Learning automata

Gabriele Puppis

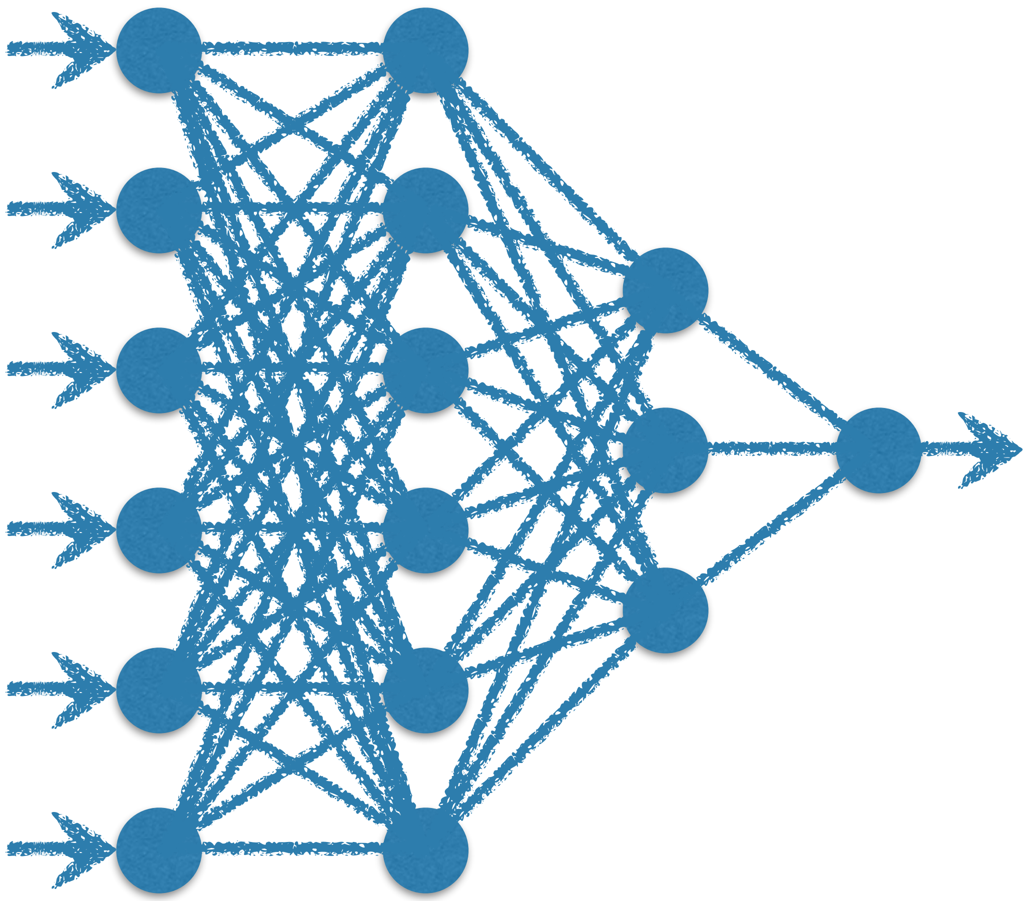
Learning automata ...and generalisations!

