Project

Clearing Restarting Automaton [1]

USER GUIDE

Peter Černo

Prague, 2012

[1]This work was partially supported by the Grant Agency of Charles University under Grant-No. 272111/A-INF/MFF and by the Czech Science Foundation under Grant-No. P103/10/0783 and Grant-No. P202/10/1333.

Contents

Preface

Restarting automata [7] were introduced as a tool for modeling some techniques used for natural language processing. In particular they are used for analysis by reduction which is a method for checking (syntactical) correctness or non-correctness of a sentence. While restarting automata are quite general (see [8] for an overview), they still lack some properties which could facilitate their wider use. One of their drawbacks is, for instance, the lack of some intuitive way how to infer their instructions. There were several attempts to learn their instructions by using genetic algorithms, but the results are far from being applicable.

Clearing restarting automata were introduced in [2, 3] as a new restricted model of restarting automata which, based on a limited context, can only delete a substring of the current content of its tape. The model is motivated by the need for simpler definitions and simultaneously by aiming for efficient machine learning of such automata. Clearing restarting automata are studied in [3]. We only mention that they can recognize all regular languages, some context-free languages and even some non-context-free languages. Moreover, the model is effectively learnable from positive samples of reductions and it is even possible to infer some non-context-free languages in this way. However, there are some context-free languages that are outside the class of languages accepted by clearing restarting automata. This limitation led to the development of the extended versions of clearing restarting automata. In [3] there were introduced two extended versions – the so-called Δ-clearing restarting automata and Δ^*-clearing restarting automata. Both of them can use a single auxiliary symbol Δ only. Δ-clearing restarting automata can leave a mark – a symbol Δ – at the place of deleting besides rewriting into the empty word λ. Δ^*-clearing restarting automata can rewrite a subword w into Δ^k where k is bounded from above by the length of w. It was shown in [3] that Δ^*-clearing restarting automata are powerful enough to recognize all context-free languages. This result was later extended in [4, 5] to hold also for the more restricted Δ-clearing restarting automata. In [1] there was proposed yet another model, the so-called subword-clearing restarting automata, which, based on a limited context, can replace a substring z of the current content of its tape by a proper substring of z. This model proved useful in some grammatical inference scenarios. It was shown that it is possible, by using a simple learning algorithm, to identify any clearing

(subword-clearing) restarting automaton in the limit from any "reasonable" presentation of positive and negative samples.

The goal of the project *Clearing Restarting Automaton* is to provide a basic development framework for implementing the algorithms concerning clearing restarting automata and other similar models (like subword-clearing restarting automata, Δ-clearing restarting automata etc.). In other words, our aim is to bring the theory closer to the real world. We do not expect that the algorithms developed in this project will be applicable to the real world data. Instead, they can be used as tools for researchers who are interested in models used in the theory of automata and formal languages. The project itself is hosted on the following website: `http://code.google.com/p/clearing-restarting-automata/`.

This guide has the following structure. Chapter 1 introduces the used automata models and fixes the notation. Chapter 2 shows how to install the application and how to use the application to define and investigate our automata models. It also shows how to infer automata from a sample computation and how to infer automata from the given set of positive and negative samples.

The project *Clearing Restarting Automaton* is freely licensed under the GNU GPL v3.0 and available for Windows, Linux and Unix platforms. It was developed in C♯ 4 by using Microsoft Visual Studio 2010 and the target platform is .NET Framework 4.0. If you want to use the application on Windows, you need to have the .NET Framework 4.0 installed on your computer. For Linux and Unix platforms, you can use the open source Mono project for running this application. You are very welcome to use and modify the source code, and even to contribute to the project itself, provided that you contact me and mention my authorship in your own projects.

Chapter 1

Theoretical Background

As our reference concerning the theory of automata and formal languages we use the monograph [6].

An *alphabet* is a finite nonempty set. The elements of an alphabet Σ are called *letters* or *symbols*. A *word* or *string* over an alphabet Σ is a finite sequence consisting of zero or more letters of Σ, whereby the same letter may occur several times. The sequence of zero letters is called the *empty word*, written λ. The set of all words (all nonempty words, respectively) over an alphabet Σ is denoted by Σ^* (Σ^+, respectively). If x and y are words over Σ, then so is their *catenation* (or *concatenation*) xy (or $x \cdot y$), obtained by juxtaposition, that is, writing x and y one after another. Catenation is an associative operation and the empty word λ acts as an identity: $w\lambda = \lambda w = w$ holds for all words w. Because of the associativity, we may use the notation w^i in the usual way. By definition, $w^0 = \lambda$.

Let u be a word in Σ^*, say $u = a_1 \dots a_n$ with $a_i \in \Sigma$. We say that n is the *length* of u and we write $|u| = n$. The sets of all words over Σ of length k, or at most k, are denoted by Σ^k and $\Sigma^{\leq k}$, respectively. By $|u|_a$, for $a \in \Sigma$, we denote the total number of occurrences of the letter a in u. The *reversal* (*mirror image*) of u, denoted u^R, is the word $a_n \dots a_1$. Finally a *factorization* of u is any sequence u_1, ..., u_t of words such that $u = u_1 \cdots u_t$.

For a pair u, v of words we define the following relations: u is a *prefix* of v, if there exists a word z such that $v = uz$; u is a *suffix* of v, if there exists a word z such that $v = zu$; and u is a *factor* (or *subword*) of v, if there exist words z and z' such that $v = zuz'$. Observe that u itself and λ are subwords, prefixes and suffixes of u. Other subwords, prefixes and suffixes are called *proper*.

Subsets of Σ^* are referred to as (*formal*) *languages* over Σ.

1.1 Context Rewriting Systems

In this section we introduce our central concept, called *context rewriting systems*, which will serve us as a framework for clearing (subword-clearing)

restarting automata and other similar models.

Definition 1.1 ([3]). *Let k be a positive integer. A* k-context rewriting system *(k-CRS for short) is a system $M = (\Sigma, \Gamma, I)$, where Σ is an input alphabet, $\Gamma \supseteq \Sigma$ is a working alphabet not containing the special symbols ¢ and \$, called* sentinels*, and I is a finite set of* instructions *of the form:*

$$(x, z \to t, y)\ ,$$

where x is called left context*, $x \in LC_k = \Gamma^k \cup ¢ \cdot \Gamma^{\leq k-1}$, y is called* right context*, $y \in RC_k = \Gamma^k \cup \Gamma^{\leq k-1} \cdot \$$ and $z \to t$ is called* instruction-rule*, $z, t \in \Gamma^*$. The* width *of the instruction $i = (x, z \to t, y)$ is $|i| = |xzty|$.*

A word $w = uzv$ can be rewritten *into utv (denoted as $uzv \vdash_M utv$) if and only if there exists an instruction $i = (x, z \to t, y) \in I$ such that x is a suffix of $¢ \cdot u$ and y is a prefix of $v \cdot \$$. We often underline the rewritten part of the word w, and if the instruction i is known we use $\vdash_M^{(i)}$ instead of $\vdash_M$, i.e. $u\underline{z}v \vdash_M^{(i)} utv$. The relation $\vdash_M \subseteq \Gamma^* \times \Gamma^*$ is called* rewriting relation*.*

Let $l \in ¢ \cdot \Gamma^* \cup \Gamma^*$, and $r \in \Gamma^* \cup \Gamma^* \cdot \$$. A word $w = uzv$ can be rewritten in the context (l, r) *into utv (denoted as $uzv \to_R utv$* in the context (l, r)*) if and only if there exists an instruction $i = (x, z \to t, y) \in I$, such that x is a suffix of $l \cdot u$ and y is a prefix of $v \cdot r$. Each definition that uses somehow the rewriting relation $\to_R$ can be relativized to any context (l, r). Unless told otherwise, we will use the* standard context $(l, r) = (¢, \$)$*.*

The language *associated with M is defined as $L(M) = \{w \in \Sigma^* \mid w \vdash_M^* \lambda\}$, where $\vdash_M^*$ is the reflexive and transitive closure of $\vdash_M$. Note that, by definition, $\lambda \in L(M)$.*

The characteristic language *associated with M is defined as $L_C(M) = \{w \in \Gamma^* \mid w \vdash_M^* \lambda\}$. Similarly, by definition, $\lambda \in L_C(M)$. Obviously, $L(M) = L_C(M) \cap \Sigma^*$.*

Remark 1.1. *We also include a special case $k = 0$ in Definition 1.1. In this case we define $LC_0 = RC_0 = \{\lambda\}$, and the rest of the definition remains the same.*

Remark 1.2. *We also extend Definition 1.1 with the following notation: if $X \subseteq LC_k$ and $Y \subseteq RC_k$ are finite nonempty sets, and Z is a finite nonempty set of rules of the form $z \to t$, $z, t \in \Gamma^*$, then we define $(X, Z, Y) = \{(x, z \to t, y) \mid x \in X, (z \to t) \in Z, y \in Y\}$. However, if $X = \{x\}$, then instead of writing $(\{x\}, Z, Y)$ we write only (x, Z, Y) for short. The same holds for the sets Z and Y, too.*

Naturally, if we increase the length of contexts used in instructions of a CRS, we can increase their power only.

