FORT 2.0

Franziska Rapp, supervised by Aart Middeldorp

University of Innsbruck
Computational Logic Group

January 10, 2018

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ?

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? (ACP, CSI, CoLL-Saigawa)

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)
- is $\mathcal{R}$ terminating ?

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)
- is $\mathcal{R}$ terminating ? (AProVE, MuTerm, NaTT, T_TT_2)

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)
- is $\mathcal{R}$ terminating ? no (AProVE, MuTerm, NaTT, T_TT_2)

Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)
- is $\mathcal{R}$ terminating ? no (AProVE, MuTerm, NaTT, T_TT_2)
- is $\mathcal{R}$ strongly confluent ?
- is $\mathcal{R}$ (weakly) normalizing ?

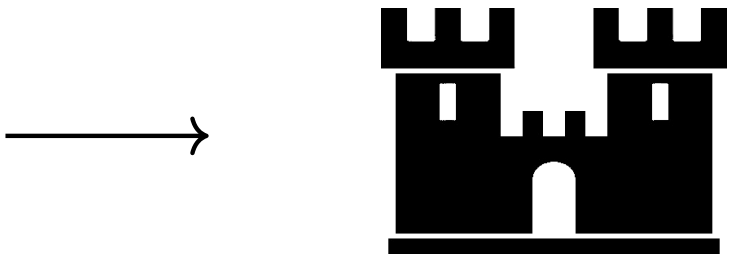
Example (COPS 111)

TRS $\mathcal{R}$

$$\begin{array}{lll} a \to b & b \to d & d \to e \\ a \to c & c \to a & g(x) \to h(a) \\ a \to e & d \to a & h(x) \to e \end{array}$$

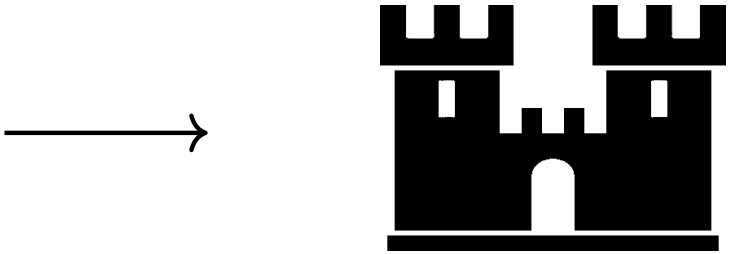
- is $\mathcal{R}$ confluent ? yes (ACP, CSI, CoLL-Saigawa)
- is $\mathcal{R}$ terminating ? no (AProVE, MuTerm, NaTT, T_TT_2)
- is $\mathcal{R}$ strongly confluent ? no
- is $\mathcal{R}$ (weakly) normalizing ? yes