Gabriele Puppis

Learning scenarios

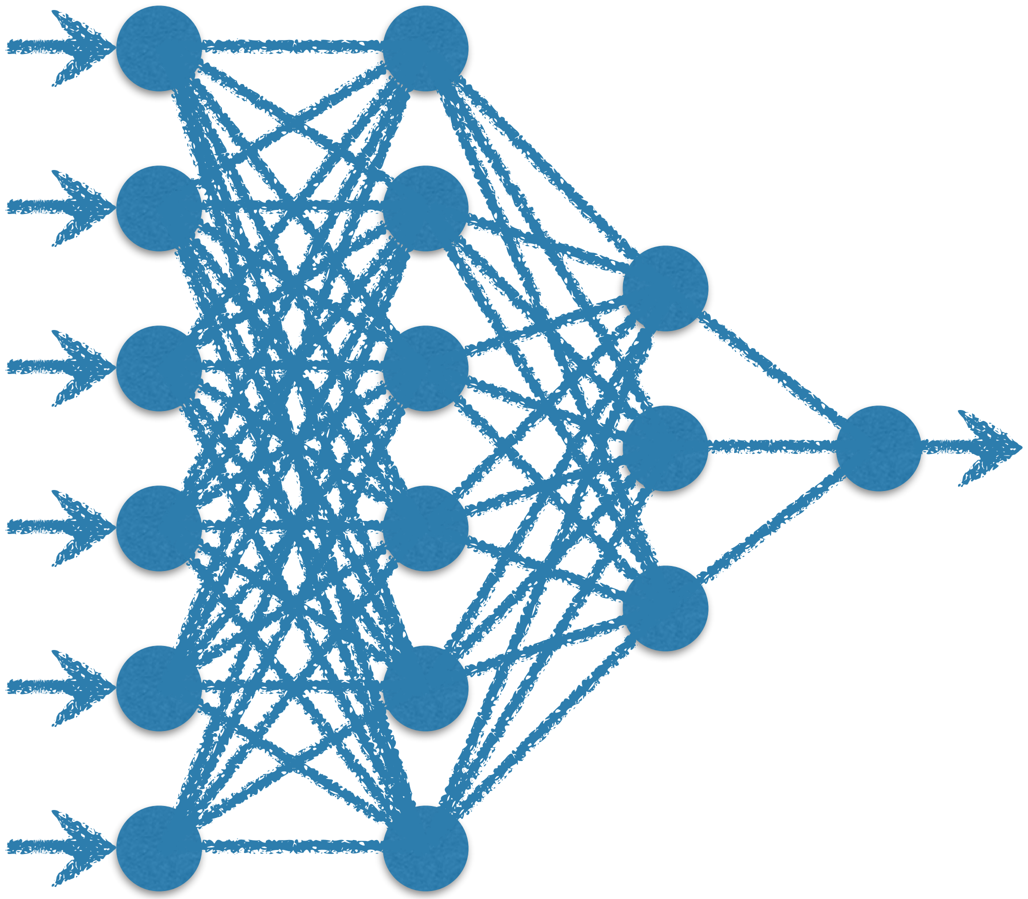
Learning scenarios

NNs

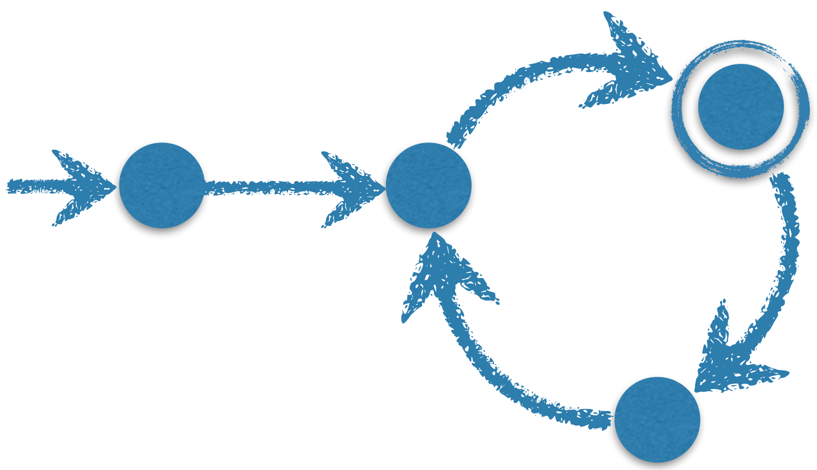


Learning scenarios

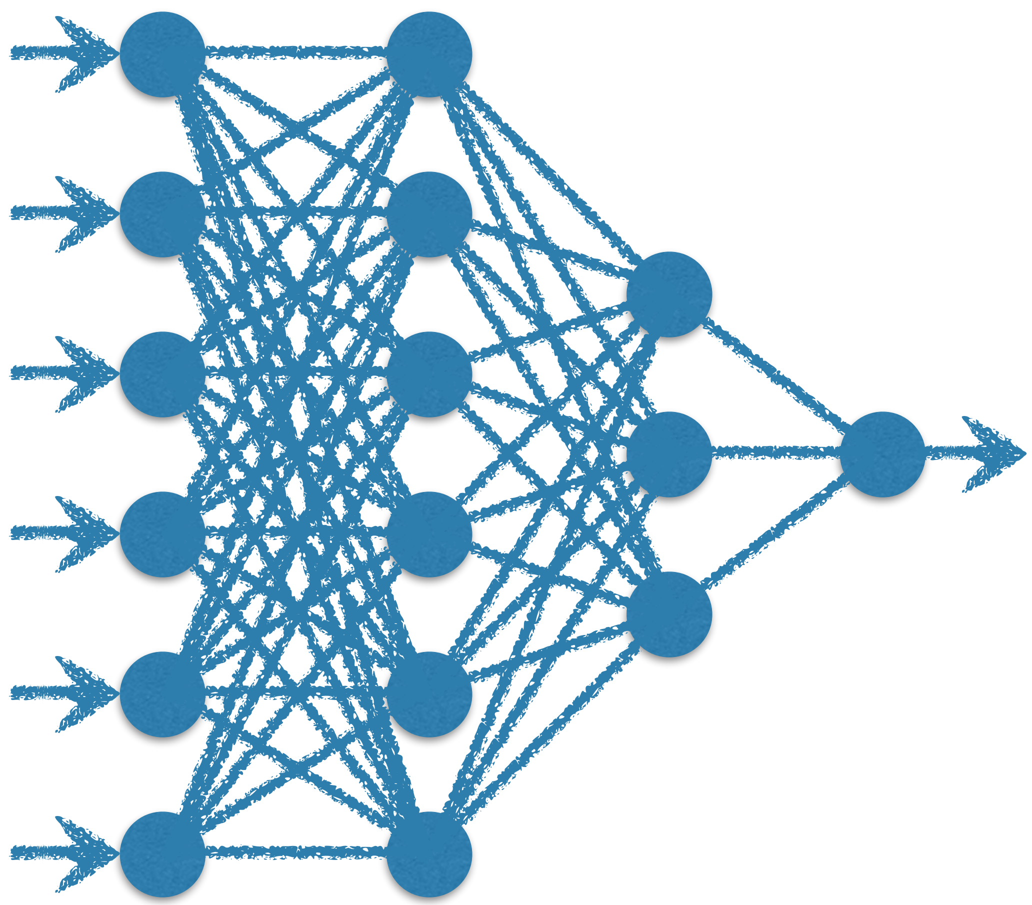
NNs



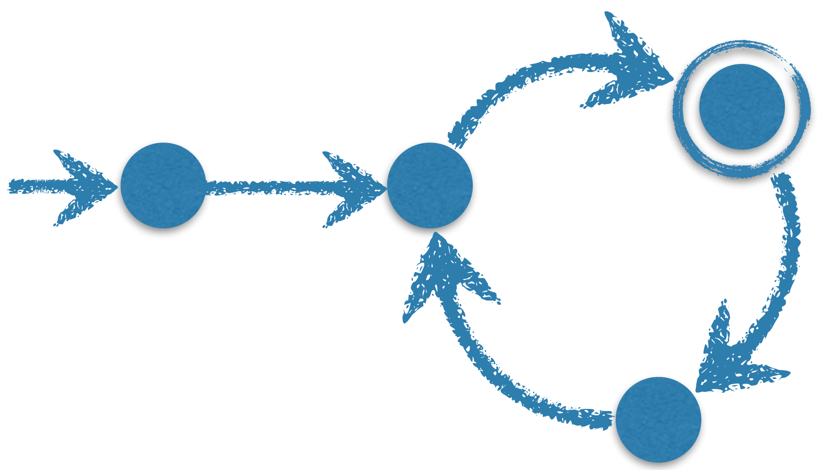
Automata



NNs



Automata

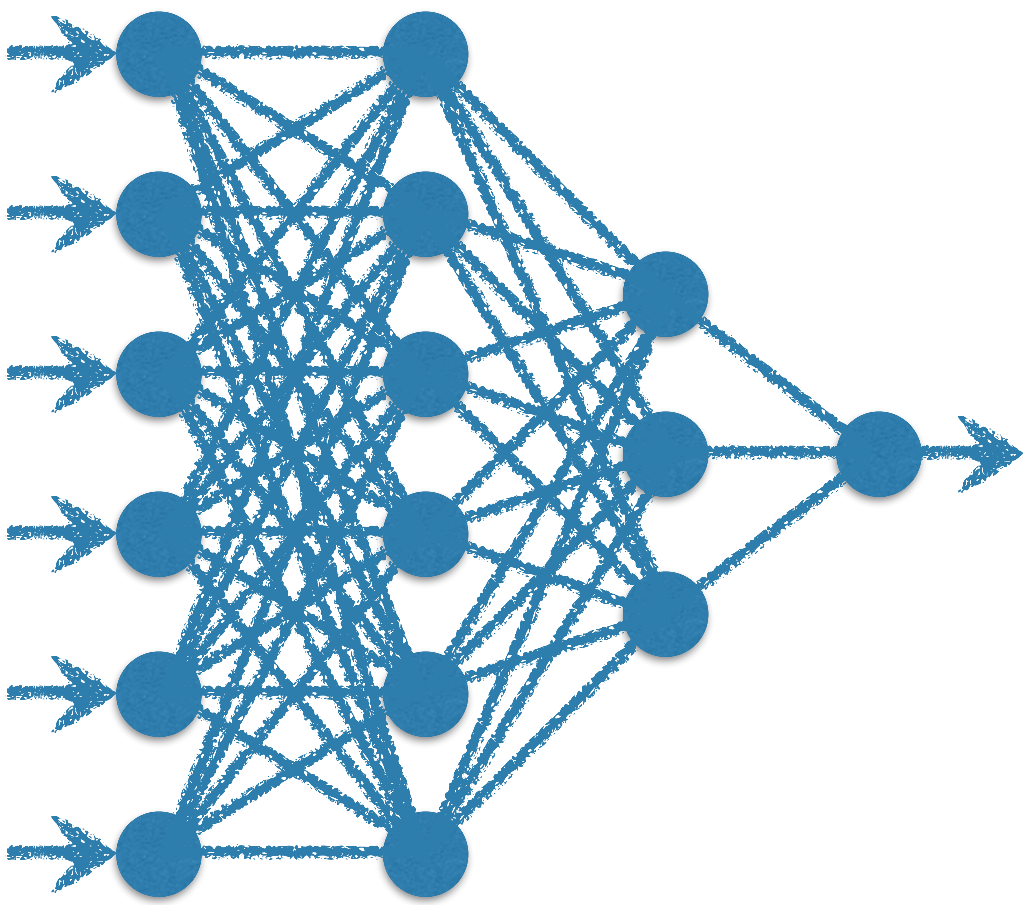


simple data, e.g. $\bar{\mathbf{x}} \in \mathbb{R}^k$

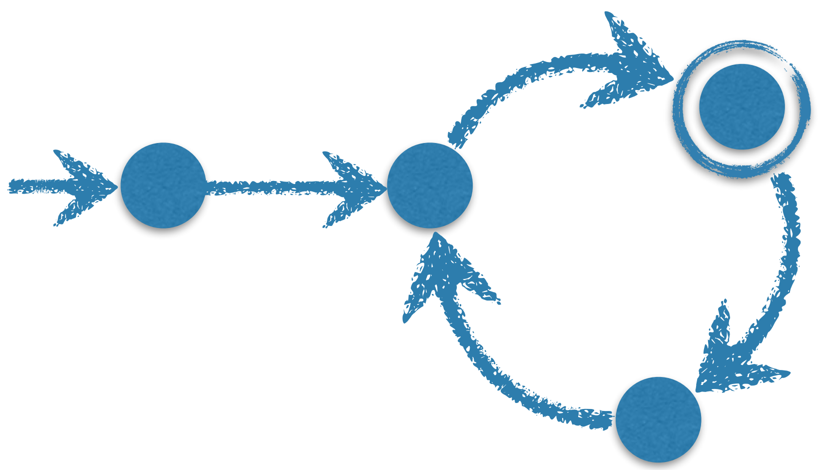
complex data, e.g. $w \in \Sigma^*$

Learning scenarios

NNs



Automata



simple data, e.g. $\bar{\mathbf{x}} \in \mathbb{R}^k$

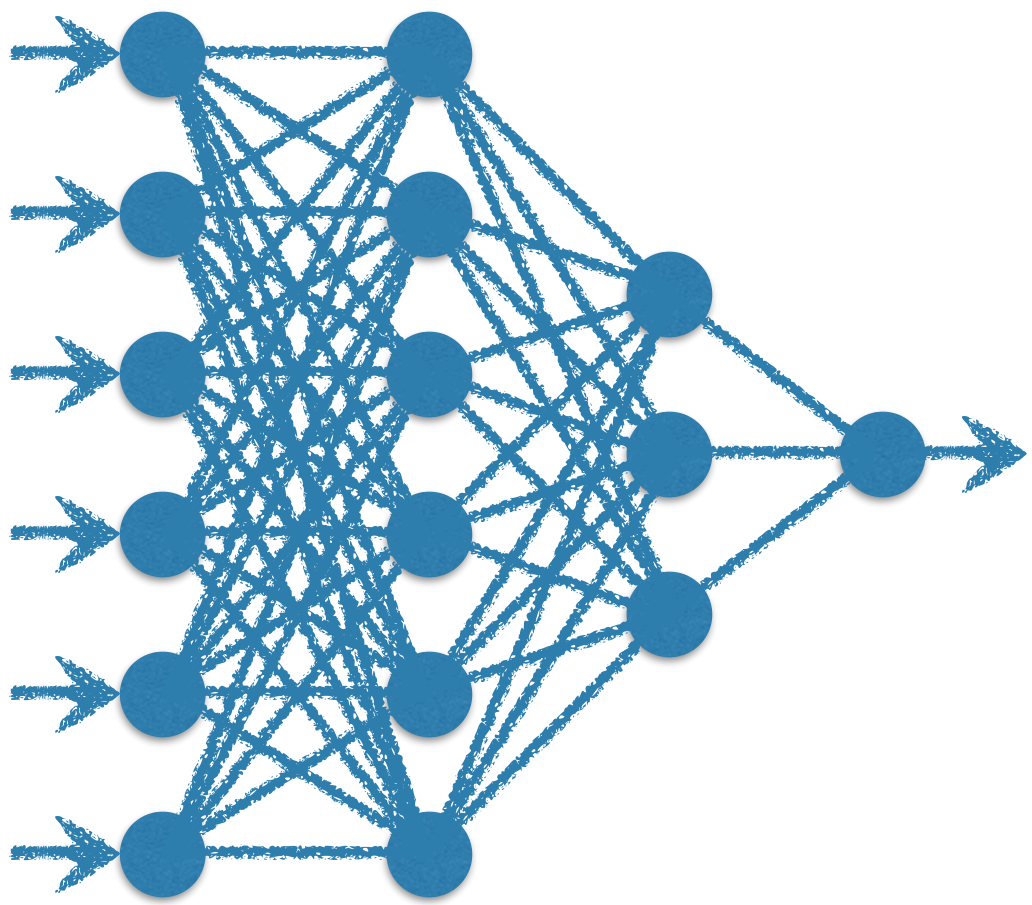
complex functions, e.g. f: $\mathbb{R}^k \to \{0,1\}$
representing pictures of dogs

complex data, e.g. $w \in \Sigma^*$

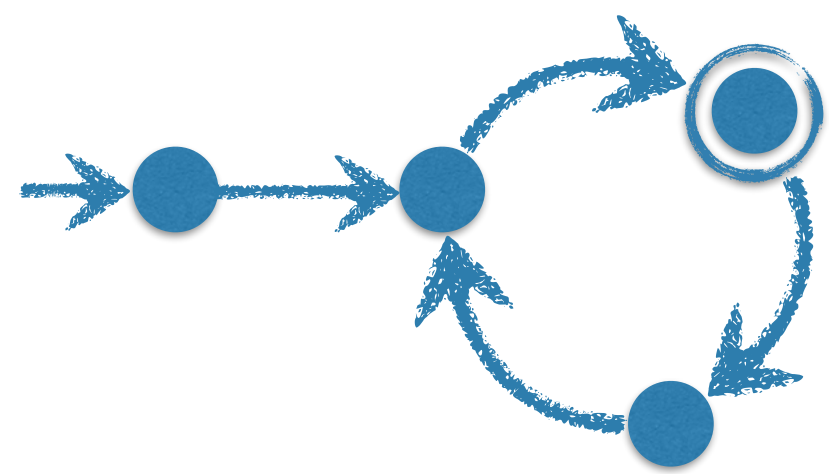
simple functions, e.g. f: $\Sigma^* \to \{0,1\}$
representing regular $L \subseteq \Sigma^*$

Learning scenarios

NNs



Automata



simple data, e.g. $\bar{\mathbf{x}} \in \mathbb{R}^k$

complex functions, e.g. f: $\mathbb{R}^k \rightarrow \{0,1\}$
representing pictures of dogs

structure (architecture) is fixed

complex data, e.g. $w \in \Sigma^*$

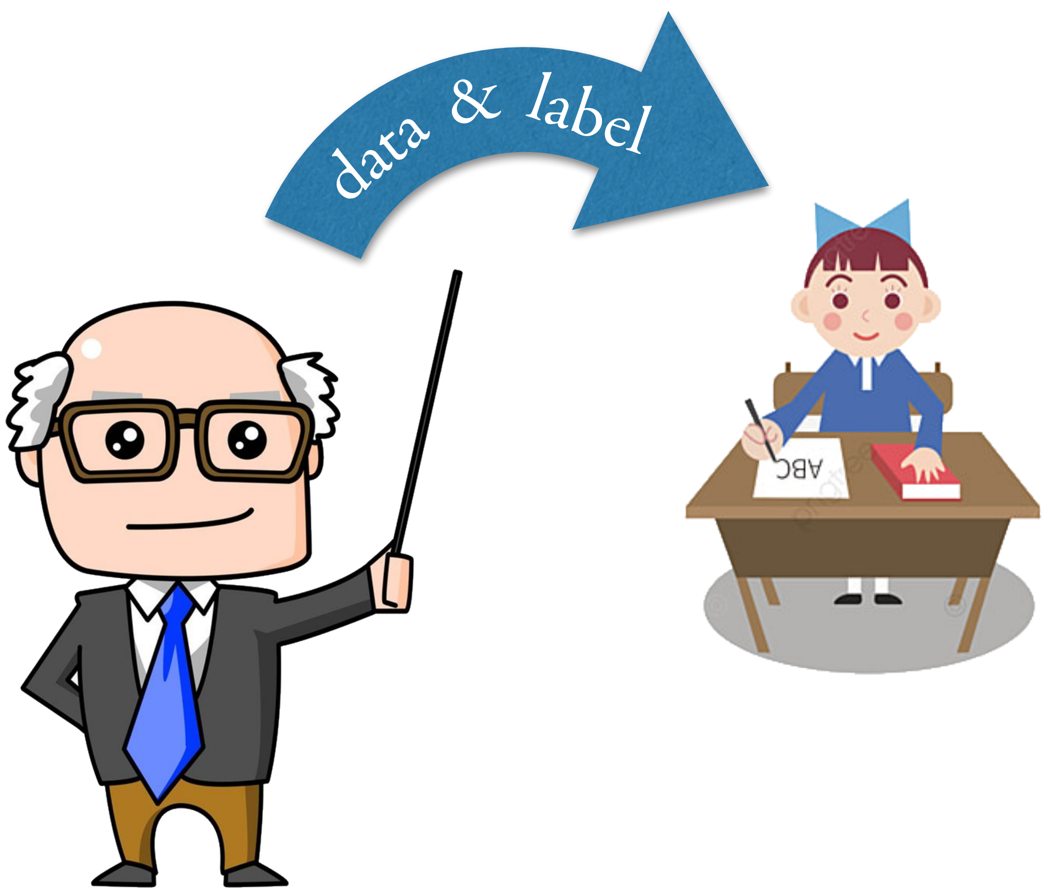
simple functions, e.g. f: $\Sigma^* \rightarrow \{0,1\}$
representing regular $L \subseteq \Sigma^*$

