Localization of (in)consistencies by monotone reducing automata*

Martin Procházka and Martin Plátek

Charles University, MFF UK, Department of Computer Science
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic prma@centrum.cz, martin.platek@mff.cuni.cz

Abstract. *A reducing automaton (red-automaton) is a deterministic automaton proposed for checking word and sub-word correctness by the use of analysis by reduction. Its monotone version characterizes the class of deterministic context-free languages (DCFL). We propose a method for a construction of a deterministic monotone enhancement of any monotone reducing automaton which is able with the help of special auxiliary symbols to localize its prefix and post-prefix (in)consistencies, and certain types of reducing conflicts. In other words this method ensures a robust analysis by reduction without spurious error messages. We formulate natural conditions for which this method ensures the localization of all prefix and post-prefix inconsistencies in any (incorrect) word with respect to a DCFL.*

1 Introduction

A reducing automaton (red-automaton for short) is a device that models the so called analysis by reduction. Analysis by reduction consists in a stepwise simplification of an extended sentence (word) until a simple sentence (word) is obtained or an error is found. It is based on another automata model – restarting automaton (R-automaton) introduced in [2]. Similarly to R-automaton, red-automaton can only delete symbols. At some place it decides to delete some of the last k visited symbols, where k is limited by a fixed constant and then restarts its computations, i.e. it enters its initial state and its head is placed on the left end of the remaining word.

Reducing automaton is formalized as an extension of deterministic finite automaton. This kind of formalization serves here as a basic tool for the method of algorithmic localization of syntactic inconsistencies (errors) for the languages from the class of DCFL. The notion of red-automata was introduced in order to present naturaly the techniques of minimization. In [7] we construct to any red-automaton M an unambiguously determined minimal red-automaton M_m which preserves the recognized language, and the set of all reductions defined by M.

For a given language L and a word $w \notin L$, it is natural to define the maximal correct prefix and prefix-inconsistency (prefix error) in w. The prefix inconsistency is the minimal incorrect prefix of w. Let $x \in \Sigma$ be the leftmost symbol in the word w such that $w = uxv$, $u, v \in \Sigma^*$, there exists a word $v' \in \Sigma^*$ for which it holds $uv' \in L$ and there is no word v'' such that $uxv'' \in L$. The u is the maximal correct prefix of w, and ux the prefix-inconsistency of w.

In a similar way we can consider (in)correct infixes for a given language L and a word $w \notin L$. We can easily see that a prefix-inconsistence can occur in a word at most once. Our effort is to study properties of reducing automata which will ensure the detection of (in)correct prefixes, and/or certain types of (in)correct infixes. The types of the (in)correct infixes studied here are studied by different techniques already in [1]. We call them here post-prefix (in)consistencies.

In this paper we use the advantage of the fact, that to any deterministic context-free language L there is a monotone reducing automaton recognizing L which is also able to detect the prefix inconsistency (error). Such type of automaton characterizes the class of DCFL. This fact is shown in [6].

The paper is structured as follows. First, in Section 2 we introduce red-automata and their basic properties. The Section 3 creates the core of this paper. Conclusion contains some remarks about connections of the presented method with the methods based on the so called head-symbols.

2 Definitions and basic properties

The *reducing automaton* has a finite *control unit* and a *working head* attached to a list with sentinels on both ends. It works in certain cycles called *stages*. At the beginning of each stage, the head points at the leftmost item behind the left sentinel, and the control unit is in a special *initial state*. In the process of the stage the automaton moves the head from the item it currently points to the next item on the right. During such a *transition* it changes the state of its control unit according to the current state and the currently scanned symbol. The stage ends as the control unit gets to any of special states called *operations*. There are three kinds of operations: `ACC`, `ERR`, and `RED`. Both `ACC` and `ERR`-operation halts the whole computation, `ACC` accepts and `ERR` rejects the word in the list. The

* This work was supported by the grant projects of the Grant Agency of the Czech Republic No. P202/10/1333 and P103/10/0783.

RED-operation $\mathtt{RED}(n)$ determines how the list should be shortened. Its parameter n – a binary word of a limited size – specifies which item on the left of the head are to be removed from the list. Bit 1 means "remove the item from the list", bit 0 means "leave the item in the list". After all items designated for deletion are removed, the automaton resets its control unit to the initial state and places the head at the leftmost item behind the left sentinel. The string $n \in (10^*)^+$ determines, which items will be deleted from the list. If the i-th symbol of n from the right is equal to 1, then the automaton deletes the i-th item to the left from the position of the head. The item scanned by the head is considered as the first one.

All final states of a reducing automaton M create a finite subset F_M of the (unbounded) set $\{\,\mathtt{ACC}, \mathtt{ERR}\,\} \cup \{\,\mathtt{RED}(n) \mid n \in (1{\cdot}0^*)^+\,\}$. Now we are able to introduce reducing automata in a formal way.

