Active Learning of Sequential Transducers with Side Information about the Domain

Raphaël Berthon[1,2] Adrien Boiret [1] Guillermo A. Perez [2]
Jean-François Raskin [1]

[1]Université libre de Bruxelles, Belgium [2]University of Antwerp, Belgium

August 19, 2021

Active Learning

Learning goal

Queries

Black box

Inputs

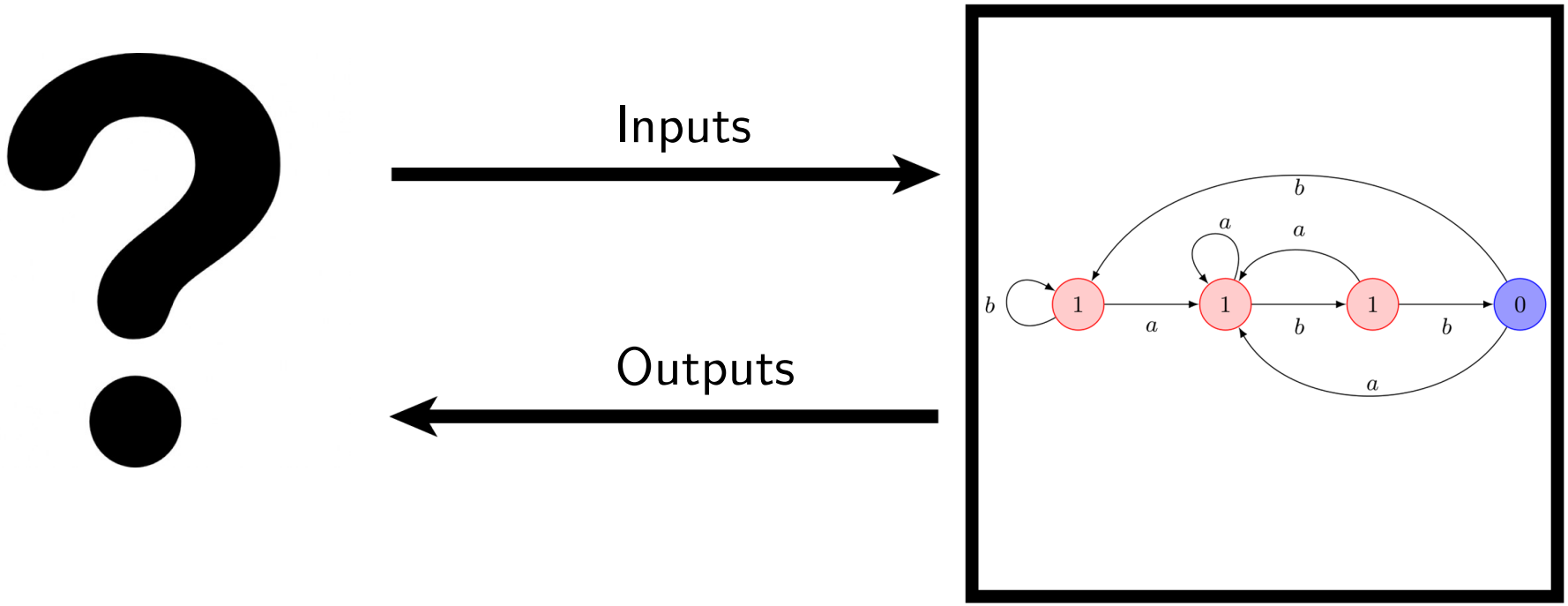
Outputs

Regular Language Learning

Learn the automaton

- Word membership
- Equivalence

DFA



Motivation

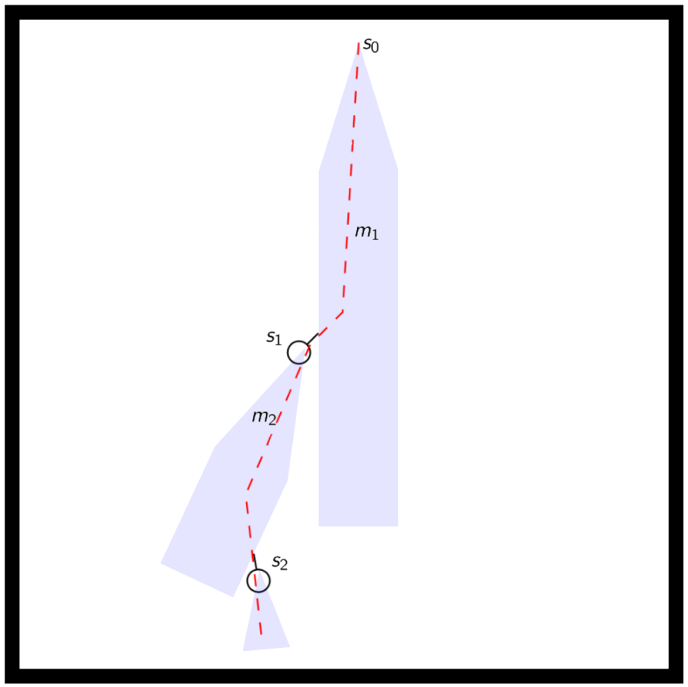
Learn an implemented strategy while knowing the game structure

Queries

Strategy

Inputs

Outputs



Graybox Subsequential Transducer Learning

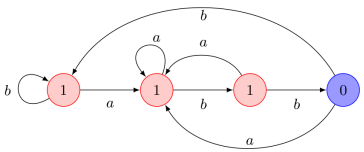
