KYBERNETIKA — VOLUME 38 (2002), NUMBER 1, PAGES 13–44

TRANSFORMATIONS OF GRAMMARS AND TRANSLATION DIRECTED BY *LR* PARSING

Bořivoj Melichar and Nguyen Van Bac

The class of *LR* translation grammars is introduced. This class is characterized by a possibility to implement a formal translation as an algorithm directed by *LR* parsing. To perform a translation, the conventional *LR* parser is extended by a facility to perform output operations within the parsing actions shift and reduce. The definitions of *Kernel*(R)- and LR-translation grammars are presented. The transformations shaking-down and postponing that enable to transform some translation grammars into *Kernel*(R)-translation grammars are described and used in the process of construction of the collection of sets of $LR(k)$ translation items. Different algorithms using these transformations are presented in an uniform way which makes it possible to compare them and to fix the hierarchy of the *LR* translation grammars.

1. INTRODUCTION

The notion of the syntax-directed translation was a highly influential idea in the theory of formal translations. Models for a description of formal translations are the syntax-directed translation schemes. A special case of syntax-directed translation schemes are the simple syntax-directed translation schemes, which can be written in the form of translation grammars. For an arbitrary translation described by a translation grammar with an $LL(k)$ input grammar, it is possible to create a one-pass translation algorithm by a simple extension of the parsing algorithm for $LL(k)$ grammars. A similar approach to $LR(k)$ grammars led to the result that it is only possible to perform a one-pass formal translation during $LR(k)$ parsing in case the translation grammar has a postfix property ([1, 8, 9, 15]). Nevertheless, there is a possibility to make an extension of the $LR(k)$ parsing algorithm in which the output of output symbols can be performed not only as a part of the operation reduce but also as a part of the operation shift. Translation grammars for which such an extension may be applicable are called R-translation grammars ([12]). Translation grammars for which we can always construct a one-pass formal translation are called *Kernel*(R)-translation grammars ([14]). Moreover, there are transformations that enable to transform some translation grammars into *Kernel*(R)-translation grammars. The well-known transformations that can be used for this purpose are shaking-down and postponing ([13]). It is possible to include these two transformations in an algorithm for the construction of the collection of sets of $LR(k)$ translation items. This collection is then used to construct the translation and goto tables, and under control of these tables an LR translator performs a formal translation for a given input string. Translation grammars for which this method is usable are called LR-translation grammars.

The main motivation for the research described in this paper was the evaluation of attributes in attribute grammars [2, 9]. For this purpose, we suppose that the set of semantic rules for the evaluation of attributes in the particular place of the grammar rule is taken as an action symbol. Using this assumption, we can transform an attribute grammar into the translation grammar where action symbols are inserted into the grammar rules at the proper places and plays the role of output symbols. The implementation of attribute evaluation is performed as a formal translation with this exception: as soon as the output (action) symbol is prepared for the output, the corresponding semantic rules are evaluated.

This paper is organized as follows. Section 2 provides basic notions and notations of translation grammars. Section 3 presents algorithms of a computation of the collection of sets of R-translation items, a construction of the translation table, and a formal translation. In this section the definition of *Kernel*(R)-translation grammars is presented. Section 4 discusses methods that enable to transform translation grammars into *Kernel*(R)-translation grammars. In these methods the shaking-down and postponing transformations are used as main tools for this purpose. The definition of $LR(k)$ translation grammars is shown in Section 4. Algorithms which use miscellaneous combinations of the two abovementioned transformations induce five classes of translation grammars and their hierarchy is discussed in Section 5. Finally, conclusions are made in Section 6.

2. BASIC NOTIONS AND NOTATIONS

We use standard basic notions and notations of context-free grammars and parsing such as are designed [1, 2]. Below are basic notions and notations of translation grammars.

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 $A \rightarrow \alpha$, where $A \in N$, $\alpha \in (N \cup T \cup D)^*$, and $S \in N$ is the start symbol. Empty string is denoted by ε.

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:

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

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* G_i and the *output grammar* G_o of a translation grammar TG are the context-free grammars that are defined as follows.

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

$$G_o = (N, D, R_o, S), \text{ where } R_o = \{A \to h_o^{TG}(\alpha) : A \to \alpha \in R\}.$$

Note that we can omit the superscript TG when no confusion arises.

By T^{*k} we will 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 $|x|$. For input context-free grammars we define the sets $\text{FIRST}_k(\alpha)$ for $\alpha \in (N \cup T)^*$, and $\text{FOLLOW}_k(A)$ for $A \in N$ as follows:

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

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

A translation grammar is *semantically unambiguous* if there are no two distinct derivations, called the translation derivations in the theory of translation grammars, such that $A \Rightarrow^* \alpha$ and $A \Rightarrow^* \beta$ where $A \in N$, $h_i(\alpha) = h_i(\beta)$ and $h_o(\alpha) \neq h_o(\beta)$.

A translation grammar TG is called a *postfix translation grammar*, if the strings of output symbols appear only at the ends of right-hand sides of the rules. The class of postfix translation grammars is denoted as $C_{postfix}$.

A translation grammar TG is called an *R-translation grammar*, if its input grammar is an $LR(k)$ grammar and the strings of output symbols appear at the ends of right-hand sides of the rules and/or immediately in front of input symbols. More formally: R–translation grammar has rules of the form
$A \to x_1y_1A_1x_2y_2A_2 \ldots x_ny_nA_nx_{n+1}$, where $A, A_1, A_2, \ldots, A_n \in N, y_1, y_2, \ldots, y_n \in T^*$, $x_1, x_2, \ldots, x_n, x_{n+1} \in D^*$. Moreover, if $y_m = \varepsilon$, $1 \leq m \leq n$, then $x_m = \varepsilon$.

The *translation tree* is a derivation tree whose leaves can be both input and output symbols.

3. KERNEL(R)–TRANSLATION GRAMMARS

In this section we introduce *Kernel*(R)-translation grammars and a one-pass formal translation for *Kernel*(R)-translation grammars. The *Kernel*(R)-translation grammars are R-translation grammars in which a one-pass formal translation during $LR(k)$ parsing is performed successfuly. In the *Kernel*(R)-translation grammars the output symbols appear in front of input symbols and/or at the ends of the right-hand sides of rules. Thus, the postfix translation grammars are also *Kernel*(R)-translation grammars.

Before presenting algorithms for the one-pass formal translation and the definition of *Kernel*(R)-translation grammars we first introduce the notion of $LR(k)$ translation item and a translation conflict which can occur during the translation.

Definition 3.1. An $LR(k)$ *translation item* for an R-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 of the translation grammar TG,

$x \in D^*$ is a string of output symbols,

$w \in T^{*k}$ is a string of input symbols, $k \geq 0$, w is called the lookahead string.

For $k = 0$, the $LR(0)$ translation item can be written in the form $[A \to \alpha \cdot \beta, x]$. The $LR(k)$ translation item differs from the LR(k) item of the input grammar. The difference is that the $LR(k)$ translation item contains a string of output symbols.

Definition 3.2. In a set M of $LR(k)$ translation items, there is 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 $a \in T$, $x \neq y$, $\mathrm{FIRST}_k(ah_i(\beta)u) \cap \mathrm{FIRST}_k(ah_i(\delta)v) \neq \emptyset$.

The algorithm for constructing the collection of $LR(k)$ translation items performs two basic operations:

1. Constructing the *base* of set of $LR(k)$ translation items.
2. Construction the *closure* of set of $LR(k)$ translation items.

We will show several variants of algorithms for constructing the collection of $LR(k)$ translation items and therefore we will describable this two operations separately.

Algorithm 3.3. Construction of CLOSURE of $LR(k)$ translation items.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules are numbered, $k \geq 0$, and a set I of $LR(k)$ translation items.

Output: $CLOSURE(I)$.

Method:

a) $CLOSURE(I) := I$.

b) If $[A \to \alpha \cdot B\beta, \varepsilon, u] \in CLOSURE(I)$, $B \in N$, $B \to z\gamma \in R$, $z \in D^*$, $\alpha, \beta \in (N \cup T \cup D)^*$, $\gamma \in (T \cup N)(N \cup T \cup D)^* \cup \{\varepsilon\}$, then $CLOSURE(I) := CLOSURE(I) \cup \{[B \to z \cdot \gamma, z, v]: v \in \mathrm{FIRST}_k(h_i(\beta)u)\}$.

c) Repeat step b) until no new translation item can be inserted into the set $CLOSURE(I)$.

The following algorithm constructs the collection L of sets of $LR(k)$ translation items for an R-translation grammar TG.

Algorithm 3.4. Collection of sets of $LR(k)$ translation items for an R-translation grammar.

`Input:` R-translation grammar $TG = (N, T, D, R, S)$, where rules in R are numbered, and $k \geq 0$.

`Output:` Collection L of sets of $LR(k)$ translation items for the R-translation grammar TG, or a failure signalization.

`Method:`

1. Construct an augmented grammar $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \rightarrow S\}, S')$.
2. Construct the initial set of $LR(k)$ translation items as follows:
   (a) $\# := [S' \rightarrow \cdot S, \varepsilon, \varepsilon]$.
   (b) $\# := CLOSURE(\#)$.
   (c) Check if there is a *shift-translation conflict.* If yes, then finish the computation with a failure signalization of a *shift-translation conflict.* Otherwise $L := \{\#\}$, $\#$ is the initial set.
3. If the set M_i of $LR(k)$ translation items has been constructed, construct for each symbol $X \in (N \cup T)$ which is in some $LR(k)$ translation item in M_i just behind the dot, a new set of $LR(k)$ translation items X_j in the following way:
   (a) $X_j := \{[A \rightarrow \alpha X y \cdot \beta, y, u]$: $[A \rightarrow \alpha \cdot X y \beta, x, u] \in M_i$, $x, y \in D^*$, $X \in (N \cup T)$, $\alpha \in (N \cup T \cup D)^*$, $\beta \in (T \cup N)(N \cup T \cup D)^* \cup \{\varepsilon\}\}$.
   (b) $X_j := CLOSURE(X_j)$.
   (c) Check if there is a *shift-translation conflict.* If yes, then finish the computation with a failure signalization of a *shift-translation conflict.* Otherwise $L := L \cup \{X_j\}$.
4. Repeat step 3 for all sets X_j until no new set can be added into the collection L.

