CHARACTERIZATIONS OF LRR-LANGUAGES
BY CORRECTNESS-PRESERVING COMPUTATIONS

František Mráz, Friedrich Otto, and Martin Plátek

Charles University, Prague, Czech Republic

NCMA 2018

Košice

August 21–22, 2018
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata
The main result

A **superclass** of DCFL can be accepted by “well-behaved” restarting automata

the class of left-to-right regular languages (**LRR**)  

- **DCFL** can be accepted deterministically by LR(1)-analyzers with lookahead of size 1
- **LRR** can be accepted deterministically by LR-analyzers with unlimited lookahead; example

\[ \{a^n b^n c, a^n b^{2n} d \mid n \geq 0\} \]
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
A superclass of DCFL can be accepted by “well-behaved” restarting automata:

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
- even more restrictions are possible
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
- even more restrictions are possible
  - deleting only – moreover, deleting at most two continuous factor in one step
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
- even more restrictions are possible
  - deleting only – moreover, deleting at most two continuous factor in one step
  - always reduce the input into a word of small size (limited by a constant) before accepting or rejecting
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
- even more restrictions are possible
  - deleting only – moreover, deleting at most two continuous factor in one step
  - always reduce the input into a word of small size (limited by a constant) before accepting or rejecting
    - expresses the structure for correct inputs
The main result

A superclass of DCFL can be accepted by “well-behaved” restarting automata

- deterministic
- no auxiliary symbols
- length-reducing
- local changes
- monotone
- correctness-preserving
- even more restrictions are possible
  - deleting only – moreover, deleting at most two continuous factor in one step
  - always reduce the input into a word of small size (limited by a constant) before accepting or rejecting
    - expresses the structure for correct inputs
    - shows a core for an error in rejected inputs
1 Automata Models

2 Main Results

3 Strong Cyclic Form

4 Conclusions
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \).
Two-Way Restarting List Automaton

\( M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \)

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma \), \( \Sigma \subseteq \Gamma \).
Two-Way Restarting List Automaton

$M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta)$

- finite state control, set of states $Q$,
- read/write window of a fixed size $k$,
- tape = list of symbols delimited by sentinels $\triangleright$ and $\triangleleft$,
- input alphabet $\Sigma$,
- working alphabet $\Gamma$, $\Sigma \subseteq \Gamma$,
- operations: $\text{MVR}$, $\text{MVL}$, $W(v)$, $\text{SL}(v)$, Restart, Accept, Reject.
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma \), \( \Sigma \subseteq \Gamma \),
- operations: \( \text{MVR, MVL, W (v), SL (v), Restart, Accept, Reject} \).
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma, \Sigma \subseteq \Gamma \),
- operations: MVR, MVL, \( W(v) \), SL (v), Restart, Accept, Reject.
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma \), \( \Sigma \subseteq \Gamma \),
- operations: MVR, MVL, W (v), SL (v), Restart, Accept, Reject.
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma, \Sigma \subseteq \Gamma \),
- operations: MVR, MVL, W \( (v) \), SL \( (v) \), Restart, Accept, Reject.
Two-Way Restarting List Automaton

\[ M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta) \]

- finite state control, set of states \( Q \),
- read/write window of a fixed size \( k \),
- tape = list of symbols delimited by sentinels \( \triangleright \) and \( \triangleleft \),
- input alphabet \( \Sigma \),
- working alphabet \( \Gamma, \Sigma \subseteq \Gamma \),
- operations: MVR, MVL, W (v), SL (v), Restart, Accept, Reject.
Two-Way Restarting List Automaton

$M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta)$

- finite state control, set of states $Q$,
- read/write window of a fixed size $k$,
- tape = list of symbols delimited by sentinels $\triangleright$ and $\triangleleft$,
- input alphabet $\Sigma$,
- working alphabet $\Gamma$, $\Sigma \subseteq \Gamma$,
- operations: MVR, MVL, W (v), SL (v), Restart, Accept, Reject.
Accepted Languages

- A configuration of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\triangleright\} \cdot \Gamma^* \cdot \{\triangleleft\}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$
Accepted Languages

- A configuration of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\rhd \} \cdot \Gamma^* \cdot \{\lhd \}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$

- A restarting configuration: $q_0 \rhd w \lhd$, where $w \in \Gamma^*$
Accepted Languages

- a configuration of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\rhd\} \cdot \Gamma^* \cdot \{\lhd\}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$
- a restarting configuration: $q_0 \rhd w \lhd$, where $w \in \Gamma^*$
- an initial configuration: $q_0 \rhd w \lhd$, where $w \in \Sigma^*$
Accepted Languages

- a **configuration** of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\rhd\} \cdot \Gamma^* \cdot \{\lhd\}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$
- a **restarting configuration**: $q_0 \rhd w \lhd$, where $w \in \Gamma^*$
- an **initial configuration**: $q_0 \rhd w \lhd$, where $w \in \Sigma^*$
- the **input** language of $M$:

$$L(M) = \{w \in \Sigma^* \mid q_0 \rhd w \lhd \downarrow^*_M \text{ Accept}\}.$$
Accepted Languages

- A configuration of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\rangle \} \cdot \Gamma^* \cdot \{\langle \}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$
- A restarting configuration: $q_0 \triangleright w \triangleleft$, where $w \in \Gamma^*$
- An initial configuration: $q_0 \triangleright w \triangleleft$, where $w \in \Sigma^*$
- The input language of $M$:
  \[
  L(M) = \{ w \in \Sigma^* \mid q_0 \triangleright w \triangleleft \vdash_{M}^* \text{Accept} \}.
  \]

- The basic (characteristic) language of $M$:
  \[
  L_C(M) = \{ w \in \Gamma^* \mid q_0 \triangleright w \triangleleft \vdash_{M}^* \text{Accept} \}.
  \]
Accepted Languages