Learn a transducer using side information

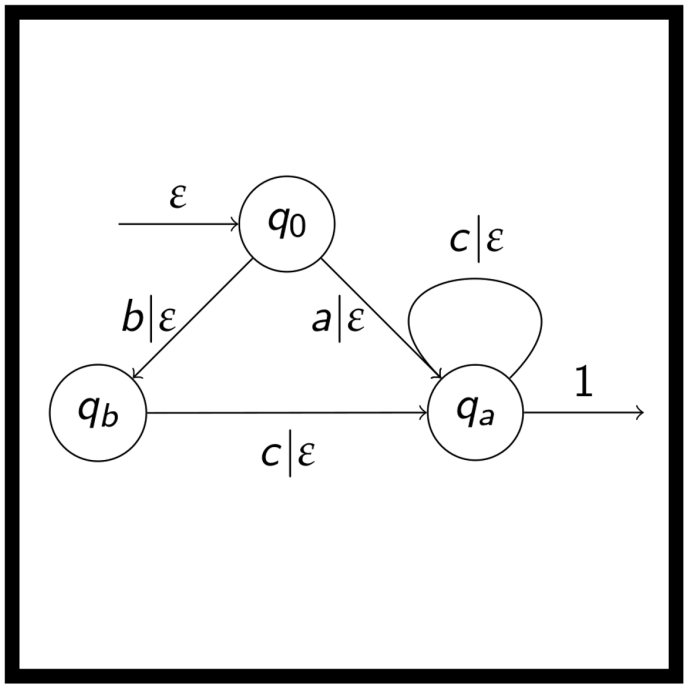
- Output of a word
- Equivalence

Transducer

Inputs

Outputs





Main result

Given a blackbox transduction τ and a domain upper-bound Up, our algorithm learns a transducer M of minimal size n such that $M_{|Up} = \tau$.

- Number of membership queries before a call to the SAT-solver: $POLY(n)$
- Number of calls to the SAT-solver before an equivalence query: $EXP(n)$
- Number of equivalence queries: n

Regular Language Learning

$L \subseteq \Sigma^*$ is an unknown regular language.

- Membership queries: does $u \in L$?
- Equivalence queries: is $\mathcal{L}(M) = L$?

Objective

Find an automaton M such that $\mathcal{L}(M) = L$.