Remark 1.3. *Based on the above observation, in Definition 1.1 we can allow contexts of any length up to k, i.e. we can use:*
$LC_{\leq k} = \Gamma^{\leq k} \cup ¢ \cdot \Gamma^{\leq k-1} = \bigcup_{i \leq k} LC_i$ *instead of* LC_k *and*
$RC_{\leq k} = \Gamma^{\leq k} \cup \Gamma^{\leq k-1} \cdot \$ = \bigcup_{i \leq k} RC_i$ *instead of* RC_k.

It is easy to see that general k-CRS can simulate any type 0 grammar (according to the Chomsky hierarchy [6]). Hence we will not consider k-CRS in their general form, since they are too powerful (they can represent all recursively enumerable languages). Instead, we will always put some restrictions on the instruction-rules and then study such restricted models. The first model we introduce is the so-called *clearing restarting automaton* which is a k-CRS such that $\Sigma = \Gamma$ and all instruction-rules are of the form $z \to \lambda$, where $z \in \Sigma^+$.

1.2 Clearing Restarting Automata

Definition 1.2 ([3]). *Let k be a nonnegative integer. A k-*clearing restarting automaton *(k-cl-RA for short) is a k-CRS $M = (\Sigma, \Sigma, I)$ (or $M = (\Sigma, I)$, for short), where for each instruction $i = (x, z \to t, y) \in I$: $z \in \Sigma^+$ and $t = \lambda$. Since t is always the empty word, we use the notation $i = (x, z, y)$.*

Remark 1.4. *Speaking about a k-cl-RA M we use "automata terminology," e.g. we say that M* accepts *a word w if $w \in L(M)$. By definition, each k-cl-RA accepts λ. If we say that a k-cl-RA M* recognizes *(or* accepts*) a language L, we always mean that $L(M) = L \cup \{\lambda\}$.*

This implicit acceptance of the empty word can be avoided by a slight modification of the definition of clearing restarting automata, or even context rewriting systems, but in principle, we would not get a more powerful model.

Example 1.1. *Let $M = (\Sigma, I)$ be a 1-cl-RA with $\Sigma = \{a, b\}$ and I consisting of the following two instructions:*

$$\begin{array}{ll} (1) & (a, ab, b), \\ (2) & (\text{¢}, ab, \$). \end{array}$$

Then we have $aaa\underline{ab}bbb \vdash_M^{(1)} aa\underline{ab}bb \vdash_M^{(1)} a\underline{ab}b \vdash_M^{(1)} \underline{ab} \vdash_M^{(2)} \lambda$ which means that $aaaabbbb \vdash_M^* \lambda$. So the word $aaaabbbb$ is accepted by M. It is easy to see that M recognizes the language $L(M) = \{a^n b^n \mid n \geq 0\}$.

Clearing restarting automata are studied in [3]. We only mention that they can recognize all regular languages, some context-free languages and even some non-context-free languages. However, there are some context-free languages that are outside the class of languages accepted by clearing restarting automata.

Theorem 1.1 ([3]). *The language $L = \{a^n c b^n \mid n \geq 0\}$ is not recognized by any k-cl-RA.*

The above limitation led to the development of the extended versions of clearing restarting automata. In [3] there were introduced two extended versions – the so-called Δ-clearing restarting automata and Δ^*-clearing restarting automata. Both of them can use a single auxiliary symbol Δ only. Δ-clearing restarting automata can leave a mark – a symbol Δ – at the place

of deleting besides rewriting into the empty word λ. Δ^*-clearing restarting automata can rewrite a subword w into Δ^k where k is bounded from above by the length of w.

1.3 Δ-Clearing Restarting Automata

Definition 1.3 ([3]). *Let k be a nonnegative integer. A k-Δ-clearing* restarting automaton *(k-Δcl-RA for short) is a system $M = (\Sigma, I)$, where $R = (\Sigma, \Gamma, I)$ is a k-CRS such that $\Delta \notin \Sigma$, $\Gamma = \Sigma \cup \{\Delta\}$, and for each instruction $i = (x, z \to t, y) \in I$: $z \in \Gamma^+$ and either $t = \lambda$, or $t = \Delta$.*

Analogously, a k-Δ^*-clearing restarting automaton *(k-Δ^*cl-RA for short) is a system $M = (\Sigma, I)$, such that for each instruction $i = (x, z \to t, y) \in I$: $z \in \Gamma^+$ and $t = \Delta^i$, where $0 \leq i \leq |z|$.*

The k-Δcl-RA (k-Δ^*cl-RA) M recognizes *the language $L(M) = \{w \in \Sigma^* \mid w \vdash_M^* \lambda\} = L(R)$, where $\vdash_M$ is the* rewriting relation $\vdash_R$ *of $R = (\Sigma, \Gamma, I)$.*

The characteristic language *of M is the language $L_C(M) = L_C(R)$.*

Example 1.2. *Let $M = (\Sigma, I)$ be the 1-Δcl-RA with $\Sigma = \{a, b, c\}$ and the set of instructions I consisting of the following instructions:*

$$\begin{array}{ll} (1) & (a, c \to \Delta, b), \\ (2) & (a, a\Delta b \to \Delta, b), \\ (3) & (\text{¢}, a\Delta b \to \Delta, \$), \\ (4) & (\text{¢}, c \to \Delta, \$), \\ (5) & (\text{¢}, \Delta \to \lambda, \$). \end{array}$$

An input word $a^n c b^n$, for arbitrary $n > 1$, is accepted by M in the following way:

$$a^n \underline{c} b^n \vdash_M^{(1)} a^{n-1} \underline{a\Delta b} b^{n-1} n \vdash_M^{(2)} a^{n-1} \Delta b^{n-1} \vdash_M^{(2)} \ldots \vdash_M^{(2)} \underline{a\Delta b} \vdash_M^{(3)} \underline{\Delta} \vdash_M^{(5)} \lambda \ .$$

First, M deletes c while marking its position by Δ. In each of the following steps, M deletes one a and one b around Δ until it obtains single-letter word Δ, which is then reduced into λ.

It is easy to see that M recognizes the language $L = \{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$.

The characteristic language of M is

$$L_C(M) = \{a^n c b^n, a^n \Delta b^n \mid n \geq 0\} \cup \{\lambda\} \ .$$

It was shown in [3] that Δ^*-clearing restarting automata are powerful enough to recognize all context-free languages. This result was later extended in [4, 5] to hold also for the more restricted Δ-clearing restarting automata.

1.4 Subword-Clearing Restarting Automata

In [1] there was proposed yet another model, the so-called subword-clearing restarting automaton, which proved useful in some grammatical inference scenarios.

Definition 1.4. *Let k be a nonnegative integer. A* k-subword-clearing restarting automaton *(k-scl-RA for short) is a k-CRS $M = (\Sigma, \Sigma, I)$, where for each instruction $i = (x, z \to t, y) \in I$: $z \in \Sigma^+$ and t is a proper subword of z.*

Subword-clearing restarting automata are strictly more powerful than clearing restarting automata. They can, for instance, recognize the language $\{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$, which lies outside the class of languages accepted by clearing restarting automata. However, they still cannot recognize all context-free languages. (Consider e.g. the language $\{ww^R \mid w \in \Sigma^*\}$).

1.5 Grammatical Inference

In this section we introduce a learning schema for clearing (subword-clearing) restarting automata and other similar models. It is possible to identify any hidden target model in the limit by using this schema. We provide only a short introduction for the purposes of this guide. For more details we refer the interested reader to [1].

The problem we are interested in can be best described as follows. Suppose that we have two finite sets of words over the alphabet Σ: the set of positive samples S^+ and the set of negative samples S^-. Our goal is to find an automaton M, such that: $S^+ \subseteq L(M)$ and $S^- \cap L(M) = \emptyset$. We may assume that $S^+ \cap S^- = \emptyset$ and $\lambda \in S^+$.