- a configuration of an RLA $M$: $\alpha q \beta$
  - $q$ is the current state
  - $\alpha \beta \in \{\rangle \} \cdot \Gamma^* \cdot \{\langle \}$ is the current contents of the tape
  - contents of the window = the first $k$ symbols of $\beta$
- a restarting configuration: $q_0 \rangle w \langle$, where $w \in \Gamma^*$
- an initial configuration: $q_0 \rangle w \langle$, where $w \in \Sigma^*$
- the input language of $M$:
  
  \[ L(M) = \{ w \in \Sigma^* \mid q_0 \rangle w \langle \vdash^*_M \text{Accept} \} . \]

- the basic (characteristic) language of $M$:
  
  \[ L_C(M) = \{ w \in \Gamma^* \mid q_0 \rangle w \langle \vdash^*_M \text{Accept} \} . \]

- obviously $L_C(M) \cap \Sigma^* = L(M)$. 
Cycles, Reductions
Error and Correctness Preserving Properties

- a cycle = a part of a computation between a restarting configuration and the configuration after a restart step

Fact 1 (Error Preserving Property for basic languages of RLAs).
Let $M$ be an RLA. If $u \Rightarrow cM v$ and $u \not\in L_C(M)$, then $v \not\in L_C(M)$.

Fact 2 (Correctness Preserving Property for basic languages of det-RLAs).
Let $M$ be a deterministic RLA. If $u \Rightarrow cM v$ and $u \in L_C(M)$, then $v \in L_C(M)$.
Cycles, Reductions
Error and Correctness Preserving Properties

- a cycle = a part of a computation between a restarting configuration and the configuration after a restart step
- a tail = a part of a computation after the last restarting configuration and a halting step (Accept or Reject)
a cycle = a part of a computation between a restarting configuration and the configuration after a restart step
a tail = a part of a computation after the last restarting configuration and a halting step (Accept or Reject)
notation:
- $q_0 \triangleright u \triangleleft \nabla^c_M q_0 \triangleright v \triangleleft$ denotes a cycle of $M$
- then we write $u \Rightarrow^c_M v$ – the cycle rewriting relation of $M$
Cycles, Reductions
Error and Correctness Preserving Properties

- a cycle = a part of a computation between a restarting configuration and the configuration after a restart step
- a tail = a part of a computation after the last restarting configuration and a halting step (Accept or Reject)
- notation:
  - $q_0 \triangleright u \triangleleft \xrightarrow{c} M q_0 \triangleright v \triangleleft$ denotes a cycle of $M$
  - then we write $u \Rightarrow^c_M v$ – the cycle rewriting relation of $M$

- if $u \Rightarrow^c_M v$ and $|u| > |v|$, then $u \Rightarrow^c_M v$ is called a reduction
Cycles, Reductions
Error and Correctness Preserving Properties

- a cycle = a part of a computation between a restarting configuration and the configuration after a restart step
- a tail = a part of a computation after the last restarting configuration and a halting step (Accept or Reject)
- notation:
  - \( q_0 \triangleright u \triangleleft \vdash^c_M q_0 \triangleright v \triangleleft \) denotes a cycle of \( M \)
  - then we write \( u \Rightarrow^c_M v \) – the cycle rewriting relation of \( M \)
  - if \( u \Rightarrow^c_M v \) and \( |u| > |v| \), then \( u \Rightarrow^c_M v \) is called a reduction

**Fact 1**

*(Error Preserving Property for basic languages of RLAs).*

Let \( M \) be an RLA. If \( u \Rightarrow^c_M v \) and \( u \notin L_C(M) \), then \( v \notin L_C(M) \).
Cycles, Reductions
Error and Correctness Preserving Properties

- A cycle is a part of a computation between a restarting configuration and the configuration after a restart step.
- A tail is a part of a computation after the last restarting configuration and a halting step (Accept or Reject).
- Notation:
  
  - $q_0 \triangleright u \triangleleft \vdash_M \triangleright q_0 \triangleright v \triangleleft$ denotes a cycle of $M$.
  - Then we write $u \Rightarrow_M v$ – the cycle rewriting relation of $M$.
  - If $u \Rightarrow_M v$ and $|u| > |v|$, then $u \Rightarrow_M v$ is called a reduction.

Fact 1

(Error Preserving Property for basic languages of RLAs).
Let $M$ be an RLA. If $u \Rightarrow_M^{c*} v$ and $u \notin L_C(M)$, then $v \notin L_C(M)$.

Fact 2

(Correctness Preserving Property for basic languages of det-RLAs).
Let $M$ be a deterministic RLA. If $u \Rightarrow_M^{c*} v$ and $u \in L_C(M)$, then $v \in L_C(M)$. 
Refinements and Constraints on RLAs

- **Restricted SL-steps:**
  - A delete-left step (DL-step) = an SL-step which can only delete symbols
  - A contextual-left step (CL-step) = an DL-step which can delete at most two factors

- Notation for \( T \subseteq \{\text{MVR}, \text{MVL}, \text{W}, \text{SL}, \text{DL}, \text{CL}, \text{Restart}\} \): \( T \)-RLA denotes RLAs which can use operations from \( T \cup \{\text{Accept, Reject}\} \)
Let $C = C_k, C_{k+1}, \ldots, C_j$ be a subcomputation and let $C_w = \triangleright \alpha q \beta \triangleleft$ be a configuration from $C$. Then $D_r(C_w) = |\beta \triangleleft|$ is the right distance of $C_w$. 

Monotonicity of rewritings: let $C_1, \ldots, C_n$ be a maximal subsequence of $C$ containing all configurations in which a rewriting occurs. $C$ is monotone if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. $M$ is monotone if all its computations are monotone. $M$ is completely monotone if $D_r(C_\ell) \geq D_r(C_\ell+1)$ holds whenever configuration $C_\ell \triangleright M C_{\ell+1}$.

Fact 3 Each $\{MVR, SL, W\}$-automaton is completely monotone.
Refinements and Constraints on RLAs
Monotonicity and Complete Monotonicity

Let \( C = C_k, C_{k+1}, \ldots, C_j \) be a subcomputation and let \( C_w = \triangleright \alpha q \beta \triangleleft \) be a configuration from \( C \). Then \( D_r(C_w) = |\beta \triangleleft| \) is the right distance of \( C_w \).