Regular Language Learning

Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table
while *True* **do**
  if $u \cdot a \cdot v$ *is a witness of non-consistency* **then**
    add av to S
    else if ua *is a witness of non-closure* **then**
      add ua to P
      else
        $M :=$ Generate Automaton$(T(P, S))$
        if u *is a non-equiv. witness for* M **then**
          add all its suffixes to S
        else
          return M

Regular Language Learning

	ε	c
ε	0	0
a	1	1

The L^* algorithm uses membership queries to fill a table. Two lines represent the same state if they are equal.

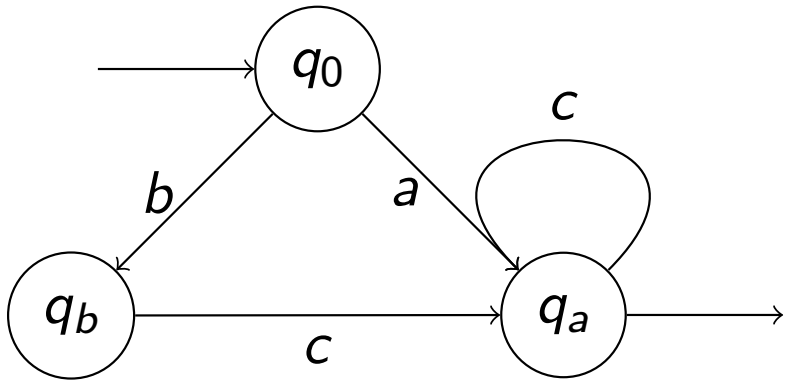
Regular Language Learning

	ε	c
ε	0	0
a	1	1
b	0	1

The L^* algorithm uses membership queries to fill a table. Two lines represent the same state if they are equal.

Regular Language Learning

	ε	c
ε	0	0
a	1	1
b	0	1
ac	1	1
bc	1	1



The L^* algorithm uses membership queries to fill a table. Two lines represent the same state if they are equal. Once there exists a unique automaton of minimal size compatible with this table, the algorithm makes an equivalence query.

Subsequential Transducer Learning

$\tau : \Sigma^* \to \Gamma^*$ is an unknown regular transduction.

- Membership queries: what is the ouput of u by τ?
- Equivalence queries: is $\mathcal{L}(M) = \tau$?

Objective

Find a transducer M such that $\mathcal{L}(M) = L$.

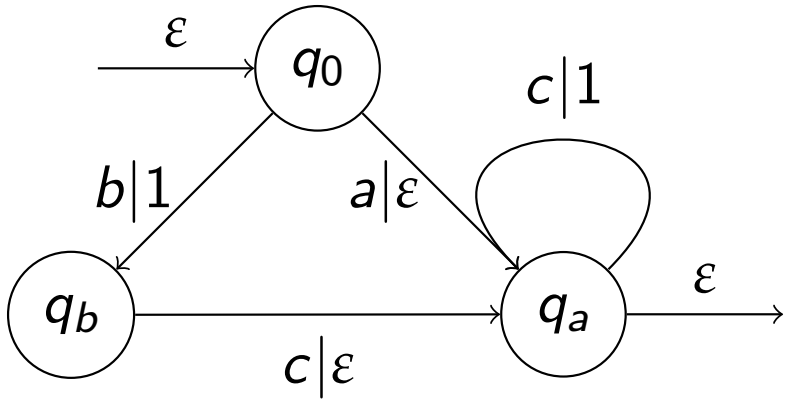
Subsequential Transducer Learning

	ε	c
ε	$\bot$	$\bot$
a	ε	1
b	$\bot$	1
ac	1	11
bc	1	11

When learning transducers, the *largest common prefix* of every line is factored out before building the candidate transducer.

Subsequential Transducer Learning

	ε	c
$\varepsilon \mid \varepsilon$	$\bot$	$\bot$
$a \mid \varepsilon$	ε	1
$b \mid \varepsilon$	$\bot$	1
$ac \mid 1$	ε	1
$bc \mid 1$	ε	1



When learning transducers, the *largest common prefix* of every line is factored out before building the candidate transducer.

Graybox Regular Language Learning

$L \subseteq \Sigma^*$ is an unknown regular language.
Up is a known regular language.