A *reducing automaton (red-automaton)* is a 7-tuple $M = (\Sigma_M, «, », S_M, s_M, F_M, f_M)$, where Σ_M is a finite input alphabet, $«, » \notin \Sigma_M$ are the (left and right) sentinels, S_M is the finite set of internal states, $s_M \in S_M$ is the (re)starting state, F_M is the finite set of final states (operations), $f_M : S_M \times (\Sigma_M \cup \{»\}) \longrightarrow (S_M \cup F_M)$ is the *transition function* of M, which fulfills the following condition:

$\forall s \in S_M : f_M(s, ») \in F_M$.

We will describe the behavior of M in more details by two functions enhancing the transition function f_M:

$\delta_M : (S_M \cup F_M \cup \{\mathtt{RED}\}) \times (\Sigma_M \cup \{«, »\}) \longrightarrow (S_M \cup F_M \cup \{\mathtt{RED}\})$

$\Delta_M : (S_M \cup F_M^*) \times (\Sigma_M \cup \{«, »\}) \longrightarrow (S_M \cup F_M^*)$

RED is a new (helping) state which is different from all states from S_M, and the set F_M^* is defined in the following way:

$F_M^* = F_M \cup \{\mathtt{RED}(n \cdot 0^k) \mid \mathtt{RED}(n) \in F_M \text{ a } k \geq 1\}$

Both functions δ_M, Δ_M for all pairs created by a state $s \in S_M$ and by a symbol $a \in (\Sigma_M \cup \{»\}$ are equal to the function f_M. We define the new functions for the remaining relevant pairs in the following way:

$$\delta_M(s, «) = s_M \qquad \Delta_M(s, «) = s_M$$

and for all $a \in (\Sigma_M \cup \{»\})$,

$$\begin{aligned} \delta_M(\mathtt{ACC}, a) &= \mathtt{ACC} & \Delta_M(\mathtt{ACC}, a) &= \mathtt{ACC} \\ \delta_M(\mathtt{ERR}, a) &= \mathtt{ERR} & \Delta_M(\mathtt{ERR}, a) &= \mathtt{ERR} \\ \delta_M(\mathtt{RED}(n), a) &= \mathtt{RED} & \Delta_M(\mathtt{RED}(n), a) &= \mathtt{RED}(n{\cdot}0) \\ \delta_M(\mathtt{RED}, a) &= \mathtt{RED} \end{aligned}$$

The first enhancement of δ_M:

$\delta_M^*(s, \lambda) = s, \quad \delta_M^*(s, ua) = \delta_M(\delta_M^*(s, u), a)$

The first enhancement of Δ_M:

$\Delta_M^*(s, \lambda) = s, \quad \Delta_M^*(s, ua) = \Delta_M(\Delta_M^*(s, u), a)$

We will often use the following conventions:

$\delta_M^*(s_M, w) = \delta_M^*(«w), \quad \Delta_M^*(s_M, w) = \Delta_M^*(«w)$.

We define for the both function a further important enhancement, namely for the final subsets S of the set $S_M \cup F_M \cup \{\mathtt{RED}\}$ resp. $S_M \cup F_M^*$:

$\delta_M^*(S, u) = \{\delta_M^*(s, u) \mid s \in S\}$

$\Delta_M^*(S, u) = \{\Delta_M^*(s, u) \mid s \in S\}$

Let us note that the tuple

$(\Sigma_M \cup \{«, »\}, S_M \cup F_M \cup \{\mathtt{RED}\}, s_M, F_M, \delta_M)$ is a finite automaton which accepts exactly all prefixes (words) which lead M to some reduction, or to an acceptation, or to a rejection.

We will consider in the following only the reducing automata which fulfills the following natural condition:

$\delta_M^*(s_M, u) = \mathtt{RED}(n) \quad \Longrightarrow \quad |u| \geq |n|$.

Let us take: $L_0(M) = \{w \in \Sigma_M^* \mid \Delta_M^*(« w ») = \mathtt{ACC}\}$.

Constant. Let $k_M = \max\{\,|n| \,|\, \mathtt{RED}(n) \in F_M\,\}$. We call k_M the *characteristic constant* of M.

The operation of reduction. We will exactly describe a reduction of a word by a binary sequence with the help of the following operation /:

$a/0 = a, \quad a/1 = \lambda, \quad \lambda/n = \lambda, \quad u/\lambda = u,$

$(u \cdot a)/(n \cdot i) = (u/n) \cdot (a/i),$

where $u \in \Sigma^*$, $a \in \Sigma$, $n \in (10^*)^+$ and $i \in \{0, 1\}$. The size of the strings u, n is here unbounded, moreover u can be longer then n, and vice versa. The reduction of the word (a) by the sequence $1 \cdot 0 \cdot 1$ is given in the following way: $(\mathtt{a})/101 = \mathtt{a}$.

Using just defined operation we can describe the way how the red-automaton M reduces a word $w \in \Sigma_M^*$.

The relation of reduction denoted by $\Rightarrow_M$ is introduced in the following way: $«w» \Rightarrow_M «w'»$, if $\Delta_M^*(«w») = \mathtt{RED}(n)$, and $«w»/n = «w'»$.