If we have no other restrictions, then the task becomes trivial even for clearing restarting automata. Just consider the instructions $I = \{(¢, w, \$) \mid w \in S^+, w \neq \lambda\}$. It follows trivially, that in this case $L(M) = S^+$, where $M = (\Sigma, I)$. Therefore, we impose the maximal allowed width $l \geq 1$ and also the specific length $k \geq 0$ of contexts for the instructions of the resulting automaton.

The learning schema itself is defined in Algorithm 1.

Algorithm 1: Learning schema $\mathsf{Infer}(S^+, S^-, l, k)$

Input : The set of positive S^+ and negative S^- samples over Σ, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$. The maximal width of instructions $l \geq 1$. The length of contexts of instructions $k \geq 0$.

Output: An automaton consistent with (S^+, S^-), or **Fail**.

```
Φ ← Assumptions(S⁺, l, k);
while ∃w₋ ∈ S⁻, w₊ ∈ S⁺, φ ∈ Φ : w₋ ⊢_φ w₊ do
    Φ ← Φ \ {φ};
Φ ← Simplify(Φ);
if Consistent(Φ, S⁺, S⁻) then
    return Automaton with instructions Φ;
Fail;
```

First, the function $\mathsf{Assumptions}(S^+, l, k)$ returns some set of instruction candidates. Let us assume, for a moment, that this set already contains all instructions of the hidden target automaton. Then in Cycle 2 we gradually remove all instructions that allow reduction from some negative sample to some positive sample. (These filtered instructions are definitely not in the set of instructions of the hidden target automaton). In Step 4 we remove redundant instructions and in Step 5 we check if the remaining set of instructions is consistent with the given input set of positive and negative samples. In other words, we check if (1) for all $w_+ \in S^+ : w_+ \vdash_\Phi^* \lambda$ and (2) for all $w_- \in S^- : w_- \not\vdash_\Phi^* \lambda$. The condition (1) always holds, provided that in Step 1 we already obtained all instructions of the hidden target automaton. However, the condition (2) may fail. The success of the above algorithm, therefore, depends both on the initial assumptions obtained in Step 1, and on the given set of positive and negative samples. Nevertheless, if we have a "reasonable" implementation of the function $\mathsf{Assumptions}$, then there is always a set of positive samples S_0^+ and a set of negative samples S_0^- such that the above schema converges to a correct solution for all sets of positive samples $S^+ \supseteq S_0^+$ and negative samples $S^- \supseteq S_0^-$ consistent with the hidden target automaton. This also implies that we can infer a correct solution in the limit from any presentation of labeled samples that covers all the samples from S_0^+ and S_0^-.

There are "reasonable" implementations of the function $\mathsf{Assumptions}$ (both for clearing and subword-clearing restarting automata) that run in a polynomial time. In fact, they run in a linear time, if the maximal width of instructions l and the length of contexts k is considered to be a fixed constant.

Example 1.3. *Here we define two functions* $\mathsf{Assumptions}$ *that we use in our inference algorithm.*

1. $\mathsf{Assumptions}_{weak}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$ *and* $\exists w_1, w_2 \in S^+ : xzy$ *is a subword of* $¢w_1\$$ *and* xy *is a subword of* $¢w_2\$\}$.

   The basic intuition behind this procedure is the assumption that if both patterns xzy *and* xy *occur in the set of positive samples, then it is somehow justified to clear the word* z *based on the context* (x, y)*. Note that the more we increase the length of contexts* k *the smaller (or equal) the number of such patterns we will find. The contexts serve here as a safety cushion against the inference of incorrect instructions.*

2. $\mathsf{Assumptions}_{strong}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$ *and* $\exists w_1, w_2 \in S^+ : w_1 = \alpha z \beta, w_2 = \alpha\beta$*,* x *is a suffix of* $¢\alpha$ *and* y *is a prefix of* $\beta\$\}$.

   This condition is more restrictive than the previous one. It basically says that the instruction (x, z, y) *is justified only in the case when there*

are positive samples $w_1, w_2 \in S^+$ such that we can obtain w_2 from w_1 by using this instruction.

All these functions can be computed in a polynomial time with respect to $\mathsf{size}(S^+) = \sum_{w \in S^+} |w|$. In fact, if l and k are fixed constants, then these functions can be computed in a linear time, since we need to consider only subwords of length bounded from above by the constant l.

The above examples can be easily extended to the model of k-scl-RA. The only difference would be that instead of patterns xzy and xy we would consider the patterns xzy and xty, where t is a proper subword of z.

Chapter 2

Application

This chapter shows how to model and investigate clearing (subword-clearing, Δ-clearing, etc.) restarting automata and, in general, all context rewriting systems. For simplicity, we use the term *automaton* to refer to any context rewriting system. In Section 2.1 we show how to install the application both on *Windows* and *UNIX* platforms. In Section 2.2 we explain how to enter the instructions into the application and how to test the properties of correctly defined automata. In Section 2.3 we showcase the inference of automata based on the set of sample reductions. Finally, in Section 2.4 we show how to infer clearing (subword-clearing) automata from the set of positive and negative samples.

2.1 Installation

In this Section we explain how to install the project *Clearing Restarting Automaton* both on *Windows* and *UNIX* platforms.

For *Windows* platform you need to have the .NET Framework of version at least 4.0 installed on your computer. You can download this framework from the Microsoft web site: `http://www.microsoft.com/`. To run the application just double-click on the application executable:

```
ClearingRestartingAutomaton.exe
```

For *UNIX* platform you need to have the Mono project installed on your computer. For more information on the project see the web page: `http://www.mono-project.com/`. If the Mono project is installed correctly you can run the application by entering the following command:

```
> mono ClearingRestartingAutomaton.exe
```

If you want to make modifications to the code we recommend to use *Microsoft Visual Studio* as the development environment.

Both the source code and the executable file can be downloaded freely from the following website:
`http://code.google.com/p/clearing-restarting-automata/`

2.2 Basics

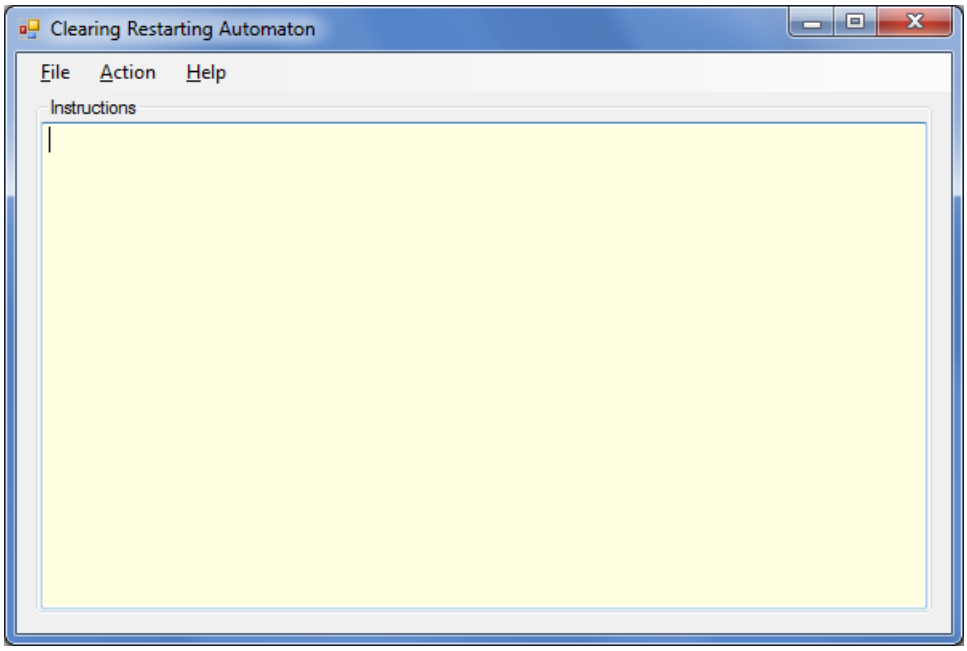

Figure 2.1: Clearing Restarting Automaton.

After starting *Clearing Restarting Automaton* application the window for entering the instructions displays as in Figure 2.1. The instructions entered into this window define the corresponding automaton. In the first part of this Section we explain the syntax of these instructions. After the automaton is correctly defined (i.e. the instructions of the automaton are correctly entered), it is possible to investigate the properties of this automaton (i.e. the language recognized by this automaton etc.). Therefore in the second part of this Section we describe *Reduce/Generate Dialog* that can be used to investigate the properties of the correctly defined automata.