FORT

property  TRS

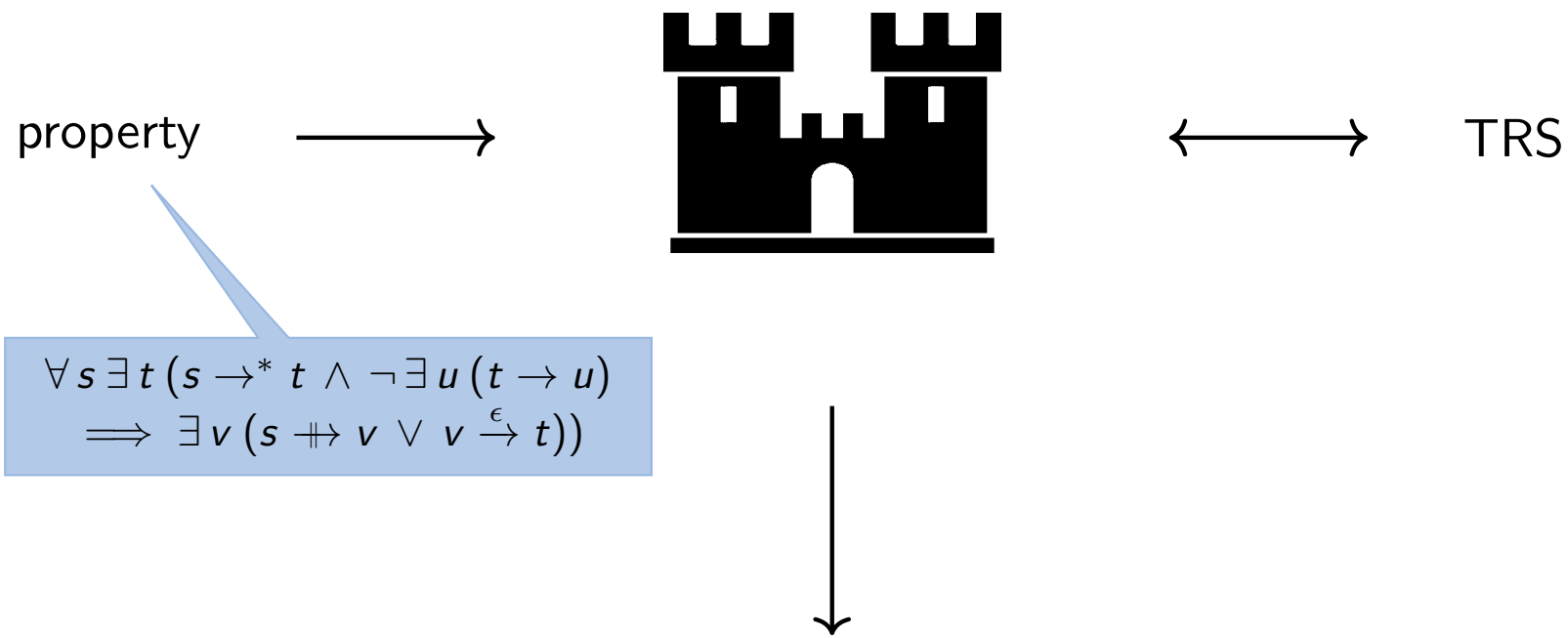
yes | no

FORT

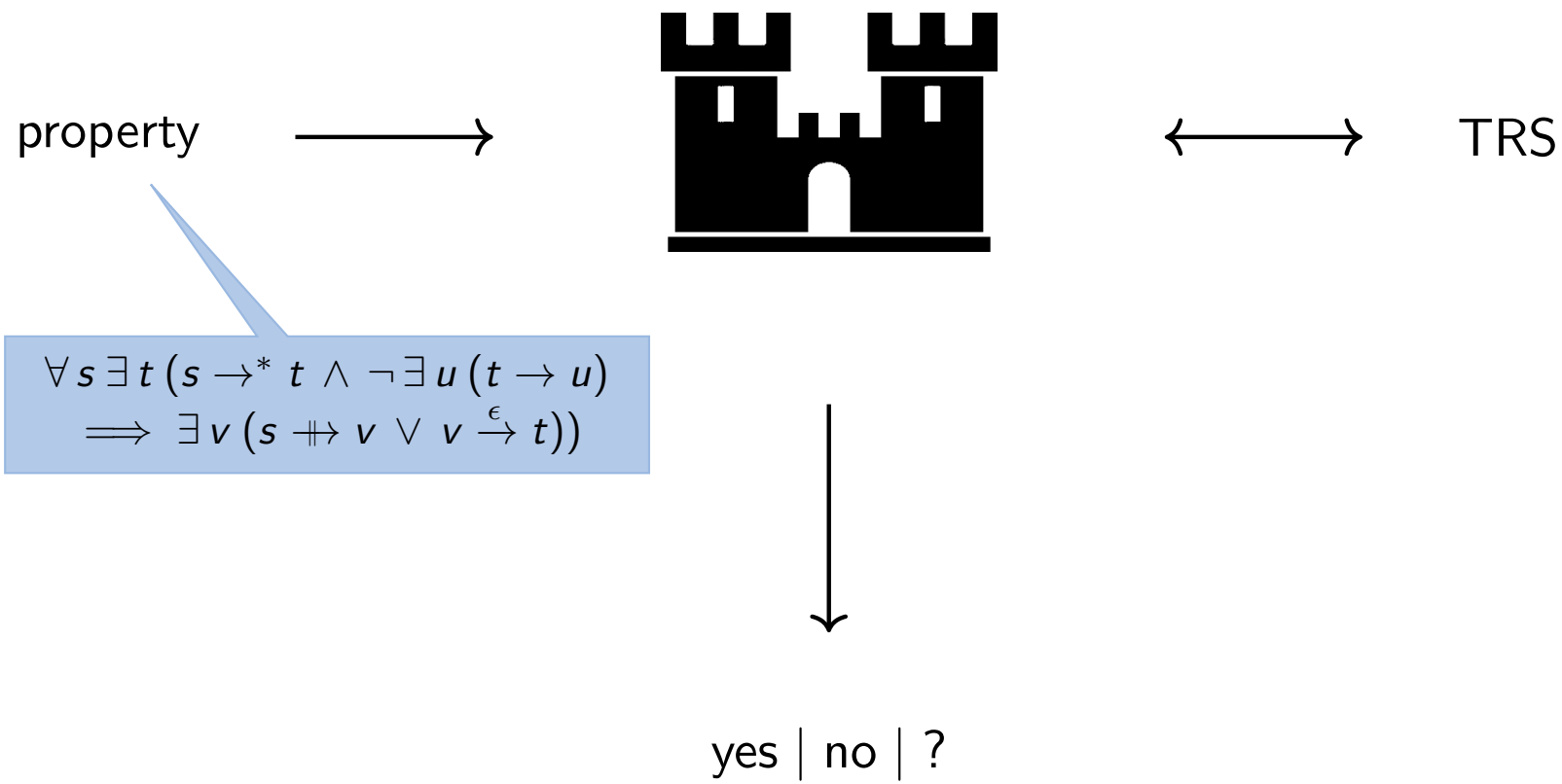
property  TRS

no | ?

FORT


property
TRS
$\forall s \exists t (s \to^* t \wedge \neg \exists u (t \to u) \implies \exists v (s \twoheadrightarrow v \vee v \xrightarrow{\epsilon} t))$
yes | no | ?