structure (states+transitions) is learned

Learning scenarios

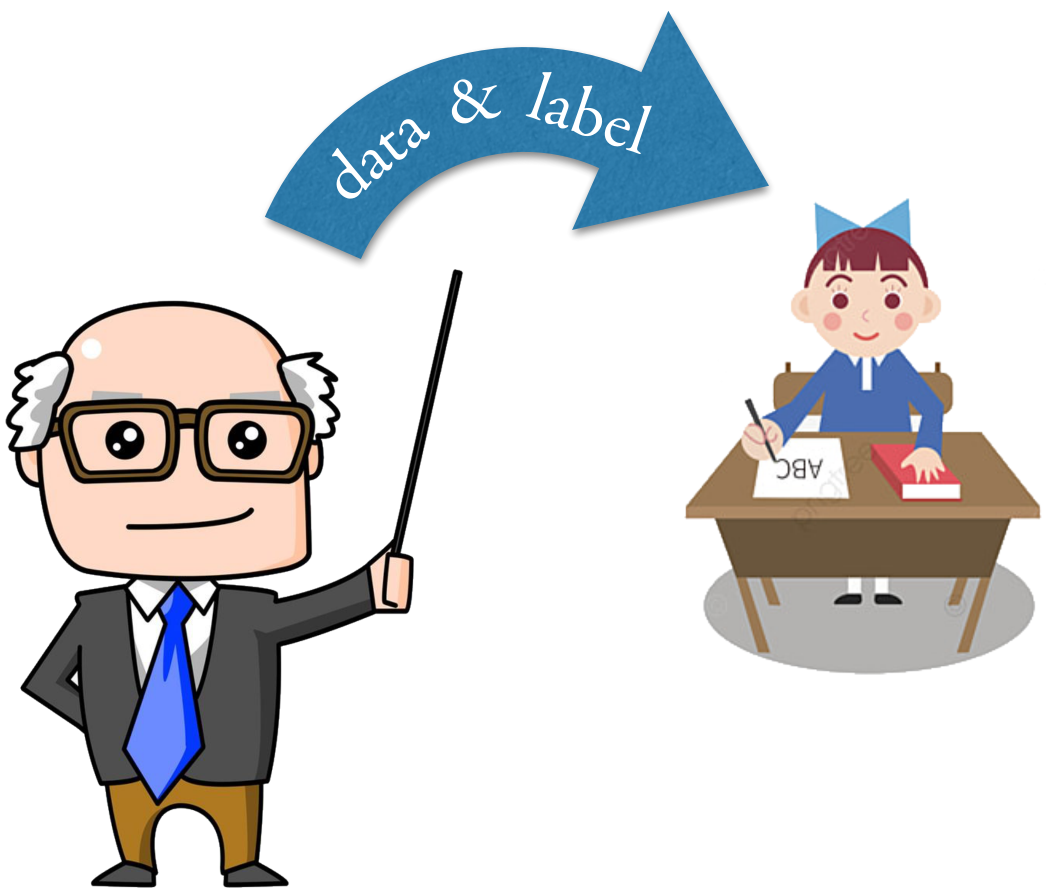
Learning scenarios

Passive

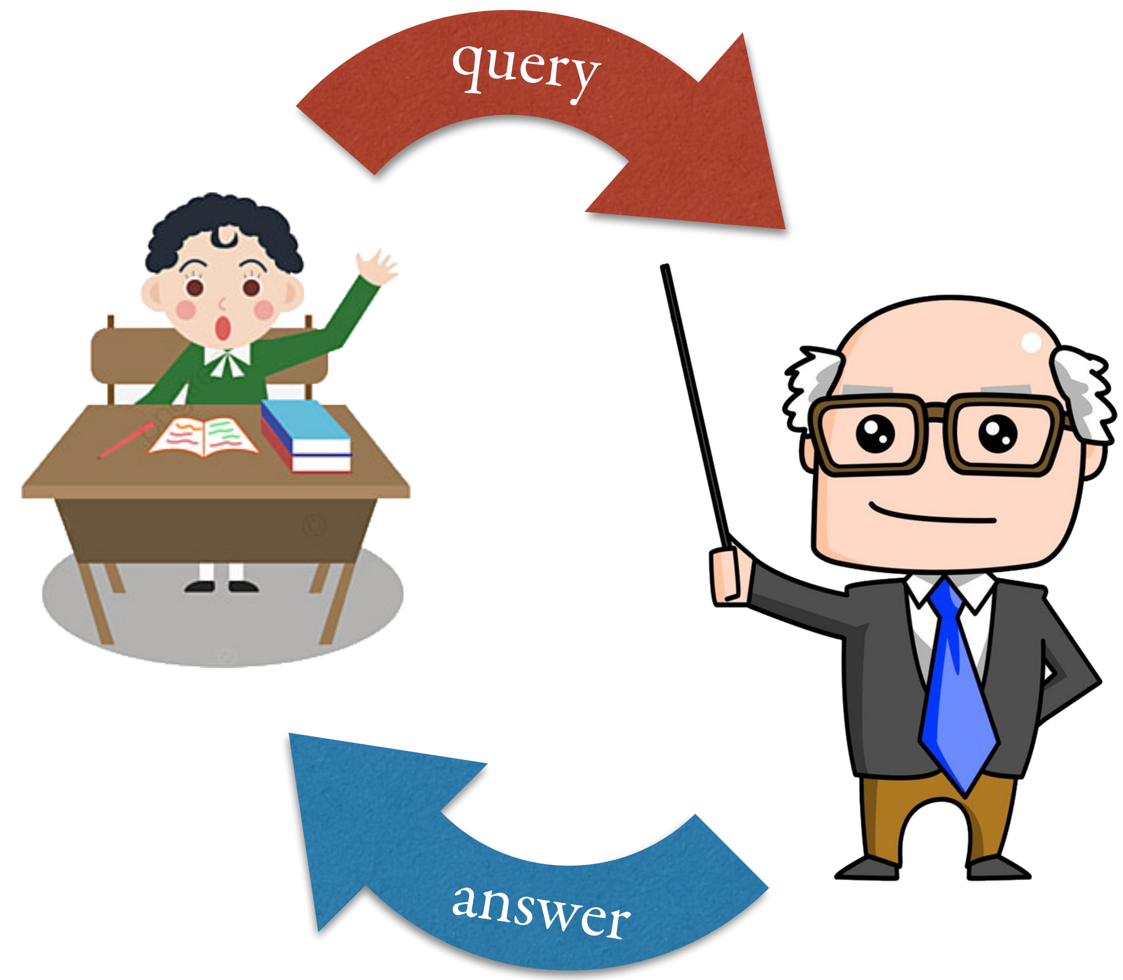


Learning scenarios

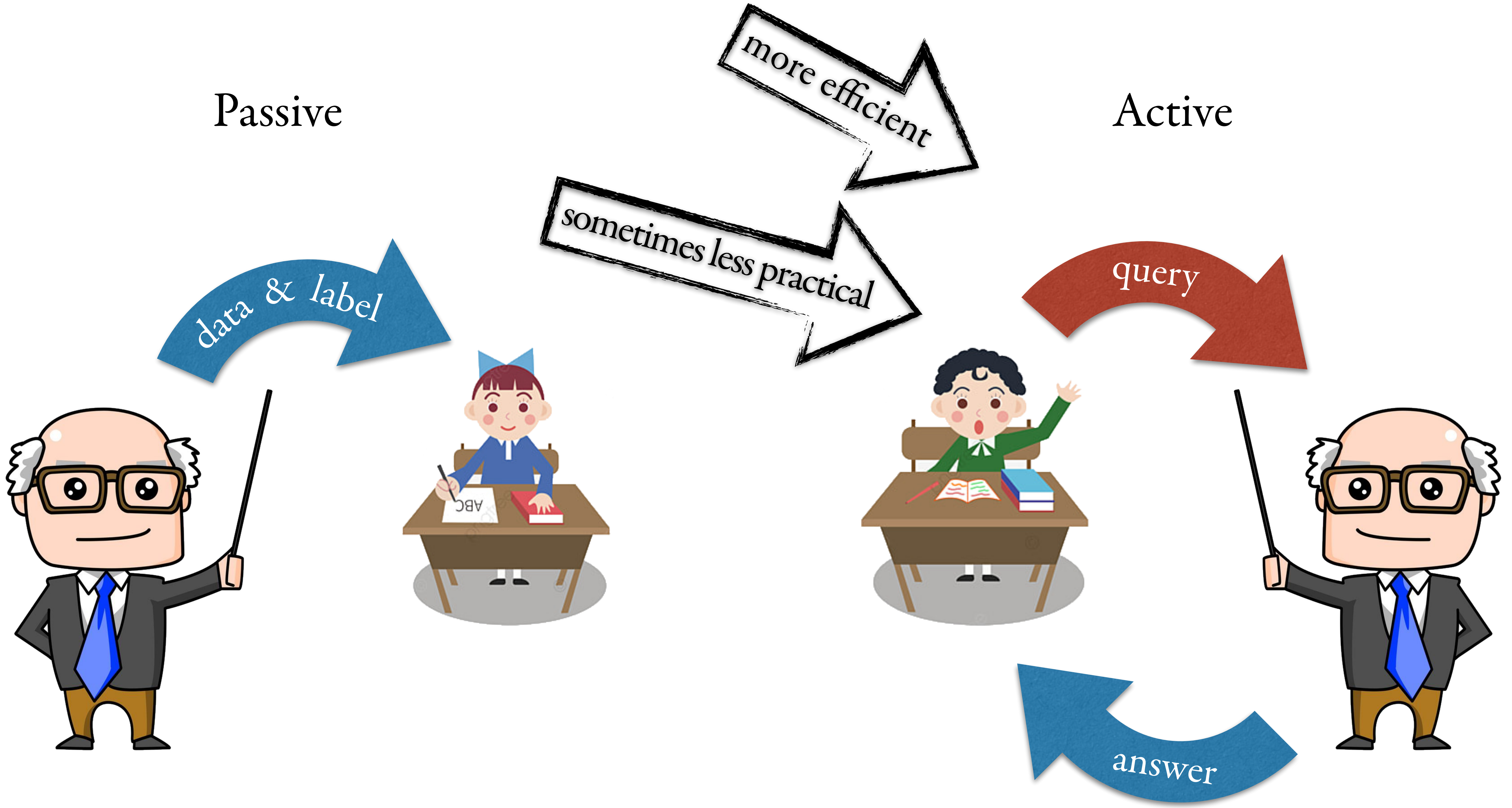
Passive



Active



Learning scenarios



A bit of history on automata learning

‘56	Moore	*Gedanken-experiments on sequential machines*
‘67	Gold	*passive learning in the limit*
‘87	Angluin	*active learning with queries*
’93…	Pitt et al.	*PAC-learning, cryptographic hardness*
’95…	Maler et al.	*learning regular ω-languages*
’96…	Vilar et al.	*learning word transformations*
’00…	Beimel et al.	*learning weighted and multiplicity automata*
’10…	Lemay et al.	*learning tree transformations*
’12…	Howar et al.	*learning languages over infinite alphabets*
’14…	Maier et al.	*learning timed languages*
’15…	Balle et al.	*spectral techniques for learning*

A bit of history on automata learning

- '56 Moore *Gedanken-experiments on sequential machines*
- '67 Gold
- '87 Angluin
- '93... Pitt et al.
- '95... Maler et a
- '96... Vilar et al
- '00... Beimel et
- '10... Lemay et
- '12... Howar et
- '14... Maier et a
- '15... Balle et al

Edward F. Moore

INTRODUCTION

This paper is concerned with finite automata[1] from the experimental point of view. This does not mean that it reports the results of any experimentation on actual physical models, but rather it is concerned with what kinds of conclusions about the internal conditions of a finite machine it is possible to draw from external experiments. To emphasize the conceptual nature of these experiments, the word "gedanken-experiments" has been borrowed from the physicists for the title.