Example 2.1. *Suppose that we want to model the* cl-RA $M = (\{a, b\}, I)$, *where the instructions* I *are:*

$$(a, ab, b),$$
$$(¢, ab, \$).$$

We can represent these instruction as:

```
[a]ab[b]
[^]ab[$]
```

Note that the symbol `^` *represents the left sentinel* ¢ *and the symbol* `$` *represents the right sentinel* \$. *The set notation for instructions is also supported. The set braces are represented by the square brackets* `[` *and* `]`, *and the elements inside these brackets are separated by whitespace or by commas:* `,` , `;` .

For instance, if we want to represent the instruction $(\{¢, a, b\}, ab, \{a, b, \$\})$*, we can do it as follows:*

```
[^ a b] ab [a b $]
```

Note that the whitespace inside the instruction (i.e. between the left and the right context) is ignored.

All instructions of CRS *are supported. For instance, the instruction* $(a, ab \to ba, b)$ *can be represented as:*

```
[a] ab -> ba [b]
```

However, only the following symbols can occur inside the words and contexts:
`a-z A-Z 0-9 ! @ # % & * ( ) ' \ / _ + : " | ? .`
The dot symbol `.` *has a special meaning. It represents the empty word* λ*. This means, for instance, that the following two instructions are equivalent:*

```
[a] ab [b]
[a] ab -> . [b]
```

You can also use the empty contexts in instructions. For instance, the following two instructions are equivalent:

```
[] ab []
[.] ab [.]
```

In the following example we illustrate how to investigate the properties of correctly defined automata.

Example 2.2. *Suppose that we have a* CRS $M = (\Sigma, \Sigma, I)$ *with* $\Sigma = \{a, b\}$ *and the following set of instructions* I*:*

$$(¢, \quad ab \to \lambda, \quad \$), \tag{2.1}$$

$$(a, \quad ab \to \lambda, \quad b), \tag{2.2}$$

$$(\{¢, a, b\}, \quad ba \to ab, \quad \{a, b, \$\}). \tag{2.3}$$

The language $L(M) = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$*. Obviously, if* $w \in L(M)$*, then* $|w|_a = |w|_b$ *(because each instruction preserves this property). On the other hand, if* $w \in \{a, b\}^*$ *and* $|w|_a = |w|_b$ *then by using the instruction (2.3) finitely many times we can get the word* $a^k b^k$ *which can be easily reduced to* λ *by using the instructions (2.1) and (2.2).*

In Clearing Restarting Automaton *application we can represent the instructions of* M *as:*

```
[^] ab [$]
[a] ab [b]
[^ a b] ba -> ab [a b $]
```

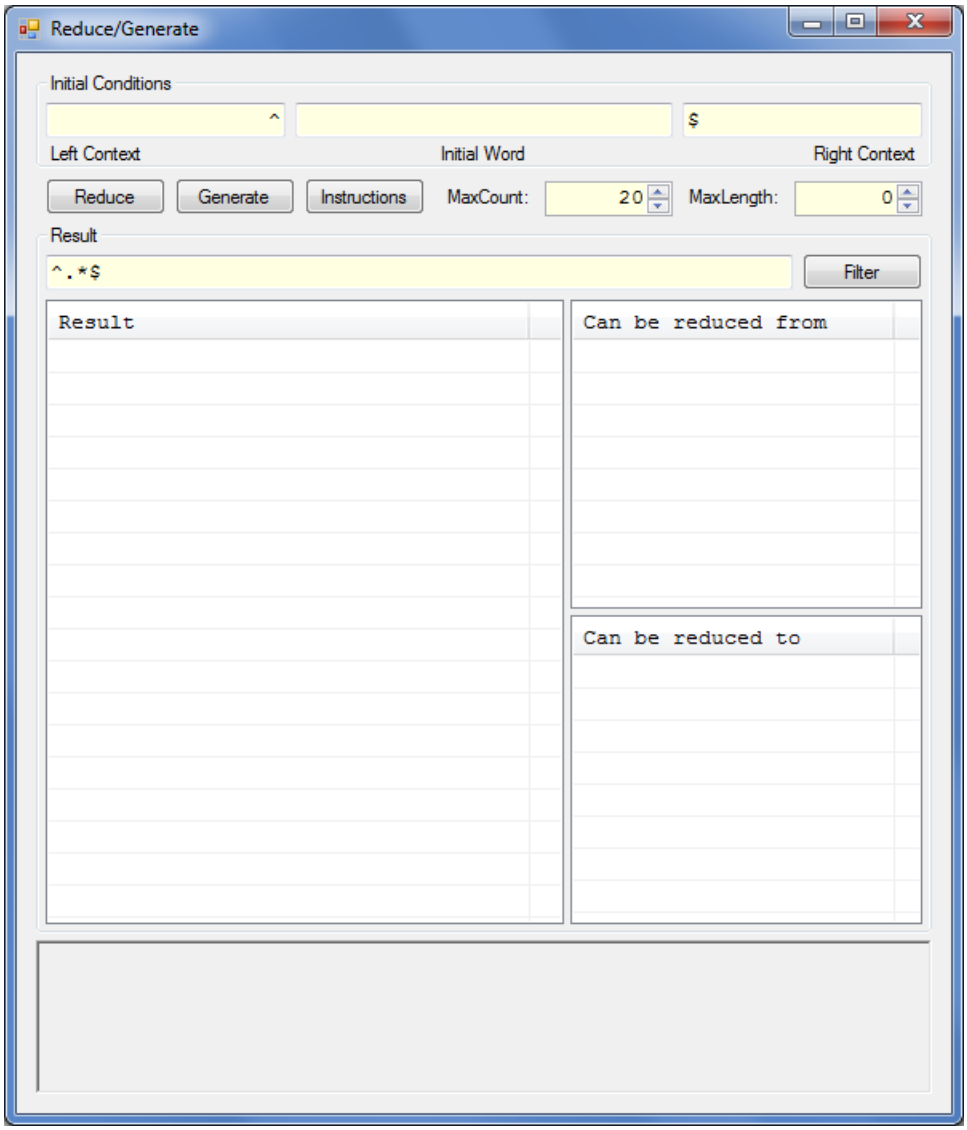

Figure 2.2: Reduce/Generate Dialog.

After entering these instructions into the instruction window click on Action *menu item and then click on* Reduce/Generate *menu item.* Reduce/Generate Dialog *will appear as in Figure 2.2.*

If you click on Generate Button*, the first* 20 *words of the language* $L(M)$ *will appear in* Result ListView *as in Figure 2.3. You can change the number of generated words by modifying* MaxCount *property (*20 *is set by default). The button is called* Generate Button *since the resulting set of words is* generated *from* Initial Word*, which is in our case set to the empty word* λ*, by using the breadth-first search technique. You can also specify the maximal length of the generated words by modifying* MaxLength *property. By default this property is set to* 0 *which means that there is no upper bound on the length of the generated words.*

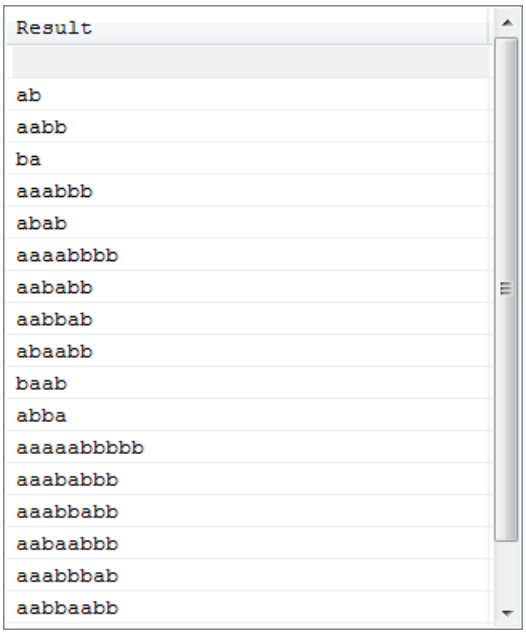

Figure 2.3: Result ListView.