- Membership queries: does $u \in L$?
- Equivalence queries: is $\mathcal{L}(M) \cap Up = L$?

Objective

Find an automaton M such that $\mathcal{L}(M) \cap Up = L$.

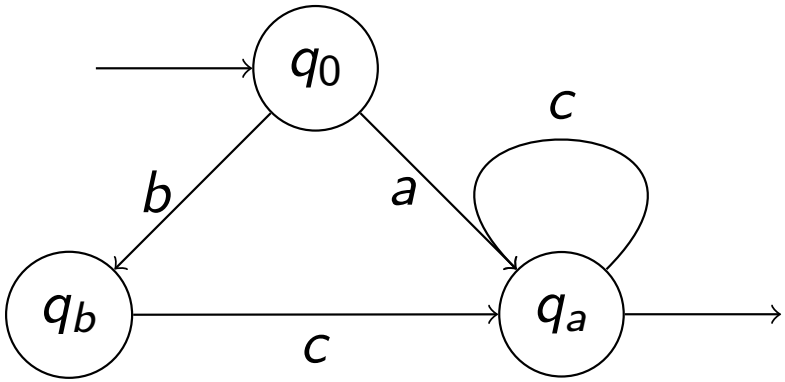
Graybox Regular Language Learning

	ε	c
ε	0	0
a	1	#
b	0	1
ac	#	#
bc	1	#

When learning a regular language in a *graybox setting*, a SAT-solver is used to find values for the # leading to a table associated to an automaton.

Graybox Regular Language Learning

ε	0
a	1
b	0
c	0
ac	$\# \leftarrow 1$
bc	1
acc	$\# \leftarrow 1$
bcc	$\# \leftarrow 1$



When learning a regular language in a *graybox setting*, a SAT-solver is used to find values for the # leading to a table associated to an automaton. It is sufficient to consider individual input words, and add the relations between them as equations in the solver.

Graybox Subsequential Transducer Learning

$\tau : \Sigma^* \to \Gamma^*$ is an unknown regular transduction.
Up is a known regular language.

- Membership queries: what is the ouput of u by τ?
- Equivalence queries: is $\mathcal{L}(M)_{|Up} = \tau$?

Objective

Find a transducer M such that $\mathcal{L}(M)_{|Up} = L$.

Graybox Subsequential Transducer Learning

	ε	c
ε	$\bot$	$\bot$
a	ε	$\#$
b	$\bot$	1

Difficulty

When learning a transducer in a graybox setting, we must guess values for the $\#$ while keeping a finite search space.

Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table

while *True* **do**

  if $u \cdot a \cdot v$ *is a witness of non-consistency* **then**

    add av to S

    else if ua *is a witness of non-closure* **then**

      add ua to P

      ???

      else

        $M :=$ Generate Transducer$(T(P, S))$

        if u *is a non-equiv. witness for* $Up \times M$ **then**

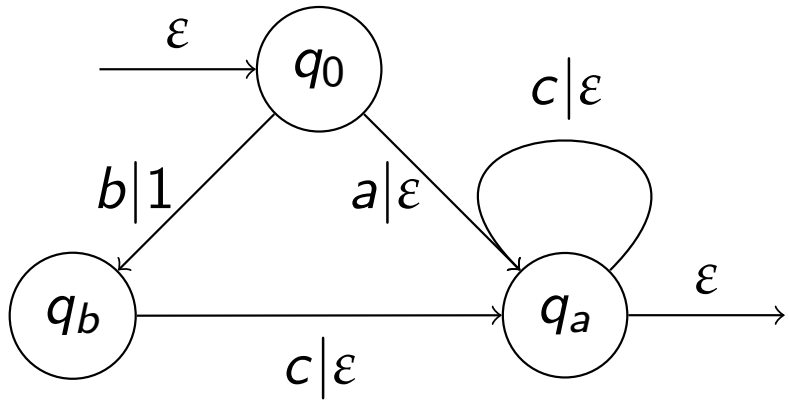
          add all its suffixes to S

        else

          return M

Merging Map

Input	Output	Partial output
$\varepsilon \in [\varepsilon]$	ε	ε
$a \in [a]$	ε	ε
$b \in [b]$	$\bot$	1
$c \in [c]$	$\bot$	$\bot$
$ac \in [a]$	$\#$	ε
$bc \in [a]$	1	1