The sequential machines considered have a finite number of states, a finite number of possible input symbols, and a finite number of possible output symbols. The behavior of these machines is strictly deterministic (i.e., no random elements are permitted in the machines) in that the present state of a machine depends only on its previous input and previous state, and the present output depends only on the present state.

The point of view of this paper might also be extended to probabilistic machines (such as the noisy discrete channel of communication theory[2]), but this will not be attempted here.

EXPERIMENTS

There will be two kinds of experiments considered in this paper. The first of these, called a simple experiment, is depicted in Figure 1.

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
Learner initially only knows the underlying alphabet Σ

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
   Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a <u>membership query</u> "Does $w \in L_0$?"
   b) or an <u>equivalence query</u> "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a membership query "Does $w \in L_0$?"
   b) or an equivalence query "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

3) Teacher answers accordingly:
   a) yes if $w \in L_0$, no otherwise
   b) yes if $L_0 = L(A)$ (game ends and Learner wins),
   otherwise gives a shortest counter-example $w \in (L_0 - L(A)) \cup (L(A) - L_0)$
   (game continues from 2)

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a <u>membership query</u> "Does $w \in L_0$?"
   b) or an <u>equivalence query</u> "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

3) Teacher answers accordingly:
   a) yes if $w \in L_0$, no otherwise
   b) yes if $L_0 = L(A)$ (game ends and Learner wins),
      otherwise gives a shortest <u>counter-example</u> $w \in (L_0 - L(A)) \cup (L(A) - L_0)$
      (game continues from 2)

Observations

- membership queries alone are not sufficient for Learner to win

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a <u>membership query</u> "Does $w \in L_0$?"
   b) or an <u>equivalence query</u> "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

3) Teacher answers accordingly:
   a) yes if $w \in L_0$, no otherwise
   b) yes if $L_0 = L(A)$ (game ends and Learner wins),
   otherwise gives a shortest <u>counter-example</u> $w \in (L_0 - L(A)) \cup (L(A) - L_0)$
   (game continues from 2)

Observations

- membership queries alone are not sufficient for Learner to win
- instead, equivalence queries alone are sufficient to win (why? how quick?)

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
   Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a membership query "Does $w \in L_0$?"
   b) or an equivalence query "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

3) Teacher answers accordingly:
   a) yes if $w \in L_0$, no otherwise
   b) yes if $L_0 = L(A)$ (game ends and Learner wins),
      otherwise gives a shortest counter-example $w \in (L_0 - L(A)) \cup (L(A) - L_0)$
      (game continues from 2)

Theorem [Angluin '87] Learner has a strategy to win in a number of rounds that is *polynomial* in $|A_0|$

Learning as a game

1) Teacher has a secret regular language L_0, e.g. represented by DFA A_0
Learner initially only knows the underlying alphabet Σ

2) Learner choses a query:
   a) either a <u>membership query</u> "Does $w \in L_0$?"
   b) or an <u>equivalence query</u> "Is $L_0 = L(A)$?" (or "Are A_0, A equivalent?")

3) Teacher answers accordingly:
   a) yes if $w \in L_0$, no otherwise
   b) yes if $L_0 = L(A)$ (game ends and Learner wins),
   otherwise gives a shortest <u>counter-example</u> $w \in (L_0 - L(A)) \cup (L(A) - L_0)$
   (game continues from 2)

in the form of an algorithm (L^* algorithm)

Theorem [Angluin '87]
Learner has a <u>strategy</u> to win in a number of rounds that is *polynomial* in $|A_0|$

Learning as a game

1) Teacher has a secret regular language L_0, e.g. rep
Learner initially only knows the underlying alph

2) Learner choses a query:
a) either a membership query "Does $w \in L_0$?
b) or an equivalence query "Is $L_0 = L(A)$?

3) Teacher answers accordingly:
a) yes if $w \in L_0$, no otherwise
b) yes if $L_0 = L(A)$ (game ends and Learner
otherwise gives a shortest counter-example
(game continues from 2)

MAT (Minimally Adequate Teacher)

Sometimes answering an equivalence query is not practical: if Teacher knew A_0, he could pass this information directly to Learner, so why bothering querying?

Equivalence queries can however be *approximated* by a series of membership queries: as long as membership in A matches membership in A_0, Learner assumes he made the correct guess.

This latter setting is often called
Black-box learning

in the form of an algorithm (L^* algorithm)

Theorem [Angluin '87] Learner has a strategy to win in a number of rounds that is *polynomial* in $|A_0|$

Learning as a game

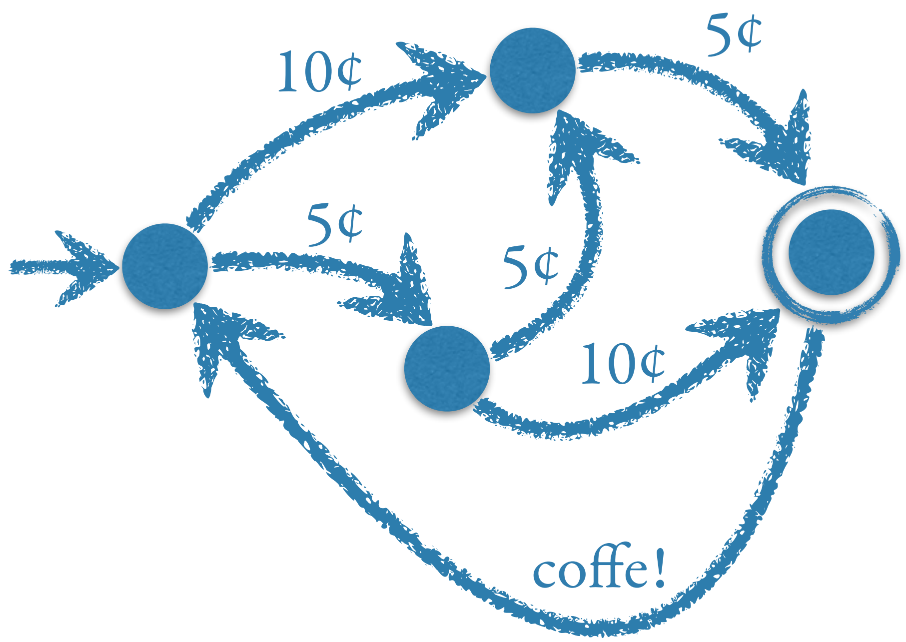
<u>MAT (Minimally Adequate Teacher)</u>

Sometimes answering an equivalence query is not practical: if Teacher knew A_0, he could pass this information directly to Learner, so why bothering querying?

Equivalence queries can however be *approximated* by a series of membership queries: as long as membership in A matches membership in A_0, Learner assumes he made the correct guess.

This latter setting is often called
<u>Black-box learning</u>

Learning as a game



<u>MAT (Minimally Adequate Teacher)</u>

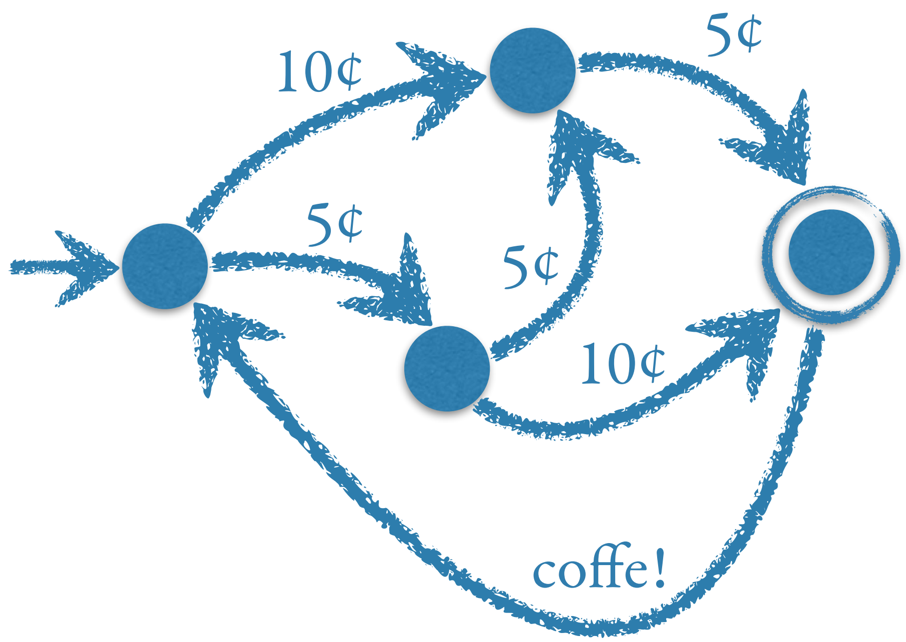
Sometimes answering an equivalence query is not practical: if Teacher knew A_0, he could pass this information directly to Learner, so why bothering querying?

Equivalence queries can however be *approximated* by a series of membership queries: as long as membership in A matches membership in A_0, Learner assumes he made the correct guess.

This latter setting is often called

<u>Black-box learning</u>

Learning as a game



Verification: model-learning
(in contrast to model-checking)

Control theory:
system identification,
diagram inference

Language theory:
grammar inference,
regular extrapolation

Security:
protocol state fuzzing

<u>MAT (Minimally Adequate Teacher)</u>

Sometimes answering an equivalence query is not practical: if Teacher knew A_0, he could pass this information directly to Learner, so why bothering querying?

Equivalence queries can however be *approximated* by a series of membership queries: as long as membership in A matches membership in A_0, Learner assumes he made the correct guess.