If $«w» \Rightarrow_M «w'»$ holds, we say, that the automaton M *reduces* the word w into the word w'. We can see that $|w| > |w'|$.

The relation $\Rightarrow^+$ is the transitive closure of $\Rightarrow$; $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

Analysis by reduction by M is any sequence of reductions $«w_1» \Rightarrow «w_2» \Rightarrow \ldots \Rightarrow «w_n»$, which cannot be further prolonged. If $w_n \in L_0(M)$, we speak about *accepting* analysis by reduction, in the other case we speak about *rejecting* analysis by reduction. Often we will speak about analysis instead of analysis by reduction.

Stages. Let us recall that each computation of a red-automaton is divided in stages. At the beginning of each stage the head points at the leftmost item behind the left sentinel, and the control unit is in the (re)starting state. The stage ends as the control unit gets to any final state (operation) from F_M. There are three kinds of operations: ACC, ERR, and RED. Accordingly, we have *accepting* (ACC-), *rejecting* (ERR-), and *reducing* (RED-)*stages.*

Recognized language. The language *recognized* by M is defined in the following way:
$L(M) = \{\, w \mid «w» \Rightarrow_M^* «w'», \;\; and \;\; w' \in L_0(M) \,\}$.

Equivalences of red-automata. Two red-automata M_1 and M_2 are *equivalent*, if $L(M_1) = L(M_2)$.

We will often consider a stronger equivalence. Let us suppose that for any $w \in (\Sigma_{M_1}^* \cup \Sigma_{M_2}^*)\cdot\{\lambda, »\}$ hold at the same time

$\delta_{M_1}^*(«w) = \mathtt{ACC} \iff \delta_{M_2}^*(«w) = \mathtt{ACC}$,
$\delta_{M_1}^*(«w) = \mathtt{RED}(n) \iff \delta_{M_2}^*(«w) = \mathtt{RED}(n)$.

We say that M_1 and M_2 are *strongly equivalent*. We can see that if M_1 and M_2 are strongly equivalent then they are equivalent, as well.

Error and correctness preserving property. We can see the following usefull property:

Lemma 1. *If $«w_1» \Rightarrow_M «w_2»$, then $w_1 \in L(M)$, exactly if $w_2 \in L(M)$.*

Monotony. Monotony is an important property that enables to characterize the class of DCFL in terms of monotonic reducing automata. This property was introduced for restarting automata in [2], first. Informally a red-automaton M is monotonic if the size of sequences of non-visited items (symbols) in individual stages of any analysis by reduction by M is non-increasing. A monotonic reducing automaton will be called a *mon-red-automaton* for short. As an example see Table 1.

3 Robust analyzer

We introduce for the robust analysis by reduction a new type of automaton – *robust analyzer*. Robust analyzer enhances the reducing automaton by the ability of inserting special auxiliary symbols, and by the ability to read any input word about its input alphabet to the end. Robust analyzer A consists of finite control unit with a finite set of states S_A, and from a working head connecting the finite control unit with a linear list of items. The list of items is bounded by a left and right sentinels « and ». All other items contain a symbol from a finite *input alphabet* X_A, or of a finite *auxiliary alphabet* Y_A. These alphabets are mutually disjunct.

Robust analyzer is able to delete some items from the list (operation of reduction). Deleted can be the item visited by the working head and some items positioned not far to the left from the working head. Operations of reductions are controlled by reducing sequences. Each operation of reduction is followed by a restart, i.e., transfer of the control unit into the (re)starting state s_A, and a placement of the working head on the left sentinel «. It means, that the robust analyzer, similarly as reducing automaton, works in stages. Therefore we can define the monotony for robust analyzers in the same way as for reducing automata.

We divide auxiliary symbols into two types. Each type serves to a different purpose:
1) for a transfer of local informations between different stages,
2) for a marking of correct and incorrect sub-words of the analyzed word, and for marking of the place of certain types of a reducing conflict.

The auxiliary symbols of the type 2 are called *signs*. Here we use two signs:
`!` – for marking of incorrectnesses,
`?` – for marking of reducing ambiguities.
We understand under the reducing ambiguity a subword for which is obtained by the robust analysis an ambiguous information about its current reduction.

There is a technical difference between reducing automata, and robust parsers. The behavior of the robust parser is described by the following three functions:

Transition function $t_A : S_A \times (X_A \cup Y_A) \longrightarrow S_A$.
t_a determines the state, into which will be transfered the finite control from its current state after scanning the symbol from the item visited by the working head.

Inserting function $i_A : S_A \times (X_A \cup Y_A) \longrightarrow I_A$.
i_A assigns to a state, and to a scanned symbol an *inserting sequence* from a (final) set $I_A \subseteq (Y_A \cdot \mathtt{0}^*)^*$.

Inserting sequences serves in a similar way as reducing sequences. They describe the inserted auxiliary symbols, and their inserting positions. If the value of the inserting function is λ, it will be nothing inserted. If the value is `!`, A inserts new item with the marking `!` immediately before (to the left) the scanned item. The value `?000` says, that the marking `?` should be inserted before the third item to the left before the scanned item.

