | xy_2\gamma_2 | \cdots | xy_n\gamma_n.$$

The following algorithm constructs the collection P of sets of translation $LR(k)$ items for a given translation grammar.

A collection of sets of $LR(k)$ items is a finite collection of finite sets for every context-free grammar, regardless of whether the grammar is $LR(k)$ or not. But there are translation grammars for which the collection of sets of translation $LR(k)$ items is an infinite collection of infinite sets. In this case, it is necessary to prevent this situation by indicating infinite loops in a construction algorithm.

Definition 2. In a set M of the collection P of sets of translation $LR(k)$ items, there is

1. a *shift-translation conflict*, if there are two items in M of the forms
$$[A \to \alpha \cdot a\beta, x, u] \text{ and } [B \to \gamma \cdot a\delta, y, v],$$
for $a \in T$, $x \neq y$, and $\text{FIRST}_k(a\beta u) \cap \text{FIRST}_k(a\delta v) \neq \emptyset$,
2. an *expansion-translation conflict*, if there are two items in M of the forms
$$[A \to \alpha \cdot B\beta, x, u] \text{ and } [C \to \gamma \cdot B\delta, y, v],$$
for $B \in N$, $x \neq y$, and $\text{FIRST}_k(\beta u) \cap \text{FIRST}_k(\delta v) \neq \emptyset$,
3. a *reduction-translation conflict*, if there are two items in M of the forms
$$[A \to \alpha \cdot, x, u] \text{ and } [A \to \alpha \cdot, y, u], \text{ for } x \neq y.$$

The algorithm constructing the collection of sets of translation $LR(k)$ items is an extended algorithm for a construction of the collection of sets of $LR(k)$ items for the LR parser. The extensions are

(a) shaking-down a string of output symbols during the computation of a closure,
(b) postponing a string of output symbols during the computation of a set,
(c) an indication of infinite loops in both cases.

We first present an algorithm computing the closure of a set of translation $LR(k)$ items. We assume that the location is appended to each output symbol in the right-hand side of each rule in R. The location is a pair (r,p), where r is the number of a rule, and p is the position of the output symbol in its right-hand side.

Algorithm 1. Computation of the closure of a set of translation $LR(k)$ items.
Input: Translation grammar $TG = (N, T, D, R, S)$, a set M of translation $LR(k)$ items, and $k \geq 0$.
Output: CLOSURE(M) with shaken-down strings marked, or a signalization of the infinite loop.
Method:

1. CLOSURE(M) := M.
2. Let $[A \rightarrow \alpha \cdot B\beta, x, u] \in$ CLOSURE(M), $B \in N$, $B \rightarrow y\ \gamma \in R$, where $y \in D^*$ and γ is either empty string or starts with an input or a nonterminal symbol, and $v \in \text{FIRST}_k\ (h_i(\beta)u)$.
   If $[A \rightarrow \alpha \cdot B\beta, x, u]$ is in an expansion-translation conflict with some item in CLOSURE(M), then the string x is not shaken-down, and
   $$\text{CLOSURE}(M) := \text{CLOSURE}(M) \cup [B \rightarrow h_i(\gamma), y, v].$$
   Otherwise, the string x is shaken-down from the item $[A \rightarrow \alpha \cdot B\beta, x, u]$, and
   $$\text{CLOSURE}(M) := \text{CLOSURE}(M) \cup [B \rightarrow h_i(\gamma), xy, v].$$
3. If the string x is shaken-down, then mark shaken-down string x as x^s in item $[A \rightarrow \alpha \cdot B\beta, x, u]$ and check if some output symbol from string y appears in string x with the same location. If there is such symbol, then finish the computation with a signalization of the infinite loop.
4. Repeat steps 2 and 3 until no new items can be inserted into the set CLOSURE(M).

We now present an algorithm constructing sets of translation $LR(k)$ items.