The collection constructed by this algorithm differs from the collection of sets of $LR(k)$ items for the input grammar. There is a string of output symbols in an item with the dot at the end of the right-hand side of the rule. In this situation the string of output symbols will be added to the output string during the operation reduce. There is also a string of output symbols in an item with the dot just in front of an input symbol. This means that the string of output symbols will be added to the output string during the operation shift if no shift-translation conflict arises.

This algorithm can be used for postfix grammars, too. In this case the output symbols will be emitted only during the reduce operation.

Example 3.5. Consider R-translation grammar $TG_1 = (\{A, B\}, \{a, b, c, d\}, \{x, y, z\}, R, A)$ whose set R contains the rules:

(1) $A \to xaBb$

(2) $A \to Byc$

(3) $B \to dz$.

Algorithm 3.4 constructs the following collection of sets of $LR(1)$ translation items.

$\# = \{[S' \to \cdot A, \varepsilon, \varepsilon],\ [A \to x \cdot aBb, x, \varepsilon],\ [A \to \cdot Byc, \varepsilon, \varepsilon],\ [B \to \cdot dz, \varepsilon, c]\}$,

$A = \{[S' \to A \cdot \varepsilon, \varepsilon]\}$,

$a = \{[A \to xa \cdot Bb, \varepsilon, \varepsilon],\ [B \to \cdot dz, \varepsilon, b]\}$,

$B_1 = \{[A \to By \cdot c, y, \varepsilon]\}$,

$d_1 = \{[B \to dz \cdot z\ , c]\}$,

$d_2 = \{[B \to dz \cdot z\ , b]\}$,

$B_2 = \{[A \to xaB \cdot b, \varepsilon, \varepsilon]\}$,

$c = \{[A \to Byc \cdot, \varepsilon, \varepsilon]\}$,

$b = \{[A \to xaBb \cdot, \varepsilon, \varepsilon]\}$.

Let us make a note concerning the names of sets of $LR(k)$ translation items. As is usual in LR parsing, we use names of the form X_i where $X \in N \cup T$ and i is an integer number. X is the closest input or nonterminal symbol to the left of the dot. This symbol is unique for each set in the collection of $LR(k)$ translation items with the only exception of the initial set. Its name is # in all cases. The index i is used in order to distinguish different sets for the same grammar symbol. If such set is just one then this index may be omitted. Names of sets of items will play the role of pushdown store symbols. Symbol # is always the initial pushdown store symbol. This way of naming of sets of items has the following advantage during reading the contents of the pushdown store:

If we omit symbol # and indices, we see the so-called viable string [1], a prefix of the right sentential form of the input grammar.

For an R-translation grammar, the translation can be performed using an algorithm that is obtained by the following two modifications of the LR parser.

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

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

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

Algorithm 3.6. Construction of the translation table for an R-translation grammar.

Input: R-translation grammar $TG = (N, T, D, R, S)$, and collection P of sets of $LR(k)$ translation items for the grammar TG.

Output: Translation table F 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. $F(M_i, u) = \mathit{Shift}(x)$, if $[A \to \alpha \cdot \beta, x, v] \in M_i$, $M_i \in P$, $\beta \in T(N \cup T)^*$, $u \in \mathrm{FIRST}_k(h_i(\beta v))$, $x \in D^*$,
2. $F(M_i, u) = \mathit{Reduce}\ j(x)$, if $j \geq 1$, $[A \to \alpha \cdot, x, u] \in M_i$, $M_i \in P$, $A \to \alpha$ is jth rule in R, $u \in T^{*k}$, a $x \in D^*$,
3. $F(M_i, \varepsilon) = \mathit{Accept}$, if $[S' \to \cdot, \varepsilon, \varepsilon] \in M_i$, $M_i \in P$,
4. $F(M_i, u) = \mathit{Error}$ otherwise.

The *goto table* G is constructed in the same way as that for the LR parser (see [1]).

Example 3.7. The goto and translation tables for the translation grammar TG_1 from Example 3.5 are shown in Table 3.1.

Table 3.1. Goto table G and translation table F for the R-translation grammar TG_1.

	A	B	a	b	c	d	a	b	c	d	ε
#	A	B_1	a			d_1	*Shift*(x)			*Shift*(ε)	
A											*Accept*
B_1					c				*Shift*(y)		
B_2				b				*Shift*(ε)			
a		B_2				d_2				*Shift*(ε)	
b											*Reduce* $1(\varepsilon)$
c											*Reduce* $2(\varepsilon)$
d_1									*Reduce* $3(z)$		
d_2								*Reduce* $3(z)$			

Algorithm 3.8. Formal translation for an R-translation grammar.

Input: The translation table F and the goto table G for an R-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. At the beginning of the formal translation the output string is empty, i.e. $y = \varepsilon$. Repeat steps 1, 2 and 3 until *Accept* or an *Error* appears. Symbol X is on the top of the pushdown store.

1. Fix the string u of the first k symbols from the unused part of the input string x.
2. (a) If $F(X, u) = \mathit{Shift}(z)$, then read one input symbol, add the string z to the output string y, and go to step 3.
   (b) If $F(X, u) = \mathit{Reduce}\ i(z)$, then pop from the pushdown store the same number of symbols as that of input and nonterminal symbols on the right-hand side of the ith rule $(i) A \to \alpha$, and add string z to the output string y. Go to step 3.
   (c) If $F(X, u) = \mathit{Accept}$, then finish the translation. The output string y is the translation of the input string x provided that the input string is read completely. Otherwise, finish the translation by an error signalization.
   (d) If $F(X, u) = \mathit{Error}$, 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 $G(Y, W) = \mathit{Error}$, then finish the translation by an error signalization.

Algorithm 3.8 is an extension of the algorithm of *LR* parser ([1]). The extension is that the output strings are emitted during two operations of translation – shift and reduce.

A configuration of Algorithm 3.8 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 $(\#, x, \varepsilon)$, the accepting configuration is a triple $(\#M_i, \varepsilon, y)$, where M_i is the symbol at the top of the pushdown store, and it holds for M_i that $F(M_i, \varepsilon) = \mathit{Accept}$.

Example 3.9. Consider the R-translation grammar TG_1 from Example 3.5. The translation and goto tables for TG_1 are in Example 3.7. Algorithm 3.8 performs the following sequence of moves for input string *adb*:

$$\begin{array}{llll}
(\#, adb, \varepsilon) & \vdash & (\#a & , db & , x) \\
 & \vdash & (\#ad_2 & , b & , x) \\
 & \vdash & (\#aB_2 & , b & , xz) \\
 & \vdash & (\#aB_2b & , \varepsilon & , xz) \\
 & \vdash & (\#A & , \varepsilon & , xz) \\
 & \vdash & \mathit{Accept}. & &
\end{array}$$

Thus, for the R-translation grammar TG_1, we can create one-pass formal translator. Such an R-translation grammar will be called *Kernel*(R)-translation grammar and is defined as follows.

Definition 3.10. A translation grammar TG is called a *Kernel*(R)-translation grammar if and only if the following conditions hold:

1. TG is an R-translation grammar.
2. There is no shift-translation conflict in the collection of sets of $LR(k)$ translation items for TG.

The class of *Kernel*(R)-translation grammars is denoted as $C_{Kernel(R)}$.

4. COLLECTION OF $LR(k)$ TRANSLATION ITEMS AND TRANSFORMATIONS OF TRANSLATION GRAMMARS

In Section 3 we have shown that it is possible for a translation grammar with LR(k) input grammar to perform a one-pass formal translation if it is a *Kernel*(R)-translation grammar. What is for translation grammars which are not R-translation grammars or are R-translation grammars but the second condition in Definition 3.10 is not satisfied?

In this section we show that among the translation grammars which are not in the class of *Kernel*(R)-translation grammars are translation grammars which can be transformed so that the second condition would be satisfied. There are transformations enabling to transform some translation grammars into equivalent *Kernel*(R)-translation grammars. The transformations that can be used for this purpose are called shaking-down and postponing. In this section we also see which translation grammars can be transformed and which ones cannot be transformed. At the end of the section we will define the class called LR-translation grammars as an extension of the class of *Kernel*(R)-translation grammars.

Theorem 4.1. Let $TG = (N, T, D, R, S)$ be a translation grammar, and let R contain a rule $A \rightarrow \alpha x C \beta$, 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 the translation grammar $TG' = (N, T, D, R', S)$, in which $R' = (R - \{A \rightarrow \alpha x C \beta\}) \cup \{A \rightarrow \alpha C x \beta\}$, is equivalent to the translation grammar TG.

Proof. The proof of this theorem is in [13]. □

The transformation defined by Theorem 4.1 is called *postponing* of the string of output symbols.

Theorem 4.2. Let $TG = (N,T,D,R,S)$ be a translation grammar, where R contains a rule $A \to \alpha xB\beta$, $x \in D^+$, $\alpha, \beta \in (N \cup T \cup D)^*$, $B \in N$, 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 $R' = (R - \{A \to \alpha xB\beta\}) \cup \{A \to \alpha[xB]\beta, [xB] \to x\gamma_1|x\gamma_2|\cdots|x\gamma_n\}$. Then $Z(TG) = Z(TG')$.

Proof. The proof of this theorem is in [13]. □

The transformation defined by Theorem 4.2 is called *shaking-down* of the string of output symbols.

The transformation shaking-down and postponing can be used to transform some translation grammars into equivalent *Kernel* (R)-translation grammar.