Reducing function $r_A : S_A \times (X_A \cup Y_A) \longrightarrow R_A$.
r_A assigns to a state, and to a scanned symbol a *reducing sequence* from a (final) set $R_A \subseteq (\mathtt{1} \cdot \mathtt{0}^*)^*$.

Reducing sequence is here interpreted in the same way as for reducing automata.

A step of the robust analyzer A consists from the sequence of the following actions:

Shift to the right. Analyzer A starts each step by a shift to the right of its working head to the next item of the working list.

Application of the inserting function. i_A by the current *situation* (state, symbol) determines the inserting sequence Is. A controlled by Is inserts new auxiliary symbols.

Application of the reduction function. r_A determines by the current situation the current reducing sequence Rs. If RS is non-empty, A controlled by Rs reduces (deletes) determined items from the

working list. After such a reduction A finishes the step by a restart, i.e., moves the working head on the left sentinel «, and transfers the control unit into the (re)starting state s_A. Then a new stage will be started on the reduced list.
If Rs is empty then A does not perform any reduction, neither the restart, and it finishes the step by the following action.

Application of the transition function. t_A determines by the current situation the new state q. Then A continues from the state q by a further step of the current stage.

The last difference of A from reducing automata consists in the fact that A contains only one halting state `END` $\in S_A$. We will see that the signs of A will refine the ability of accepting and rejecting by the states ACC and ERR of reducing automata. For this purpose we observe the signs `!` and `?` inserted in the different stages of the computation into the gradually reduced working list. We will project the signs into the original input list in such a way that we will insert the signs into the same positions (i.e. before the same items) into which they were inserted during the individual stages of the computation (robust analysis) by A.

We denote by $p_A(w)$ a word w enriched by the signs (in the way mentioned above) inserted into the list during the analysis by the robust analyzer A. We will later formulate our results using this denotation. As an example see Fig.1. We can consider $p_A(w)$ as the output word of A.

3.1 Prefix and post-prefix (in)consistencies

Assumption. We assume in the following that $L \subseteq \Sigma^*$, and any symbol of Σ is a symbol of some word from L.

We call a word v *inconsistent (incorrect)* with respect to the language $L \subseteq \Sigma^*$, if for any $u, w \in \Sigma^*$ is $uvw \notin L$.

We can see that incorrect words can obtain proper incorrect sub-words. This fact lead us to the following notion.
We say that a word v is an *incorrect core* of the word w with respect to the language L, if it is a subword of w, if it is incorrect with respect to the language L, and if it is minimal by the ordering "to be a sub-word".

On the other hand, a word v is a *correct* sub-word of a word w with respect to the language L, if $w = xvy$, and for some x', y' is $x'vy' \in L$. We say that v is a *correct core* of a word w with respect to the language L, if it is a correct sub-word of w with respect to L, and it is maximal by the ordering "o be a sub-word". The assumption that each symbol of Σ is a symbol of some word of the language L ensures that each symbol of any word $w \in \Sigma^*$ is contained in some correct core of this word.

Prefix consistence is the longest correct prefix v of the analyzed word w. *Prefix inconsistence* is the shortest incorrect prefix of the analyzed word w, i.e., it is the prefix va of w, where $a \in \Sigma$. *Post-prefix consistence* is a suffix x of a correct core behind (to the right of) the prefix consistence, or behind some of the previous post-prefix consistencies. We assign to the post-prefix consistency x the incorrect sub-word xa of w. We say that xa is a post-prefix inconsistence of w (with respect to L).

Our effort in the following is to deterministically, in a monotonic way, and exactly to localize the prefix and post-prefix (in)consistencies in the analyzed words from DCFL.

3.2 Post-prefix robust analyzer A

Prefix consistence. A red-automaton M is *prefix-consistent* when for each word u and each symbol a (including the right sentinel) it holds the following: if $\Delta_M(s_M, u) \in S_M$ and $\Delta_M(s_M, ua) \neq$ `ERR`, then ua is a prefix of some word from $L(M) \cdot \{»\}$.

The following proposition is derived from the main result from [2]. The detailed proof is in [6]. There is also connected with some other propositions.

Proposition 1. *Monotone, prefix consistent, red-automata characterize the class of DCFL.*

It is shown in [7] that the notion of red-automata is useful for the techniques of minimization. There is to any red-automaton M constructed an unambiguously determined state-minimal red-automaton M_m which is strongly equivalent with M.

We will show informally in the next part a method how construct for a given monotone, prefix-correct, state-minimal reducing automaton M a robust analyzer A which determines in any word $w \in «\Sigma_M^*»$ the prefix-(in)consistence, and (not obligatory all) post-prefix (in)consistencies. We suppose for the construction that $L(M) \neq \emptyset$.

At first A will use the prefix-consistency of M for the finding of the prefix-(in)consistency of the analyzed word w.