This latter setting is often called
<u>Black-box learning</u>

Learning as a killer app of Myhill-Nerode theorem

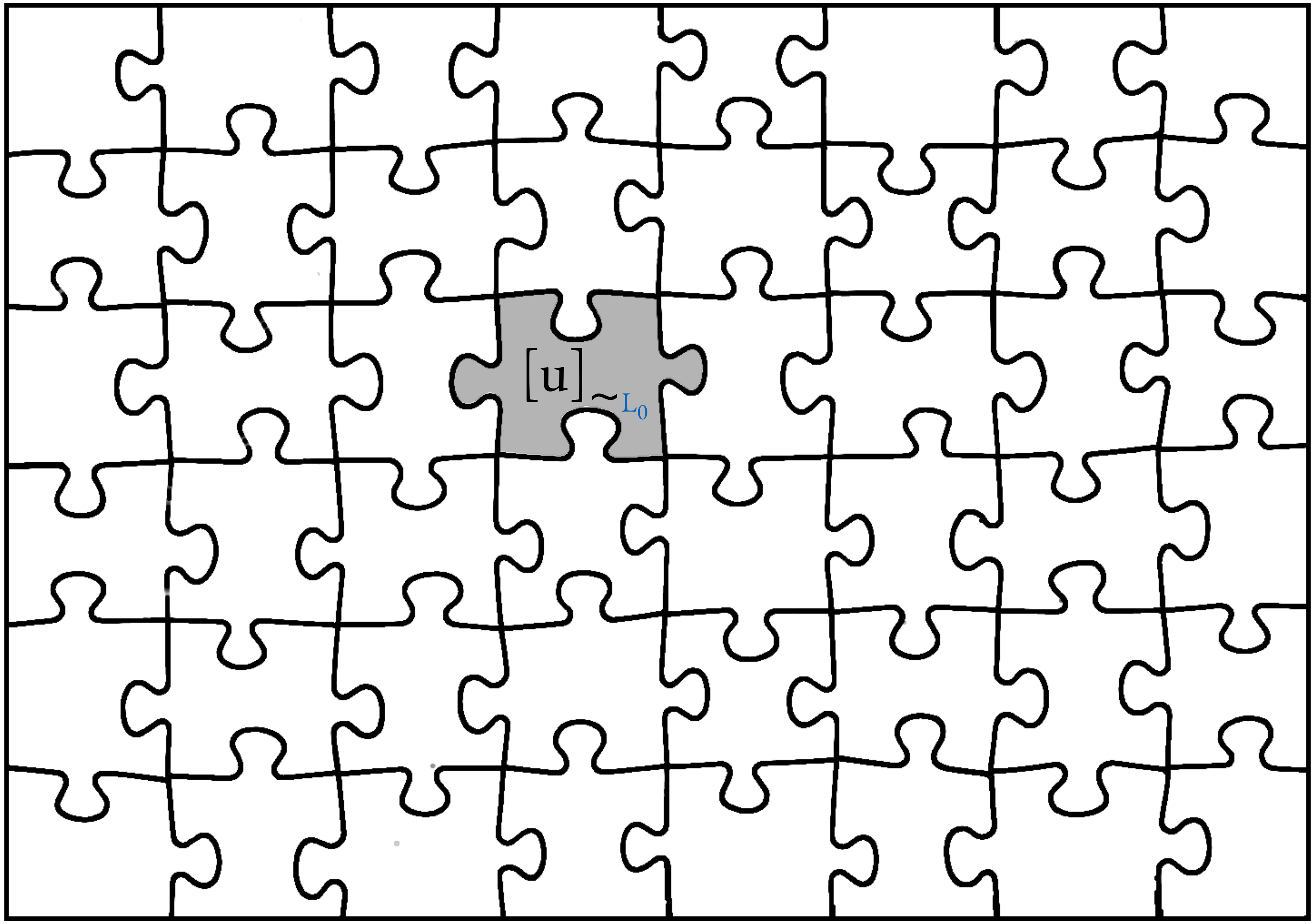
Myhill-Nerode equivalence: $u \sim_{L_0} v$ if ....?

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

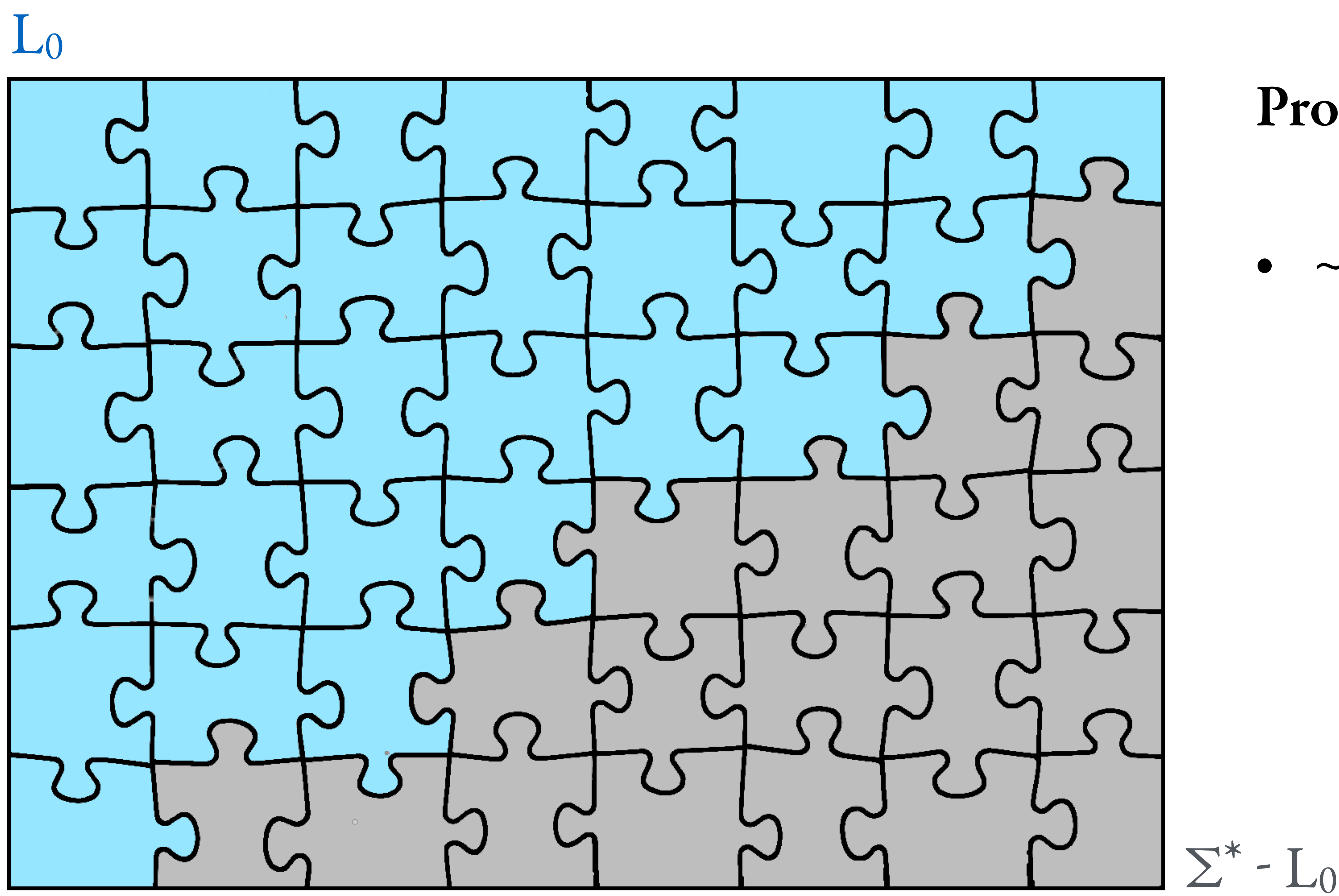
Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; ut \in L_0 \leftrightarrow vt \in L_0$



Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

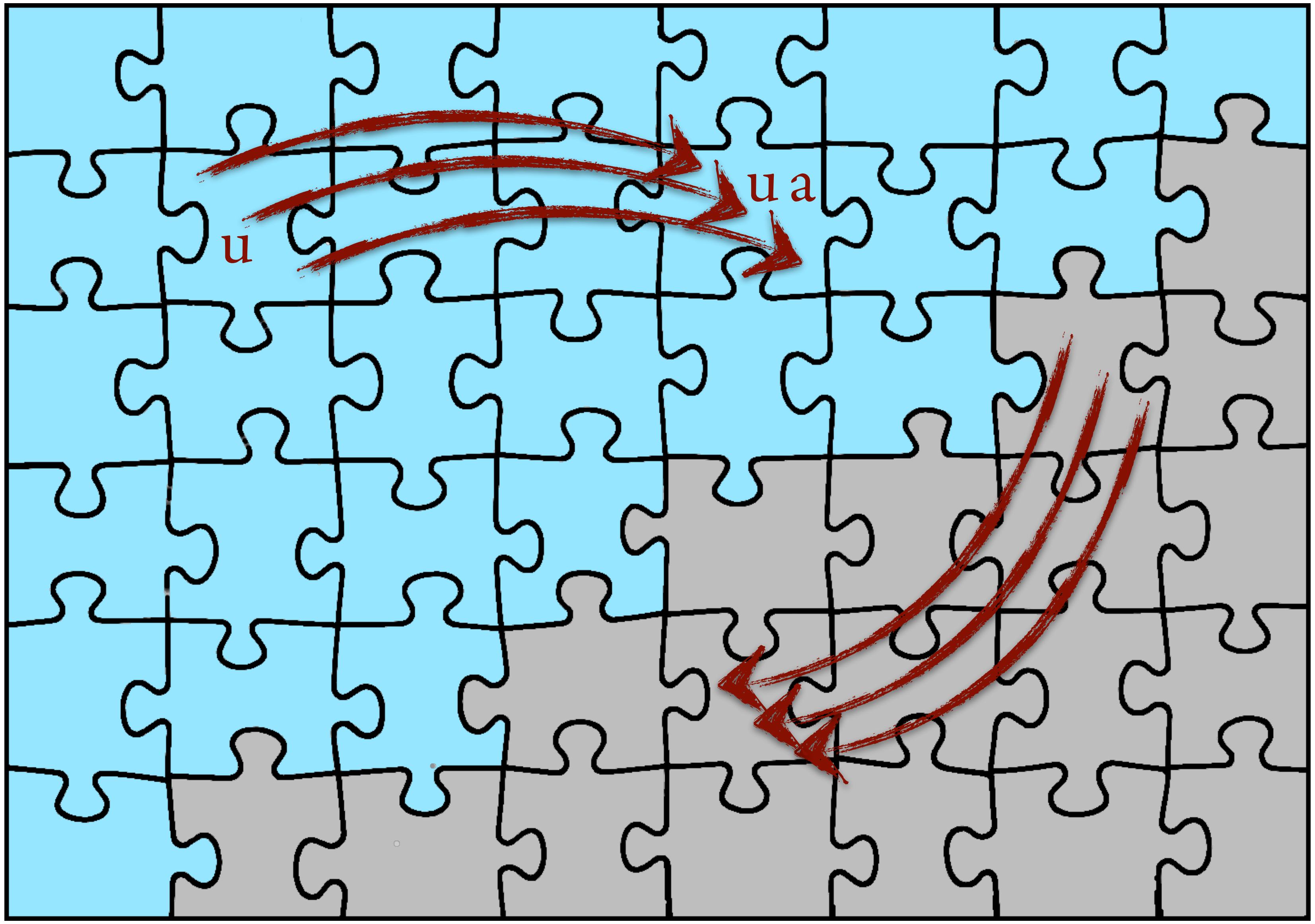


Properties:

- $\sim_{L_0}$ refines $=_{L_0}$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

