Non-termination of String and Cycle Rewriting by Automata

Hans Zantema and Alexander Fedotov

Eindhoven, Nijmegen
The Netherlands

WST, September, 2016

Non-termination

Non-termination

A rewrite system is *non-terminating* if there is an infinite sequence $t_0, t_1, t_2, \ldots$ such that $t_i \rightarrow t_{i+1}$ for all i:

$$t_0 \rightarrow t_1 \rightarrow t_2 \cdots$$

Non-termination

A rewrite system is *non-terminating* if there is an infinite sequence $t_0, t_1, t_2, \ldots$ such that $t_i \to t_{i+1}$ for all i:

$$t_0 \to t_1 \to t_2 \cdots$$

First we focus on *string rewriting*:

- a set $\mathcal{R}$ of rules $\ell \to r$ are given, for strings ℓ, r
- a step is a replacement of an occurrence of ℓ by r for a rule $\ell \to r$ in $\mathcal{R}$:
  $u\ell v \to_R urv$ for all $\ell \to r$ in $\mathcal{R}$ and all strings u, v

Non-termination

A rewrite system is *non-terminating* if there is an infinite sequence $t_0, t_1, t_2, \ldots$ such that $t_i \rightarrow t_{i+1}$ for all i:

$$t_0 \rightarrow t_1 \rightarrow t_2 \cdots$$

First we focus on *string rewriting*:

- a set $\mathcal{R}$ of rules $\ell \rightarrow r$ are given, for strings ℓ, r
- a step is a replacement of an occurrence of ℓ by r for a rule $\ell \rightarrow r$ in $\mathcal{R}$:
  $u\ell v \rightarrow_R urv$ for all $\ell \rightarrow r$ in $\mathcal{R}$ and all strings u, v

Later we will consider *cycle rewriting*

Techniques for proving non-termination of string and term rewriting

Techniques for proving non-termination of string and term rewriting

- search for a *loop* = finite computation from a term t to a term containing an instance of t as a subterm: $t \to^* C[t\sigma]$,

Techniques for proving non-termination of string and term rewriting

- search for a *loop* = finite computation from a term t to a term containing an instance of t as a subterm: $t \rightarrow^* C[t\sigma]$,
- search for other patterns from which infinite computations easily follow (Emmes, Enger, Giesl, 2012)

Techniques for proving non-termination of string and term rewriting

- search for a *loop* = finite computation from a term t to a term containing an instance of t as a subterm: $t \to^* C[t\sigma]$,
- search for other patterns from which infinite computations easily follow (Emmes, Enger, Giesl, 2012)
- search for automaton for which accepting language implies non-termination (Endrullis, Zantema, RTA 2015)

Techniques for proving non-termination of string and term rewriting

- search for a *loop* = finite computation from a term t to a term containing an instance of t as a subterm: $t \rightarrow^* C[t\sigma]$,
- search for other patterns from which infinite computations easily follow (Emmes, Enger, Giesl, 2012)
- search for automaton for which accepting language implies non-termination (Endrullis, Zantema, RTA 2015)

We focus on this last approach, and will exend it to cycle rewriting

Example

$ab \to bbaa$ is non-terminating since it admits a loop:

$$abb \to bbaab \to bb\underbrace{abb}aa$$

goes on forever since the instance abb can be rewritten forever

Example

$ab \rightarrow bbaa$ is non-terminating since it admits a loop:

$$abb \rightarrow bbaab \rightarrow bb\underbrace{abb}aa$$

goes on forever since the instance abb can be rewritten forever

Are all non-terminating systems looping?

Example

$ab \to bbaa$ is non-terminating since it admits a loop:

$$abb \to bbaab \to bb\underbrace{abb}aa$$

goes on forever since the instance abb can be rewritten forever

Are all non-terminating systems looping?

No [GeserZantema1999]

Example

$ab \to bbaa$ is non-terminating since it admits a loop:

$$abb \to bbaab \to bb\underbrace{abb}aa$$

goes on forever since the instance abb can be rewritten forever

Are all non-terminating systems looping?