Such a situation can occur after one, or after more stages if M will be transfered into the final rejecting state `ERR`. The computation (analysis) of M on the word w until this moment we describe in the following way:
1) At first M (possibly) gradually reduces the word w into the word w', i.e., $w \Rightarrow_M^* w'$.
2) Then in the next stage M transfers over some prefix x of the word w' into some non-final state $s \in S_M$, i.e., $\delta_M^*(s_M, x) = s \in S_M$,

3) Finally from the state s transfers over the next symbol a into the final state ERR, i.e., $\delta_M(s,a) = \texttt{ERR}$.

We can see that A has founded by the previous simulation of M the prefix inconsistency of w. For marking of the prefix inconsistency A inserts the sign ! between the correct prefix «x and the symbol a.

The prefix-consistency of M ensures that M has visited in the last step described above the symbol a at the first time. Therefore if $w' = «xay$ for some y then ay is a suffix of the original input word w.

Let us now informaly describe how A continues in the robust analysis over the mentioned suffix ay of the word w.

We will use the function δ_M for this aim. This function was introduced as an enhancement of the transition function f_M. It describes not only the transfers between the individual states, but also the tranfers between the indiviual subsets of the set $S_M \cup F_M \cup \{\texttt{RED}\}$, i.e., of the set of all final and non-final states, and of a special state RED. We will use it in the following in order to describe the all possible (partial) computation of M over the suffix ay at the same time.

We let A to compute the function δ_M over the suffix $ay = a_0 a_1 \dots a_{|y|}$ starting from the set S_M of all non-final states of M. A will control the computation in the following way. Let us initially take the set S_M as a set further denoted as S_I.

Let us denote the following part of the computation of A as a cycle C_1. The cycle C_1 is performed until for the set $S = \delta^*_M(S_I, a_0 \dots a_i)$, where $0 \le i \le |y|$, holds that $\emptyset \subset S \subseteq S_M \cup \{ERR\}$, and S contains some non-final state. Then A performs the following action: the head of A will be placed to the next item to the right, and as (the current value of) the set S will be taken the set $\delta_M(S, a_{i+1})$. Here ends the description of C_1.

The core of the post-prefix analysis by A are the following four cases where is not fulfilled the condition for the continuation of the computation by cycle C_1.

Correct suffix. The set S contains the accepting state ACC; i.e., $\texttt{ACC} \in S$.
If $\texttt{ACC} \in S$, then the current suffix of the analyzed word w by A is a suffix of some word from $L(M)$. Therefore, the current suffix cannot contain any further inconsistency. The work of A on w is finished at this moment.

An unambiguous inconsistency. The set S contains a single state – the rejecting state ERR; i.e., $S = \{\texttt{ERR}\}$.

All the possible computations of M over the word w behind the previous inconsistency has ended at the same time in the state ERR. We have found a suffix of a correct core of the analyzed word, i.e., one of its post-prefix (in)consistency. At this moment A inserts the sign ! immediately before the position of its working head. The automaton A will look for a new post-prefix (in)consistency behind (to the right from) the currently inserted sign !. A will take instead of the set $\{\texttt{ERR}\}$ as the current value of the set S the set $\delta_M(S_M, a)$, where a is the symbol scanned by the working head. A will continue in the robust analysis of the remaining suffix by the schema of the cycle C_1.

An ambiguous reduction. S does not contain ACC, and either does contain two different reducing states of M, or does contain at least one non-final state, and at least one reducing state; i.e.,
$\texttt{ACC} \notin S$, and $\exists n : \texttt{RED}(n) \in S \not\subseteq \{\texttt{RED}(n), \texttt{ERR}\}$.
We say that S fulfilling the condition above is an *ambiguous* set.
The task for A is to work without false inconsistency messages. From that reason A separates the ambiguous part from the remaining suffix. It inserts the sign for the ambiguity ? in the place of the current ambiguity, i.e., immediately to the left from the position of the working head (if the sign is not already placed there in some of the previous stages). At this moment A takes for the set S the complete set S_M, and continues in the robust analysis behind the sign ? by the scheme of the cycle C_1.

An unambiguous reduction. The set S contains exactly one reducing operation, and possibly beside it the final state ERR; i.e., $\exists n : \texttt{RED}(n) \in S \subseteq \{\texttt{RED}(n), \texttt{ERR}\}$.

Let us denote as u the sub-word which is created by the input symbols positioned between the last sign ! or ?, and the position of the working head including the scanned symbol. The sub-word u is because of the prefix-consistency, and because of the state minimality of M a sub-word of some word from $L(M)$. Moreover, the u is reduced in any word $w \in L(M)$ of the form $w = vux$ by the reducing seguence n, i.e, the reducing sequence and the position of the reduction are determined unambiguously. A will reduce also by n, but only the symbols from u if we consider the case that n can be longer then u.