FORT



FORT is based on tree automata techniques (Dauchet and Tison, LICS 1990)

Outline

- First-Order Theory of Rewriting
- Automation
- Witness Generation
- Combinations of TRSs
- Extra Variables
- Summary

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^{+} \qquad \twoheadrightarrow \qquad \leftrightarrow^{*} \qquad = \qquad \cdots$$

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \not\to \qquad \leftrightarrow^* \qquad = \qquad \cdots$$

- models of $\mathcal{L}$ are finite TRSs $(\mathcal{F}, \mathcal{R})$ such that

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \Rightarrow \qquad \leftrightarrow^* \qquad = \qquad \cdots$$

- models of $\mathcal{L}$ are finite TRSs $(\mathcal{F}, \mathcal{R})$ such that
  - $\mathcal{R}$ is left-linear and right-ground

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \not\to \qquad \leftrightarrow^* \qquad = \qquad \cdots$$

- models of $\mathcal{L}$ are finite TRSs $(\mathcal{F}, \mathcal{R})$ such that
  - $\mathcal{R}$ is left-linear and right-ground
  - $\mathcal{T}(\mathcal{F}) \neq \varnothing$

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \not\to \qquad \leftrightarrow^* \qquad = \qquad \cdots$$

- models of $\mathcal{L}$ are finite TRSs $(\mathcal{F}, \mathcal{R})$ such that
  - $\mathcal{R}$ is left-linear and right-ground
  - $\mathcal{T}(\mathcal{F}) \neq \varnothing$
- set of ground terms serves as domain for variables in formulas over $\mathcal{L}$

First-Order Theory of Rewriting

- first-order logic over a language $\mathcal{L}$ without function symbols
- $\mathcal{L}$ contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \not\to \qquad \leftrightarrow^* \qquad = \qquad \cdots$$

- models of $\mathcal{L}$ are finite TRSs $(\mathcal{F}, \mathcal{R})$ such that
  - $\mathcal{R}$ is left-linear and right-ground
  - $\mathcal{T}(\mathcal{F}) \neq \varnothing$
- set of ground terms serves as domain for variables in formulas over $\mathcal{L}$

Derived Predicates

$$s \to^* t \iff s \to^+ t \vee s = t \qquad s \leftrightarrow t \iff s \to t \vee t \to s$$

$$s \to^! t \iff s \to^* t \wedge \neg \exists u\,(t \to u) \qquad s \downarrow t \iff \exists u\,(s \to^* u \wedge t \to^* u)$$

$$\mathrm{CR} \iff \forall s \forall t \forall u\,(s \to^* t \wedge s \to u \implies t \downarrow u)$$

SN can be expressed via additional finiteness predicate

Outline

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\not\Rightarrow$
  - RR_n automata for all n-ary relations

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\nrightarrow$
  - RR_n automata for all n-ary relations
- implications and universal quantifiers are eliminated

$$\varphi \Rightarrow \psi \;\equiv\; \neg\varphi \vee \psi \qquad\qquad \forall x\, \varphi \;\equiv\; \neg\exists x \neg\varphi$$

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\nrightarrow$
  - RR_n automata for all n-ary relations
- implications and universal quantifiers are eliminated

$$\varphi \Rightarrow \psi \;\equiv\; \neg\varphi \vee \psi \qquad\qquad \forall x\, \varphi \;\equiv\; \neg \exists x \neg \varphi$$

- negations are pushed inside and double negations are eliminated

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\Rightarrow\!\!\!\!\!\!+$
  - RR_n automata for all n-ary relations
- implications and universal quantifiers are eliminated

$$\varphi \Rightarrow \psi \;\equiv\; \neg\varphi \vee \psi \qquad\qquad \forall x\, \varphi \;\equiv\; \neg\exists x \neg\varphi$$

- negations are pushed inside and double negations are eliminated
- remaining propositional connectives are implemented by corresponding closure operations on RR_n automata

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\nrightarrow$
  - RR_n automata for all n-ary relations
- implications and universal quantifiers are eliminated

$$\varphi \Rightarrow \psi \;\equiv\; \neg\varphi \vee \psi \qquad\qquad \forall x\, \varphi \;\equiv\; \neg \exists x \neg \varphi$$

- negations are pushed inside and double negations are eliminated
- remaining propositional connectives are implemented by corresponding closure operations on RR_n automata
- existential quantifiers are implemented using projection

Translation

- binary predicates are RR_2 relations and implemented via tree automata
  - ground tree transducers (GTTs) for $\nrightarrow$
  - RR_n automata for all n-ary relations
- implications and universal quantifiers are eliminated

$$\varphi \Rightarrow \psi \;\equiv\; \neg\varphi \vee \psi \qquad\qquad \forall x\, \varphi \;\equiv\; \neg \exists x \neg \varphi$$

- negations are pushed inside and double negations are eliminated
- remaining propositional connectives are implemented by corresponding closure operations on RR_n automata
- existential quantifiers are implemented using projection

Remark

formulas are not transformed into prenex normal form, since this increases the dimension of involved relations

Outline

Witness Generation for Formulas

- free variables (existentially quantified)

Example (1)

`-D -w COPS/39.trs "s ->! t & s ->! u & ~t = u"`

$f(b) \to a$ $\quad f(c) \to f(b)$ $\quad b \to e$ $\quad f(e) \to a$

$f(b) \to f(c)$ $\quad f(c) \to d$ $\quad c \to e'$ $\quad f(e') \to d$

Witness Generation for Formulas

- free variables (existentially quantified)

Example (1)