Moreover, these transformations can be embeded in the process of construction of the collection of sets of $LR(k)$ translation items. During this process, translation conflicts are indicated and in some cases, they can be resolved by a transformation of rules of translation grammar. In all the following algorithms the two above mentioned transformations are used to remove translation conflicts, if possible, otherwise the algorithms finish with a failure signalization.

We know that the $LR(k)$ items associated with the viable prefixes of an $LR(k)$ grammar are the main key to create the action and goto tables of $LR(k)$ parser ([1]). $LR_{sp}(k)$ translation items defined below are also the key to create translation and goto tables of $LR(k)$ translator. The subscript *sp* stands for shaking-down and postponing transformations.

Definition 4.3. An *$LR_{sp}(k)$ translation item* for a translation grammar $TG = (N,T,D,R,S)$ is an object of the form $[A \to \alpha \cdot \beta, x^\delta, w]$, where:

$A \to \alpha\beta$ is a rule of translation grammar TG,

$x \in D^*$ is a string of output symbols,

$w \in T^{*k}$ is a string of input symbols, $k \geq 0$,

δ is a flag, and

$\delta = post$ means that x was and/or will be postponed,

$\delta = shake$ means that x was and/or will be shaken-down,

$\delta = out$ means that x will be appended to the string of output symbols during parsing.

For $k = 0$, the $LR_{sp}(0)$ translation item can be written in the form $[A \to \alpha \cdot \beta, x^\delta]$. The $LR_{sp}(k)$ translation item differs from the $LR(k)$ translation item in that the $LR_{sp}(k)$ translation item contains output symbols with a flag δ. By definition $\varepsilon = \varepsilon^{shake} = \varepsilon^{post} = \varepsilon^{out}$.

Definition 4.4. Let $TG = (N, T, D, R, S)$ be a translation grammar. The set of all nonterminal symbols of the translation grammar TG which generate only strings of input symbols is called *NPost* and formally defined as follows: $NPost(TG) = \{A\colon A \in N,\ A \Rightarrow^{+} w,\ w \in T^{*}\}$.

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 $A \to \alpha x C y \beta$ in R, where $\alpha, \beta \in (N \cup T \cup D)^{*}$, $x, y \in D^{*}$, $C \in (T \cup NPost)$, and α does not end with an output symbol. Let us have a set of translation items M that contains item $[A \to \alpha x \cdot C y \beta, x^{post}, u]$, β does not start with an output symbol. The set $\mathrm{GOTO}(M, C)$ contains item $[A \to \alpha x C y \cdot \beta, (xy)^{\delta}, u]$, where $x, y \in D^{*}$ and δ can be *shake*, *post* or *out*. This item corresponds to the rule $A \to \alpha C x y \beta$ that may be obtained by postponing the string x in the rule $A \to \alpha x C y \beta$.

Similarly, if there is an item $[B \to \alpha x \cdot C \beta, x^{shake}, u]$ in set M and $C \to y_1\gamma_1 | y_2\gamma_2 | \cdots | y_n\gamma_n \in R$, where $y_i \in D^{*}$, $\gamma_i \in (N \cup T)(N \cup T \cup D)^{*}$, $1 \le i \le n$, then translation items

$$\begin{array}{l}
[C \to y_1 \cdot \gamma_1, (xy_1)^{\delta}, v],\\
[C \to y_2 \cdot \gamma_2, (xy_2)^{\delta}, v],\\
\vdots\\
[C \to y_n \cdot \gamma_n, (xy_n)^{\delta}, v],
\end{array}$$

are in M, where $v \in \mathrm{FIRST}_k(h_i(\beta u))$ and δ can be *shake*, *post* or *out*, and they correspond to the following rules obtained by shaking-down of string x from the rule $B \to \alpha x C \beta$ to rules:

$$C \to x y_1 \gamma_1 | x y_2 \gamma_2 | \cdots | x y_n \gamma_n.$$

Before describing the algorithms constructing the collection of sets of $LR_{sp}(k)$ translation items for a given translation grammar we cite some cases that are associated with using postponing and shaking-down. A collection of sets of $LR(k)$ items is always finite regardless of whether the grammar is $LR(k)$ or not. Nevertheless, there are translation grammars for which the collection of sets of $LR_{sp}(k)$ translation items is an infinite collection of infinite sets. In this case, it is necessary to prevent such situation by indicating infinite loops in a construction algorithm. Let us demonstrate two sources of such infinite loops via examples.

Example 4.5. The first source of an infinite loop is a shaking-down in connection with an output symbol in front of a left-recursive nonterminal. Consider translation grammar $TG_2 = (\{A\}, \{a, b\}, \{x\}, R, A)$, whose set R contains the rules:

(1) $A \to xAa$

(2) $A \to b$.

The initial set of the collection of sets of $LR_{sp}(1)$ translation items for the grammar TG_2 is infinite, and some of the items it contains are:

$$\begin{array}{ll}
[S' \rightarrow \cdot A, \varepsilon, \varepsilon], & \\
[A \rightarrow x \cdot A\ a, x^{shake}, \varepsilon], & \\
[A \rightarrow \cdot b, \varepsilon, \varepsilon], & \\
[A \rightarrow x \cdot A\ a, (xx)^{shake}, a], & (*) \\
[A \rightarrow \cdot b, x^{post}, a], & \\
[A \rightarrow x \cdot A\ a, (xxx)^{shake}, a], & \\
[A \rightarrow \cdot b, (xx)^{post}, a], & \\
\vdots &
\end{array}$$

The complete initial set for the translation grammar TG_2 can be written as

$$\begin{array}{ll}
\# = \{ & [S' \rightarrow \cdot A, \varepsilon, \varepsilon], \\
& [A \rightarrow x \cdot A\ a, x^{shake}, \varepsilon], \\
& [A \rightarrow \cdot b, \varepsilon, \varepsilon], \\
& [A \rightarrow x \cdot A\ a, (x^n)^{shake}, a], \\
& [A \rightarrow \cdot b, (x^m)^{post}, a] : m \geq 1,\ n \geq 2\}.
\end{array}$$

This initial set of the collection of sets of $LR_{sp}(k)$ translation items for the translation grammar TG_2 is infinite for any $k \geq 0$.

There is a possibility to indicate an infinite loop during the construction of such infinite set. The loop only becomes infinite, if in some $LR_{sp}(k)$ translation item $[A \rightarrow \cdot\alpha, w, u]$, an output symbol positioned in the particular rule at a particular place appears in string w at least twice. In our example, symbol x from the first rule appears for the second time in item $(*)$.

Example 4.6. The second source of an infinite loop is a combination of postponing and shaking-down transformations in connection with a recursive nonterminal symbol. Consider a translation grammar $TG_3 = (\{A\}, \{a, b\}, \{x\}, R, A)$, whose set R contains rules

(1) $A \rightarrow xaA$

(2) $A \rightarrow b.$

The initial set of the collection of sets of $LR(1)$ translation items is finite, but there is an infinite number of sets $a_1 = \text{GOTO}(\#, a)$, $a_2 = \text{GOTO}(\text{GOTO}(\#, a), a)$, $a_3 = \text{GOTO}(\text{GOTO}(\text{GOTO}(\#, a), a), a), \ldots$, that differ from each other only in output strings. Initial set $\#$, and sets a_1, a_2 have the following form:

$$\begin{aligned}
\# &= \{\,[S' \to \cdot A, \varepsilon, \varepsilon],\\
&\quad [A \to x \cdot aA, x^{post}, \varepsilon],\\
&\quad [A \to \cdot b, \varepsilon, \varepsilon]\},\\
a_1 &= \{\,[A \to xa \cdot A, x^{shake}, \varepsilon],\\
&\quad [A \to x \cdot aA, (xx)^{post}, \varepsilon],\\
&\quad [A \to \cdot b, x^{out}, \varepsilon]\},\\
a_2 &= \{\,[A \to xa \cdot A, (xx)^{shake}, \varepsilon],\\
&\quad [A \to x \cdot aA, (xxx)^{post}, \varepsilon],\\
&\quad [A \to \cdot b, (xx)^{out}, \varepsilon]\}.
\end{aligned}$$

The number of sets

$$\begin{aligned}
a_i &= \{\,[A \to xa \cdot A, (x^i)^{shake}, \varepsilon],\\
&\quad [A \to x \cdot aA, (x^{i+1})^{post}, \varepsilon],\\
&\quad [A \to \cdot b, (x^i)^{out}, \varepsilon]\}
\end{aligned}$$

is infinite because there is such a set for each $i \geq 1$.

An infinite loop caused by a combination of the postponing and the shaking-down transformations can be indicated using the same principle as in case of presence of an output symbol in front of a left-recursive nonterminal. Again, if an output symbol from a particular place in some grammar rule appears at least twice in the output string of some $LR_{sp}(k)$ translation item, an infinite loop starts. In Example 4.6, we can indicate that the output symbol x occurs twice in the second item of set a_1. This symbol is the first symbol of the right-hand side in the first rule of the translation grammar TG_3.

To enable an indication of infinite loops, we append to each output symbol some information about its location in translation grammar rules. The location of an output symbol is a pair (r, p), where r is the number of the grammar rule, and p is the position of the output symbol within the right-hand side of this rule. Positions in the right-hand side of a rule are numbered naturally by 1, 2, ...

As the algorithm constructing the collection of sets of LR_{sp} translation items includes the postponing and shaking-down transformations, besides *shift-translation conflict* mentioned in Section 3 the following translation conflicts can occur during its construction.