Algorithm 2. Contruction of the collection of sets of translation $LR(k)$ items.
Input: Translation grammar $TG = (N, T, D, R, S)$, where rules in R are numbered, and $k \geq 0$.
Output: Collection P of sets of translation $LR(k)$ items for the translation grammar TG, or a failure signalization.
Method:

1. Construct an augmented grammar
   $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \rightarrow S\}, S')$. To each output symbol on the right-hand side of each rule in R append its location which is a pair (r,p), where r is the number of a rule, and p is the position of the output symbol in its right-hand side.
2. Construct the initial set of translation $LR(k)$ items as follows:
   (a) $\# :=$ CLOSURE($\{[S' \rightarrow \cdot S, \epsilon, \epsilon]\}$).
   (b) If a signalization of the infinite loop appears during the computation of the closure, finish the computation with a failure signalization. Otherwise $P := \{\#\}$.
3. If a set M_i of translation $LR(k)$ items has been constructed, construct for each symbol $X \in (N \cup T)$ which follows the dot in some item in M_i a new set of translation $LR(k)$ items M_j, in this way:

(a) $M_j := \emptyset$.
(b) Select the subset Y of items from M_i with symbol X following the dot:
$$Y = \{[A \to \alpha \cdot \gamma, x, u] : [A \to \alpha \cdot \gamma, x, u] \in M_i, \gamma = X \cdot \beta\}.$$
(c) For each item $[A \to \alpha \cdot X\beta, x, u] \in Y$ do:
Let $A \to \alpha' Xy\beta'$ be a rule of translation grammar TG and $y \in D^*$, $\alpha = h_i(\alpha'), \beta = h_i(\beta')$, i.e., $A \to \alpha X\beta$ is the rule in the input grammar of TG for this rule. Let β' starts with an input or a nonterminal symbol.
If either the string x is marked as a shaken-down string, or X is an input symbol and item $[A \to \alpha \cdot X\beta, x, u]$ is not in a shift-translation conflict, then the string x is not postponed, and
$M_j := M_j \cup [A \to \alpha X \cdot \beta, y, u]$. Otherwise, if either X is a nonterminal symbol generating only strings of input symbols, or X is an input symbol, then the string x is postponed, and
$$M_j := M_j \cup [A \to \alpha X \cdot \beta, xy, u].$$
In case when the string x cannot be postponed because the nonterminal in question generates strings of both input and output symbols, finish the computation with a failure signalization.
Mark the postponed string x as x^p in item $[A \to \alpha \cdot X\beta, x, u]$.
(d) $M_j :=$ CLOSURE(M_j).
(e) If a signalization of the infinite loop appears during the computation of the closure, finish the computation with a failure signalization.
(f) $P := P \cup \{M_j\}$.

4. Repeat step 3. for all sets M_i until no new sets can be added into the collection P.
5. Replace strings of output symbols marked either as shaken-down or as postponed by empty strings in all items of all sets.

Definition 3. Translation grammar TG is called an *LR(k) translation grammar* if and only if the input grammar of TG is an $LR(k)$ grammar, and there is no translation-conflict in any set of translation $LR(k)$ items of the collection P for TG.

Algorithm 2 constructs the collection of sets of translation $LR(k)$ items for a given translation grammar. This collection differs from the collection of sets of $LR(k)$ items for the input grammar. Each of its items contains a string of output symbols.

There is a string y of output symbols in an item with the dot at the end of the right-hand side of a rule. String y is either a string of output symbols from the end of the rule in question, or a string of shaken-down or postponed output symbols. This situation means that a reduce operation will be executed during the translation, and the string y will be added to the output string.

There is also a string x of output symbols in an item with the dot in front of an input symbol. In this case, string x is either a string of output symbols from the rule in question, placed in front of the input symbol following the dot, or a string of shaken-down or postponed output symbols. The existence of such an item in some set of translation $LR(k)$ items means that a shift operation will be executed during the translation, and the string x will be added to the output string.

4 Algorithm of the Formal Translation

For an $LR(k)$ translation grammar, the translation can be performed using an algorithm that is obtained by the following modifications of the LR parser.

1. During a reduce operation, add the string of output symbols from the translation $LR(k)$ item corresponding to the reduce operation performed.
2. During a shift operation, add the string of output symbols from the translation $LR(k)$ item corresponding to the shift operation performed.

Strings of output symbols may be inserted into entries of the action table of the LR parser. The resulting table will be called a translation table.

Algorithm 3. Construction of the translation table for an $LR(k)$ translation grammar.
Input: $LR(k)$ translation grammar $TG = (N, T, D, R, S)$, and collection P of sets of translation $LR(k)$ items grammar TG.
Output: Translation table p for the translation grammar TG.
Method: Translation table has a row for each set of items from P, columns are for all elements of the set T^{*k}.