Monotonicity of rewritings: let \( C_1, \ldots, C_n \) be a maximal subsequence of \( C \) containing all configurations in which a rewriting occurs. \( C \) is monotone if \( D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n) \).

\( M \) is monotone if all its computations are monotone.
Refinements and Constraints on RLAs
Monotonicity and Complete Monotonicity

- Let \( C = C_k, C_{k+1}, \ldots, C_j \) be a subcomputation and let \( C_w = \triangleright \alpha q \beta \triangleleft \) be a configuration from \( C \). Then \( D_r(C_w) = |\beta \triangleleft| \) is the right distance of \( C_w \).
- Monotonicity of rewritings: let \( C_1, \ldots, C_n \) be a maximal subsequence of \( C \) containing all configurations in which a rewriting occurs. \( C \) is monotone if \( D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n) \). \( M \) is monotone if all its computations are monotone.
- \( M \) is completely monotone if \( D_r(C_\ell) \geq D_r(C_{\ell+1}) \) holds whenever configuration \( C_\ell \vdash_M C_{\ell+1} \).
Let $C = C_k, C_{k+1}, \ldots, C_j$ be a subcomputation and let $C_w = \triangleright \alpha q \beta \triangleleft$ be a configuration from $C$. Then $D_r(C_w) = |\beta| <|$ is the right distance of $C_w$.

Monotonicity of rewritings: let $C_1, \ldots, C_n$ be a maximal subsequence of $C$ containing all configurations in which a rewriting occurs. $C$ is monotone if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. $M$ is monotone if all its computations are monotone.

$M$ is completely monotone if $D_r(C_\ell) \geq D_r(C_{\ell+1})$ holds whenever configuration $C_\ell \vdash_M C_{\ell+1}$.

Fact 3

Each $\{MVR, SL, W\}$-automaton is completely monotone.
Refinements and Constraints on RLAs
Monotonicity and Complete Monotonicity

- Let $C = C_k, C_{k+1}, \ldots, C_j$ be a subcomputation and let $C_w = \triangleright \alpha q \beta \triangleleft$ be a configuration from $C$. Then $D_r(C_w) = |\beta \triangleleft|$ is the right distance of $C_w$.

- Monotonicity of rewritings: let $C_1, \ldots, C_n$ be a maximal subsequence of $C$ containing all configurations in which a rewriting occurs. $C$ is monotone if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. $M$ is monotone if all its computations are monotone.

- $M$ is completely monotone if $D_r(C_\ell) \geq D_r(C_{\ell+1})$ holds whenever configuration $C_\ell \vdash_M C_{\ell+1}$.

Fact 3

Each $\{MVR, SL, W\}$-automaton is completely monotone.

- A $\{MVR, SL, W\}$-automaton with a window of size $k \geq 2$ can be interpreted as a pushdown automaton with a $k$-lookahead and with a limited look under the top of the pushdown.

- A deterministic PDA can be simulated by a $\text{det-}\{MVR, SL, W\}$-automaton with a window of size 1.
An **RLWW-automaton** $M$ is an RLA

1. No $W$-steps (rewritings only by $SL$-steps).
2. Exactly one $SL$-step in each cycle.
3. At most one $SL$-step in each tail computation.
An **RLWW-automaton** $M$ is an RLA

1. No $W$-steps (rewritings only by $SL$-steps).
2. Exactly one $SL$-step in each cycle.
3. At most one $SL$-step in each tail computation.

For **RLWW**-automata, all cycle-rewritings are reductions.
RLWW-automata

- An **RLWW-automaton** $M$ is an RLA
  1. No $W$-steps (rewritings only by SL-steps).
  2. Exactly one SL-step in each cycle.
  3. At most one SL-step in each tail computation.

- For **RLWW**-automata, all cycle-rewritings are reductions.

Different variants of **RLWW**-automata

<table>
<thead>
<tr>
<th>auxiliary symbols possible ($\neg$WW)</th>
<th>SL-steps</th>
<th>DL-steps only</th>
<th>CL-steps only</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVL-steps ($RL\neg$)</td>
<td>RLWW</td>
<td>RLWWD</td>
<td>RLWWC</td>
</tr>
<tr>
<td>no MVL-steps, rewrite followed by restart ($R\neg$)</td>
<td>RWW</td>
<td>RWWD</td>
<td>RWWC</td>
</tr>
<tr>
<td>no auxiliary symbols ($\neg$W)</td>
<td>MVL-steps ($RL\neg$)</td>
<td>RLW</td>
<td>RLWD</td>
</tr>
<tr>
<td>no MVL-steps, rewrite followed by restart ($R\neg$)</td>
<td>RW</td>
<td>RWD</td>
<td>RWC</td>
</tr>
</tbody>
</table>

- For each **RLW**-automaton $M$, $L(M) = L_C(M)$. 

F. Mráz, F. Otto, M. Plátek (Prague) Characterizations of LRR-Languages NCMA 2018 9/24
Correctness Preserving Properties

(a) An RLA-automaton $M$ satisfies the *Complete Weak Correctness Preserving Property (CWCPP) for its basic (input) language* if, for each accepting computation $C_0, C_1, \ldots, C_n$ of $M$, $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, where $u_j$ is the contents of the tape in configuration $C_j$ $(0 \leq j \leq n)$.
Correctness Preserving Properties

(a) An RLA-automaton $M$ satisfies the \textit{Complete Weak Correctness Preserving Property (CWCPP)} for its basic (input) language if, for each accepting computation $C_0, C_1, \ldots, C_n$ of $M$, $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, where $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

(b) An RLA-automaton $M$ satisfies the \textit{Complete Strong Correctness Preserving Property (CSCPP)} for its basic (input) language if, for each computation $C_0, C_1, \ldots, C_n$ of $M$, we have that $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, if $u_i \in L_C(M)$ ($u_i \in L(M)$) for some $i$. Here $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).
Correctness Preserving Properties

(a) An RLA-automaton $M$ satisfies the Complete Weak Correctness Preserving Property (CWCPP) for its basic (input) language if, for each accepting computation $C_0, C_1, \ldots, C_n$ of $M$, $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, where $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