Definition 4.7. In a set M of $LR_{sp}(k)$ translation items, there is

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

There are several algorithms using miscellaneous combinations of two above-mentioned transformations in the process of construction of the collection of sets of LR_{sp} translation items. Each such combination defines a particular class of translation grammars:

1. the algorithm using only the shaking-down transformation ([10]) and defining class C_S,
2. the algorithm using only the postponing transformation ([3]) and defining class C_P,
3. the algorithm using both the postponing and shaking-down transformations, but the postponing is used when the shaking-down transformation cannot be used ([3]) and defining class C_{SP},
4. the algorithm using both the postponing and shaking-down transformations but the shaking-down transformation is used when the postponing cannot be used ([3]), and defining class C_{PS} and
5. the algorithms using both the postponing and shaking-down transformations which are used with the same priority, and defining class C_{SEP}.

The following algorithm ([3, 10]) uses only shaking-down transformation. Thus, an output symbol will be emitted if it is in front of an input symbol and will be shaken down if it is in front of a nonterminal symbol. The shaking-down transformation affects mainly the computation of the closure. Therefore, we can use Algorithm 3.4 where the operation *CLOSURE* will be substituted by the operation *SCLOSURE*, where S stands for the shaking-down. Moreover, during the computation of the base of a new set we must set into the item $[A \to \alpha X y \cdot \beta, \gamma^{\delta}, u]$ the value of flag δ:

$\delta = shake$ when $\beta \in N(N \cup T \cup D)^*$,
$\delta = out$ when $\beta \in T(N \cup T \cup D)^* \cup \{\varepsilon\}$.
We will denote the resulting collection of sets of $LR_{sp}(k)$ translation items by$S(TG)$.

Algorithm 4.8. Construction of *SCLOSURE* of $LR_{sp}(k)$ translation items.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules are numbered, $k \geq 0$, and a set I of $LR_{sp}(k)$ translation items.

Output: *SCLOSURE(I)*.

Method:

a) $SCLOSURE(I) := I$.

b) If $[A \to \alpha \cdot B\beta, y^{shake}, u] \in SCLOSURE(I)$, $B \in N$ and $B \to zC\gamma \in R$, $\alpha, \beta, \gamma \in (N \cup T \cup D)^*$, $y, z \in D^*$, $C \in (N \cup T)$ or $C\beta = \varepsilon$, then $SCLOSURE(I) := SCLOSURE(I) \cup \{[B \to z \cdot C\beta, (yz)^{\delta}, v]$: $v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$, $\delta = shake$ when $C \in N$, $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$;

check in the new created $LR_{sp}(k)$ translation item whether some output symbol from a particular place in some grammar rule appears at least twice in the output string. If there is such a symbol, then finish the computation with a failure signalization of an infinite loop.

c) Repeat step b) until no new translation item can be inserted into the set $SCLOSURE(I)$.

Example 4.9. Consider translation grammar $TG_4 = (\{S, A, B\}, \{c, d\}, \{x, y\}, R, S)$ where set R contains the following rules:

(1) $S \to xA$

(2) $S \to yB$

(3) $A \to c$

(4) $B \to d.$

This translation grammar describes translation $Z(TG_4) = \{(cd, x), (db, y)\}$.
The collection of sets $S(TG_4)$ of $LR_{sp}(1)$ translation items contains the following sets:

$$\begin{aligned}
\# &= \{\, [S' \to \cdot S, \varepsilon, \varepsilon], \\
&\quad\;\; [S \to x \cdot A, x^{shake}, \varepsilon], \\
&\quad\;\; [S \to y \cdot B, y^{shake}, \varepsilon], \\
&\quad\;\; [A \to \cdot c, x^{out}, \varepsilon], \\
&\quad\;\; [B \to \cdot d, y^{out}, \varepsilon]\}, \\
S &= \{\, [S' \to S\cdot, \varepsilon, \varepsilon]\}, \\
A &= \{\, [S \to xA\cdot, \varepsilon, \varepsilon]\}, \\
B &= \{\, [S \to yB\cdot, \varepsilon, \varepsilon]\}, \\
c &= \{\, [A \to c\cdot, \varepsilon, \varepsilon]\}, \\
d &= \{\, [A \to d\cdot, \varepsilon, \varepsilon]\}.
\end{aligned}$$

There is no translation conflict in the set $S(TG_4)$. Output symbols x and y has been shaken down to the items $[A \to \cdot c, x, \varepsilon]$ and $[B \to \cdot d, y, \varepsilon]$, respectively.

Algorithm 4.11 ([3, 4]) uses only the postponing transformation. An output symbol will be emitted if it is in front of an input symbol and it is not in a shift-translation conflict. An output symbol must be postponed if it is in front of a nonterminal symbol or it is in a shift-translation conflict. We know that an output symbol cannot be postponed if it is in front of a nonterminal symbol that produces an output string. In such a case we can finish immediately the computation with an error signalization. The flag δ in an $LR_{sp}(k)$ translation item of the form $[A \to \alpha \cdot C\beta, x^{\delta}, u]$ will be *post* if $C \in NPost$ or *out* if $C \in T$ or $C\beta = \varepsilon$. The postponing transformation affects both the computation of bases and closures of sets of $LR(k)$ translation items and therefore we will define the operation *PCLOSURE*.

Algorithm 4.10. Construction of *PCLOSURE* of $LR_{sp}(k)$ translation items.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules are numbered, $k \geq 0$, and a set I of $LR_{sp}(k)$ translation items.

Output: *PCLOSURE*(I).

Method:

a) *PCLOSURE*$(I) := I$.

b) If $[A \to \alpha \cdot B\beta, y^{post}, u] \in PCLOSURE(I)$, $B \in N$ and $B \to zC\gamma \in R$, $\alpha, \beta, \gamma \in (N \cup T \cup D)^*$, $y, z \in D^*$, $C \in (N \cup T)$ or $C\beta = \varepsilon$, if $B \notin NPost$ then finish the computation with an error signalization else $PCLOSURE(I) := PCLOSURE(I) \cup \{[B \to z \cdot C\gamma, (yz)^{\delta}, v]$: $v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$, $\delta = post$ when $C \in N$, $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$;
check in the new created $LR_{sp}(k)$ translation item whether some output symbol from a particular place in some grammar rule appears at least twice in the output string. If there is such a symbol, then finish the computation with a failure signalization of an infinite loop.

c) Repeat step b) until no new translation item can be inserted into the set *PCLOSURE*(I).

Algorithm 4.11. Construction of the collection P of sets of $LR_{sp}(k)$ translation items. The postponing transformation is used.

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 $LR_{sp}(k)$ translation items for the translation grammar TG, or a reduction-translation conflict, or an error signalization.

Method:

1. Construct an augmented grammar $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \to S\}, S')$.
2. Construct a set *NPost* for TG'.
3. Construct the initial set of $LR_{sp}(k)$ translation items as follows:
   (a) $\# := [S' \to \cdot S, \varepsilon, \varepsilon]$.
   (b) $\# := PCLOSURE(\#)$.
   (c) Check all translation items of the set $\#$ with an input symbol following immediately the dot, whether there is a *shift-translation conflict.* If yes, then rewrite *out* of all translation items in the shift-translation conflict to *post*.
   Otherwise $P := \{\#\}$, $\#$ is the initial set.

4. If the set M_i of $LR_{sp}(k)$ translation items has been constructed, construct for each symbol $X \in (N \cup T)$ which is in some $LR_{sp}(k)$ translation item in M_i just behind the dot, a new set of $LR_{sp}(k)$ translation items X_j in the following way:

   (a) Construction of the basis of the set X_j is divided into two cases:
      i. $X_j := \{[A \to \alpha Xy \cdot C\beta, (xy)^\delta, u]$: $[A \to \alpha \cdot XyC\beta, x^{post}, u] \in M_i$, $y \in D^*$, $x \in D^+$, $X \in (NPost \cup T)$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.
      ii. $X_j := \{[A \to \alpha Xy \cdot C\beta, y^\delta, u]$: $[A \to \alpha \cdot XyC\beta, x^{out}, u] \in M_i$, $x, y \in D^*$, $X \in T$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.

      In both cases $\delta = post$ when $C \in NPost$, $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$.

   (b) $X_j := PCLOSURE(X_j)$.

   (c) Check if there is a *shift-translation conflict* in the set X_j. If yes, then rewrite *out* over the output symbols in these translation items to *post*. Otherwise $P := P \cup \{X_j\}$.

5. Repeat step 4 for all sets X_j until no new set can be added into the collection P.

Example 4.12. Consider translation grammar $TG_5 = (\{S, A\}, \{a, b, c\}, \{x, y\}, R, S)$, where set R contains the following rules:

(1) $S \to xAa$

(2) $S \to yAb$

(3) $A \to c$.

Construct the collection $P(TG_5)$ of sets of $LR_{sp}(1)$ translation items as follows:
$N_{POST} = \{A\}$,

$$
\begin{aligned}
\# &= \{[S' \to \cdot S, \varepsilon, \varepsilon],\\
&\quad\; [S \to x \cdot Aa, x^{post}, \varepsilon],\\
&\quad\; [S \to y \cdot Ab, y^{post}, \varepsilon],\\
&\quad\; [A \to \cdot c, \varepsilon, \{a, b\}]\},\\
A &= \{[S \to xA \cdot a, x^{out}, \varepsilon],\\
&\quad\; [S \to yA \cdot b, y^{out}, \varepsilon]\},\\
a &= \{[S \to xAa \cdot, \varepsilon, \varepsilon]\},\\
b &= \{[S \to yAb \cdot, \varepsilon, \varepsilon]\},\\
c &= \{[A \to c \cdot, \varepsilon, \{a, b\}]\}.
\end{aligned}
$$

The next algorithm ([3, 4]) uses the shaking-down and postponing transformations. Shaking-down is always used if an output symbol is in front of a nonterminal symbol. Postponing is used if there is a shift-translation conflict. The flag δ in an $LR_{sp}(k)$ translation item of the form $[A \to \alpha \cdot C\beta, x^\delta, u]$ will be *shake* if $C \in N$ or *out* if $C \in T$ or $C\beta = \varepsilon$.

Algorithm 4.13. Construction of the collection SP of sets of $LR_{sp}(k)$ translation items. The shaking-down and postponing transformations are used. The shaking-down transformation has greater priority than the postponing transformation.