1. $p(M_i, u) = shift\ (x)$, if $[A \to \alpha \cdot \beta, x, v] \in M_i$, $\beta \in T(N \cup T)^*$, $u \in \mathrm{FIRST}_k(\beta v)$, $x \in D^*$,
2. $p(M_i, u) = reduce\ j(x)$, if $j \geq 1$, $[A \to h_i(\alpha)\cdot, x, u] \in M_i$, $A \to \alpha$ is j-th rule in R, $u \in T^{*k}$, and $x \in D^*$,
3. $p(M_i, \epsilon) = accept$, if $[S' \to S\cdot, \epsilon, \epsilon] \in M_i$,
4. $p(M_i, u) = error$ in all other cases.
   Note: The goto table may be constructed in the same way as that for the LR parser (see [1]).

Algorithm 4. Formal translation for $LR(k)$ translation grammar.
Input: The translation table p and the goto table g for a translation grammar $TG = (N, T, D, R, S)$, and an input string $x \in T^*$, $k \geq 0$.
Output: Output string y in case that for $x \in L(G_i)$, $(x, y) \in Z(TG)$. Otherwise an error signalization.
Method: The symbol # is the initial symbol in the pushdown store. Repeat steps 1, 2, and 3 until accept or error appears. Symbol X is on the top of the pushdown store.

1. Fix the string u of first k symbols from the unused part of the input string.
2. (a) If $p(X, u) = shift\ (x)$, then read one input symbol, add the string x to the output string, and go to step 3.
   (b) If $p(X, u) = reduce\ i(x)$, then pop from the pushdown store the same number of symbols as that of input and nonterminal symbols in the right-hand side of the i-th rule $(i) A \to \alpha$, and add string x to the output string. Go to step 3.
   (c) If $p(X, u) = accept$, then finish the translation. The output string is the translation of the input string provided that the input string is read completely. Otherwise finish the translation by an error signalization.
   (d) If $p(X, u) = error$, then finish the translation by an error signalization.

3. If W is a symbol that is to be pushed on the pushdown store (the read symbol in 2(a) or the left-hand side of a rule used in the reduction in 2(b), and Y is the symbol at the top of the pushdown store, then:
   (a) If $g(Y, W) = M$, then push M on the top of the pushdown store, and go to step 1.
   (b) If $g(Y, W) = error$, then finish the translation by an error signalization.

A configuration of Algorithm 4 is a triple (α, x, y), where α is the contents of the pushdown store, x is the unused part of the input string, and y is the created part of the output string.

The initial configuration is a triple $(\#, x, \epsilon)$, the accepting configuration is a triple $(\#M_i, \epsilon, y)$, where M_i is the symbol at the top of the pushdown store, and it holds for M_i that $p(M_i, \epsilon) = accept$.

5 Conclusion

An approach similar to that for $LR(k)$ translation grammars may also be used to define $LALR(k)$ translation grammars. A slightly different approach must be used in case of $SLR(k)$ translation grammars. An inspection of translation conflicts must be performed during the computation of translation $LR(0)$ items in order to postpone output symbols.

The class of $LR(k)$ translation grammars does not contain all translation grammars with $LR(k)$ input grammars. For example, all translation grammars with output symbols in front of left-recursive nonterminal symbols do not belong to this class.

References

[1] Aho, A.V., Ullman, J.D. (1971,1972) The theory of parsing, translation and compiling. Vol.1: Parsing, Vol.2: Compiling, New York: Prentice – Hall.

[2] Lewis, P.M., Stearns, R.E (1968) Syntax directed transductions. Journal of the ACM, Vol. 15, No. 3, pp. 465 – 488, July 1968.

[3] Lewis, P.M., Rosenkrantz, D.J.,Stearns, R.E. (1976) Compiler design theory. London, Addison – Wesley.

[4] Melichar, B. (1992) Formal translation directed by LR parsing. Kybernetika, Vol. 28, No.1, pp. 50 – 61, January 1992.

[5] Melichar, B. (1992) Transformations of translation grammars. Kybernetika (to appear).

[6] Purdom, P., Brown, C.A. (1980) Semantic routines and LR(k) parsers. Acta Informatica, Vol. 14, No. 4, pp. 229 - 315.