(b) An RLA-automaton $M$ satisfies the Complete Strong Correctness Preserving Property (CSCPP) for its basic (input) language if, for each computation $C_0, C_1, \ldots, C_n$ of $M$, we have that $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, if $u_i \in L_C(M)$ ($u_i \in L(M)$) for some $i$. Here $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

- Complete $\equiv$ each and every operation of the automaton $M$ considered preserves the property of the tape contents to belong to the language $L_C(M)$ ($L(M)$).
Correctness Preserving Properties

(a) An RLA-automaton $M$ satisfies the **Complete Weak Correctness Preserving Property (CWCPP)** for its basic (input) language if, for each accepting computation $C_0, C_1, \ldots, C_n$ of $M$, $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, where $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

(b) An RLA-automaton $M$ satisfies the **Complete Strong Correctness Preserving Property (CSCPP)** for its basic (input) language if, for each computation $C_0, C_1, \ldots, C_n$ of $M$, we have that $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, if $u_i \in L_C(M)$ ($u_i \in L(M)$) for some $i$. Here $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

- Complete $\equiv$ each and every operation of the automaton $M$ considered preserves the property of the tape contents to belong to the language $L_C(M)$ ($L(M)$).
- No intermediate information is stored on the tape.
Correctness Preserving Properties

(a) An RLA-automaton $M$ satisfies the **Complete Weak Correctness Preserving Property (CWCPP)** for its basic (input) language if, for each accepting computation $C_0, C_1, \ldots, C_n$ of $M$, $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, where $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

(b) An RLA-automaton $M$ satisfies the **Complete Strong Correctness Preserving Property (CSCCPP)** for its basic (input) language if, for each computation $C_0, C_1, \ldots, C_n$ of $M$, we have that $u_j \in L_C(M)$ ($u_j \in L(M)$) for all $j = 0, 1, \ldots, n$, if $u_i \in L_C(M)$ ($u_i \in L(M)$) for some $i$. Here $u_j$ is the contents of the tape in configuration $C_j$ ($0 \leq j \leq n$).

- Complete $\equiv$ each and every operation of the automaton $M$ considered preserves the property of the tape contents to belong to the language $L_C(M)$ ($L(M)$).
- No intermediate information is stored on the tape.
- Complete Weak and Strong Correctness Preserving Properties do not depend on the operation of restart.
Example

$L = \{a^n b^n c, a^n b^{2n} d \mid n \geq 0\}$ is accepted by a monotone RWW-automaton $M$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{c, d\}$:
Example

$L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \}$ is accepted by a monotone RWW-automaton $M$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{ c, d \}$:

* either
  * performs the cycle $q_0 \triangleright a^m b^n x \triangleleft \triangleright_M^c q_0 \triangleright a^{m-1} C b^{n-1} x \triangleleft$, where $C$ is an auxiliary symbol (guessing that $x = c$);
  * repeatedly performs CL-steps rewriting $aCb$ into $C$
  * accepts on $\triangleright Cc \triangleleft$
Example

$L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \}$
is accepted by a monotone RWW-automaton $M$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{c, d\}$:

- either
  - performs the cycle $q_0 \triangleright a^m b^n x \triangleleft \vdash^c_M q_0 \triangleright a^{m-1} C b^{n-1} x \triangleleft$, where $C$ is an auxiliary symbol (guessing that $x = c$);
  - repeatedly performs CL-steps rewriting $aCb$ into $C$
  - accepts on $\triangleright Cc \triangleleft$

- or
  - performs the cycle $q_0 \triangleright a^m b^n x \triangleleft \vdash^c_M q_0 \triangleright a^{m-1} D b^{n-2} x \triangleleft$, where $D$ is an auxiliary symbol (guessing that $x = d$);
  - repeatedly performs CL-steps rewriting $aDbb$ into $D$
  - accepts on $\triangleright Dd \triangleleft$
Example

\[ L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \} \]
is accepted by a monotone RWW-automaton \( M \); on input \( a^m b^n x \), where \( m, n \geq 2 \) and \( x \in \{ c, d \} \):

- either
  - performs the cycle \( q_0 \xrightarrow{a^m b^n x} q_0 \xleftarrow{C} a^{m-1} C b^{n-1} x \)
  - repeatedly performs CL-steps rewriting \( aC \) into \( C \)
  - accepts on \( \triangleright Cc \)

- or
  - performs the cycle \( q_0 \xrightarrow{a^m b^n x} q_0 \xleftarrow{D} a^{m-1} D b^{n-2} x \)
  - repeatedly performs CL-steps rewriting \( aDb \) into \( D \)
  - accepts on \( \triangleright Dd \)

In an accepting computation of \( M \), all but the initial configuration contain an occurrence of an auxiliary symbol \( \Rightarrow \) does not satisfy the Complete Weak Correctness Preserving Property for its input language.
Example

$L = \{ a^nb^n c, a^nb^{2n} d \mid n \geq 0 \}$
is accepted by a monotone RWW-automaton $M$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{ c, d \}$:

- either
  - performs the cycle $q_0 \triangleright a^m b^n x \triangleleft \vdash_c^M q_0 \triangleright a^{m-1} C b^{n-1} x \triangleleft$, where $C$ is an auxiliary symbol (guessing that $x = c$);
  - repeatedly performs CL-steps rewriting $aCb$ into $C$
  - accepts on $\triangleright C c \triangleleft$

- or
  - performs the cycle $q_0 \triangleright a^m b^n x \triangleleft \vdash_c^M q_0 \triangleright a^{m-1} D b^{n-2} x \triangleleft$, where $D$ is an auxiliary symbol (guessing that $x = d$);
  - repeatedly performs CL-steps rewriting $aDb b$ into $D$
  - accepts on $\triangleright D d \triangleleft$

In an accepting computation of $M$, all but the initial configuration contain an occurrence of an auxiliary symbol $\Rightarrow$ does not satisfy the Complete Weak Correctness Preserving Property for its input language.