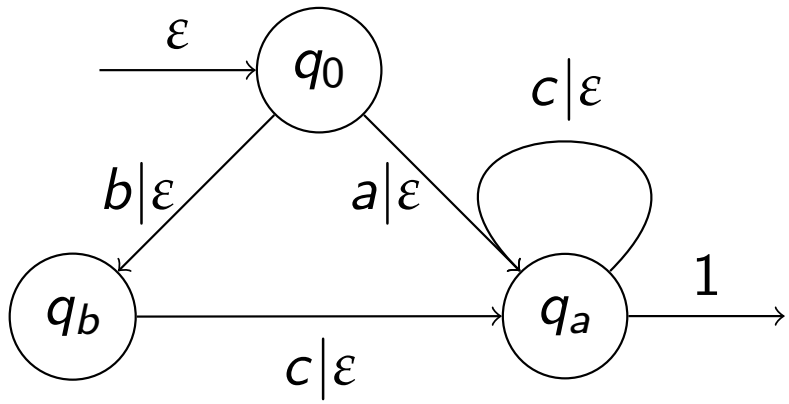


A *merging map* consists in guessing an equivalence class and a partial output for every input.

We define the transducer resulting from a given merging map.

Merging Map

Input	Output	Partial output
$\varepsilon \in [\varepsilon]$	ε	ε
$a \in [a]$	ε	ε
$b \in [b]$	$\bot$	ϵ
$c \in [c]$	$\bot$	$\bot$
$ac \in [a]$	$\#$	ϵ
$bc \in [a]$	1	ϵ



Different merging maps may lead to different resulting transducers, but there exists a finite number of merging maps associated to a given table.

Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table

while *True* **do**

  if $u \cdot a \cdot v$ *is a witness of non-consistency* **then**

    add av to S

    else if ua *is a witness of non-closure* **then**

      add ua to P

      else if *u distinguishes two merging maps* **then**

        add u and its suffixes to S

        ???