Now click on the word `abba`*. Two smaller* ListView*s on the right side of* Result ListView *show the list of words which the selected word* `abba` *can be* reduced from *and the list of words which the selected word* `abba` *can be* reduced to*. If a word in one of these two smaller* ListView*s has a gray color it means that this word is not in* Result ListView*. For instance, the word* `abba` *can be reduced from the word* `baba`*, but the word* `baba` *is not in* Result ListView*. For illustration see Figure 2.4.*

If you click on the word `baba` *then the reduction step from the word* `baba` *to the word* `abba` *will appear in* Bottom TextBox *together with the used instruction as in Figure 2.5.*

If you double-click on the word `baba` *you will add it to* Result ListView*. However, the word will still have a gray color. In this way you can explore the derivation path of a word in both directions.*

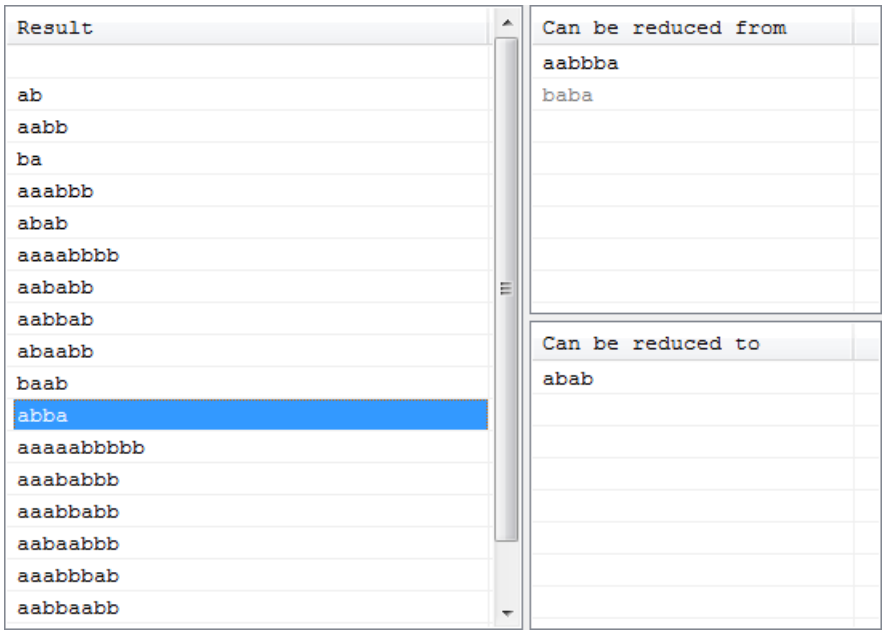

Figure 2.4: Smaller ListViews.

Figure 2.5: Reduction Step.

If you double-click on the word `abba` *in* Result ListView *then the reduction path of this word will appear in* Bottom TextBox *as in Figure 2.6. Only one reduction path is shown in* Bottom TextBox.

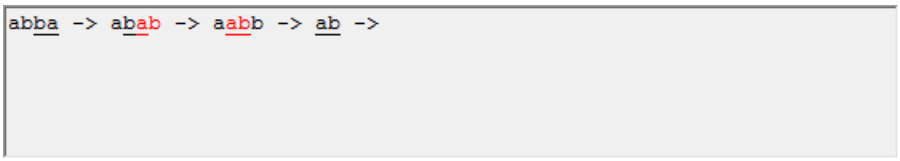

Figure 2.6: Reduction Path.

If the result list of words is large you can use a regular expression *to filter the output. Set* MaxCount *property to* 200 *and click on* Generate Button. *Now enter the regular expression* `^(ba)*$` *to* Filter TextBox *next to* Filter Button *and then click on* Filter Button. Result ListView *is shown in Figure 2.7.*

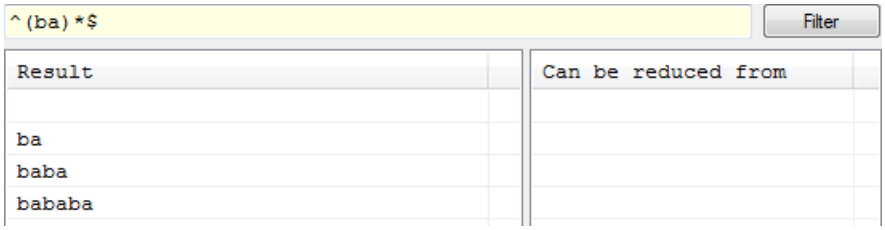

Figure 2.7: Filtered Result ListView.

Reduce Button *can be used in the same way as* Generate Button. *The only difference is that* Reduce Button *is used to find all words that can be reduced from* Initial Word.

If you click on Instructions Button, Information Dialog *will appear with the list of instructions of the used automaton as in Figure 2.8. Note that the set notation is not used here because this list of instructions represents the internal representation of the automaton.*

2.3 Learning from Reductions

In this section we show how to infer the automaton from the set of *sample reductions.*

Example 2.3. *Suppose that we have the following sequence of reductions:*

$$ababababababa\underline{ab} \vdash_M abababababa\underline{ab}abb \vdash_M ababab\underline{ab}abbabb \vdash_M$$
$$ab\underline{ab}abbabbabb \vdash_M abbabbabba\underline{bb} \vdash_M abbabba\underline{bb}ab \vdash_M$$
$$abba\underline{bb}abab \vdash_M a\underline{bb}ababab \vdash_M abababa\underline{ab} \vdash_M$$
$$aba\underline{ab}abb \vdash_M abba\underline{bb} \vdash_M a\underline{bb}ab \vdash_M$$
$$aba\underline{ab} \vdash_M a\underline{bb} \vdash_M \underline{ab} \vdash_M \lambda \text{ accept.}$$

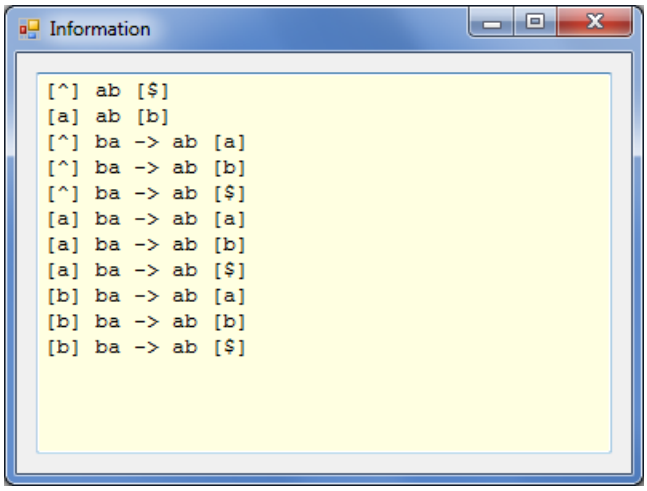

Figure 2.8: Instructions.

From this sample computation*, we can collect* 15 *reductions. All these reductions have unambiguous factorizations (the deleted symbols are underlined).*

To enter these reduction into the application click on Action *menu item and then click on* Learn from Samples of Reductions *menu item.* Learning Dialog *will appear as in Figure 2.9.*

First enter the word `abababababababab` *into* Initial Word TextBox *and then click on* Start Button*. After clicking on* Start Button Learning Step TextBox *will contain this word and also* Learning Process TextBox *will contain this word as the first and the only word. For illustration see Figure 2.10.*

If you want to enter the first reduction:

$$abababababababab \vdash_M abababababababb$$

you only need to select the underlined letter in Learning Step TextBox *as in Figure 2.11 and then you need to click on* Reduce Button*. The situation after clicking on* Reduce Button *is illustrated in Figure 2.12.*

Now we are left with the word `abababababababb` *in* Learning Step TextBox *and we can repeat the same procedure with this word etc. until we enter the whole* sample computation*. The result of this process is shown in Figure 2.13.*

If we want to append another sample computation we just need to enter the first word of this sample computation into Initial Word TextBox *and then click on* Append Button*. The whole process of entering the sample computation will be the same as was described above. Note that if we click on* Start Button *instead of* Append Button *then the whole* Learning Process TextBox *will be cleared in order to enable entering a new sample computation from scratch.*

After we have entered all sample computations we wanted, everything is prepared to infer the corresponding automaton. The only variable we have to