The automaton is monotone.
Example

\[ L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \} \]
is accepted by a deterministic monotone RLW-automaton \( M' \); on input \( a^m b^n x \), where \( m, n \geq 2 \) and \( x \in \{ c, d \} \):

- Scans the given input completely and accepts or rejects words of length 1.
Example

\[ L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \} \]

is accepted by a deterministic monotone RLW-automaton \( M' \); on input \( a^m b^n x \), where \( m, n \geq 2 \) and \( x \in \{ c, d \} \):

- Scans the given input completely and accepts or rejects words of length 1.
- If the last letter is a \( c \), it deletes \( ab \) and restarts;
  if the last letter is a \( d \), it deletes \( abb \) and restarts.
Example

\[ L = \{ a^n b^n c, a^n b^{2n} d \mid n \geq 0 \} \]

is accepted by a deterministic monotone \textit{RLW-automaton} \( M' \); on input \( a^m b^n x \), where \( m, n \geq 2 \) and \( x \in \{ c, d \} \):

- Scans the given input completely and accepts or rejects words of length 1.
- If the last letter is a \( c \), it deletes \( ab \) and restarts; if the last letter is a \( d \), it deletes \( abb \) and restarts.
- The automaton is monotone.
$L = \{a^n b^n c, a^n b^{2n} d \mid n \geq 0\}$ is accepted by a deterministic monotone RLW-automaton $M'$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{c, d\}$:

- Scans the given input completely and accepts or rejects words of length 1.
- If the last letter is a $c$, it deletes $ab$ and restarts; if the last letter is a $d$, it deletes $abb$ and restarts.

- The automaton is monotone.
- The (accepting) computations of $M'$ are much more transparent than those of the RWW-automaton $M$. 
Example

$L = \{a^n b^n c, a^n b^{2n} d \mid n \geq 0\}$
is accepted by a deterministic monotone \textit{RLW}-automaton $M'$; on input $a^m b^n x$, where $m, n \geq 2$ and $x \in \{c, d\}$:

- Scans the given input completely and accepts or rejects words of length 1.
- If the last letter is a $c$, it deletes $ab$ and restarts; if the last letter is a $d$, it deletes $abb$ and restarts.

- The automaton is monotone.
- The (accepting) computations of $M'$ are much more transparent than those of the \textit{RWW}-automaton $M$.
- The \textit{det-RLW}-automaton $M'$ satisfies the Complete Strong Correctness Preserving Property for its input language.
Complete Correctness Preserving Properties for RLWW-Automata

- Each RLW-automaton can be turned into an RLW-automaton that satisfies the Complete Weak Correctness Preserving Property for its input language.
  - Take care of tails – do not rewrite in tails.
- Each deterministic RLW-automaton can be turned into a deterministic RLW-automaton that satisfies the Complete Strong Correctness Preserving Property for its input language.
Complete Correctness Preserving Properties for RLWW-Automata

- Each RLW-automaton can be turned into an RLW-automaton that satisfies the Complete Weak Correctness Preserving Property for its input language.
  - Take care of tails – do not rewrite in tails.

- Each deterministic RLW-automaton can be turned into a deterministic RLW-automaton that satisfies the Complete Strong Correctness Preserving Property for its input language.
Characterization of LRR

- Already known:
  (a) [Jančar, Mráz, Plátek, Vogel, ’99]
    \[ \text{DCFL} = \mathcal{L}(\text{det-mon-RWC}) \subseteq \mathcal{L}(\text{det-mon-RLWC}) \]

- New:
  \[ \text{Theorem 4} \]
  For each det-mon-RLWW-automaton \(M_a\), there exists a det-mon-RLWC-automaton \(M_b\) such that \(L(M_a) = L(M_b)\).

  \[ \text{Proof:} \]
  \(L = L(M_a)\) belongs to the class LRR [ˇCulík II, Cohen, ’73] there exists a deterministic sequential right-to-left transducer \(G\) such that \(L_1 = G(L)\) is a deterministic context-free language.
  We construct a \(\{MVR, MVL, W, CL, \text{Restart}\}\)-automaton \(M_2\) such that \(M_2\) accepts on input \(w\) iff \(G(w) \in L_1\) iff \(w \in L\).
  We simulate \(M_2\) by a det-mon-RLWC-automaton \(M_3\).
Characterization of LRR

Already known:

(a) [Jančar, Mráz, Plátek, Vogel, ’99]
\[ \text{DCFL} = \mathcal{L}(\text{det-mon-RWC}) \subsetneq \mathcal{L}(\text{det-mon-RLWC}) \]

(b) [Otto, ’09] \[ \text{LRR} = \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-RLWD}) \]

New:
Characterization of LRR

- Already known:
  (a) [Jančar, Mráz, Plátek, Vogel, ’99] \( \text{DCFL} = \mathcal{L}(\text{det-mon-RWC}) \subsetneq \mathcal{L}(\text{det-mon-RLWC}) \)
  (b) [Otto,’09] \( \text{LRR} = \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-RLWD}) \)

- New:

**Theorem 4**

*For each det-mon-RLWW-automaton \( M_a \), there exists a det-mon-RLWC-automaton \( M_b \) such that \( L(M_a) = L(M_b) \).*

Proof:
Main Results

Characterization of LRR

- Already known:
  1. [Jančar, Mráz, Plátek, Vogel, ’99] \( DCFL = \mathcal{L}(\text{det-mon-RWC}) \subseteq \mathcal{L}(\text{det-mon-RLWC}) \)
  2. [Otto,’09] \( \text{LRR} = \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-RLWD}) \)

- New:

Theorem 4

For each det-mon-RLWW-automaton \( M_a \), there exists a det-mon-RLWC-automaton \( M_b \) such that \( L(M_a) = L(M_b) \).

Proof:

- \( L = L(M_a) \) belongs to the class LRR
Main Results

Characterization of LRR

- Already known:
  (a) [Jančar, Mráz, Plátek, Vogel, ’99]
    \[ \text{DCFL} = \mathcal{L} \text{(det-mon-RWC)} \subsetneq \mathcal{L} \text{(det-mon-RLWC)} \]
  (b) [Otto,’09] \[ \text{LRR} = \mathcal{L} \text{(det-mon-RLWW)} = \mathcal{L} \text{(det-mon-RLWD)} \]

- New:

Theorem 4

For each det-mon-RLWW-automaton \( M_a \), there exists a det-mon-RLWC-automaton \( M_b \) such that \( L(M_a) = L(M_b) \).