`Input:` Translation grammar $TG = (N, T, D, R, S)$, where rules in R are numbered, and $k \geq 0$.

`Output:` Collection SP of sets of $LR_{sp}(k)$ translation items for the translation grammar TG, or a reduction-translation conflict.

`Method:`

1. Construct an augmented grammar $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \rightarrow S\}, S')$.
2. Construct the initial set of $LR_{sp}(k)$ translation items as follows:
   (a) $\# := [S' \rightarrow .S, \varepsilon, \varepsilon]$.
   (b) $\# := SCLOSURE(\#)$.
   (c) Check all translation items of the set $\#$ with an input symbol following immediately the dot, whether there is a *shift-translation conflict*. If yes, then rewrite *out* of all translation items in the shift-translation conflict to *post*.
   (d) Check if there is a *reduction-translation conflict*. If yes, then finish the computation with a failure signalization of a reduction-translation conflict. Otherwise $SP := \{\#\}$, $\#$ is the initial set.
3. If the set M_i of $LR_{sp}(k)$ translation items has been constructed, construct for each symbol $X \in (N \cup T)$ which is in some $LR_{sp}(k)$ translation item in M_i just behind the dot, a new set of $LR_{sp}(k)$ translation items X_j in the following way:
   (a) Construction of the basis of the set X_j is divided into three cases:
      i. $X_j := \{[A \rightarrow \alpha X y \cdot C\beta, y^{\delta}, u]$: $[A \rightarrow \alpha \cdot XyC\beta, x^{shake}, u] \in M_i$, $x, y \in D^*$, $X \in N$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.
      ii. $X_j := \{[A \rightarrow \alpha X y \cdot C\beta, y^{\delta}, u]$: $[A \rightarrow \alpha \cdot XyC\beta, x^{out}, u] \in M_i$, $x, y \in D^*$, $X \in T$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.
      iii. $X_j := \{[A \rightarrow \alpha X y \cdot C\beta, (xy)^{\delta}, u]$: $[A \rightarrow \alpha \cdot XyC\beta, x^{post}, u] \in M_i$, $x \in D^+$, $y \in D^*$, $X \in T$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.

      In all cases above $\delta = shake$ when $C \in N$ and $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$.
   (b) $X_j := SCLOSURE(X_j)$.
   (c) Check if there is a *shift-translation conflict* in the set X_j. If yes, then rewrite *out* over the output symbols in these translation items to *post*.
   (d) Check if there is a *reduction-translation conflict*. If yes, then finish the computation with a failure signalization of a reduction-translation conflict. Otherwise $SP := SP \cup \{X_j\}$.
4. Repeat step 3 for all sets X_j until no new set can be added into the collection SP.

Example 4.14. Consider translation grammar

$TG_6 = (\{S, A\}, \{a, b, c\}, \{x, y\}, R, S)$, where set R contains the following rules:

(1) $S \to xBa$

(2) $S \to yCb$

(3) $B \to zD$

(4) $C \to wE$

(5) $D \to c$.

Construct the collection $SP(TG_6)$ of sets of $LR_{sp}(1)$ translation items as follows:

$$
\begin{aligned}
N_{POST} &= \{D\}, \\
\# &= \{[S' \to \cdot S, \varepsilon, \varepsilon], \\
&\quad [S \to x \cdot Ba, x^{shake}, \varepsilon], \\
&\quad [S \to y \cdot Cb, y^{shake}, \varepsilon], \\
&\quad [B \to z \cdot D, (xz)^{shake}, a], \\
&\quad [C \to w \cdot D, (yw)^{shake}, b], \\
&\quad [D \to \cdot c, (xz)^{post}, a], \\
&\quad [D \to \cdot c, (yw)^{post}, b]\}, \\
S &= \{[S' \to S \cdot, \varepsilon, \varepsilon]\}, \\
B &= \{[S \to xB \cdot a, \varepsilon, \varepsilon]\}, \\
C &= \{[S \to yC \cdot b, \varepsilon, \varepsilon]\}, \\
D &= \{[B \to zD \cdot, \varepsilon, a], \\
&\quad [C \to wD \cdot, \varepsilon, b]\}, \\
c &= \{[D \to c \cdot, (xz)^{out}, a], \\
&\quad [D \to C \cdot, (yw)^{out}, b]\}, \\
a &= \{[S \to xBa \cdot, \varepsilon, \varepsilon]\}, \\
b &= \{[S \to yCb \cdot, \varepsilon, \varepsilon]\}.
\end{aligned}
$$

Algorithm 4.16 ([3, 4]) uses shaking-down and postponing. The shaking-down transformation is used if an output symbol is in front of a nonterminal symbol over which the postponing transformation cannot be used. The flag δ in an $LR_{sp}(k)$ translation item of the form $[A \to \alpha \cdot C\beta, x^{\delta}, u]$ will be *shake* if $C \in (N - NPost)$, *post* if $C \in NPost$ or *out* if $C \in T$ or $C\beta = \varepsilon$. We must cover two situations in this algorithm, because the output strings may be either shaken down or postponed.

This is the reason to define another version of the closure computation.

Algorithm 4.15. Construction of PSCLOSURE of $LR_{sp}(k)$ translation items.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules are numbered, $k \geq 0$, and a set I of $LR_{sp}(k)$ translation items.

Output: $PSCLOSURE(I)$.

Method:

a) $PSCLOSURE(I) := I$.

b) If $[A \to \alpha \cdot B\gamma, x^{shake}, u] \in PSCLOSURE(I)$, $B \in (N - NPost)$ and $B \to yC\beta$, $x, y \in D^*$, $\alpha, \beta, \gamma \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}$, then $PSCLOSURE(I) := PSCLOSURE(I) \cup \{[B \to y \cdot C\beta, (xy)^{\delta}, v]$: $v \in \mathrm{FIRST}_k(h_i(\gamma)u)\}$, $\delta = post$ when $C \in NPost$, $\delta = shake$ when $C \in (N - NPost)$ and $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$.
Check in the new created $LR_{sp}(k)$ translation item whether some output symbol from a particular place in some grammar rule appears at least twice in the output string. If there is such a symbol, then finish the computation with a failure signalization of an infinite loop.

c) If $[A \to \alpha \cdot B\gamma, x^{post}, u] \in PSCLOSURE(I)$, $x \in D^+$, α, $\gamma \in (N \cup T \cup D)^*$, $B \in NPost$ and $B \to \beta$, $\beta \in (NPost \cup T)^*$, then $PSCLOSURE(I) := PSCLOSURE(I) \cup \{[B \to \cdot\beta, \varepsilon, v]$: $v \in \mathrm{FIRST}_k(h_i(\gamma)u)\}$.

d) Repeat steps b) and c) until no new translation item can be inserted into the set $PSCLOSURE(I)$.

Algorithm 4.16. Construction of the collection PS of sets of $LR_{sp}(k)$ translation items. Shaking-down and postponing transformations are used. The postponing transformation has greater priority than the shaking-down transformation.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules in R are numbered, and $k \geq 0$.

Output: Collection PS of sets of $LR_{sp}(k)$ translation items for the translation grammar TG, or a reduction-translation conflict.

Method:

1. Construct an augmented grammar $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \to S\}, S')$.
2. Construct a set $NPost$ for TG'.
3. Construct the initial set of $LR_{sp}(k)$ translation items as follows:
   (a) $\# := [S' \to \cdot S, \varepsilon, \varepsilon]$.
   (b) $\# := PSCLOSURE(\#)$

(c) Check all translation items of the set # with an input symbol following immediately the dot, whether there is a *shift-translation conflict.* If yes, then rewrite *out* of the all translation items in the shift-translation conflict to *post.*

(d) Check if there is a *reduction-translation conflict.* If yes, then finish the computation with a failure signalization of a reduction-translation conflict. Otherwise $PS := \{\#\}$, # is the initial set.

4. If the set M_i of $LR_{sp}(k)$ translation items has been constructed, construct for each symbol $X \in (N \cup T)$ which is in some $LR_{sp}(k)$ translation item in M_i just behind the dot, a new set of $LR_{sp}(k)$ translation items X_j in the following way:

   (a) Construction of the basis of the set X_j is divided into three cases:

      i. $X_j := \{[A \to \alpha Xy \cdot C\beta, y^\delta, u]$: $[A \to \alpha \cdot XyC\beta, x^{shake}, u] \in M_i$, $x, y \in D^*$, $X \in (N - NPost)$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.

      ii. $X_j := \{[A \to \alpha Xy \cdot C\beta, y^\delta, u]$: $[A \to \alpha \cdot XyC\beta, x^{out}, u] \in M_i$, $x, y \in D^*$, $X \in T$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.

      iii. $X_j := \{[A \to \alpha Xy \cdot C\beta, (xy)^\delta, u]$: $[A \to \alpha \cdot XyC\beta, x^{post}, u] \in M_i$, $x \in D^+$, $y \in D^*$, $X \in (NPost \cup T)$, $\alpha, \beta \in (N \cup T)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.

   In all cases above $\delta = post$ when $C \in NPost$, $\delta = shake$ when $C \in (N - NPost)$ and $\delta = out$ when $C \in T$ or $C\beta = \varepsilon$.

   (b) $X_j := PSCLOSURE(X_j)$

   (c) Check if there is a *shift-translation conflict* in the set X_j. If yes, then rewrite *out* over the output symbols in these translation items to *post.*

   (d) Check if there is a *reduction-translation conflict.* If yes, then finish the computation with a failure signalization of a reduction-translation conflict. Otherwise $PS := PS \cup \{X_j\}$.

5. Repeat step 4 for all sets X_j until no new set can be added into the collection PS.

Example 4.17. Consider translation grammar $TG_7 = (\{S, A, B, C\}, \{a, b, c\}, \{x, y, z\}, R, S)$, whose set R contains the rules:

(1) $S \to xAb$	(4) $B \to Bb$
(2) $S \to yAc$	(5) $B \to b$
(3) $A \to BzC$	(6) $C \to a$.