        else

          $M :=$ Generate Transducer$(T(P, S))$

          if u *is a non-equiv. witness for* $Up \times M$ **then**

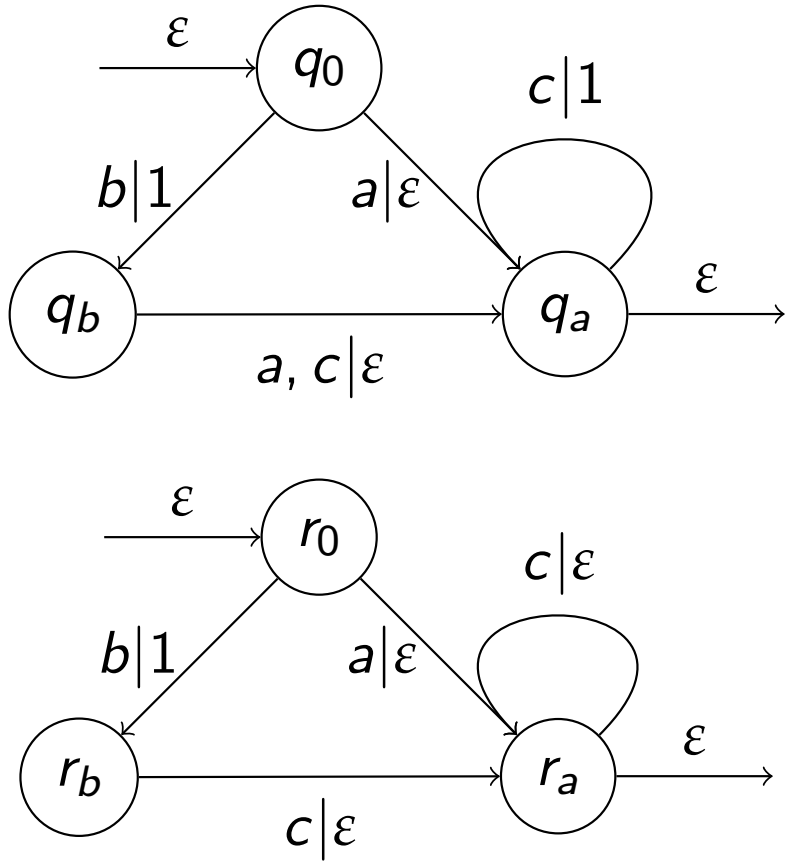
            add all its suffixes to S

          else

            return M

Merging Map

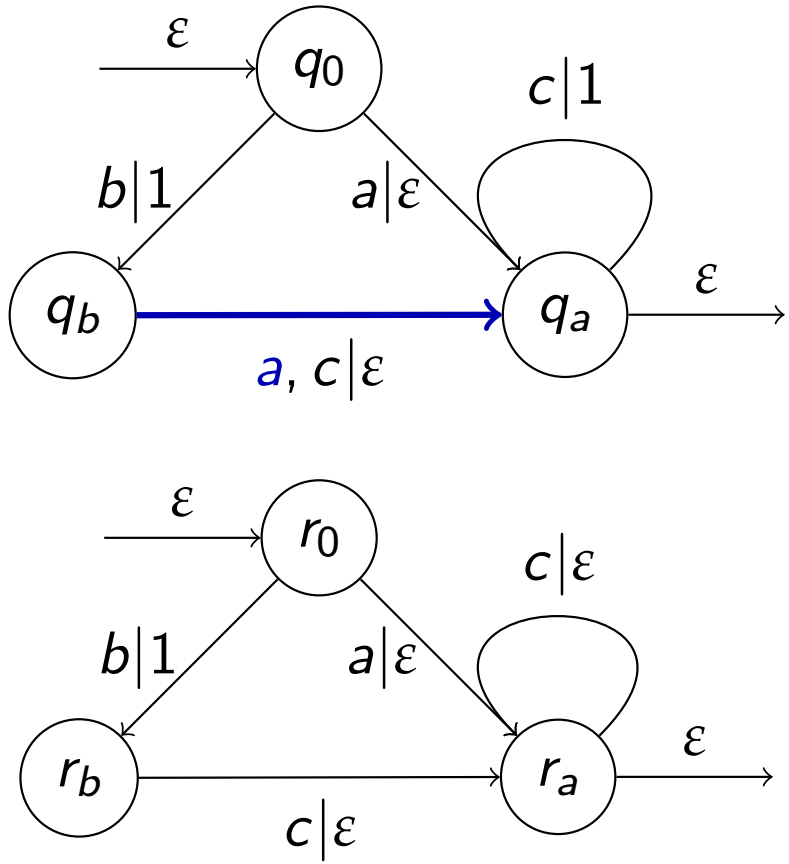
Input	Output	Partial output
$\varepsilon \in [\varepsilon]$	ε	ε
$a \in [a]$	ε	ε
$b \in [b]$	$\bot$	ε
$c \in [c]$	$\bot$	$\bot$
$ac \in [a]$	$\#$	ε
$bc \in [a]$	1	1



A merging maps may be induced by a transducer that is not equivalent to the resulting transducer of this merging map.

Merging Map

Input	Output	Partial output
$\varepsilon \in [\varepsilon]$	ε	ε
$a \in [a]$	ε	ε
$b \in [b]$	$\bot$	ε
$c \in [c]$	$\bot$	$\bot$
$ac \in [a]$	$\#$	ε
$bc \in [a]$	1	1



An open transition may be removed without changing the behaviour on the table.