 `-D -w COPS/39.trs "s ->! t & s ->! u & ~t = u"`

$f(b) \to a$ $\quad f(c) \to f(b)$ $\quad b \to e$ $\quad f(e) \to a$

$f(b) \to f(c)$ $\quad f(c) \to d$ $\quad c \to e'$ $\quad f(e') \to d$

`Witnesses:` s : $f(b)$ $\quad t$: a $\quad u$: d

SAT

Witness Generation for Formulas

- free variables (existentially quantified)

Example (1)

```
-D -w COPS/39.trs "exists s, t, u (s ->! t & s ->! u & ~t = u)"
```

$f(b) \to a$ $\quad$ $f(c) \to f(b)$ $\quad$ $b \to e$ $\quad$ $f(e) \to a$

$f(b) \to f(c)$ $\quad$ $f(c) \to d$ $\quad$ $c \to e'$ $\quad$ $f(e') \to d$

Witness Generation for Formulas

- free variables (existentially quantified)
- universally quantified variables (in negative case)

Example (2)

`-D -w COPS/39.trs CR`

$f(b) \to a$ $\quad f(c) \to f(b)$ $\quad b \to e$ $\quad f(e) \to a$

$f(b) \to f(c)$ $\quad f(c) \to d$ $\quad c \to e'$ $\quad f(e') \to d$

Witness Generation for Formulas

- free variables (existentially quantified)
- universally quantified variables (in negative case)

Example (2)

```
-D -w COPS/39.trs CR
```

$f(b) \to a$ $\quad$ $f(c) \to f(b)$ $\quad$ $b \to e$ $\quad$ $f(e) \to a$

$f(b) \to f(c)$ $\quad$ $f(c) \to d$ $\quad$ $c \to e'$ $\quad$ $f(e') \to d$

```
Witnesses for formula:  exists s, t, u (s -> t & s ->* u & ~t join u)
```

s : $f(c)$ $\quad$ t : $f(e')$ $\quad$ u : $f(e)$

NO

Function `find_terms`

Input:

- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;

$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

avoids ending up in loops

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

`while` $\mathcal{C} \neq \varnothing$ `do`

`od; fail`

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

`while` $\mathcal{C} \neq \varnothing$ `do`

`select` $fs(\alpha_1, \ldots, \alpha_m) \to \alpha$ `from` $\mathcal{C}$ `with minimal` m;

`od; fail`

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

add α to Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

while $\mathcal{C} \neq \varnothing$ do

select $fs(\alpha_1, \ldots, \alpha_m) \to \alpha$ from $\mathcal{C}$ with minimal m;

for all $1 \leqslant i \leqslant m$ try $\overrightarrow{t_i} = \mathtt{find\_terms}(\mathcal{A}, \alpha_i, Q_v)$;

od; fail

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

`while` $\mathcal{C} \neq \varnothing$ `do`

`select` $fs(\alpha_1, \ldots, \alpha_m) \to \alpha$ `from` $\mathcal{C}$ `with minimal` m;

`for all` $1 \leqslant i \leqslant m$ `try` $\overrightarrow{t_i} =$ `find_terms`$(\mathcal{A}, \alpha_i, Q_v)$;

inner calls might end up in loops

`od; fail`

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;
`while` $\mathcal{C} \neq \varnothing$ `do`

$fs = f_1 \cdots f_n$

`select` $fs(\alpha_1, \ldots, \alpha_m) \to \alpha$ `from` $\mathcal{C}$ `with minimal` m;
`for all` $1 \leqslant i \leqslant m$ `try` $\overrightarrow{t_i} =$ `find_terms`$(\mathcal{A}, \alpha_i, Q_v)$;
`return` $f_1(\pi_1 \overrightarrow{t_1}, \ldots, \pi_1 \overrightarrow{t_{a(f_1)}}), \ldots, f_n(\pi_n \overrightarrow{t_1}, \ldots, \pi_n \overrightarrow{t_{a(f_n)}})$
`od; fail`

Function `find_terms`

Input:
- $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ RR_n automaton
- $\alpha \in Q$
- $Q_v \subseteq Q$

Output: n-tuple of terms accepted in α not using states in Q_v

---

`add` α `to` Q_v;
$\mathcal{C} = \{fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v$, and if $Q_v = \{\alpha\}$ then $\bot \notin fs\}$;

`while` $\mathcal{C} \neq \varnothing$ `do`

`select` $fs(\alpha_1, \ldots, \alpha_m) \to \alpha$ `from` $\mathcal{C}$ `with minimal` m;

`for all` $1 \leqslant i \leqslant m$ `try` $\overrightarrow{t_i} =$ `find_terms`$(\mathcal{A}, \alpha_i, Q_v)$;

`return` $f_1(\pi_1 \overrightarrow{t_1}, \ldots, \pi_1 \overrightarrow{t_{a(f_1)}}), \ldots, f_n(\pi_n \overrightarrow{t_1}, \ldots, \pi_n \overrightarrow{t_{a(f_n)}})$

`od; fail`

Witnesses for RR_n Automaton

Use `find_terms`$(\mathcal{A}, \alpha, \varnothing)$ with some $\alpha \in Q_f$ to generate witnesses for $\mathcal{A}$.