Construct the collection $SP(TG_7)$ of sets of $LR_{sp}(1)$ translation items as follows:

NPost= $\{B, C\}$,

$$
\begin{aligned}
\# = \{ & [S' \to \cdot S, \varepsilon, \varepsilon], \\
& [S \to x \cdot Ab, x^{shake}, \varepsilon], \\
& [S \to y \cdot Ac, y^{shake}, \varepsilon], \\
& [A \to \cdot BzC, x^{post}, b], \\
& [A \to \cdot BzC, y^{post}, c], \\
& [B \to \cdot Bb, \varepsilon, \{a, b\}], \\
& [B \to \cdot b, \varepsilon, \{a, b\}]\},
\end{aligned}
$$

$$S = \{[S' \to S \cdot \varepsilon, \varepsilon]\},$$

$$
\begin{aligned}
A = \{ & [S \to xA \cdot b, \varepsilon, \varepsilon], \\
& [S \to yA \cdot c, \varepsilon, \varepsilon]\},
\end{aligned}
$$

$$
\begin{aligned}
B = \{ & [A \to Bz \cdot C, (xz)^{post}, b], \\
& [A \to Bz \cdot C, (yz)^{post}, c], \\
& [B \to B \cdot b, \varepsilon, \{a, b\}], \\
& [C \to \cdot a, \varepsilon, \{b, c\}]\},
\end{aligned}
$$

$$b_1 = \{[B \to b \cdot \varepsilon, \{a, b\}]\},$$

$$b_2 = \{[S \to xAb \cdot \varepsilon, \varepsilon]\},$$

$$c = \{[S \to yAc \cdot, \varepsilon, \varepsilon]\},$$

$$
\begin{aligned}
C = \{ & [A \to BzC \cdot, (xz)^{out}, b], \\
& [A \to BzC \cdot, (yz)^{out}, c]\},
\end{aligned}
$$

$$a = \{[C \to a \cdot, \varepsilon, \{b, c\}]\},$$

$$b_3 = \{[B \to Bb \cdot, \varepsilon, \{a, b\}]\}.$$

Algorithm 4.20 prevents the translation conflicts during constructing a set of $LR_{sp}(k)$ translation items. If the created set includes an *expansion translation* conflict, then the algorithm uses postponing to remove it.

Definition 4.18. Let $TG = (N, T, D, R, S)$ be a translation grammar and M is a set of $LR_{sp}(k)$ translation items for TG. The set of all nonterminal symbols of TG, which are in *NPost* and induce an expansion-translation conflict in M, is called *NConflict*$_M$ and formally defined as follows: *NConflict*$_M$ = $\{A$: $A \in$ *NPost* and there is an expansion-translation conflict for A in $M\}$.

Algorithm 4.19. Construction of *ECLOSURE* of $LR(k)$ translation items.

`Input:` Translation grammar $TG = (N, T, D, R, S)$, where rules are numbered, $k \geq 0$, and a set I of $LR_{sp}(k)$ translation items.

`Output:` $ECLOSURE(I)$.

`Method:`

a) $ECLOSURE(I) := I$

b) If $[A \to \alpha \cdot B\gamma, y^{shake}, u] \in ECLOSURE(I)$, $B \in N$ and $B \to zC\beta$, $y, z \in D^*$, $\alpha, \beta, \gamma \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}$ then

   i. If $C \in NConflict_{ECLOSURE(I)}$, then $ECLOSURE(I) := ECLOSURE(I) \cup \{[B \to z \cdot C\beta, (yz)^{post}, v]\colon v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$.

   ii. If $C \in (N - NConflict_{ECLOSURE(I)})$, then $ECLOSURE(I) := ECLOSURE(I) \cup \{[B \to z \cdot C\beta, (yz)^{shake}, v]\colon v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$.
   Check in the new created $LR_{sp}(k)$ translation item whether some output symbol from a particular place in some grammar rule appears at least twice in the output string. If there is such a symbol, then finish the computation with a failure signalization of an infinite loop. Check the new translation item with all others for any *expansion-translation conflict.* If yes, and $C \notin NPost$, then finish the computation with a failure signalization of an expansion-translation conflict. If yes, and $C \in NPost$, then put C into $NConflict_{ECLOSURE(I)}$, rewrite *shake* over the output symbols in all translation items with the nonterminal symbol C following immediately after the dot to *post* and remove all output symbols in all translation items deduced from C.

   iii. If $C \in T$ or $C\beta = \varepsilon$, then $ECLOSURE(I) := ECLOSURE(I) \cup \{[B \to z \cdot C\beta, (yz)^{out}, v]\colon v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$.

c) If $[A \to \alpha \cdot B\gamma, y^{post}, u] \in ECLOSURE(I)$, $\alpha, \gamma \in (N \cup T \cup D)^*$, $y \in D^+$, $B \in NPost$ and $B \to \beta$, $\beta \in (NPost \cup T)^*$, then $ECLOSURE(I) := ECLOSURE(I) \cup \{[B \to \cdot\beta, \varepsilon, v]\colon v \in \mathrm{FIRST}_k(h_i(\alpha)u)\}$.

d) Repeat steps (d) and (e) until no new translation item can be inserted into the set $ECLOSURE(I)$.

Algorithm 4.20. Construction of the collection *SEP* of sets of $LR_{sp}(k)$ translation items. The shaking-down and postponing transformations are used. The shaking-down and postponing transformations have the same priority.

Input: Translation grammar $TG = (N, T, D, R, S)$, where rules in R are numbered, and $k \geq 0$.

Output: Collection SEP of sets of $LR_{sp}(k)$ translation items for the translation grammar TG, or a reduction-translation conflict, or an expansion-translation conflict.

Method:

1. Construct an augmented grammar $TG' = (N \cup \{S'\}, T, D, R \cup \{S' \to S\}, S')$.
2. Construct a set *NPost* for TG'.
3. Construct the initial set of $LR_{sp}(k)$ translation items as follows:
   (a) $\# := [S' \to \cdot S, \varepsilon, \varepsilon]$, $NConflict_{\#} = \emptyset$.
   (b) $\# := ECLOSURE(\#)$.
   (c) Check all translation items of the set $\#$ with input symbol following immediately the dot for any *shift-translation conflict.* If yes, then rewrite *out* of all translation items in the shift-translation conflict to *post.*
   (d) Check if there is a *reduction-translation conflict.* If yes, then finish the computation with a failure signalization of a reduction-translation conflict.
   Otherwise $SEP := \{\#\}$, $\#$ is the initial set.
4. If the set M_i of $LR_{sp}(k)$ translation items has been constructed, construct for each symbol $X \in (N \cup T)$ which is in some $LR_{sp}(k)$ translation item in M_i just behind the dot, a new set of $LR_{sp}(k)$ translation items X_j in the following way:
   (a) Put $NConflict_{X_j} = \emptyset$ and the basis of the set X_j is constructed as follows:
      i. $X_j := \{[A \to \alpha X y \cdot C\beta, y^{\delta}, u]$: $[A \to \alpha \cdot XyC\beta, x^{shake}, u] \in M_i$, $y \in D^*$, $x \in D^+$, $X \in N$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$,
      ii. $X_j := \{[A \to \alpha X y \cdot C\beta, (xy)^{\delta}, u]$: $[A \to \alpha \cdot XyC\beta, x^{post}, u] \in M_i$, $y \in D^*$, $x \in D^+$, $X \in (T \cup NPost)$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$,
      iii. $X_j := \{[A \to \alpha X y \cdot C\beta, y^{\delta}, u]$: $[A \to \alpha \cdot XyC\beta, x^{out}, u] \in M_i$, $x, y \in D^*$, $X \in T$, $\alpha, \beta \in (N \cup T \cup D)^*$, $C \in (N \cup T) \cup \{\varepsilon\}\}$.
   (b) Check the basis of the set X_j, whether there is an *expansion-translation conflict.* If yes, and $C \notin NPost$, then finish the computation with a failure signalization of a reduction-translation conflict. If yes, and $C \in NPost$, then put C into $NConflict_{X_j}$. This step is repeated for all nonterminal symbols following immediately after the dot.
   (c) The computation the δ after steps (a) and (b) is as follows. If $C \in NConflict_{X_j}$, then $\delta = post$. If $C \in (N - NConflict_{X_j})$, then $\delta = shake$. If $C \in T$ or $C\beta = \varepsilon$, then $\delta = out$.

(d) $X_j := ECLOSURE(X_j)$.

(e) Check if there is a *shift-translation conflict* in the set X_j. If yes, then rewrite *out* over the output symbols in these translation items to *post.*

(f) Check if there is a *reduction-translation conflict.* If yes, then finish the computation with a failure signalization of a reduction-translation conflict.
Otherwise $SEP := SEP \cup \{X_j\}$.

5. Repeat step 4 for all sets X_j until no new set can be added into the collection SEP.

Example 4.21. Consider translation grammar $TG_8 = (\{S, A, B, C\}, \{a, b, c\}, \{x, y, z\}, R, S)$, whose set R contains the rules:

(1) $S \to xAb$ (4) $B \to Bb$

(2) $S \to yAc$ (5) $B \to b$

(3) $A \to BzC$ (6) $C \to a$.

The collection $SEP(TG_8)$ of sets of $LR_{sp}(1)$ translation items for the grammar TG_8 is constructed as follows:

NPost$= \{B, C\}$,

$$
\begin{aligned}
\# = \{ & [S' \to \cdot S, \varepsilon, \varepsilon], \\
& [S \to x \cdot Ab, x^{shake}, \varepsilon], \\
& [S \to y \cdot Ac, y^{shake}, \varepsilon], \\
& [A \to \cdot BzC, x^{shake}, b], \\
& [A \to \cdot BzC, y^{shake}, c],\}.
\end{aligned}
$$

NConflict$_{\#}=\{B\}$.