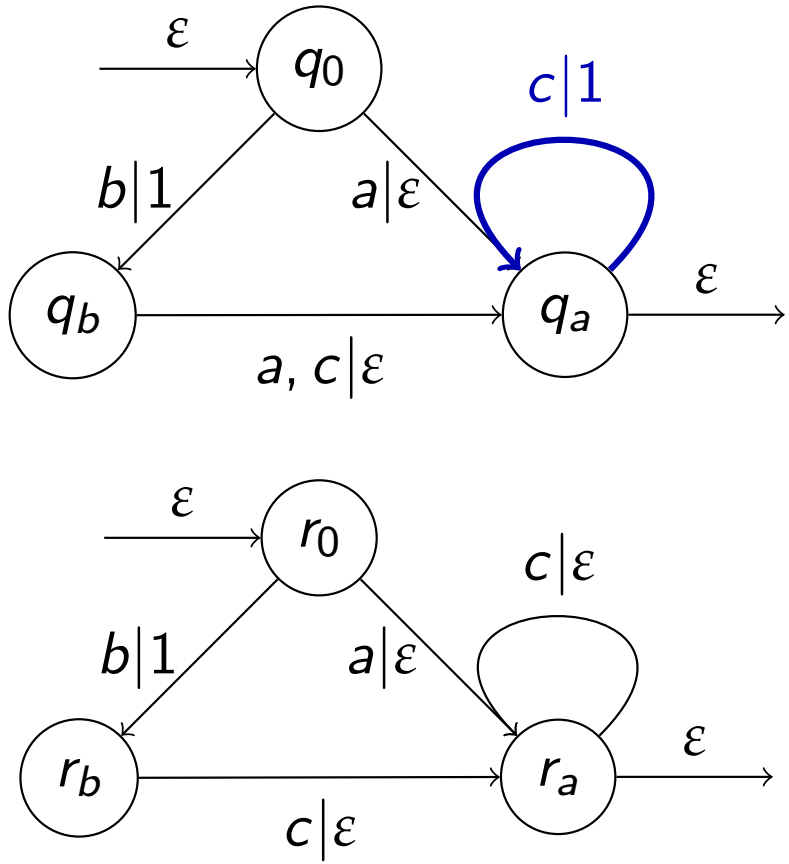
Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table

while *True* **do**

- **if** $u \cdot a \cdot v$ *is a witness of non-consistency* **then**
  - add av to S
  - **else if** ua *is a witness of non-closure* **then**
    - add ua to P
    - **else if** u *distinguishes two merging maps* **then**
      - add u and its suffixes to S
      - **else if** u *uses an open transition* **then**
        - add u and its suffixes to S
        - ???
        - **else**
          - $M :=$ Generate Transducer$(T(P, S))$
          - **if** u *is a non-equiv. witness for* $Up \times M$ **then**
            - add all its suffixes to S
          - **else**
            - return M

Merging Map

Input	Output	Partial output
$\varepsilon \in [\varepsilon]$	ε	ε
$a \in [a]$	ε	ε
$b \in [b]$	$\bot$	ε
$c \in [c]$	$\bot$	$\bot$
$ac \in [a]$	$\#$	ε
$bc \in [a]$	1	1



A muted transition may have an output of ε without changing the behaviour on the table.

Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table

while *True* **do**

- **if** $u \cdot a \cdot v$ *is a witness of non-consistency* **then**
  - add av to S
  - **else if** ua *is a witness of non-closure* **then**
    - add ua to P
    - **else if** u *distinguishes two merging maps* **then**
      - add u and its suffixes to S
      - **else if** u *uses an open transition* **then**
        - add u and its suffixes to S
        - **else if** *u uses a muted transition* **then**
          - add u and its suffixes to S
          - **else**
            - $M :=$ Generate Transducer$(T(P, S))$
            - **if** u *is a non-equiv. witness for* $Up \times M$ **then**
              - add all its suffixes to S
            - **else**
              - return M

Main result

Given a blackbox transduction τ and a domain upper-bound Up, our algorithm learns a transducer M of minimal size n such that $M_{|Up} = \tau$.

- Number of membership queries before a call to the SAT-solver: $POLY(n)$
- Number of calls to the SAT-solver before an equivalence query: $EXP(n)$
- Number of equivalence queries: n

Future works

- Better Evaluation of the number of calls to a SAT-solver.
- Work on an implementation.