No [GeserZantema1999]

Example

$$bL \to bR,\ Ra \to aR,\ Rb \to Lab,\ aL \to La$$

is non-terminating:

$$bLb \to bRb \to bLab \to bRab \to baRb \to baLab \to bLaab$$

$$\to^+ bLaaab \to^+ bLaaaab \to^+ \cdots$$

but does not admit a loop

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

- $\mathcal{L} \neq \emptyset$

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

- $\mathcal{L} \neq \emptyset$
- $\mathcal{L}$ is closed under rewriting: if $t \in \mathcal{L}$ and $t \to u$ then $u \in \mathcal{L}$, and

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

- $\mathcal{L} \neq \emptyset$
- $\mathcal{L}$ is closed under rewriting: if $t \in \mathcal{L}$ and $t \to u$ then $u \in \mathcal{L}$, and
- $\mathcal{L}$ does not contain normal forms: if $t \in \mathcal{L}$ then $\exists u : t \to u$

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

- $\mathcal{L} \neq \emptyset$
- $\mathcal{L}$ is closed under rewriting: if $t \in \mathcal{L}$ and $t \to u$ then $u \in \mathcal{L}$, and
- $\mathcal{L}$ does not contain normal forms: if $t \in \mathcal{L}$ then $\exists u : t \to u$

then the system is non-terminating: start by $t_0 \in \mathcal{L}$, and for $i = 0, 1, 2, \ldots$ choose t_{i+1} such that $t_i \to t_{i+1}$

Basic idea our approach

If there is a set $\mathcal{L}$ of terms or strings such that

- $\mathcal{L} \neq \emptyset$
- $\mathcal{L}$ is closed under rewriting: if $t \in \mathcal{L}$ and $t \to u$ then $u \in \mathcal{L}$, and
- $\mathcal{L}$ does not contain normal forms: if $t \in \mathcal{L}$ then $\exists u : t \to u$

then the system is non-terminating: start by $t_0 \in \mathcal{L}$, and for $i = 0, 1, 2, \ldots$ choose t_{i+1} such that $t_i \to t_{i+1}$

Even non-WN

The approach

Search for a *regular* $\mathcal{L}$ described by a finite automaton by expressing the above requirements in a SAT formula, and let a SAT solver do the work

The approach

Search for a *regular* $\mathcal{L}$ described by a finite automaton by expressing the above requirements in a SAT formula, and let a SAT solver do the work

Example

The approach

Search for a *regular* $\mathcal{L}$ described by a finite automaton by expressing the above requirements in a SAT formula, and let a SAT solver do the work

Example

For

$$bL \to bR,\ Ra \to aR,\ Rb \to Lab,\ aL \to La$$

a compatible language $\mathcal{L}$ is described by the regular expression

$$b\ a^*\ (L + R)\ a^*\ b$$

The approach

Search for a *regular* $\mathcal{L}$ described by a finite automaton by expressing the above requirements in a SAT formula, and let a SAT solver do the work

Example

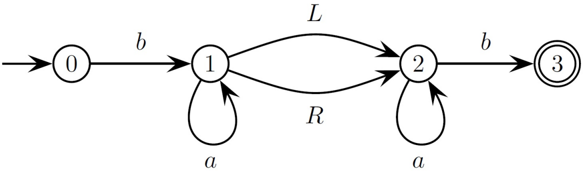
For

$$bL \to bR,\ Ra \to aR,\ Rb \to Lab,\ aL \to La$$

a compatible language $\mathcal{L}$ is described by the regular expression

$$b\ a^*\ (L + R)\ a^*\ b$$

also described by the automaton



SAT solving

SAT solving

Fixing a rewrite system and a number n, we want to find an automaton A on n states fully automatically such that its accepting language $\mathcal{L}$ satisfies all our requirements, hence proving non-termination

SAT solving

Fixing a rewrite system and a number n, we want to find an automaton A on n states fully automatically such that its accepting language $\mathcal{L}$ satisfies all our requirements, hence proving non-termination

How to do this?

SAT solving

Fixing a rewrite system and a number n, we want to find an automaton A on n states fully automatically such that its accepting language $\mathcal{L}$ satisfies all our requirements, hence proving non-termination

How to do this?

Introduce mn^2 boolean variables v_{ija} describing whether there is an a-transition from state i to state j, for i, j running over all n states, and a running over all m symbols

SAT solving

Fixing a rewrite system and a number n, we want to find an automaton A on n states fully automatically such that its accepting language $\mathcal{L}$ satisfies all our requirements, hence proving non-termination

How to do this?

Introduce mn^2 boolean variables v_{ija} describing whether there is an a-transition from state i to state j, for i, j running over all n states, and a running over all m symbols

Build a formula on these variables that is satisfiable if and only if the accepting language $\mathcal{L}$ satisfies all our requirements

SAT solving

Fixing a rewrite system and a number n, we want to find an automaton A on n states fully automatically such that its accepting language $\mathcal{L}$ satisfies all our requirements, hence proving non-termination

How to do this?