Proof:

- \( L = L(M_a) \) belongs to the class LRR
- [Čulík II, Cohen, ’73] there exists a deterministic sequential right-to-left transducer \( G \) such that \( L_1 = G(L) \) is a deterministic context-free language.
Characterization of LRR

- Already known:
  - (a) [Jančar, Mráz, Plátek, Vogel, ’99] 
    \[ \text{DCFL} = \mathcal{L}(\text{det-mon-RWC}) \subseteq \mathcal{L}(\text{det-mon-RLWC}) \]
  - (b) [Otto,’09] \( \text{LRR} = \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-RLWD}) \)

- New:

**Theorem 4**

*For each det-mon-RLWW-automaton \( M_a \), there exists a det-mon-RLWC-automaton \( M_b \) such that \( L(M_a) = L(M_b) \).*

Proof:

- \( L = L(M_a) \) belongs to the class LRR
- [Čulík II, Cohen, ’73] there exists a deterministic sequential right-to-left transducer \( G \) such that \( L_1 = G(L) \) is a deterministic context-free language.
- We construct a \{MVR, MVL, W, CL, Restart\}-automaton \( M_2 \) such that \( M_2 \) accepts on input \( w \) iff \( G(w) \in L_1 \) iff \( w \in L \).
Characterization of LRR

- Already known:
  
  (a) [Jančar, Mráz, Plátek, Vogel, ’99] DCFL = \( L(\text{det-mon-RWC}) \subset L(\text{det-mon-RLWC}) \)
  
  (b) [Otto,’09] \( LRR = L(\text{det-mon-RLWW}) = L(\text{det-mon-RLWD}) \)

- New:

**Theorem 4**

*For each det-mon-RLWW-automaton \( M_a \), there exists a det-mon-RLWC-automaton \( M_b \) such that \( L(M_a) = L(M_b) \).*

**Proof:**

- \( L = L(M_a) \) belongs to the class LRR
- [Čulík II, Cohen, ’73] there exists a deterministic sequential right-to-left transducer \( G \) such that \( L_1 = G(L) \) is a deterministic context-free language.
- We construct a \( \{\text{MVR, MVL, W, CL, Restart}\} \)-automaton \( M_2 \) such that \( M_2 \) accepts on input \( w \) iff \( G(w) \in L_1 \) iff \( w \in L \).
- We simulate \( M_2 \) by a det-mon-RLWC-automaton \( M_3 \).
Main Results

Characterization of LRR

- $L = L(M_a)$ belongs to the class LRR
- [Čulík II, Cohen, ’73] there exists a deterministic sequential right-to-left transducer $G$ such that $L_1 = G(L)$ is a deterministic context-free language.

\[
\begin{array}{cccc}
  a_1 & \ldots & a_{n-1} & a_n \\
  a_1 & \ldots & a_{n-1} & a_n, p_n \\
  a_1 & \ldots & a_{n-1}, p_{n-1} & a_n, p_n \\
  a_1, p_1 & \ldots & a_{n-1}, p_{n-1} & a_n, p_n \\
\end{array}
\]

$w = G(w)$

- it can be simulated by a \{MVR, MVL, W\}-RLA
- $L_1 = G(L)$ is from DCFL $\Rightarrow$ it is accepted by a det-mon-RWC-automaton $M_1$ – operations \{MVR, CL, Restart\}
- an \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can simulate the composition of $G$ and $M_1$
Characterization of LRR

The \( \{\text{MVR, MVL, W, CL, Restart}\} \)-automaton \( M_2 \) can be simulated by a det-mon-RLWC-automaton \( M_3 \).
Main Results

Characterization of LRR

- the \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can be simulated by a det-mon-RLWC-automaton $M_3$.
  - $M_3$ behaves like $M_2$, but without rewrites of the form $a \rightarrow (a, p_a)$
the \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can be simulated by a det-mon-RLWC-automaton $M_3$.

- $M_3$ behaves like $M_2$, but without rewrites of the form $a \rightarrow (a, p_a)$
- It scans tape from right to left and simulates the transducer $G$ in its finite-state control and remembers the output of $G$ just for the letters inside the window of $M_1$
the \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can be simulated by a det-mon-RLWC-automaton $M_3$.

- $M_3$ behaves like $M_2$, but without rewrites of the form $a \rightarrow (a, p_a)$
- It scans tape from right to left and simulates the transducer $G$ in its finite-state control and remembers the output of $G$ just for the letters inside the window of $M_1$
- The first step of $M_1$ on $G(w)$ can be simulated,
Characterization of LRR

the \{MVR, MVL, W, CL, Restart\}-automaton \(M_2\) can be simulated by a det-mon-RLWC-automaton \(M_3\).

- \(M_3\) behaves like \(M_2\), but without rewrites of the form \(a \rightarrow (a, p_a)\)
- It scans tape from right to left and simulates the transducer \(G\) in its finite-state control and remembers the output of \(G\) just for the letters inside the window of \(M_1\)
- The first step of \(M_1\) on \(G(w)\) can be simulated,
- but if \(M_1\) perform a MVR-step, it must reconstruct the contents of the window
the \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can be simulated by a det-mon-RLWC-automaton $M_3$.

- $M_3$ behaves like $M_2$, but without rewrites of the form $a \rightarrow (a, p_a)$
- It scans tape from right to left and simulates the transducer $G$ in its finite-state control and remembers the output of $G$ just for the letters inside the window of $M_1$
- The first step of $M_1$ on $G(w)$ can be simulated,
- but if $M_1$ perform a MVR-step, it must reconstruct the contents of the window
- for that a mirrored version of a theorem [Aho, Hopcroft, Ullman,’69] can be used
Main Results

Characterization of LRR

- the \{MVR, MVL, W, CL, Restart\}-automaton \(M_2\) can be simulated by a det-mon-RLWC-automaton \(M_3\).