Observation. The reducing sequence n deletes at least one symbol from u. This observation follows from the unambiguity of the reduction of u. Let us suppose the opposite. Then for some z, and n' is $n = n' \cdot \texttt{0}^{|u|}$, and $\delta^*_M(s_M, zu) = \texttt{RED}(n)$.
Let z' be the shortest z with the properties described above. Since M is prefix-correct «$z'u$ is a prefix of some word from «$L(M)$». In the next stage occurs one of the following variant:

$$\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{RED}(n'') \quad \text{for some } n''$$
$$\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{RED}$$
$$\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{ACC}$$
$$\delta^*_M(s_M, (z'/n') \cdot u) \in S_M$$

Each of the presented variant leads to a contradiction with the unambiguity of the reduction of the word u. If occurs the first one, then n'' reduces some symbol of the sub-word u (since z' cannot be shorter), therefore $n'' \neq n$ and $\texttt{RED}(n'') \in S \not\subseteq \{\texttt{RED}(n), \texttt{ERR}\}$. By the remaining variants is the contradiction obvious.

Apart from the reduction of u by n, A will insert into the list a new item with an auxiliary symbol – the set U of pairs of an internal state and a word over an input alphabet of the length k_M at most. This auxiliary symbol will be used in the next stage to adjust the set of states computed by the function δ_M. Our goal is to avoid situation when $\delta^*_M(s,u) = \texttt{ERR} \neq \delta^*_M(s,u/n)$ for some $s \in S_M$. Such internal states s must be eliminated. The set U is defined by the following way:

- If $|u| \geq |n|$, then A reduces the working list of items by n and A puts a new item with the auxiliary symbol U just in front of the leftmost deleted item. $U = \{(s,\lambda) \mid \exists s' \in S_M : \delta^*_M(s',u_1) = s \text{ and } \delta^*_M(s,u_2) = \texttt{RED}(n)\}$ where $u = u_1u_2$ and $|u_2| = |n|$.
- If $|u| < |n|$, then A cannot reduce by the whole n as such a reduction would impact a part of the working list in front of u; this part would be reduced by n_1 such that $n = n_1n_2$ and $|n_2| = |u|$. But a part of the working list in front of the rightmost marker `!` or `?` can be reduced in some word of $L(M)$ in other way or even not at all. So, A will reduce items behind the rightmost marker `!` or `?` by the reducing sequence n_2 and it will insert a new item containing an auxiliary symbol U just to the right of the rightmost marker. $U = \{(s,x) \mid \exists v \in \Sigma^*_M : x = v/n_1 \text{ a } |v| = |n_1| \text{ a } \delta^*_M(s,vu) = \texttt{RED}(n)\}$.

Insertion of the set U into the working list is important as it ensures the continuity of subsequent stages of computation. In next stage, A will use this set to adjust the set S of internal states computed by function δ_M. As soon as A reach the item with U, it substitute S by $S' = \{\delta^*_M(s,v) \mid (s,v) \in U\}$. It guarantees that A enter a part of the list impacted by the last reduction in such states only that led to the last reduction of u by n resp. n_2.

A uses just defined set U in such a case only when the new item with U is inserted just behind an item containing a symbol of the input alphabet or a marker. Otherwise, when this item contains an auxiliary symbol U' different from both markers, then (instead of insertion of U) A replaces U' with U computed in the following way:

$U = \{(s,x) \mid \exists y,(s',x') \in U' : x = yx'/n_1$, and
$\delta^*_M(s,y) = s'$, and $\delta^*_M(s',x'u) = \texttt{RED}(n)$, and
$|y| = \max\{0, |n_1| - |x'|\}\}$,

where n_1 is a prefix of n of the length $|n| - |u|$. In all cases, the length of the word x contained in any pair of inserted set U is bounded by the characteristic constant k_M which ensures that U is finite.

Let us note that A stores in its finite control a suitable suffix of its working list before the position of its working head. The length of this suffix need not be longer then $2 \cdot k_M$. It contains the input items, and the inserted values of the set S in the last $2 \cdot k_M$ steps.

Recall, that we suppose that the automaton M is minimal and prefix-consistent. The minimality of M ensures that each state of M is reachable. The state-reachability, and the prefix-consistence of the automaton M ensure for any word u, that u is a sub-word of some word of $L(M)$, if $\delta^*_M(S_M,u) \not\subseteq \{\texttt{ERR}, \texttt{RED}\}$.

If A inserts the sign `!` or `?` immediately before the right sentinel » it finishes its computation. Since we suppose that $L(M)$ is non-empty, is » a suffix of some word of the language «$L(M)$» (i.e.,$L(M)$ with sentinels).

Now we have outlined the behavior of A in the first stage after the localization of the prefix-inconsistence. In the next stages we need also to consider the signs and the other auxiliary symbols inserted in the previous stages. We will not describe here these details.

We illustrate the outlined method by the following example.

Example 1. We explain the presented method by two prefix-consistent, state-minimal mon-red-automata. We will present two different robust analyses of the following inconsistent word «`a++(a)+)(a`».

The transition functions of the automata M_1 and M_3 from [6] which we use in this example are defined by the tables in the figure 1. We do not present here the automaton M_2 from [6].