Items $[A \to \cdot BzC, x^{shake}, b]$ and $[A \to \cdot BzC, y^{shake}, c]$ of the closure of the set $\#$ are in the *expansion-translation conflict.* The nonterminal symbol B following the dot is the element of the set *NPost*, therefore we can postpone the output symbol over the nonterminal symbol B.

$$
\begin{aligned}
\# = \{ & [S' \to \cdot S, \varepsilon, \varepsilon], \\
& [S \to x \cdot Ab, x^{shake}, \varepsilon], \\
& [S \to y \cdot Ac, y^{shake}, \varepsilon], \\
& [A \to \cdot BzC, x^{post}, b], \\
& [A \to \cdot BzC, y^{post}, c], \\
& [B \to \cdot Bb, \varepsilon, \{a, b\}], \\
& [B \to \cdot b, \varepsilon, \{a, b\}]\},
\end{aligned}
$$

$S = \{[S' \to S \cdot, \varepsilon, \varepsilon]\}$,

$A = \{[S \to xA \cdot b, \varepsilon, \varepsilon]$,

$$
\begin{aligned}
& [S \to yA \cdot c, \varepsilon, \varepsilon]\}, \\
B = & \{[A \to Bz \cdot C, (xz)^{shake}, b], \\
& [A \to Bz \cdot C, (yz)^{shake}, c], \\
& [B \to B \cdot b, \varepsilon, \{a, b\}], \\
& [C \to \cdot a, (xz)^{out}, b], \\
& [C \to \cdot a, (yz)^{out}, c]\},
\end{aligned}
$$

Items $[C \to \cdot a, (xz)^{out}, b]$ and $[C \to \cdot a, (yz)^{out}, c]$ of the closure of set B are in the *shift-translation conflict*. We postpone the output symbols over the terminal symbol a.

$$
\begin{aligned}
B = & \{[A \to Bz \cdot C, (xz)^{shake}, b], \\
& [A \to Bz \cdot C, (yz)^{shake}, c], \\
& [B \to B \cdot b, \varepsilon, \{a, b\}], \\
& [C \to \cdot a, (xz)^{post}, b], \\
& [C \to \cdot a, (yz)^{post}, c]\}, \\
b_1 = & \{[B \to b\cdot, \varepsilon, \{a, b\}]\}, \\
b_2 = & \{[S \to xAb\cdot, \varepsilon, \varepsilon]\}, \\
c = & \{[S \to yAc\cdot, \varepsilon, \varepsilon]\}, \\
C = & \{[A \to BzC\cdot, \varepsilon, b], \\
& [A \to BzC\cdot, \varepsilon, c]\}, \\
a = & \{[C \to a\cdot, (xz)^{out}, b], \\
& [C \to a\cdot, (yz)^{out}, c]\}, \\
b_3 = & \{[B \to Bb\cdot, \varepsilon, \{a, b\}]\}.
\end{aligned}
$$

So that there is no translation conflict in the collection $SEP(TG_8)$ of sets of translation items.

Lemma 4.22. Algorithm 4.20 either constructs a finite collection of finite sets of $LR(k)$ translation items, or indicates an infinite loop and stops for each translation grammar.

Proof. The proof is very similar to the proof in [11, 14]. Algorithm 4.20 is an extension of the algorithm for the construction of the collection of sets of $LR(k)$ items for a context-free grammar. This collection is always finite. An $LR_{sp}(k)$ translation item is an $LR(k)$ item augmented with a string of output symbols with a flag. If an output symbol from a particular location in a grammar rule appears more than once in some $LR_{sp}(k)$ translation item of a set of $LR_{sp}(k)$ translation items M, then either a shaking-down transformation is used by rules with output symbol before left-recursive nonterminal or a combination of postponing and skaking-down transformations in connection with a recursive nonterminal symbol. The algorithm either removes translation conflicts by using a postponing transformation or finishes with a failure signalization.

The length of the string of output symbols in each $LR_{sp}(k)$ translation item is limited by the number of output symbols appearing at distinct positions in the grammar rules. As the number of output symbols is limited in all grammar rules, the length of the output string in each $LR_{sp}(k)$ translation item is limited too. □

Lemma 4.23. For an $LR(k)$ translation grammar TG, and for an input string x, Algorithm 3.8 using the translation table based upon the collection SEP of sets of $LR_{sp}(k)$ translation items constructed by Algorithm 4.20 creates an output string y such that $(x, y) \in Z(TG)$.

Proof. The proof is very similar to the proofs in [3, 11]. □

Definition 4.24. A translation grammar TG is called an $LR(k)$ translation grammar if the following conditions hold:

1. The input grammar of the translation grammar TG is an $LR(k)$ grammar.
2. There is no output symbol in front of a left-recursive nonterminal in TG.
3. There is no reduction-translation conflict in the collection of sets of $LR_{sp}(k)$ translation items.

For each $LR(k)$ translation grammar we can construct a deterministic bottom-up translator that operates in linear time.

5. THE HIERARCHY OF LR TRANSLATION GRAMMARS

The hierarchy of LR-translation grammars is described in the following theorems.

Theorem 5.1. $C_{postfix} \subset C_{Kernel(R)}$.

Proof. The containment is implied from their definitions. The following translation grammar $TG_{Kernel(R)} = (\{A\}, \{a\}, \{x\}, R, A)$ whose set R contains the rule

(1) $A \rightarrow xa.$

is in $C_{Kernel(R)}$ but not in $C_{postfix}$. Therefore, the inclusion is proper. □

Theorem 5.2. $C_{Kernel(R)} \subset C_S$.

Proof. Assume TG is a translation grammar and $TG \in C_{Kernel(R)}$. Algorithm 3.4 with the operation $SCLOSURE$ differs from Algorithm 3.4 in that, if there is an output symbol appearing immediately in front of a nonterminal symbol, Algorithm 3.4 with the operation $SCLOSURE$ uses the shaking-down transformation so that the output symbol will appear immediately before the input symbol while Algorithm

3.4 cannot compute the set of $LR_{sp}(k)$ translation items because the translation grammar is not an R-translation grammar. If the collection of sets of $LR_{sp}(k)$ translation items for TG is computed by using Algorithm 3.4, then no shaking-down is used and this collection does not include any shift-translation conflict because $TG \in C_{Kernel(R)}$. So that $TG \in C_S$.

The translation grammar $TG_S = (\{A, B\}, \{a, b\}, \{x, y\}, R, A)$ whose set R contains the rules

(1) $A \to xBa$

(2) $B \to yb$.

is in C_S but not in $C_{Kernel(R)}$. Therefore the inclusion is proper. □

Theorem 5.3. $C_S \subset C_{SP}$.

Proof. Assume TG is a translation grammar and $TG \in C_S$. Algorithm 4.13 differs from Algorithm 3.4 with the operation $SCLOSURE$ in that if in some set of $LR_{sp}(k)$ translation items is any shift-translation conflict, then Algorithm 4.13 uses the postponing transformation while Algorithm 3.4 with the operation $SCLOSURE$ finishes the computation with a failure signalization of a reduction-translation conflict. In other cases both algorithms behave in the same way. So $TG \in C_{SP}$.

The translation grammar $TG_{SP} = (\{A, B\}, \{a, b, c\}, \{x, y, z\}, R, A)$ whose set R contains the rules

(1) $A \to xBa$

(1) $A \to yBb$

(2) $B \to zc$.

is in C_{SP} but not in C_S. Therefore the inclusion is proper. □

Theorem 5.4. $C_{Kernel(R)} \subset C_P$.

Proof. Assume TG is a translation grammar and $TG \in C_{Kernel(R)}$. Algorithm 4.11 differs from Algorithm 3.4 in that if in some set of LR_{sp} translation items there is any shift-translation conflict, then Algorithm 4.11 uses the postponing transformation while Algorithm 3.4 finishes with a failure signalization of a shift-translation conflict. In other cases both algorithms behave in the same way. So $TG \in C_P$.

The translation grammar $TG_P = (\{S, A, B\}, \{a, b, c\}, \{x, y\}, R, S)$ whose set R contains the rules

(1) $S \to xA\ a$

(2) $S \to yA\ b$

(3) $A \to Bb$

(4) $B \to c$.

is in C_P but not in $C_{Kernel(R)}$. Therefore the inclusion is proper. □

Theorem 5.5. $C_P \subset C_{SP}$.

Proof. Assume TG is a translation grammar and $TG \in C_P$. Algorithm 4.13 differs from Algorithm 4.11 in that: if there is an output symbol appearing immediately in front of a nonterminal symbol, which produces a non-empty string of output symbols, Algorithm 4.13 uses the shaking-down transformation so that the output symbol will appear immediately before the symbol that is in $(NPost \cup T)$ while Algorithm 4.11 finishes computation with an error signalization. If the collection of sets of $LR_{sp}(k)$ translation items for TG is computed using Algorithm 4.13, then the shaking-down transformation need not to be used and this collection is the same as when it is computed by Algorithm 4.11. So $TG \in C_{SP}$.

The grammar TG_{SP} presented in the proof of Theorem 5.3 is in C_{SP} but not in C_P. Therefore the inclusion is proper. □

Theorem 5.6. $C_{Kernel(R)} \subset C_S \cap C_P$.

Proof. From Theorem 5.2 we have $C_{Kernel(R)} \subset C_S$. From Theorem 5.4 we have $C_{Kernel(R)} \subset C_P$. So that $C_{Kernel(R)} \subseteq C_S \cap C_P$.

The translation grammar $TG_{S \cap P} = (\{A, B\}, \{a, b\}, \{x\}, R, A)$ whose set R contains the rules

(1) $A \rightarrow xBa$

(3) $B \rightarrow b$.

is in $C_S \cap C_P$ but not in $C_{Kernel(R)}$. Therefore the inclusion is proper. □

Theorem 5.7. $C_{SP} \not\subseteq C_P$, $C_P \not\subseteq C_{SP}$.