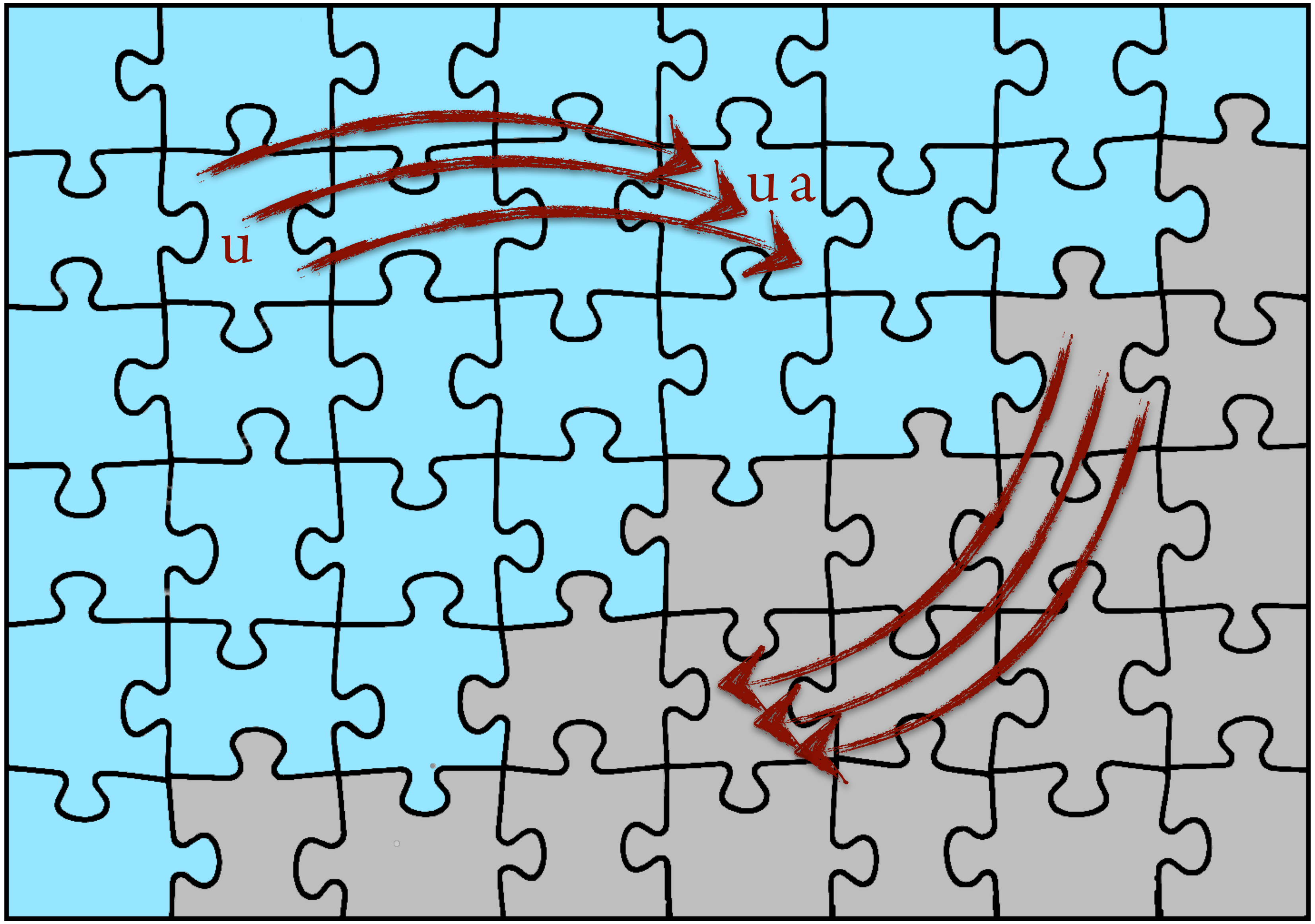


Properties:

- $\sim_{L_0}$ refines $=_{L_0}$
- $\sim_{L_0}$ is right-invariant (i.e. $u \sim_{L_0} v \rightarrow u\,a \sim_{L_0} v\,a$)

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

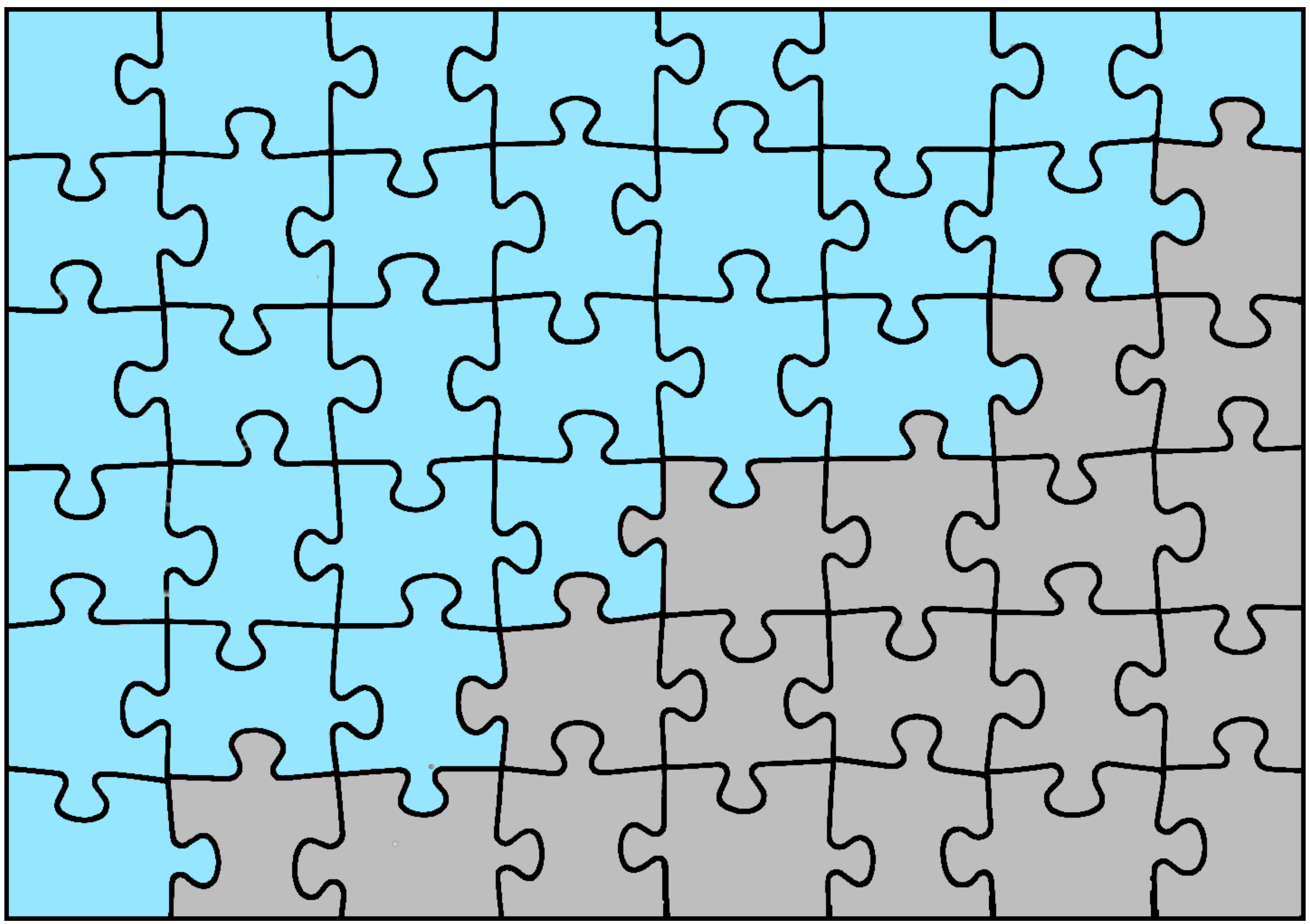


Properties:

- $\sim_{L_0}$ refines $=_{L_0}$
- $\sim_{L_0}$ is right-invariant (i.e. $u \sim_{L_0} v \rightarrow u\,a \sim_{L_0} v\,a$)
- $\sim_{L_0}$ has finite index iff L_0 is regular

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; u\,t \in L_0 \leftrightarrow v\,t \in L_0$



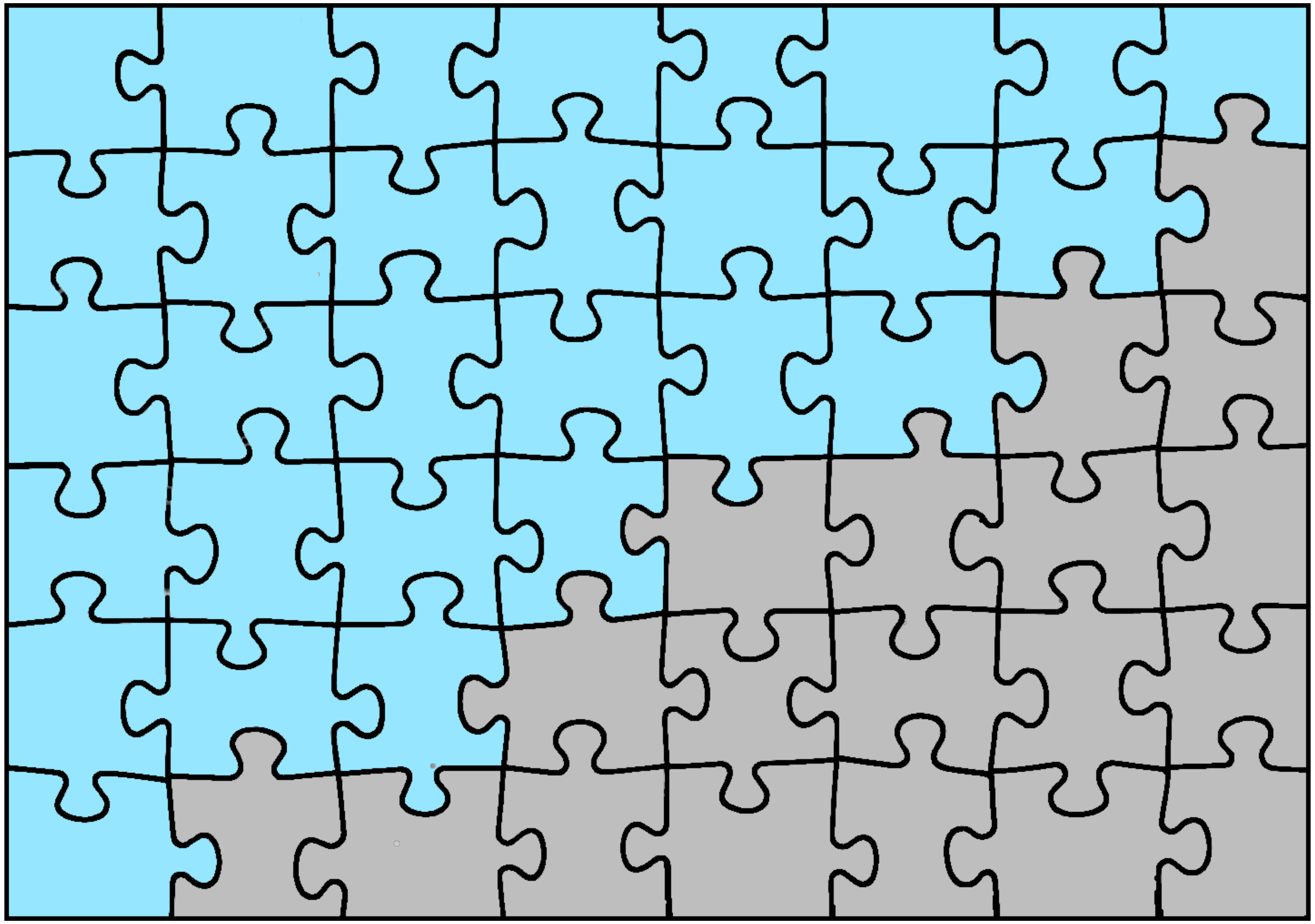
Properties:

- $\sim_{L_0}$ refines $=_{L_0}$
- $\sim_{L_0}$ is right-invariant (i.e. $u \sim_{L_0} v \rightarrow u\,a \sim_{L_0} v\,a$)
- $\sim_{L_0}$ has finite index iff L_0 is regular

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; ut \in L_0 \leftrightarrow vt \in L_0$