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

`find_terms`$(\mathcal{A}, ✓, \varnothing)$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state $\checkmark$ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to \checkmark$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(\checkmark, \alpha) \to \checkmark$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, \checkmark, \varnothing)$

$Q_v = \{\checkmark\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark, \mathsf{ff}(\alpha, \gamma) \to \checkmark\}$

$\texttt{find\_terms}(\mathcal{A}, \alpha, \{\checkmark\})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\texttt{find\_terms}(\mathcal{A}, \alpha, \{✓\})$

$Q_v = \{✓, \alpha\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state $\checkmark$ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to \checkmark \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(\checkmark, \alpha) \to \checkmark \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, \checkmark, \varnothing)$

$Q_v = \{\checkmark\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark, \mathsf{ff}(\alpha, \gamma) \to \checkmark\}$

$\texttt{find\_terms}(\mathcal{A}, \alpha, \{\checkmark\})$

$Q_v = \{\checkmark, \alpha\}$

$\mathcal{C} = \{\mathsf{aa} \to \alpha\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state $\checkmark$ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to \checkmark \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(\checkmark, \alpha) \to \checkmark \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, \checkmark, \varnothing)$

$Q_v = \{\checkmark\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark, \mathsf{ff}(\alpha, \gamma) \to \checkmark\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\texttt{find\_terms}(\mathcal{A}, \alpha, \{\checkmark\})$

$Q_v = \{\checkmark, \alpha\}$

$\mathcal{C} = \{\mathsf{aa} \to \alpha\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to ✓ \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to ✓ \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(✓, \alpha) \to ✓ \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to ✓ \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to ✓ \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(✓, \alpha) \to ✓ \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

$\texttt{find\_terms}(\mathcal{A}, \beta, \{✓, \gamma\}) = (\mathsf{b}, \mathsf{b})$

$\texttt{find\_terms}(\mathcal{A}, \alpha', \{✓, \gamma\}) = (\bot, \mathsf{a})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to ✓ \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to ✓ \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(✓, \alpha) \to ✓ \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\overrightarrow{t_2} = (\mathsf{g}(\mathsf{b}),\ \ \ \ \ \ \ \ )$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

$\texttt{find\_terms}(\mathcal{A}, \beta, \{✓, \gamma\}) = (\mathsf{b}, \mathsf{b})$

$\texttt{find\_terms}(\mathcal{A}, \alpha', \{✓, \gamma\}) = (\bot, \mathsf{a})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to ✓ \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to ✓ \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(✓, \alpha) \to ✓ \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing)$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

$\texttt{find\_terms}(\mathcal{A}, \beta, \{✓, \gamma\}) = (\mathsf{b}, \mathsf{b})$

$\texttt{find\_terms}(\mathcal{A}, \alpha', \{✓, \gamma\}) = (\bot, \mathsf{a})$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\overrightarrow{t_2} = (\mathsf{g}(\mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a}))$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\begin{array}{llll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to ✓ \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{ff}(\alpha, \gamma) \to ✓ \\ & & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(✓, \alpha) \to ✓ \end{array}$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing) = (\mathsf{f}(\mathsf{a}, \mathsf{g}(\mathsf{b})),\ \ )$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\overrightarrow{t_2} = (\mathsf{g}(\mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a}))$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

$\texttt{find\_terms}(\mathcal{A}, \beta, \{✓, \gamma\}) = (\mathsf{b}, \mathsf{b})$

$\texttt{find\_terms}(\mathcal{A}, \alpha', \{✓, \gamma\}) = (\bot, \mathsf{a})$

Example (Witness Generation)

Consider the RR_2 automaton $\mathcal{A}$ with final state ✓ and Δ:

$$\mathsf{aa} \to \alpha \qquad \bot\mathsf{a} \to \alpha' \qquad \mathsf{fg}(\alpha, \beta') \to \gamma \qquad \mathsf{fh}(\alpha, \beta, \alpha') \to ✓$$
$$\mathsf{bb} \to \beta \qquad \mathsf{b}\bot \to \beta' \qquad \mathsf{gf}(\beta, \alpha') \to \gamma \qquad \mathsf{ff}(\alpha, \gamma) \to ✓$$
$$\mathsf{gg}(\gamma) \to \gamma \qquad \mathsf{ff}(✓, \alpha) \to ✓$$

over the signature $\mathcal{F} = \{a : 0,\ b : 0,\ g : 1,\ f : 2,\ h : 3\}$.

$\texttt{find\_terms}(\mathcal{A}, ✓, \varnothing) = (\mathsf{f}(\mathsf{a}, \mathsf{g}(\mathsf{b})), \mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{b}, \mathsf{a})))$

$Q_v = \{✓\}$

$\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to ✓, \mathsf{ff}(\alpha, \gamma) \to ✓\}$

$\overrightarrow{t_1} = (\mathsf{a}, \mathsf{a})$

$\overrightarrow{t_2} = (\mathsf{g}(\mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a}))$

$\texttt{find\_terms}(\mathcal{A}, \gamma, \{✓\})$

$Q_v = \{✓, \gamma\}$

$\mathcal{C} = \{\mathsf{fg}(\alpha, \beta') \to \gamma, \mathsf{gf}(\beta, \alpha') \to \gamma\}$

$\texttt{find\_terms}(\mathcal{A}, \beta, \{✓, \gamma\}) = (\mathsf{b}, \mathsf{b})$