- \(M_3\) behaves like \(M_2\), but without rewrites of the form \(a \rightarrow (a, p_a)\)
- It scans tape from right to left and simulates the transducer \(G\) in its finite-state control and remembers the output of \(G\) just for the letters inside the window of \(M_1\)
- The first step of \(M_1\) on \(G(w)\) can be simulated,
- but if \(M_1\) perform a MVR-step, it must reconstruct the contents of the window
- for that a mirrored version of a theorem [Aho, Hopcroft, Ullman,’69] can be used
   - for each (one-way) deterministic finite-state automaton \(A\) there exists a two-way deterministic finite-state automaton \(B\) such that if \(A\) arrives at some position \(i\) of its input \(x\) in a state \(q_i\), then \(B\) starting at the same position in state \(q_i\), finishes at position \(i - 1\) in the state \(q_{i-1}\)
Characterization of LRR

- The \{MVR, MVL, W, CL, Restart\}-automaton $M_2$ can be simulated by a det-mon-RLWC-automaton $M_3$.
  - $M_3$ behaves like $M_2$, but without rewrites of the form $a \rightarrow (a, p_a)$
  - It scans tape from right to left and simulates the transducer $G$ in its finite-state control and remembers the output of $G$ just for the letters inside the window of $M_1$
  - The first step of $M_1$ on $G(w)$ can be simulated,
  - but if $M_1$ perform a MVR-step, it must reconstruct the contents of the window
  - for that a mirrored version of a theorem [Aho, Hopcroft, Ullman,’69] can be used
    - for each (one-way) deterministic finite-state automaton $A$ there exists a two-way deterministic finite-state automaton $B$ such that if $A$ arrives at some position $i$ of its input $x$ in a state $q_i$, then $B$ starting at the same position in state $q_i$, finishes at position $i - 1$ in the state $q_{i-1}$
  - The first cycle of $M_1$ on $G(w)$ can be simulated; the resulting contents of the tape $x$ is such that $G(x) \in L_1$ iff $G(w) \in L_1$ hence $x \in L$ iff $w \in L$. 
Corollary 5

Each det-mon-RLWC-automaton can be turned into det-mon-\{MVR, MVL, CL\}-automaton satisfying the Complete Strong Correctness Preserving Property for its input language.

Proof:

- Each deterministic RLWC-automaton satisfies the Complete Strong Correctness Preserving Property for its input language.
- An RLWC-automaton can use MVR-, CL- and Restart-steps only
- Simulate each Restart-step by MVL-steps!
Corollary 5

Each det-mon-RLWC-automaton can be turned into det-mon-\{MVR, MVL, CL\}-automaton satisfying the Complete Strong Correctness Preserving Property for its input language.

Proof:
- Each deterministic RLWC-automaton satisfies the Complete Strong Correctness Preserving Property for its input language.
- An RLWC-automaton can use MVR-, CL- and Restart-steps only.
- Simulate each Restart-step by MVL-steps!

Proposition 1

For each det-mon-\{MVR, MVL, SL\}-automaton $M_a$, there exists a det-mon-RLWW-automaton $M_b$ such that $L(M_a) = L(M_b)$.

Proof:
Corollary 5

Each det-mon-RLWC-automaton can be turned into det-mon-\{MVR, MVL, CL\}-automaton satisfying the Complete Strong Correctness Preserving Property for its input language.

Proof:
- Each deterministic RLWC-automaton satisfies the Complete Strong Correctness Preserving Property for its input language.
- An RLWC-automaton can use MVR-, CL- and Restart-steps only
- Simulate each Restart-step by MVL-steps!

Proposition 1

For each det-mon-\{MVR, MVL, SL\}-automaton $M_a$, there exists a det-mon-RLWW-automaton $M_b$ such that $L(M_a) = L(M_b)$.

Proof:
- $M_b$ must restart after simulating an SL-step of $M_a$
- Use the monotonicity! It encodes the state of $M_a$ after an SL-step on its tape – together with the rightmost symbol of the rewritten part
- The next cycle of $M_b$ starts by finding the rightmost tape field encoding also a state
Characterisation of LRR by Automata With Complete Strong Correctness Preserving Property

cscpp- denotes the Complete Strong Correctness Preserving Property.

Corollary 6

\[
\begin{align*}
\text{LRR} & \quad \subseteq (\text{det-mon-RLWW}) \\
& \quad \subseteq (\text{det-mon-\{MVR,MVL,SL \}}) \\
& \quad \subseteq (\text{det-mon-cscpp-RLWC}) \\
& \quad \subseteq (\text{det-mon-cscpp-\{MVR,MVL,SL \}}) \\
& \quad \subseteq (\text{det-mon-cscpp-\{MVR,MVL,CL \}}).
\end{align*}
\]
an RLA is in \textit{weak cyclic form} if before \textit{accepting} it always shortens its tape contents so that it fits in its window.
an RLA is in *weak cyclic form* if before accepting it always shortens its tape contents so that it fits in its window

an RLA is in *strong cyclic form* if before accepting or rejecting it always shortens its tape contents so that it fits in its window
an RLA is in \textit{weak cyclic form} if before accepting it always shortens its tape contents so that it fits in its window

an RLA is in \textit{strong cyclic form} if before accepting or rejecting it always shortens its tape contents so that it fits in its window

\begin{theorem}
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{\text{scf}}$ in strong cyclic form such that $L(M) = L(M_{\text{scf}})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{\text{scf}}} v$.
\end{theorem}
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$. 
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$.  
- The set of words rejected by $M$ in a tail computation is regular – it can accepted by a finite state automaton $A_-$.  

There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$.
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^{c}_{M} v$, also $u \Rightarrow^{c}_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$. 
- The set of words rejected by $M$ in a tail computation is regular – it can accepted by a finite state automaton $A_-$. 
- There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$.
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$. 
- The set of words rejected by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_-$. 
- There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$.