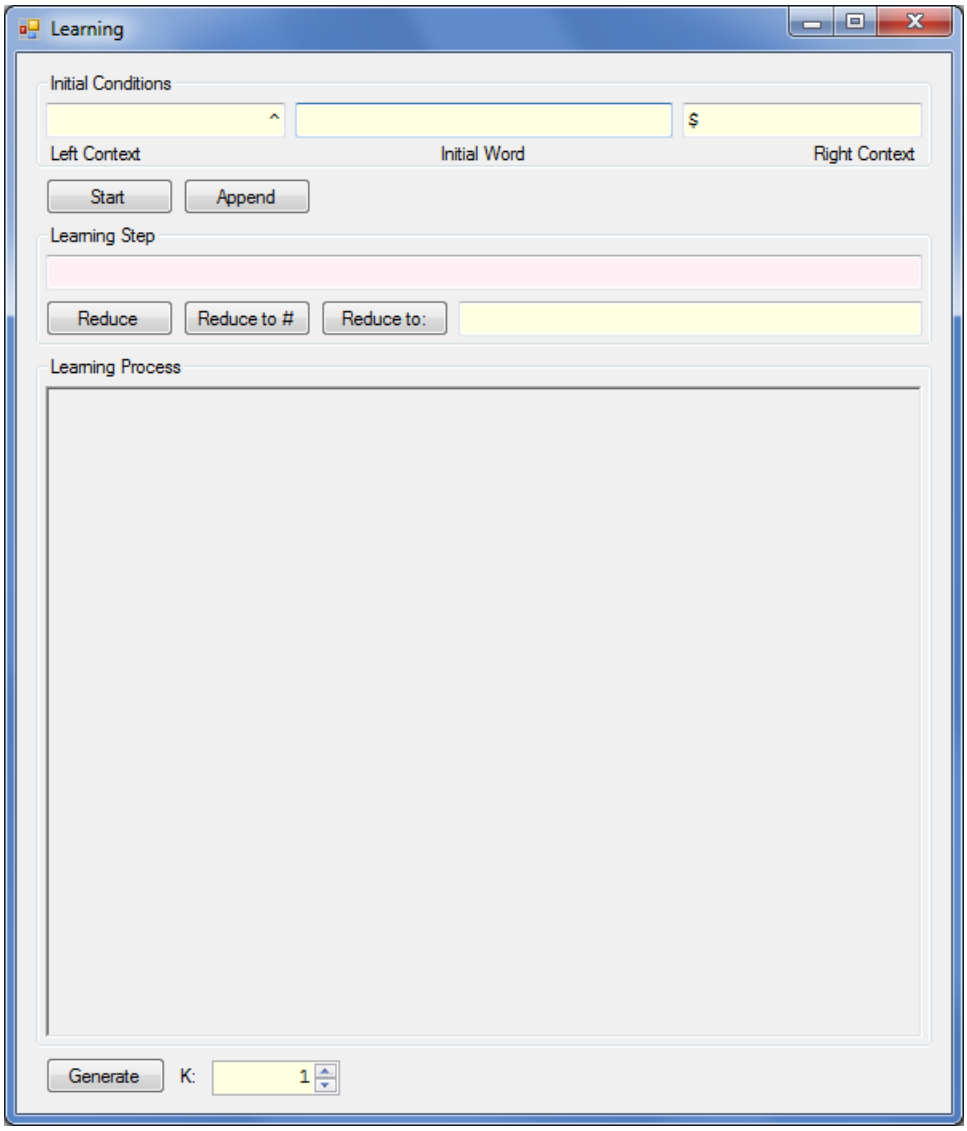

Figure 2.9: Learning Dialog.

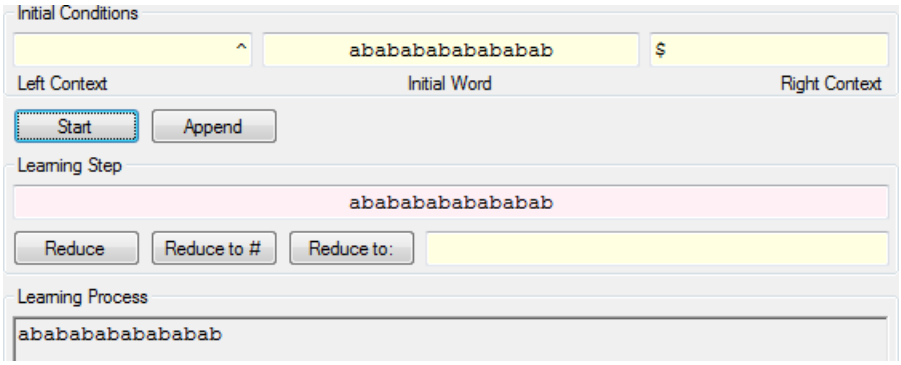

Figure 2.10: Learning Dialog Start.

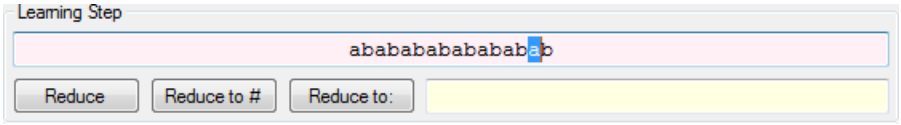

Figure 2.11: Learning Step Select.

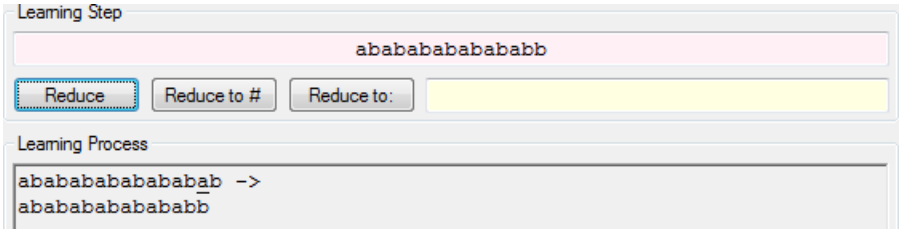

Figure 2.12: Learning Step.

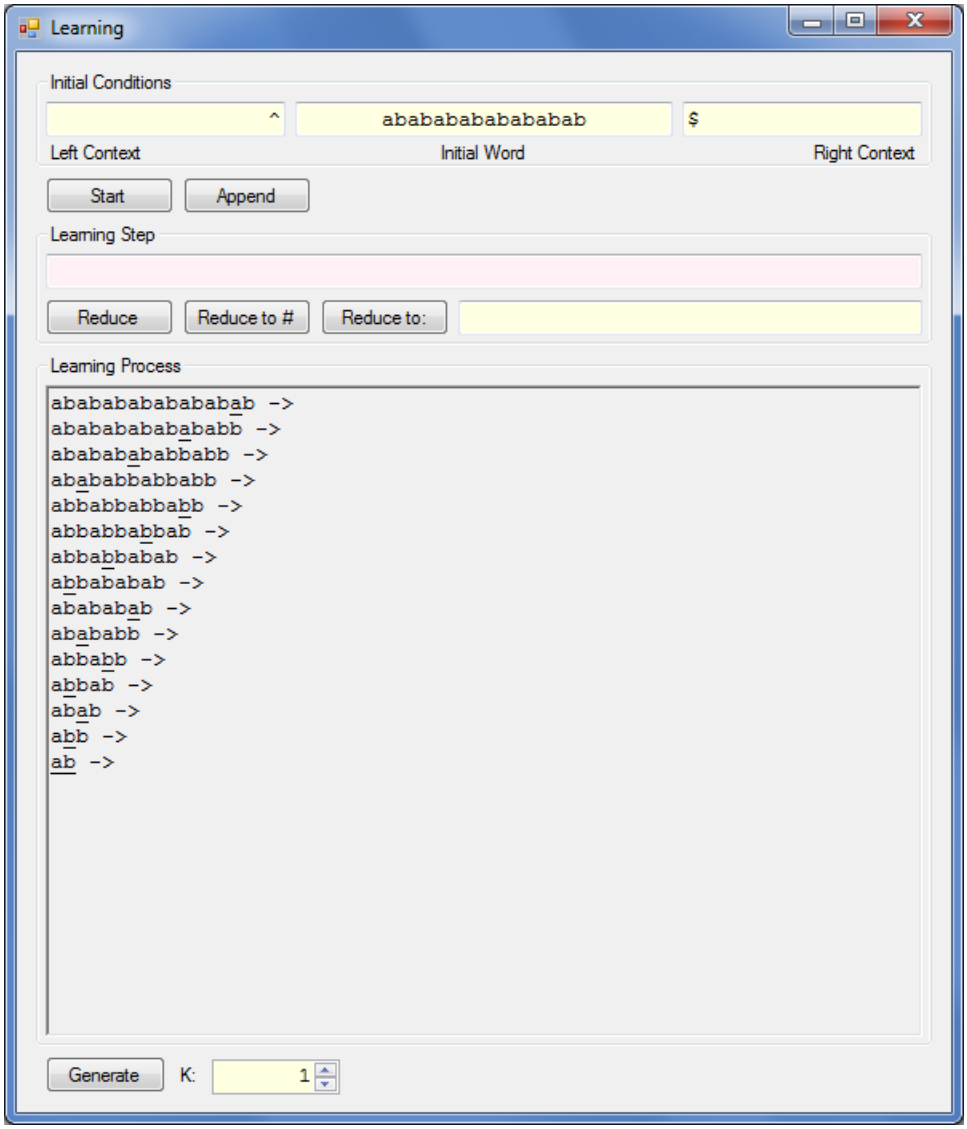

Figure 2.13: Learning Result.