The robust reduction analyses of the considered word are on figures 1a, and 1b. Both figures contain also the input word enriched by the signs `!` and `?` on the corresponding places.

Let us note that in [6] is a detailed description of a construction which constructs to a given prefix consistent, state minimal mon-red-automaton M its robust analyzer A. This construction implements the outlined method. The robust analyzer A is by this construction given unambiguously for a given M.

3.3 Guarantees of the presented method

We formulate the guarantees of the presented method as theorems. The detailed proofs can be found in [6].

Theorem 1. *Let M be a prefix consistent, state-minimal mon-red-automaton and A its post-prefix robust analyzer. For any $w \in$ «Σ^*» the following propositions hold:*

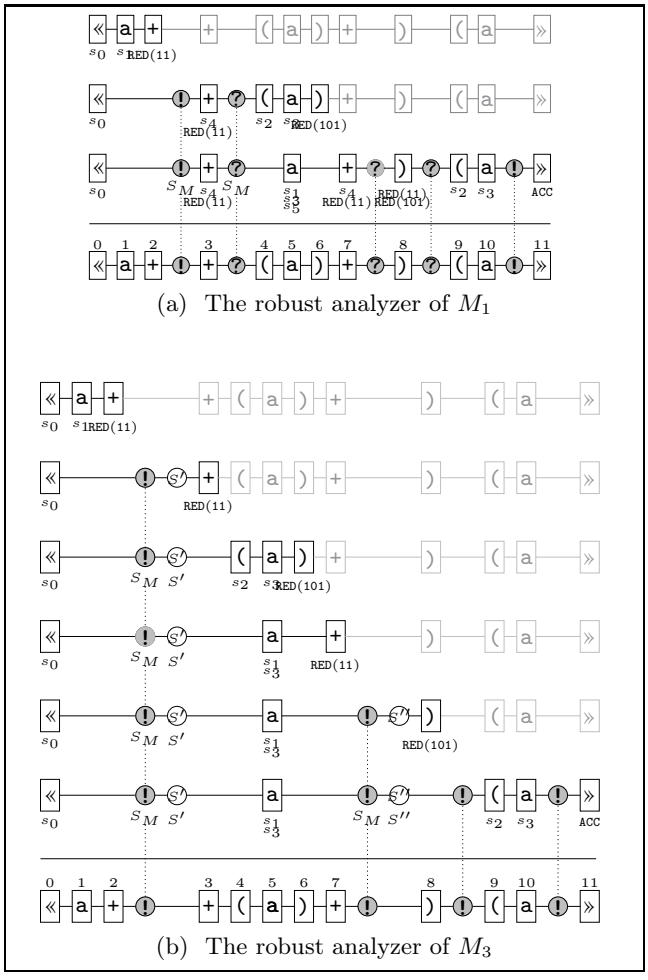
	a	+	(	)	»
⇒ s_0	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	s_4	ERR	RED(101)	ERR
s_4	s_5	ERR	s_2	ERR	ERR
s_5	ERR	s_4	ERR	RED(110)	ERR

(a) automaton M_1 reducing the word a+ without brackets around it "from the left" and the +a with brackets around "from the right"

	a	+	(	)	»
⇒ s_0	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	RED(11)	ERR	RED(101)	ERR

(b) (strongly) monotone automaton M_3 reducing a+ only "from the left"

Table 1: The transition functions for M_1 and M_3.



(a) The robust analyzer of M_1

(b) The robust analyzer of M_3

Fig. 1: The robust analysis of «a++(a)+)(a».

1. *The analyzer A reads (in one or more stages) the complete word «w» and finishes its computation in a special state* END. *Any computation of A is monotone.*
2. *If $p_A(«w»)$ does not contain any sign* !, *then $p_A(«w») = «w»$ and w is from $L(M)$.*
3. *If $«w» \in «L(M)»$ then $p_A(«w») = «w»$.*
4. *If «u! is a prefix of $p_A(«w»)$ and u does not contain the sign* ! *then u does not contain the sign* ? *as well, and «u is the longest correct-prefix of the word «w» with respect to the language $«L(M)»$.*
5. *If !u! or ?u! is a sub-word of the word $p_A(«w»)$ and u does not contain any sign* ! *or* ? *then u is a suffix of some corect core of the word «w» with respect to the language $«L(M)»$.*
6. *If !u» or ?u» is a suffix of the word $p_A(«w»)$, and u does not contain any sign* ! *or* ? *then u» is a sufix of some word from $«L(M)»$.*
7. *If !u? or ?u? is a sub-word of $p_A(«w»)$ and u does not contain any sign* ! *or* ? *then u is a sub-word of some word from $«L(M)»$.*

We will discuss the meaning of the sign ?, and we will formulate properties of the mon-red-automaton M which ensure that its post-prefix robust analyzer A does not use the sign ? at all. This sign serves as the right sentinel for the correct sub-words of $L(M)$ which lead A to some of two following types of a reduction conflict. Let u be such a sub-word which is followed by ?. The first type of the conflict means that there are two different words of $L(M)$ containing u which lead M by reading the complete u and its prefixes to two different reductions. The second type of the conflict means that there is a transfer trough u by M which leads to a reduction, and at the same time there is an another transfer leading to the shift to the right of M from u to the next symbol.