$\texttt{find\_terms}(\mathcal{A}, \alpha', \{✓, \gamma\}) = (\bot, \mathsf{a})$

Outline

Properties Involving Multiple TRSs

- express properties on combinations of TRSs

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`
- indices correspond to TRSs (union of indicated TRSs)

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`
- indices correspond to TRSs (union of indicated TRSs)
- no index $\Longrightarrow$ union of all TRSs

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`
- indices correspond to TRSs (union of indicated TRSs)
- no index $\Longrightarrow$ union of all TRSs
- no nesting of indices:

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`
- indices correspond to TRSs (union of indicated TRSs)
- no index $\Longrightarrow$ union of all TRSs
- no nesting of indices: `[0]([1]SN & CR)` $\times$

Properties Involving Multiple TRSs

- express properties on combinations of TRSs
- `[0,1]SN & [1,2](SN & CR) & [0]CR & ~CR`
- indices correspond to TRSs (union of indicated TRSs)
- no index $\Longrightarrow$ union of all TRSs
- no nesting of indices: `[0]([1]SN & CR)` $\times$

Remark

Search space for synthesis explodes.

Related Work

Carpa+ (Zantema, 2013) is tool for synthesizing TRSs that satisfy properties which can be encoded as SMT problems

Related Work

Carpa+ (Zantema, 2013) is tool for synthesizing TRSs that satisfy properties which can be encoded as SMT problems

- TRSs that can be synthesized form small extension of class of ARSs: one unary function symbol f is permitted and rules must have form

$$a \to b \qquad a \to f(b) \qquad f(a) \to b$$

Related Work

Carpa+ (Zantema, 2013) is tool for synthesizing TRSs that satisfy properties which can be encoded as SMT problems

- TRSs that can be synthesized form small extension of class of ARSs: one unary function symbol f is permitted and rules must have form

$$a \to b \qquad a \to f(b) \qquad f(a) \to b$$

- properties are restricted to those that can be encoded into conjunctive fragment of SMT-LRA (linear real arithmetic)

Related Work

Carpa+ (Zantema, 2013) is tool for synthesizing TRSs that satisfy properties which can be encoded as SMT problems

- TRSs that can be synthesized form small extension of class of ARSs: one unary function symbol f is permitted and rules must have form

$$a \to b \qquad a \to f(b) \qquad f(a) \to b$$

- properties are restricted to those that can be encoded into conjunctive fragment of SMT-LRA (linear real arithmetic)
- predecessor tool Carpa synthesized combinations of ARSs with SAT solver

Examples (1)

-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

Examples (1)

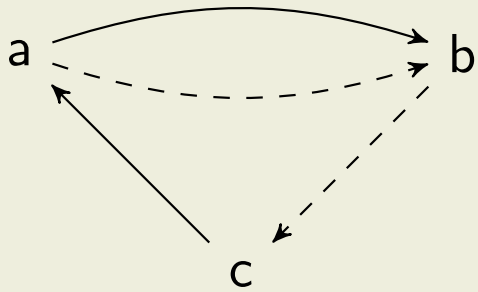
-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

input for Carpa:
3 (nr of elements)
2 (nr of basic relations)
x1=tc(1)
x2=tc(2)
irr(x1)
irr(x2)
x3=union(1,2)
x3=tc(x3)
nirr(x3)
x4=comp(1,2)
x5=union(x1,x2)
subs(x4,x5)

Examples (1)

-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

  generates TRSs

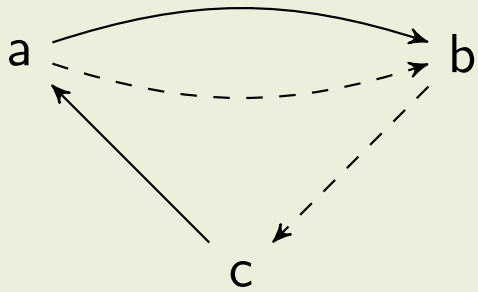


TRS 0
TRS 1

Examples (1)

-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

  generates TRSs



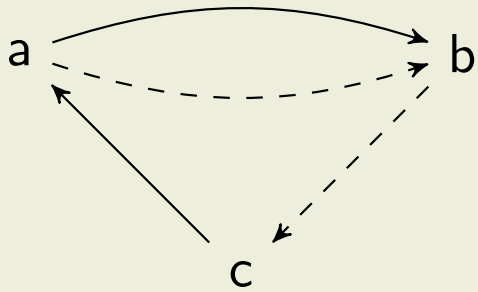
TRS 0
TRS 1

-  `-S -g "0 0 0" "[0]CR & [1]CR & LCom(0,1) & ~Com(0,1)"`

Examples (1)