Introduce mn^2 boolean variables v_{ija} describing whether there is an a-transition from state i to state j, for i, j running over all n states, and a running over all m symbols

Build a formula on these variables that is satisfiable if and only if the accepting language $\mathcal{L}$ satisfies all our requirements

Apply a SAT solver to this formula: if it is satisfiable then $\mathcal{L}$ satisfies the requirements proving non-termination

Expressing the requirements (RTA2015)

Expressing the requirements (RTA2015)

- $\mathcal{L} \neq \emptyset$: there is a path in A from the initial state to a final state

Expressing the requirements (RTA2015)

- $\mathcal{L} \neq \emptyset$: there is a path in A from the initial state to a final state
- Closed under rewriting (overapproximation):
  for every rule $\ell \to r$ and every two states i, j in A for which there is a path labeled by ℓ from i to j, there is also a path labeled by r from i to j

Expressing the requirements (RTA2015)

- $\mathcal{L} \neq \emptyset$: there is a path in A from the initial state to a final state
- Closed under rewriting (overapproximation):
  for every rule $\ell \rightarrow r$ and every two states i, j in A for which there is a path labeled by ℓ from i to j, there is also a path labeled by r from i to j
- $\mathcal{L}$ contains no normal forms:
  In a preprocessing build an automaton A' exactly accepting the normal forms, then in the product automaton $A \times A'$ there should be no path from the initial state to a state (f, f') for which f is final in A and f' is final in A'

Observations (RTA2015)

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail
- Extends to *term rewriting* by applying *tree automata*, yielding the first automatic non-termination proof for the S-rule

  $$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

  being one of the building blocks of combinatory logic

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail
- Extends to *term rewriting* by applying *tree automata*, yielding the first automatic non-termination proof for the S-rule

  $$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

  being one of the building blocks of combinatory logic
- Not only for non-WN: variants also apply to prove non-termination for systems that are WN

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail
- Extends to *term rewriting* by applying *tree automata*, yielding the first automatic non-termination proof for the S-rule

  $$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

  being one of the building blocks of combinatory logic
- Not only for non-WN: variants also apply to prove non-termination for systems that are WN

New insights (2016)

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail
- Extends to *term rewriting* by applying *tree automata*, yielding the first automatic non-termination proof for the S-rule

  $$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

  being one of the building blocks of combinatory logic
- Not only for non-WN: variants also apply to prove non-termination for systems that are WN

New insights (2016)

- Exploit simulations

Observations (RTA2015)

- This works well: for many examples a corresponding proof of non-termination is found fully automatically by a SAT solver, where all earlier techniques fail
- Extends to *term rewriting* by applying *tree automata*, yielding the first automatic non-termination proof for the S-rule

$$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

  being one of the building blocks of combinatory logic
- Not only for non-WN: variants also apply to prove non-termination for systems that are WN

New insights (2016)

- Exploit simulations
- Extend the approach to *cycle rewriting*

Simulations

A relation $\sim$ on the states of an automaton is called a *simulation* if for every p, q, r, a such that $p \sim q$ and $p \xrightarrow{a} r$, there exists a state s such that $q \xrightarrow{a} s$ and $r \sim s$:

$$\begin{array}{ccc} p & \xrightarrow{a} & r \\ \wr & & \wr \\ q & \xrightarrow{a} & s \end{array}$$

Simulations

A relation $\sim$ on the states of an automaton is called a *simulation* if for every p, q, r, a such that $p \sim q$ and $p \xrightarrow{a} r$, there exists a state s such that $q \xrightarrow{a} s$ and $r \sim s$:

$$\begin{array}{ccc} p & \xrightarrow{a} & r \\ \wr & & \wr \\ q & \xrightarrow{a} & s \end{array}$$

Consequence: for any string u we have

$$\begin{array}{ccc} p & \xrightarrow{u} & r \\ \wr & & \wr \\ q & \xrightarrow{u} & s \end{array}$$

Simulations

A relation $\sim$ on the states of an automaton is called a *simulation* if for every p, q, r, a such that $p \sim q$ and $p \xrightarrow{a} r$, there exists a state s such that $q \xrightarrow{a} s$ and $r \sim s$:

$$\begin{array}{ccc} p & \xrightarrow{a} & r \\ \wr & & \wr \\ q & \xrightarrow{a} & s \end{array}$$