Now we gradually introduce the notions of *unambiguously reducible sub-word* and *unambiguously reducing red-automaton*, and we will show that a post-prefix robust analyzer A of M which is unambiguously reducing, need not to use the sign ? at all.

We say that a sub-word w is *reducible* by M if for some n holds that $\mathtt{RED}(n) \in \delta^*_M(S_M, w)$.

We say that a sub-word w is *unambiguously reducible* by M if for some n holds that $\mathtt{RED}(n) \in \delta^*_M(S_M, w) \subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$.

We say that a sub-word which is reducible but it is not unambiguously reducible is an *ambiguously reducible* sub-word.

We say that M is *unambiguously reducing* if any of its reducible sub-words is unambiguously reducible.

Example 2. The automaton M_1 from the example 1 is not unambiguously reducing. All its ambiguously reducible sub-words are presented in Table 2a. Let us

note the length of this words is not limited by any constant, since for any $i \geq 0$ holds that

$\delta^*_{M_1}(S_{M_1}, (\texttt{+a})^i\texttt{+}) = \{\texttt{ERR}, \texttt{RED(110)}, s_4\}$.

The minimal unambiguously reducible sub-words of M_1 are in Table 2b.

sub-word w	)	a)	+	a+	+a+	...
$\delta^*_{M_1}(S_{M_1}, w)$	RED(110)	RED(110)	RED(11)	RED(11)	RED	...
	RED(101)	RED(101)	s_4	s_4	s_4	

(a) ambiguously reducible sub-words

sub-word w	+a)	(a)	«a+
$\delta^*_{M_1}(S_{M_1}, w)$	RED(110)	RED(101)	RED(11)

(b) minimal unambiguously reducible sub-words

Table 2: Reducible sub-words of M_1.

We can see that in this example the robust analyzer of M_3 has founded all inconsistencies in the word «a + +(a)+)(a»., i.e., it has not used the sign ? at all. This example illustrates the fact that M_3 is an unambiguously reducing red-automaton. We can see that directly from Table 1b.

The meaning of the reducing unambiguity for the localization of the post-pefix inconsistencies summarizes the following theorem.

Theorem 2. *Let M be a prefix consistent, state-minimal mon-red-automaton and A its post-prefix robust analyzer. If M is at the same time unambiguously reducing then its post-prefix robust analyzer A does not use the sign* ? *at all. That means, that A in any word from «Σ^*_M» determines the prefix inconsistency and all its post-prefix inconsistencies with respect to the language $L(M)$.*

Corollary 1. *Let M be a mon-red-automaton which is at the same time prefix-consistent and state-minimal, and A be its robust analyzer. Then there is a deterministic push-down transducer which translates any word w from Σ^*_M on the word $p_A(w)$.*

4 Conclusion

The presented method can be considered as a direct generalization of the method presented in [9], and as an essential refinement and a generalization of the method from [5]. The method in [9] is based on monotone reducing automata, the method in [5] is based on (monotone) list automata with auxiliary symbols. Both the methods are based on the so called head-symbols. The head-symbol (in)consistencies from [9] create a very special type of (in)consistencies considered by the method presented in this paper. In the close future we will show that the set of languages recognized by unambiguously reducing, prefix consistent, state-minimal mon-red-automata creates a proper subclass of DCFL.

References

1. G. V. Cormack: *An LR substring parser for noncorrecting syntax error recovery.* In: Proc. of PLDI '89, 1989, 161–169.
2. P. Jančar, F. Mráz, M. Plátek, J. Vogel: *Restarting automata.* In Proc. FCT'95, Dresden, Germany, August 1995, LNCS 965, Springer Verlag 1995, 283–292.
3. F. Otto: *Restarting automata.* In: Z. Ésik, C. Martin-Vide, and V. Mitrana (Eds.), Recent Advances in Formal Languages and Applications, Studies in Computational Intelligence, Vol. 25, Springer, Berlin, 2006, 269–303.
4. Gh. Păun: Marcus Contextual Grammars, Kluwer, Dordrecht, Boston, London, 1997.
5. M. Plátek: *Construction of a robust parser from a deterministic reduced parser.* Kybernetika 33(3), 1997, 311–332.
6. M. Procházka: *Redukční automaty a syntaktické chyby.* (in Czech) Text for PhD Dissertation, it will be to achieve soon on the web.
7. M. Procházka: *Concepts of syntax error recovery for monotonic reducing automata.* MIS 2004, 94–103, http://ulita.ms.mff.cuni.cz/pub/MIS/MIS2004.pdf
8. M. Procházka, M. Plátek: *Redukční automaty – monotonie a redukovanost.* ITAT 2002, 23–32.
9. M. Procházka, M. Plátek: *Redukční automaty a syntaktické chyby.* Proceedings of ITAT 2011 (in Czech), 2011, 23–30.