-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

  generates TRSs



TRS 0

TRS 1

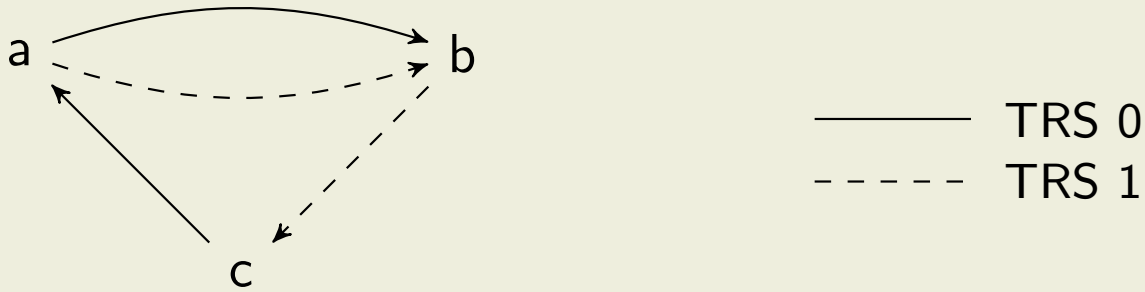
-  `-S -g "0 0 0" "[0]CR & [1]CR & LCom(0,1) & ~Com(0,1)"`

```
LCom = forall s,t,u([0]s -> t & [1]s -> u => exists v([1]t ->* v & [0]u ->* v))
Com = forall s,t,u([0]s ->* t & [1]s ->* u => exists v([1]t ->* v & [0]u ->* v))
```

Examples (1)

-  `-S -g "0 0 0" "[0]SN & [1]SN & ~SN & forall s, t, u ([0]s -> t & [1]t -> u => [0]s ->+ u | [1]s ->+ u)"`

  generates TRSs

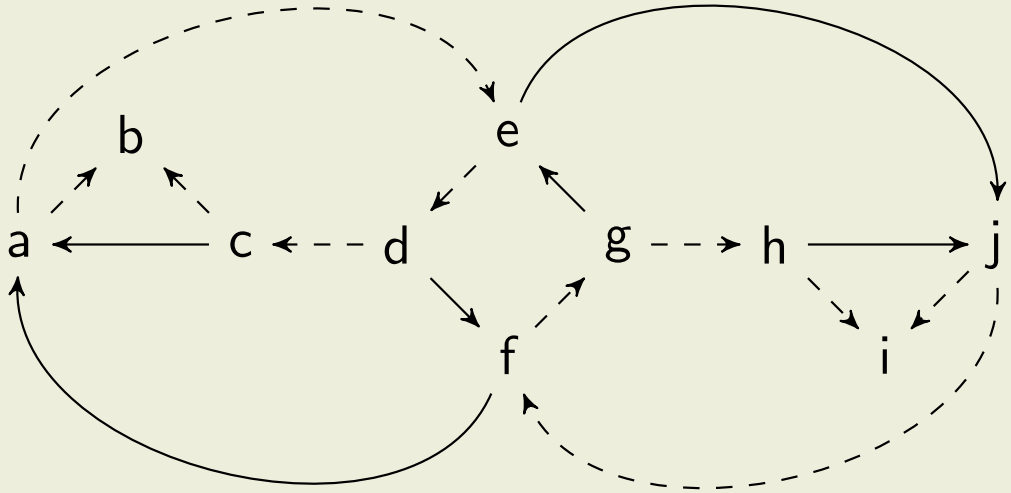


-  `-S -g "0 0 0" "[0]CR & [1]CR & LCom(0,1) & ~Com(0,1)"`

  generates TRSs

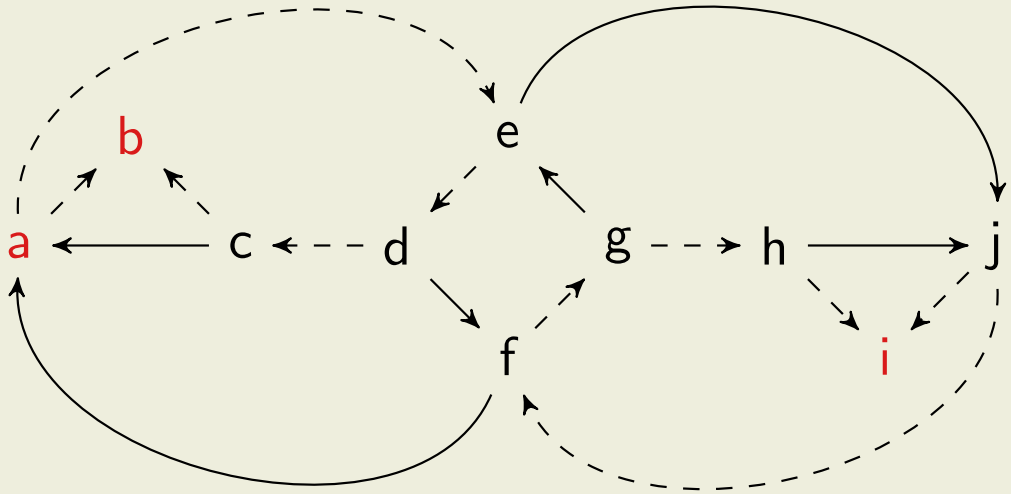
Examples (2)

```
-D -w Zantema/example8.trs "[0](CR & SN) & [1](CR & SN) & ~CR &
forall s, t, u ([0]s -> t & [1]s -> u =>
                exists v (([1]t -> v | [0]t -> v) & [0]u ->* v))"
```



Examples (2)

```
-D -w Zantema/example8.trs "[0](CR & SN) & [1](CR & SN) & ~CR &
forall s, t, u ([0]s -> t & [1]s -> u =>
                exists v (([1]t -> v | [0]t -> v) & [0]u ->* v))"
```




```
Witnesses for formula:  exists s, t, u (s -> t & s ->* u & ~t join u)
```

s : a t : b u : i

YES

Commutation

- Formula: `forall s,t,u ([0]s ->* t & [1]s ->* u =>`
  `exists v ([1]t ->* v & [0]u ->* v))`