Consequence: for any string u we have

$$\begin{array}{ccc} p & \xrightarrow{u} & r \\ \wr & & \wr \\ q & \xrightarrow{u} & s \end{array}$$

A *backward simulation* is a simulation in the automaton obtained by reversing all arrows

Simulations are helpful for the properties for string rewriting:

Simulations are helpful for the properties for string rewriting:

- Closed under rewriting:
  for every rule $\ell \to r$ and every two states i, j in A for which there is a path labeled by ℓ from i to j, no path labeled by r from i to j is required, but only from i' to j' for i', j' related to i, j by suitable simulations

Simulations are helpful for the properties for string rewriting:

- Closed under rewriting:
  for every rule $\ell \to r$ and every two states i, j in A for which there is a path labeled by ℓ from i to j, no path labeled by r from i to j is required, but only from i' to j' for i', j' related to i, j by suitable simulations

- $\mathcal{L}$ contains no normal forms:
  No big product automaton required any more: if M accepts non-normal forms, then for $\mathcal{L} = L(A)$ we require $\mathcal{L} \subseteq L(M)$, to be expressed by simulating any path from initial to final in A by a similar path in M

Cycle rewriting

Cycle rewriting

Cycle rewriting is string rewriting modulo cyclic shift, that is, the equivalence relation $\simeq$ defined by $uv \simeq vu$ for all u, v

Cycle rewriting

Cycle rewriting is string rewriting modulo cyclic shift, that is, the equivalence relation $\simeq$ defined by $uv \simeq vu$ for all u, v

Termination of cycle rewriting is a stronger property than termination of string rewriting, for instance,

$$ab \to ba$$

is terminating in string rewriting, but not in cycle rewriting

Cycle rewriting

Cycle rewriting is string rewriting modulo cyclic shift, that is, the equivalence relation $\simeq$ defined by $uv \simeq vu$ for all u, v

Termination of cycle rewriting is a stronger property than termination of string rewriting, for instance,

$$ab \to ba$$

is terminating in string rewriting, but not in cycle rewriting

[ZBK14] and [SZ15] present techniques for proving termination of cycle rewriting, but how to prove non-termination?

Cycle rewriting is string rewriting modulo cyclic shift, that is, the equivalence relation $\simeq$ defined by $uv \simeq vu$ for all u, v

Termination of cycle rewriting is a stronger property than termination of string rewriting, for instance,

$$ab \to ba$$

is terminating in string rewriting, but not in cycle rewriting

[ZBK14] and [SZ15] present techniques for proving termination of cycle rewriting, but how to prove non-termination?

First approach: search for pattern $uv \to^+ vu^+$

Cycle rewriting

Cycle rewriting is string rewriting modulo cyclic shift, that is, the equivalence relation $\simeq$ defined by $uv \simeq vu$ for all u, v

Termination of cycle rewriting is a stronger property than termination of string rewriting, for instance,

$$ab \rightarrow ba$$

is terminating in string rewriting, but not in cycle rewriting

[ZBK14] and [SZ15] present techniques for proving termination of cycle rewriting, but how to prove non-termination?

First approach: search for pattern $uv \rightarrow^+ vu^+$

We adapt our automata based approach for cycle rewriting

Non-termination of cycle rewriting

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

- $L \neq \emptyset$,

Non-termination of cycle rewriting

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

- $L \neq \emptyset$,
- *if* $\ell \rightarrow r \in R$, $u, v \in \Sigma^*$ *and* $u\ell v \in L$ *then either* $urv \in L$ *or there exist* $r_1 \neq \epsilon \neq r_2$ *such that* $r = r_1 r_2$ *and* $r_2 v u r_1 \in L$,

Non-termination of cycle rewriting

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

- $L \neq \emptyset$,
- *if* $\ell \to r \in R$, $u, v \in \Sigma^*$ *and* $u\ell v \in L$ *then either* $urv \in L$ *or there exist* $r_1 \neq \epsilon \neq r_2$ *such that* $r = r_1 r_2$ *and* $r_2 v u r_1 \in L$,
- *if* $\ell_1 \ell_2 \to r \in R$, $u \in \Sigma^*$ *and* $\ell_2 u \ell_1 \in L$, *then there exist* r_1, r_2 *such that* $r = r_1 r_2$ *and* $r_2 u r_1 \in L$,