choose is k – the length of the contexts for the instructions. For the purposes of this example we set this parameter to $k = 4$. If we click on Generate Button, Information Dialog *will appear containing the instructions of the resulting inferred automaton as in Figure 2.14.*

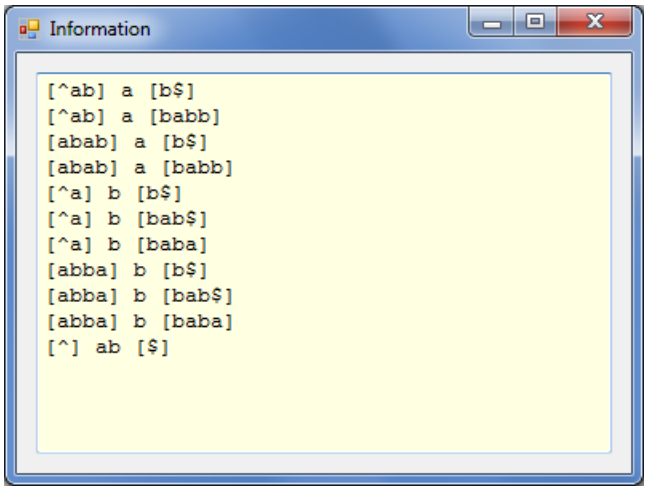

Figure 2.14: Resulting Inferred Automaton.

The resulting inferred 4-cl-RA $M = (\{a, b\}, I)$ *has the following set of instructions* I*:*

$$(\{¢ab, abab\}, a, \{b\$, babb\}), \quad (\{¢a, abba\}, b, \{b\$, bab\$, baba\}), \quad (¢, ab, \$).$$

It can be shown that the following holds:

$$L(M) \cap \{(ab)^n \mid n > 0\} = \{(ab)^{2^l} \mid l \geq 0\}.$$

Suppose that we want to test this hypothesis. First we copy-paste the inferred instructions from Figure 2.14 into the main window of Clearing Restarting Automaton *application (see Figure 2.1). Then we click on* Action *menu item and then on* Reduce/Generate *menu item. We generate, for instance,* 600 *words and then filter them with the following regular expression:* `^(ab)*$`. Result ListView *shown in Figure 2.15 supports our hypothesis.*

Note that there are two other buttons in Learning Dialog*:* Reduce to ♯ Button *and* Reduce to: Button*. These can be used to incorporate the generalized reductions of* Δcl-RA *or even* CRS*. The symbol* ♯ *is usually used to represent the symbol* Δ *of* Δcl-RA*.*

2.4 Learning from Data

In this Section we show how to infer clearing (subword-clearing) automata from the set of positive and negative samples. We use the learning schema described in Section 1.5.

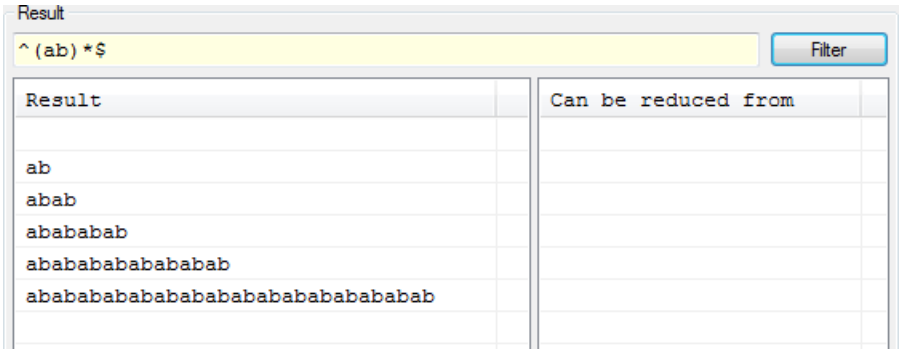

Figure 2.15: Testing the Hypothesis.

Example 2.4. *Consider that we want to infer a* k-cl-RA $M = (\{a, b\}, I)$ *recognizing the language* $L = \{a^n b^n \mid n \geq 0\}$.

First, click on Action *menu item and then click on* Infer from Positive and Negative Samples *menu item.* Inference Dialog *will appear as in Figure 2.16. Note that in* Assumptions SelectBox Weak Clearing Assumptions *item is selected by default. There are several other options, as is shown in Figure 2.17.* Weak Clearing Assumptions *item corresponds to the function* $\mathsf{Assumptions}_{weak}$ *and* Strong Clearing Assumptions *item corresponds to the function* $\mathsf{Assumptions}_{strong}$*, both defined in Example 1.3. The other two items are the equivalents to these two functions for subword-clearing restarting automata. The last two items are only for experimental purposes and are not covered in this Section. The item* SEARCH Clearing Automaton *(*SEARCH Delta-Clearing Automaton*) is used for the exhaustive search for clearing (*Δ*-clearing) restarting automata consistent with the given set of positive and negative samples.*

Since we want to find a clearing restarting automaton, we leave this selection set to the default Weak Clearing Assumptions *item. Our first attempt will be to enter the positive samples:* `ab`, `aabb` *and* `aaabbb` *into* Postive Samples TextBox. *These words can be separated by whitespace or commas:* `,` , `;` . *Since it is not allowed to leave the other* Negative Samples TextBox *empty, we enter the following negative samples into it:* `aab`, `abb`. *After that, we click on* Infer Button. *The result is shown in Figure 2.18 and the inferred consistent automaton is shown in Figure 2.19.*

It is easy to see, that the resulting automaton is consistent with the given set of positive and negative samples, but it does not recognize the target language L*. We only mention that it is sufficient to add the following negative samples:* `aaab`, `abbb`, `aaabb`, `aabbb`, *to* Negative Samples TextBox *in order to obtain the desired automaton. The reason is that these negative samples will filter out the undesired instructions, as is shown in* Debug Output *in Figure 2.20.*

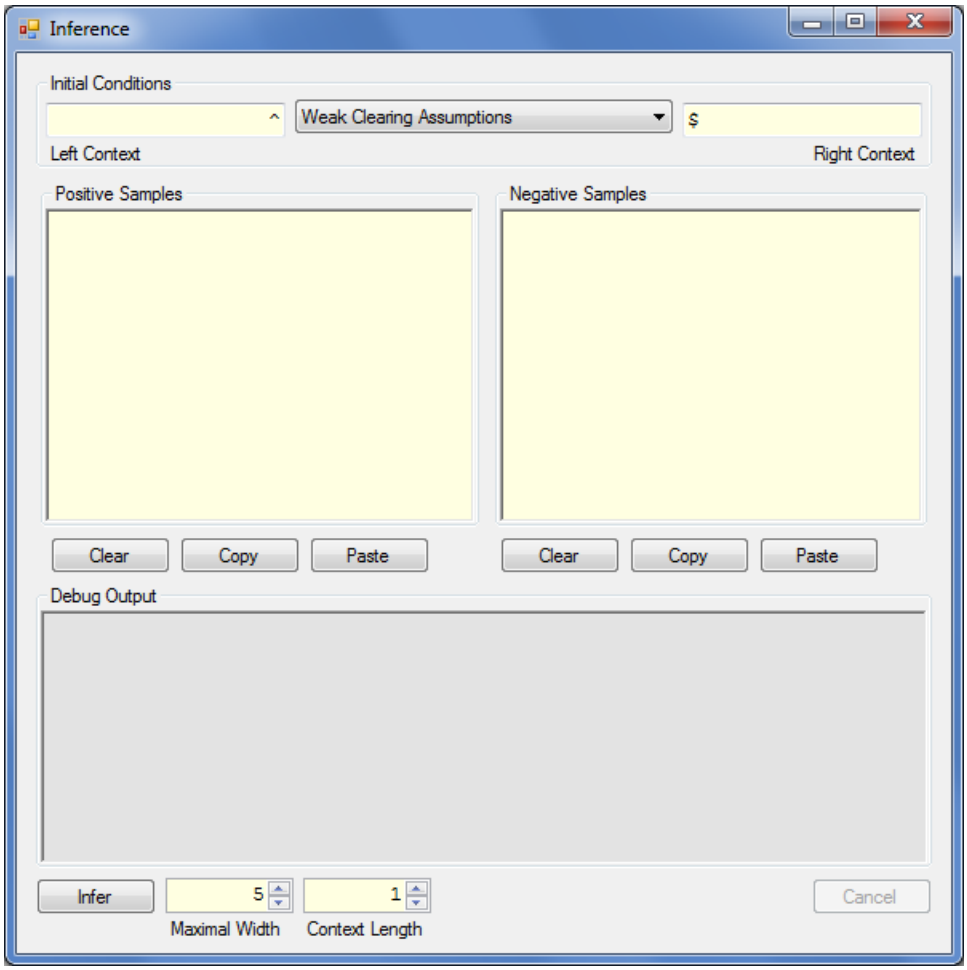

Figure 2.16: Inference Dialog.