1. $M_{scf}$ accepts or rejects all “short” words.
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow_M^c v$, also $u \Rightarrow_{M_{scf}}^c v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$. 
- The set of words rejected by $M$ in a tail computation is regular – it can accepted by a finite state automaton $A_-$. 
- There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$.

1. $M_{scf}$ accepts or rejects all “short” words
2. On “long” words if tests whether $A_+$ or $A_-$ would accept it; if yes, it cuts out $v$ from the tape suffix and restarts.
Strong Cyclic Form

For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$.
- The set of words rejected by $M$ in a tail computation is regular – it can accepted by a finite state automaton $A_-$.
- There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$

1. $M_{scf}$ accepts or rejects all “short” words
2. On “long” words if tests whether $A_+$ or $A_-$ would accepted it; if yes, it cuts out $v$ from the tape suffix and restarts.

- Monotonicity is preserved.
For each det-mon-RLWC-automaton $M$, there is a det-mon-RLWC-automaton $M_{scf}$ in strong cyclic form such that $L(M) = L(M_{scf})$ and, for all $u \Rightarrow^c_M v$, also $u \Rightarrow^c_{M_{scf}} v$.

- The set of words accepted by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_+$.  
- The set of words rejected by $M$ in a tail computation is regular – it can be accepted by a finite state automaton $A_-$.  
- There exists a constant $c$ such that for each word $z$ from $L(A_+) \cup L(A_-)$ of length at least $c$, there is a factorization $z = uvw$ such that $|vw| \leq c$, $|v| \geq 1$, if $z \in L(A_+)$, then $uw \in L(A_+)$ and if $z \in L(A_-)$, then $uw \in L(A_-)$  
  1. $M_{scf}$ accepts or rejects all “short” words  
  2. On “long” words if tests whether $A_+$ or $A_-$ would accept it; if yes, it cuts out $v$ from the tape suffix and restarts.  
  3. Otherwise, it simulates the next cycle of $M$.  
- Monotonicity is preserved.
Conclusions

RLWW-automata

Corollary 8

For all \( Y \in \{\lambda, \text{scf, scf-cscpp}\} \), the following holds:

\[
\mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-Y-RLW}) = \mathcal{L}(\text{det-mon-Y-RLWD}) = \mathcal{L}(\text{det-mon-Y-RLWC}) = \text{LRR}.
\]

- RLWC-, RLWD-, and RLW-automata can always be modified to satisfy the Complete Weak Correctness Preserving Property for input and basic languages.
- **Deterministic** RLWC-, RLWD-, and RLW-automata can always be modified to satisfy the Complete **Strong** Correctness Preserving Property for input and basic languages.
- General – nondeterministic – RLWW-automata can be modified to satisfy CSCPP only for basic languages.
Corollary 9

For all $X \in \{\{\text{MVR, MVL, SL}\}, \{\text{MVR, MVL, DL}\}, \{\text{MVR, MVL, CL}\}\}$ and all $Y \in \{\lambda, \text{scf, scf-cscpp}\}$, the following holds:

$$\mathcal{L}(\text{det-mon-Y-X}) = \text{LRR}.$$ 

- No Restart-steps
- the language class LRR is robust – characterized by automata both with and without Complete Strong Correctness Preserving Property
Conclusions

Why?

- **det-mon-**\textit{RLWC}-automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:

Further research:
Conclusions

Why?

- **det-mon-RLWC**-automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.

Further research:

To study **det-mon-RLWC**-automata in strong cyclic form having minimal look-ahead window and minimal reductions for a given LRR-language.
Conclusions

Why?

- **det-mon-** _RLWC_ -automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.
  - **det-mon-cscpp-scf-** {**MVR**, **MVL**, **CL**}-automata have this ability, too.

Further research:

To study **det-mon-** _RLWC_ -automata in strong cyclic form having minimal look-ahead window and minimal reductions for a given LRR-language.
Conclusions

Why?

- **det-mon-\textit{RLWC}-automata** in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.
  - **det-mon-cscpp-scf-\{MVR, MVL, CL\}-automata** have this ability, too.
- **det-mon-\textit{RLWC}-automata** in strong cyclic form ensure a deterministic analysis by reduction for the complement of any LRR-language.

Further research:
Conclusions

Why?

- **det-mon-**RLWC-automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.
  - **det-mon-cscpp-scf-\{MVR, MVL, CL\}-automata have this ability, too.**
- **det-mon-**RLWC-automata in strong cyclic form ensure a deterministic analysis by reduction for the complement of any LRR-language.
  - Again, this also holds for
    - **det-mon-cscpp-scf-\{MVR, MVL, CL\}-automata.**

Further research:

- To study **det-mon-**RLWC-automata in strong cyclic form having minimal look-ahead window and minimal reductions for a given LRR-language.

---

F. Mráz, F. Otto, M. Plátek (Prague) Characterizations of LRR-Languages NCMA 2018 23/24
Conclusions

Why?

- **det-mon-** _RLWC_ -automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.
  - **det-mon-cscpp-scf-** \{MVR, MVL, CL\} -automata have this ability, too.
- **det-mon-** _RLWC_ -automata in strong cyclic form ensure a deterministic analysis by reduction for the complement of any LRR-language.
  - Again, this also holds for **det-mon-cscpp-scf-** \{MVR, MVL, CL\} -automata.
  - Localization of syntactical errors and for syntactic error recovery.

Further research:
Conclusions

Why?

- **det-mon-RLWC**-automata in strong cyclic form ensure a deterministic analysis by reduction for LRR-languages:
  - immediate constituents correspond to reductions and the final irreducible sentence.
  - **det-mon-cscpp-scf-**\{MVR, MVL, CL\}-automata have this ability, too.
- **det-mon-RLWC**-automata in strong cyclic form ensure a deterministic analysis by reduction for the complement of any LRR-language.
  - Again, this also holds for **det-mon-cscpp-scf-**\{MVR, MVL, CL\}-automata.
  - Localization of syntactical errors and for syntactic error recovery.

Further research:

- To study **det-mon-RLWC**-automata in strong cyclic form having minimal look-ahead window and minimal reductions for a given LRR-language.
Thank you for your attention!