Commutation

- Formula: `forall s,t,u ([0]s ->* t & [1]s ->* u =>`
  `exists v ([1]t ->* v & [0]u ->* v))`
- CoCo participant CoLL
  proves confluence via (strong) commutation

Commutation

- Formula: `forall s,t,u ([0]s ->* t & [1]s ->* u =>`
  `exists v ([1]t ->* v & [0]u ->* v))`
- CoCo participant CoLL-Saigawa
  proves confluence via (strong) commutation

Commutation

- Formula: `forall s,t,u ([0]s ->* t & [1]s ->* u =>`
  `exists v ([1]t ->* v & [0]u ->* v))`
- CoCo participant CoLL-Saigawa
  proves confluence via (strong) commutation
- Combining 65 TRSs from COPS and TPDB
  (2145 after removing symmetric ones)

Commutation

- Formula: `forall s,t,u ([0]s ->* t & [1]s ->* u =>`
  `exists v ([1]t ->* v & [0]u ->* v))`
- CoCo participant CoLL-Saigawa
  proves confluence via (strong) commutation
- Combining 65 TRSs from COPS and TPDB
  (2145 after removing symmetric ones)

	FORT	CoLL
YES	613	451
NO	1379	81
MAYBE	153	1613

uniformly renamed TRSs

Outline

Extension

- variables on right hand sides, but rule has to be linear

Extension

- variables on right hand sides, but rule has to be linear

no variable should occur
more than once within a rule

Extension

- variables on right hand sides, but rule has to be linear
- $\mathsf{a} \to x \quad \mathsf{f}(x, y) \to \mathsf{g}(z)$

Extension

- variables on right hand sides, but rule has to be linear
- $\mathsf{a} \to x \quad \mathsf{f}(x, y) \to \mathsf{g}(z)$ ✓

Extension

- variables on right hand sides, but rule has to be linear
- $\mathsf{a} \to x \quad \mathsf{f}(x, y) \to \mathsf{g}(z)$ ✓
- $\mathsf{f}(x, y) \to \mathsf{g}(x)$

Extension

- variables on right hand sides, but rule has to be linear
- $\mathsf{a} \to x \quad \mathsf{f}(x, y) \to \mathsf{g}(z)$ ✓
- $\mathsf{f}(x, y) \to \mathsf{g}(x)$ ×

Extension

- variables on right hand sides, but rule has to be linear
- $a \to x \quad f(x, y) \to g(z)$ ✓
- $f(x, y) \to g(x)$ ×

Possible Applications

- approximation mappings for call by need strategies

Extension

- variables on right hand sides, but rule has to be linear
- $a \to x \quad f(x, y) \to g(z)$ ✓
- $f(x, y) \to g(x)$ ×

Possible Applications

- approximation mappings for call by need strategies
- dependency graph approximations for termination analysis

Extension

- variables on right hand sides, but rule has to be linear
- $\mathsf{a} \to x \quad \mathsf{f}(x, y) \to \mathsf{g}(z)$ ✓
- $\mathsf{f}(x, y) \to \mathsf{g}(x)$ ×

Possible Applications

- approximation mappings for call by need strategies
- dependency graph approximations for termination analysis
- infeasability testing of conditional critical pairs for confluence analysis

Extension

- variables on right hand sides, but rule has to be linear
- $a \to x \quad f(x, y) \to g(z)$ ✓
- $f(x, y) \to g(x)$ ×

Possible Applications

- approximation mappings for call by need strategies
- dependency graph approximations for termination analysis
- infeasability testing of conditional critical pairs for confluence analysis
- inside FORT: compute $\leftrightarrow^*$ via $(\nparallel\!\!\!\rightarrow_{\mathcal{R} \cup \mathcal{R}^{-1}})^+$ instead of $(\nparallel\!\!\!\rightarrow_{\mathcal{R}} \cup \nparallel\!\!\!\rightarrow_{\mathcal{R}}^{-1})^+$

Extension

- variables on right hand sides, but rule has to be linear
- $a \to x \quad f(x, y) \to g(z)$ ✓
- $f(x, y) \to g(x)$ ×

Possible Applications

- approximation mappings for call by need strategies
- dependency graph approximations for termination analysis
- infeasability testing of conditional critical pairs for confluence analysis
- inside FORT: compute $\leftrightarrow^*$ via $(\twoheadrightarrow_{\mathcal{R} \cup \mathcal{R}^{-1}})^+$ instead of $(\twoheadrightarrow_{\mathcal{R}} \cup \twoheadrightarrow_{\mathcal{R}}^{-1})^+$

Example (COPS 81)

$g(f(a)) \to f(g(f(a)))$ $\quad$ $g(f(a)) \to f(f(a))$ $\quad$ $f(f(a)) \to f(a)$

produced an RR_2 automaton of size 118 for $\leftrightarrow^*$ instead of size 427

Outline

Summary

- GUI

Summary

- GUI
- witness generation

Summary

- GUI
- witness generation
- combinations of TRSs
  - Carpa(+)
  - commutation (CoLL)

Summary

- GUI
- witness generation
- combinations of TRSs
  - Carpa(+)
  - commutation (CoLL)
- extra variables