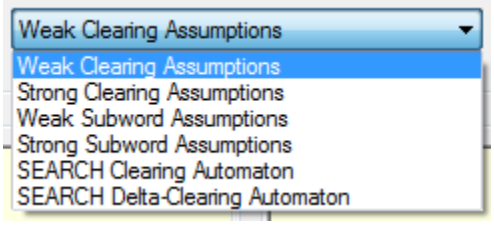

Figure 2.17: Assumptions.

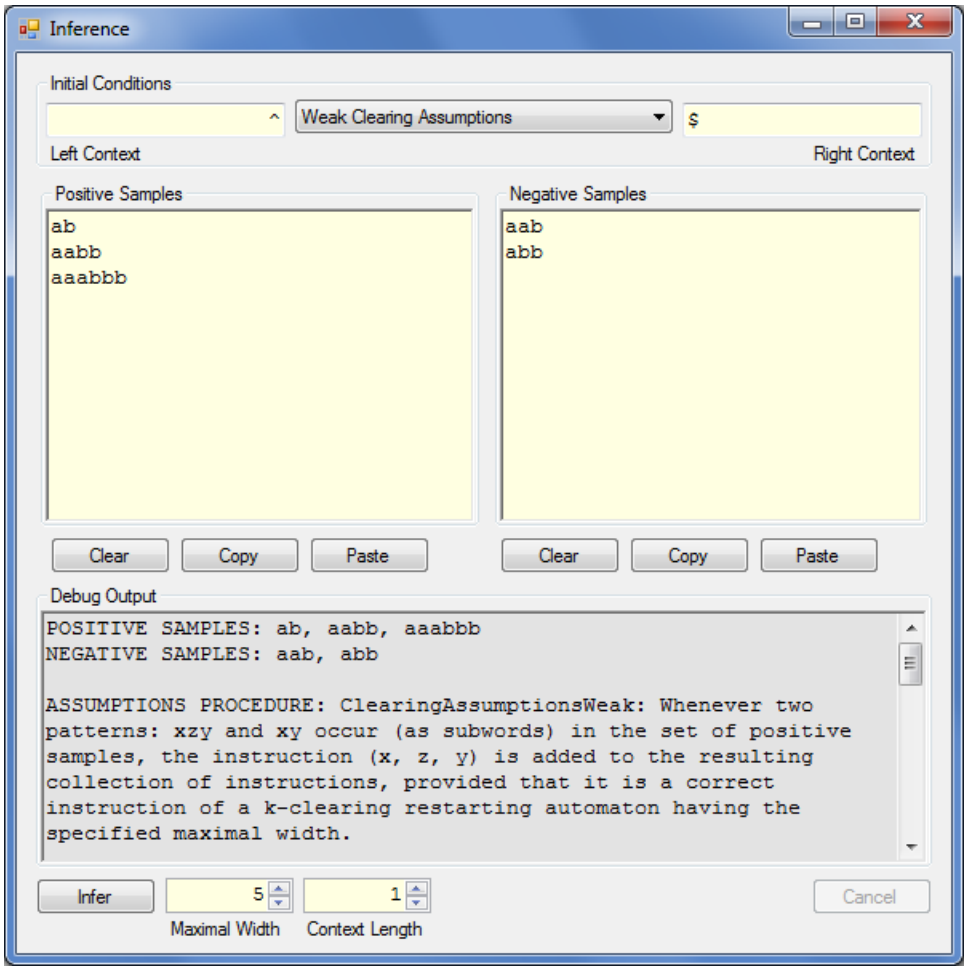

Figure 2.18: Inference Dialog.

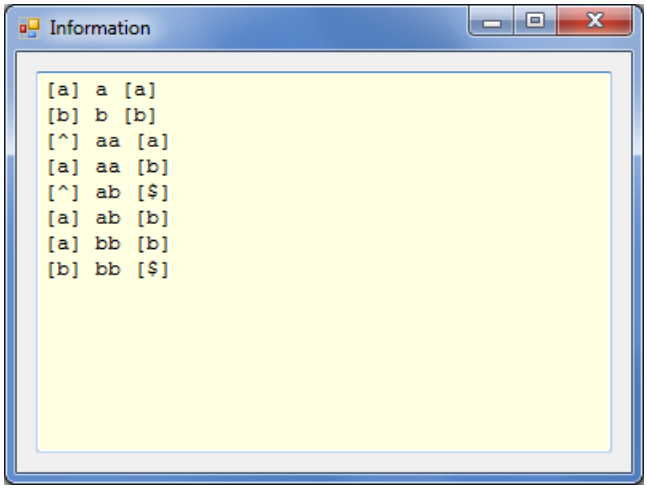

Figure 2.19: Inferred Consistent Automaton.

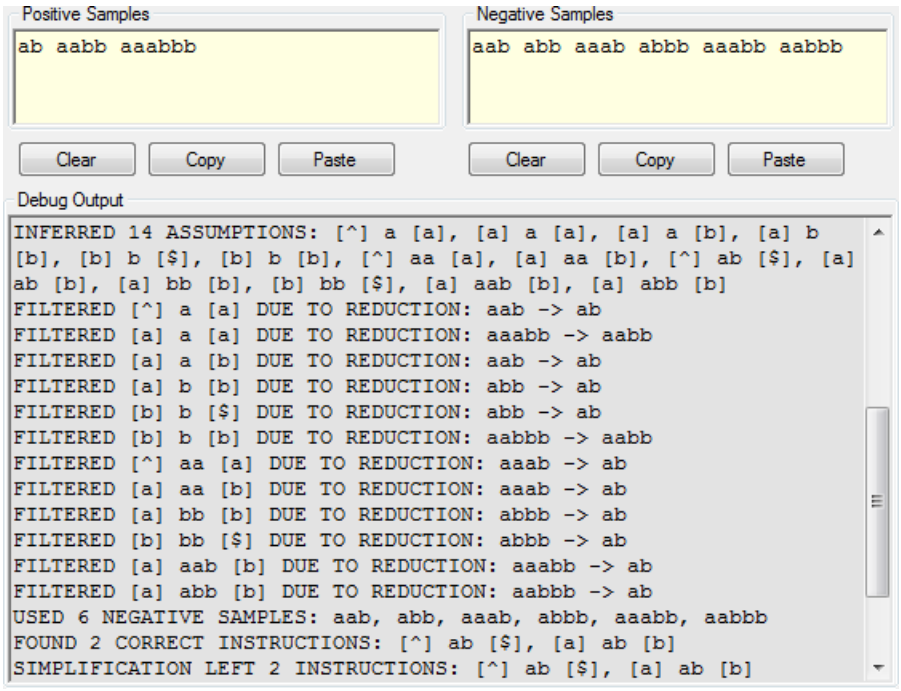

Figure 2.20: Debug Output.

Bibliography

[1] P. Černo. Clearing restarting automata and grammatical inference. Technical report, Charles University, Faculty of Mathematics and Physics, Prague, 2012.

[2] P. Černo and F. Mráz. Clearing restarting automata. In H. Bordinh, R. Freund, M. Holzer, M. Kutrib, and F. Otto, editors, *Workshop on Non-Classical Models for Automata and Applications (NCMA)*, volume 256 of *books@ocg.at*, pages 77–90. Österreichisches Computer Gesellschaft, 2009.

[3] P. Černo and F. Mráz. Clearing restarting automata. *Fundamenta Informaticae*, 104(1):17–54, 2010.

[4] P. Černo and F. Mráz. Delta-clearing restarting automata and cfl. In G. Mauri and A. Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 153–164. Springer Berlin / Heidelberg, 2011.

[5] P. Černo and F. Mráz. Delta-clearing restarting automata and cfl. Technical report, Charles University, Faculty of Mathematics and Physics, Prague, 2011.

[6] J. E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, Reading, 1969.

[7] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel, editor, *FCT'95*, volume 965 of *LNCS*, pages 283–292, Dresden, Germany, August 1995. Springer.

[8] F. Otto. Restarting automata. In Z. Ésik, C. Martín-Vide, and V. Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, Berlin, 2006.