Non-termination of cycle rewriting

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

- $L \neq \emptyset$,
- *if* $\ell \to r \in R$, $u, v \in \Sigma^*$ *and* $u\ell v \in L$ *then either* $urv \in L$ *or there exist* $r_1 \neq \epsilon \neq r_2$ *such that* $r = r_1 r_2$ *and* $r_2 v u r_1 \in L$,
- *if* $\ell_1 \ell_2 \to r \in R$, $u \in \Sigma^*$ *and* $\ell_2 u \ell_1 \in L$, *then there exist* r_1, r_2 *such that* $r = r_1 r_2$ *and* $r_2 u r_1 \in L$,
- $L \subseteq L'$ *for* L' *consisting of all strings for which a cyclic shift can be rewritten*

Non-termination of cycle rewriting

Theorem

Let R be an SRS over Σ and $L \subseteq \Sigma^*$ satisfy

- $L \neq \emptyset$,
- *if* $\ell \to r \in R$, $u, v \in \Sigma^*$ *and* $u\ell v \in L$ *then either* $urv \in L$ *or there exist* $r_1 \neq \epsilon \neq r_2$ *such that* $r = r_1 r_2$ *and* $r_2 v u r_1 \in L$,
- *if* $\ell_1 \ell_2 \to r \in R$, $u \in \Sigma^*$ *and* $\ell_2 u \ell_1 \in L$, *then there exist* r_1, r_2 *such that* $r = r_1 r_2$ *and* $r_2 u r_1 \in L$,
- $L \subseteq L'$ *for* L' *consisting of all strings for which a cyclic shift can be rewritten*

Then R is not cycle terminating

By building an automaton M satisfying $L(M) = L'$ in advance, all these properties can be expressed in SAT by exploiting forward and backward simulations

By building an automaton M satisfying $L(M) = L'$ in advance, all these properties can be expressed in SAT by exploiting forward and backward simulations

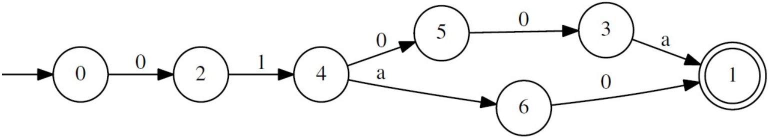
Example

$R = \{00a \to a0, 1a \to a01\}$

By building an automaton M satisfying $L(M) = L'$ in advance, all these properties can be expressed in SAT by exploiting forward and backward simulations

Example
$R = \{00a \to a0, 1a \to a01\}$

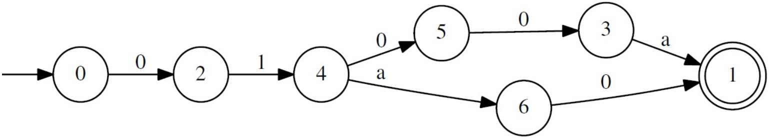
The above approach yields



By building an automaton M satisfying $L(M) = L'$ in advance, all these properties can be expressed in SAT by exploiting forward and backward simulations

Example
$R = \{00a \to a0, 1a \to a01\}$

The above approach yields



Indeed $L = \{0100a, 01a0\}$ satisfies all properties

Conclusions

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015
- Instead of looking for a particular infinite computation, we look for a (regular) language (= set of strings) with some properties from which non-termination easily follows

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015
- Instead of looking for a particular infinite computation, we look for a (regular) language (= set of strings) with some properties from which non-termination easily follows
- Properties are expressed in a SAT formula; as often in (non-)termination the real search for a proof is done by a SAT solver

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015
- Instead of looking for a particular infinite computation, we look for a (regular) language (= set of strings) with some properties from which non-termination easily follows
- Properties are expressed in a SAT formula; as often in (non-)termination the real search for a proof is done by a SAT solver
- We exploited simulations to improve this approach

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015
- Instead of looking for a particular infinite computation, we look for a (regular) language (= set of strings) with some properties from which non-termination easily follows
- Properties are expressed in a SAT formula; as often in (non-)termination the real search for a proof is done by a SAT solver
- We exploited simulations to improve this approach
- We extended the approach to cycle rewriting, again exploiting simulations

Conclusions

- We summarized the method for non-termination of string rewriting from RTA2015
- Instead of looking for a particular infinite computation, we look for a (regular) language (= set of strings) with some properties from which non-termination easily follows
- Properties are expressed in a SAT formula; as often in (non-)termination the real search for a proof is done by a SAT solver
- We exploited simulations to improve this approach
- We extended the approach to cycle rewriting, again exploiting simulations
- Thank you