Proof. The grammar TG_{SP} described in the proof of Theorem 5.3 is in C_{SP} but not in C_P.

The grammar TG_P described in the proof of Theorem 5.4 is in C_P but not in C_{SP}. □

Theorem 5.8. $C_{SP} \subset C_{SEP}$.

Proof. Assume TG is a translation grammar and $TG \in C_{SP}$. Algorithm 4.20 differs from Algorithm 4.13 in that if during the computation of the sets of $LR_{sp}(k)$ translation items there is any expansion-translation conflict, then Algorithm 4.20 uses the postponing transformation while Algorithm 4.13 uses the shaking-down transformation. It is known that if the shaking-down transformation is used for an output symbol which is in an expansion-translation conflict, then the collection includes a reduction-translation conflict. In other cases both algorithms behave in the same way. Because $TG \in C_{SP}$, no expansion-translation conflict occurs when the collection of sets of $LR_{sp}(k)$ translation items for TG is computed using Algorithm 4.20. So that $C_{SP} \subset C_{SEP}$.

The grammar TG_P presented in the proof of Theorem 5.4 is in C_{SEP} but not in C_{SP}. □

Theorem 5.9. $C_{SP} = C_{SEP}$.

Proof. We describe briefly the differences between Algorithm 4.20 and Algorithm 4.16. Algorithm 4.20 checks during the computation of the sets of $LR_{sp}(k)$ translation items for any expansion translation conflict. If there is such a conflict, then Algorithm 4.20 uses the postponing transformation to postpone the output symbols, otherwise it uses the shaking-down transformation for the output symbols. Algorithm 4.16 uses always the postponing transformation if it is possible. That is, the postponing transformation is also used in the case when in all sets of $LR_{sp}(k)$ translation items there is no expansion translation conflict (note: on this account we call Algorithm 4.16 a lazy variant of translation). In the case when an output symbol appears immediately before a grammar symbol that is in $((N - NPost) \cup T)$ both algorithms behave in the same way.

Now we prove Theorem 5.9 by contradition. Assume TG is a translation grammar and $TG \in C_{SP}$. Assume $TG \notin C_{SEP}$, i.e., Algorithm 4.20 finishes in a case when in a set M there is an expansion-translation conflict as: $[B \to \alpha_i \cdot A\ \beta_i, x^{shake}, u\ ]$, $[C \to \alpha'_i \cdot A\ \beta'_i, y^{shake}, u']$, $\mathrm{FIRST}_k(\beta_i u) = \mathrm{FIRST}_k(\beta'_i u')$ and $A \notin NPost$. The two rightmost derivations of the initial symbol S are $S \Rightarrow^* \alpha\alpha_i A\ \beta_i\beta$ and $S \Rightarrow^* \alpha'\alpha'_i A\ \beta'_i\beta'$. $TG \in C_{SP}$ implies: $h_i(\alpha\alpha_i A\ \beta_i\beta) = h_i(\alpha'\alpha'_i A\ \beta'_i\beta')$, $h_o(\alpha\alpha_i A\ \beta_i\beta) = h_o(\alpha'\alpha'_i A\ \beta'_i\beta')$. Because $A \notin NPost$ no output symbols can be postponed over A, Algorithm 4.16 must emit all output symbols before reading $h_i(\beta_i\beta)$ and $TG \in C_{SP}$ we have $h_o(\alpha\alpha_i) = h_o(\alpha'\alpha'_i)$ and $h_i(\alpha\alpha_i) = h_i(\alpha'\alpha'_i)$. The collection PS must contain the set M' that includes the translation items such as: $[B \to \alpha_i \cdot A\ \beta_i, z^{shake}, u\ ]$, $[C \to \alpha'_i \cdot A\ \beta'_i, z^{shake}, u']$, $z \in D^+$. Because $x \neq y$, on the same position in $\alpha\alpha_i$ and $\alpha'\alpha'_i$ Algorithm 4.20 emits different substrings of $h_o(\alpha'\alpha'_i)$. But it cannot happen because in such a case Algorithm 4.20 must finish earlier before the computation of the set M on account of an expansion or a reduction translation conflict. So $TG \in C_{SEP}$.

Assume TG is a translation grammar and $TG \in C_{SEP}$. Assume $TG \notin C_{SP}$, i.e., Algorithm 4.16 finishes computation in a case when in a set M there is a reduction translation conflict. We know that a reduction translation conflict can occur only if the algorithm uses the shaking-down tranformation for an output symbol of a translation item which is in an expansion translation conflict. The PS includes such translation items as: $[B \to \alpha_i \cdot A\ \beta_i, x^{shake}, u\ ]$, $[C \to \alpha'_i \cdot A\ \beta'_i, y^{shake}, u']$, $\mathrm{FIRST}_k(\beta_i u) = \mathrm{FIRST}_k(\beta'_i u')$ and $A \notin NPost$. The two rightmost derivations of the initial symbol S are $S \Rightarrow^* \alpha\alpha_i A\ \beta_i\beta$ and $S \Rightarrow^* \alpha'\alpha'_i A\ \beta'_i\beta'$. $TG \in C_{SEP}$ implies: $h_i(\alpha\alpha_i A\ \beta_i\beta) = h_i(\alpha'\alpha'_i A\ \beta'_i\beta')$, $h_o(\alpha\alpha_i A\ \beta_i\beta) = h_o(\alpha'\alpha'_i A\ \beta'_i\beta')$. Because $A \notin NPost$ no output symbols can be postponed over A, Algorithm 4.20 must emit all output symbols before reading $h_i(\beta_i\beta)$ and $TG \in C_{SEP}$ we have $h_o(\alpha\alpha_i) = h_o(\alpha'\alpha'_i)$ and $h_i(\alpha\alpha_i) = h_i(\alpha'\alpha'_i)$. The collection SEP must include the set M' that includes the translation items such as: $[B \to \alpha_i \cdot A\ \beta_i, z^{shake}, u\ ]$, $[C \to \alpha'_i \cdot A\ \beta'_i, z^{shake}, u']$, $z \in D^+$. Because $x \neq y$, on the same position in $\alpha\alpha_i$ and $\alpha'\alpha'_i$ Algorithm 4.16 emits different substrings of $h_o(\alpha'\alpha'_i)$. But it cannot happen because in this case Algorithm 4.16 must finish earlier before the computation of the set M on account of an expansion or a reduction translation conflict. So $TG \in C_{SP}$. □

6. 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 $LR_{sp}(0)$ 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, if we modify the grammar TG_7 from Example 4.17 such that we replace rule (5) $B \to b$ by $B \to xb$, then the resulting translation grammar is no longer an $LR(1)$ but an $LR(2)$ translation grammar, although its input grammar is the same as the input grammar of translation grammar TG_7 and both input grammars are $LR(1)$ grammars. Implementation of all translations described by translation grammars with $LR(k)$ input grammars can be done by a combination of output of output symbols and temporary storing them in the memory until they can be added to the output string ([5, 6, 7]).

(Received February 18, 2000.)

REFERENCES

[1] A. V. Aho and J. D. Ullman: The Theory of Parsing, Translation and Compiling. Vol. 1: Parsing, Vol. 2: Compiling. Prentice–Hall, New York 1971, 1972.

[2] A. V. Aho, R. Sethi, and J. D. Ullman: Compiler–Principles, Techniques and Tools. Addison–Wesley, Reading, 1987.

[3] N. V. Bac: *LR* Translation Grammars. MSc Thesis. Department of Computer Science and Engineering, CTU, Prague 1994. In Czech.

[4] N. V. Bac and B. Melichar: Hierarchy of *LR(k)* Translation Grammars. Research Report DC-96-09, Department of Computer Science and Engineering, CTU, Prague 1996.

[5] J. Janoušek: One–pass formal translation directed by *LR* parsing. In: WORKSHOP'97, CTU, Prague 1997, pp. 139–140.

[6] J. Janoušek and B. Melichar: Formal translations described by translation grammars with *LR(k)* input grammars. In: Ninth International Symposium on Programming Languages, Implementations, Logics and Programs (Lecture Notes in Computer Science 1338), Springer–Verlag, Berlin 1997, pp. 421–422.

[7] J. Janoušek and B. Melichar: The output-store formal translator directed by LR parsing. In: SOFSEM'97: Theory and Practice of Informatics (Lecture Notes in Computer Science 1338), Springer–Verlag, Berlin 1997, pp. 432–439.

[8] P. M. Lewis and R. E. Stearns: Syntax directed transductions. J. Assoc. Comput. Mach. *15* (1967), 3, 465–488.

[9] P. M. Lewis, D. J. Rosenkrantz, and R. E. Stearns: Compiler Design Theory. Addison–Wesley, London 1976.

[10] B. Melichar: Compilers. Publishing House of the Czech Technical University, Prague 1991. In Czech.

[11] B. Melichar: *LR* Translation Grammars. Reseach Report DC-92-03, Department of Computer Science and Engineering, CTU, Prague 1992.

[12] B. Melichar: Formal translation directed by *LR* parsing. Kybernetika *28* (1992), 1, 50–61.

[13] B. Melichar: Transformations of translation grammars. Kybernetika *30* (1994), 1, 53–62.

[14] B. Melichar and N. V. Bac: Transformations of Grammars and Translation Directed by *LR* Parsing. Research Report DC-96-02, Department of Computer Science and Engineering, CTU, Prague 1996.

[15] P. Purdom and C. A. Brown: Semantic routines and *LR*(k) parsers. Acta Inform. *14* (1980), 4, 229–315.

Prof. Ing. Bořivoj Melichar, DrSc. and Ing. Nguyen Van Bac, CSc., Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo nám. 13, 121 35 Praha 2. Czech Republic.
e-mail: melichar@fel.cvut.cz