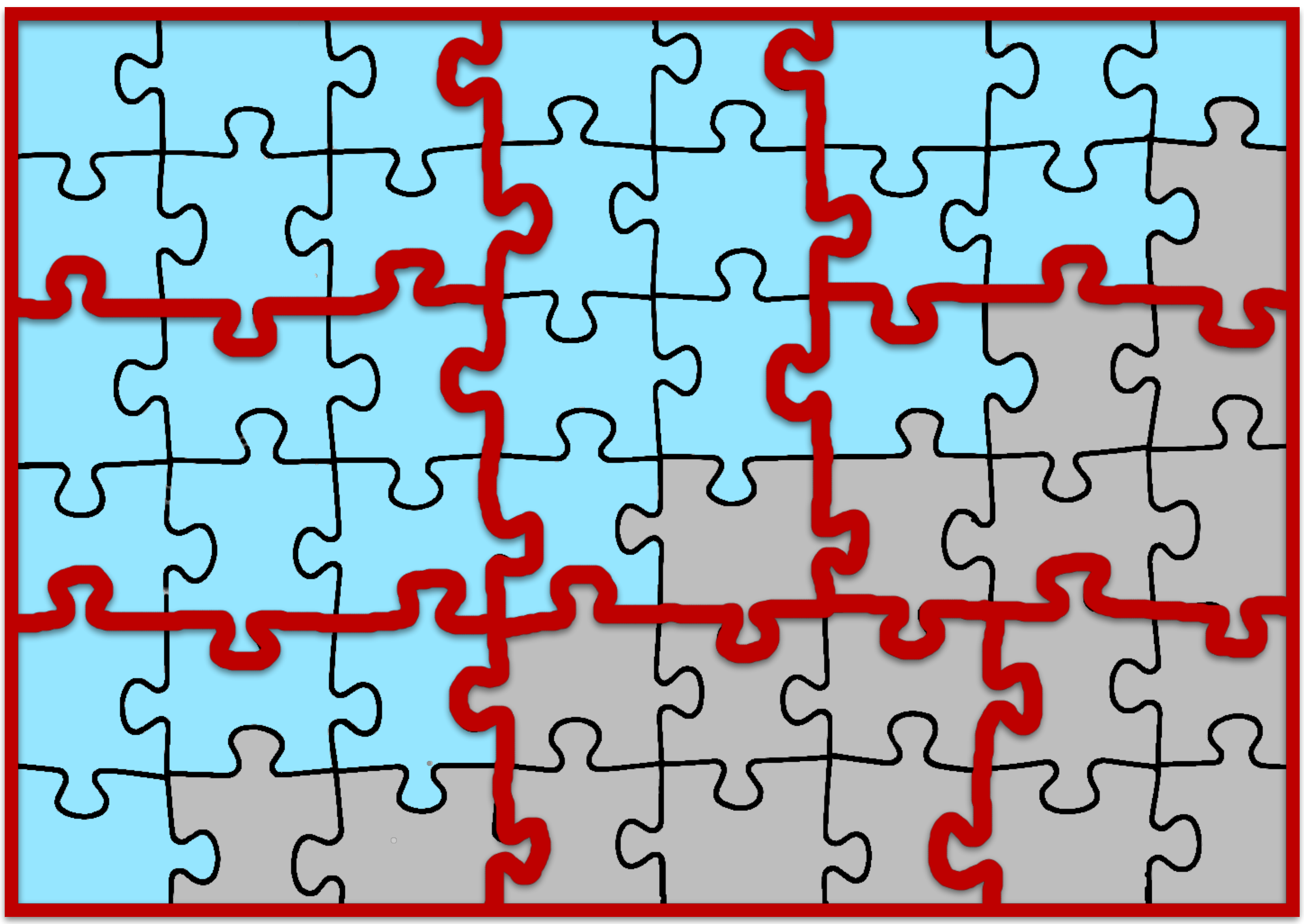
Properties:

- $\sim_{L_0}$ refines $=_{L_0}$
- $\sim_{L_0}$ is right-invariant (i.e. $u \sim_{L_0} v \rightarrow ua \sim_{L_0} va$)
- $\sim_{L_0}$ has finite index iff L_0 is regular

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$



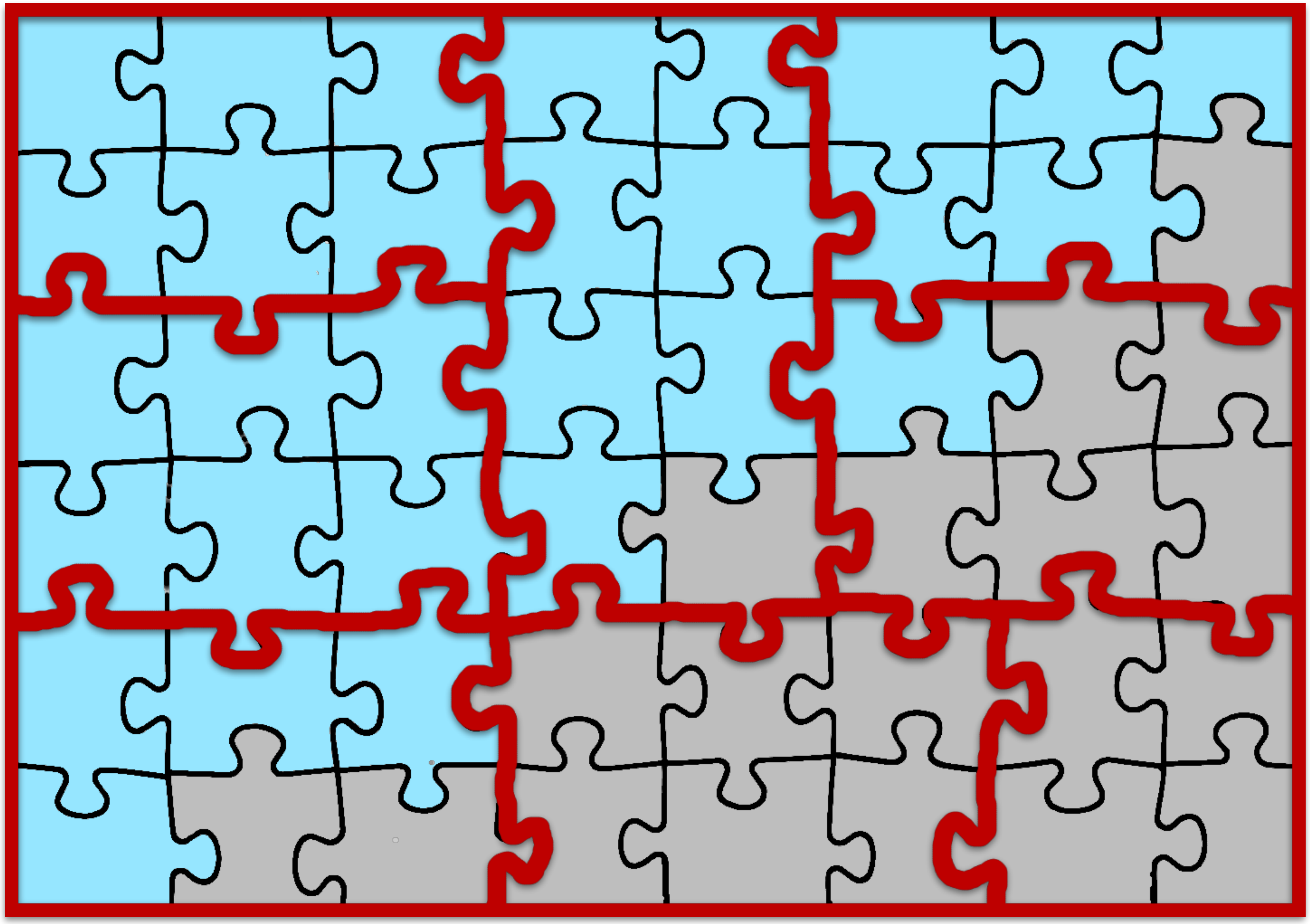
Properties:

- $\sim_{L_0}$ refines $=_{L_0}$
- $\sim_{L_0}$ is right-invariant (i.e. $u \sim_{L_0} v \rightarrow u\,a \sim_{L_0} v\,a$)
- $\sim_{L_0}$ has finite index iff L_0 is regular
- $\approx_{L_0,T}$ is coarser than $\sim_{L_0}$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $ut \in L_0 \leftrightarrow vt \in L_0$

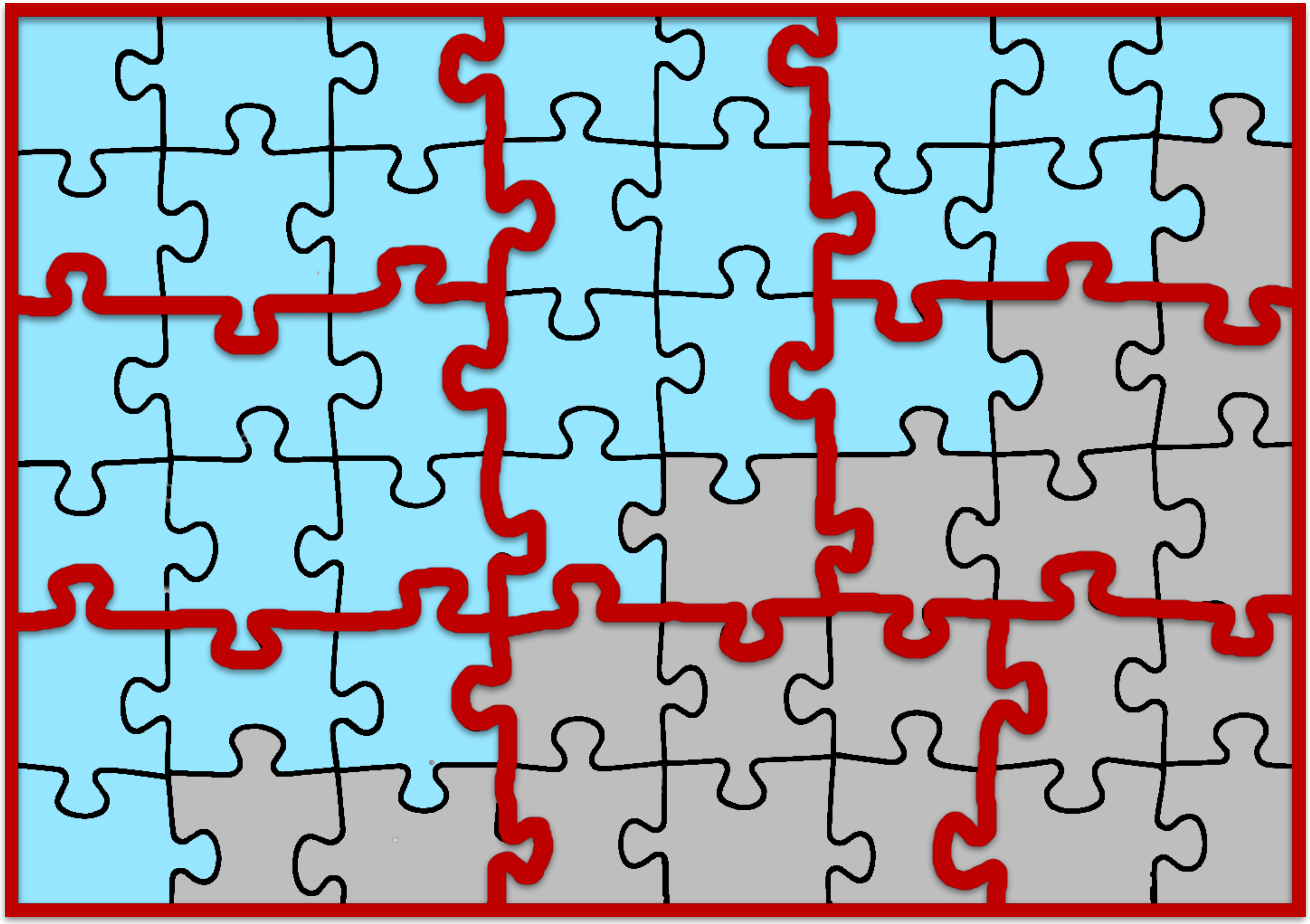
Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $ut \in L_0 \leftrightarrow vt \in L_0$



Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

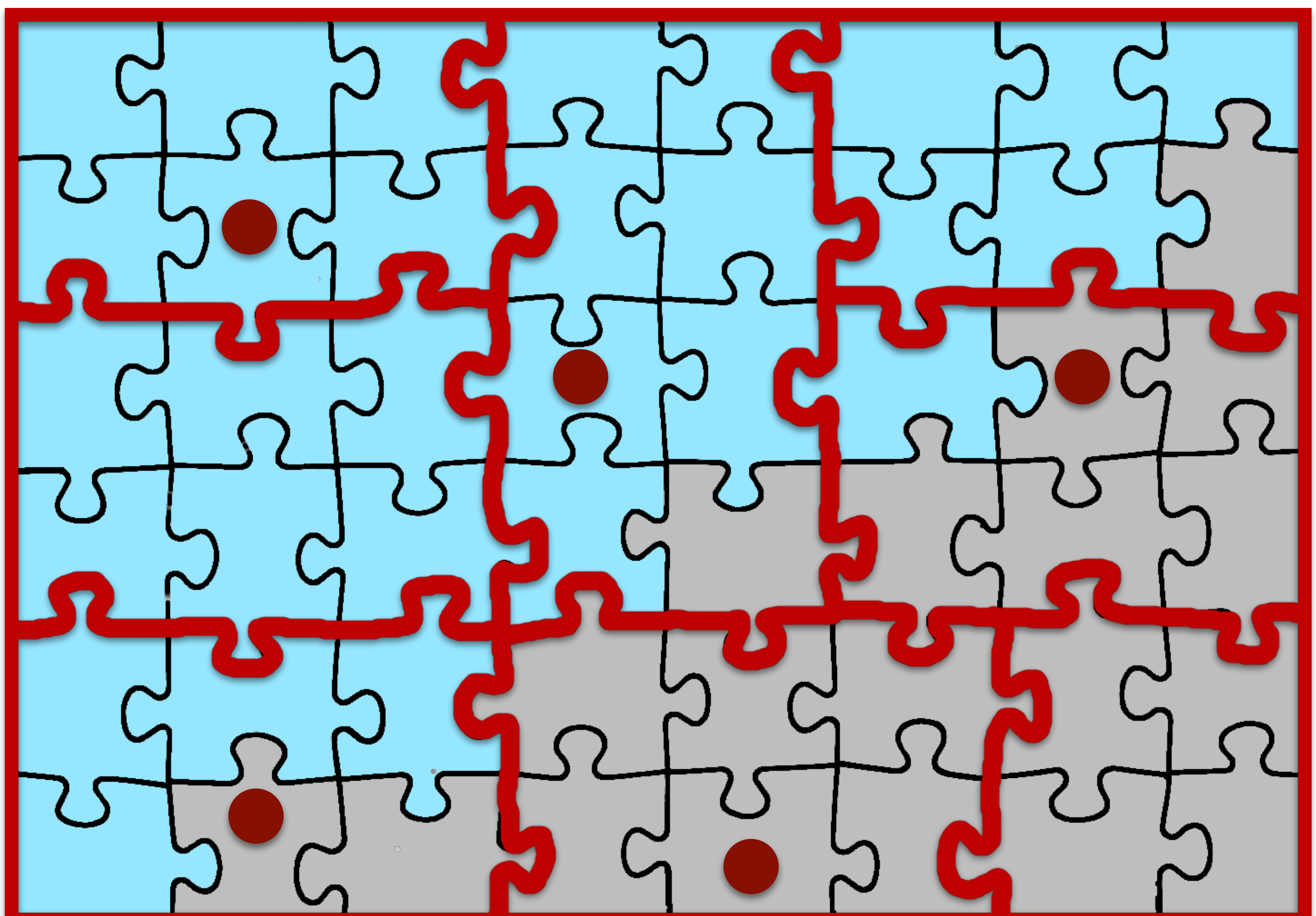


Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; ut \in L_0 \leftrightarrow vt \in L_0$



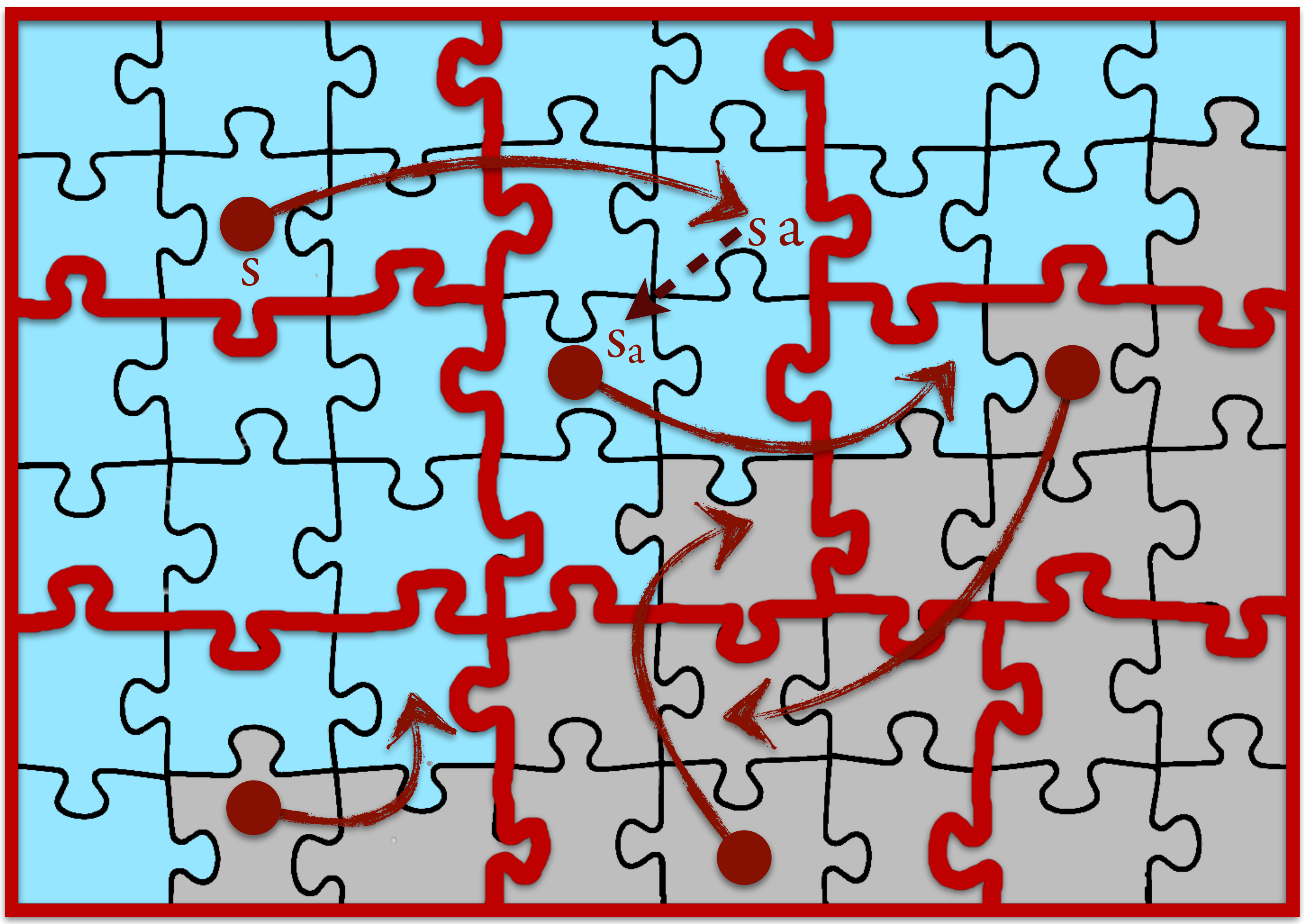
Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is <u>T-minimal</u> if $\forall s \neq s' \in S \;\; s \not\approx_{L_0,T} s'$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$



Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal
if $\forall s \neq s' \in S \quad s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete
if $\forall s \in S \quad \forall a \in \Sigma$
$\exists s_a \in S \quad s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Learner-strategy

$S = T = \{\varepsilon\}$ // S is T-minimal, possibly not T-complete

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal
if $\forall s \neq s' \in S \;\; s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete
if $\forall s \in S \;\; \forall a \in \Sigma$
$\exists s_a \in S \;\; s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Learner-strategy

```
S = T = {ε}   // S is T-minimal, possibly not T-complete
loop
  while S NOT T-complete
    let s ∈ S and a ∈ Σ such that
        ∀s'∈S ∃t∈T Membership(s a t) ≠ Membership(s' t)
    S = S ∪ {s a}
```

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal if $\forall s \neq s' \in S \;\; s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete if $\forall s \in S \;\; \forall a \in \Sigma$
$\exists\, s_a \in S \;\; s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \ \ u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \ \ u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Learner-strategy

S = T = {ε} // S is T-minimal, possibly not T-complete

loop

while S NOT T-complete

let s ∈ S and a ∈ Σ such that

∀s'∈S ∃t∈T Membership(s a t) ≠ Membership(s' t)

S = S ∪ {s a}

A = DFA with state set S, transitions $\delta(s,a) = s_a$

initial state ε, final states s' s.t. Membership(s')

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal if $\forall s \neq s' \in S \ \ s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete if $\forall s \in S \ \ \forall a \in \Sigma$

$\exists s_a \in S \ \ s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $ut \in L_0 \leftrightarrow vt \in L_0$

Learner-strategy

S = T = {ε} // S is T-minim

loop

while S NOT T-complete

let s ∈ S and a ∈ Σ such that

∀s'∈S ∃t∈T Membership(s a t) ≠ Mem ership(s' t)

S = S ∪ {s a}

A = DFA with state set S, transitions $\delta(s,a) = s_a$

initial state ε, final states s' s.t. Membership(s')

s_a is the unique word in S such that $s_a \approx_{L_0,T} s\,a$

(S T-complete → s_a exists

S T-minimal → s_a unique)

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal

if $\forall s \neq s' \in S$ $s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete

if $\forall s \in S$ $\forall a \in \Sigma$

$\exists s_a \in S$ $s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

<u>Learner-strategy</u>

S = T = {ε} // S is T-minim

loop

while S NOT T-complete

let s ∈ S and a ∈ Σ such that

∀s'∈S ∃t∈T Membership(s a t) ≠ Mem ership(s' t)

S = S ∪ {s a}

A = DFA with state set S, transitions $\delta(s,a) = s_a$

initial state ε, final states s' s.t. Membership(s')

if Equivalence(A) // this surely happens

return A // when |S| = index($\sim_{L_0}$)

else

let w be the counter-example of equivalence

T = T ∪ {suffixes of w}

s_a is the unique word in S such that $s_a \approx_{L_0,T} s\,a$

(S T-complete → s_a exists

S T-minimal → s_a unique)

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is <u>T-minimal</u>

if $\forall s \neq s' \in S \;\; s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is <u>T-complete</u>

if $\forall s \in S \;\; \forall a \in \Sigma$

$\exists\, s_a \in S \;\; s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $u t \in L_0 \leftrightarrow v t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $u t \in L_0 \leftrightarrow v t \in L_0$

Learner-strategy

```
S = T = {ε}   // S is T-minimal, possibly not T-complete
loop          // this will loop at most index(~L0) times
  while S NOT T-complete
    let s ∈ S and a ∈ Σ such that
        ∀s'∈S ∃t∈T Membership(s a t) ≠ Membership(s' t)
    S = S ∪ {s a}
  A = DFA with state set S, transitions δ(s,a) = sa
      initial state ε, final states s' s.t. Membership(s')
  if Equivalence(A)    // this surely happens
    return A           // when |S| = index(~L0)
  else
    let w be the counter-example of equivalence
    T = T ∪ {suffixes of w}  // S becomes T-incomplete
                             // and will grow at next iteration...
```

Learner constructs automata from pairs (S,T) where $S \subseteq \Sigma^*$ is T-minimal & T-complete

$S \subseteq \Sigma^*$ is T-minimal
if $\forall s \neq s' \in S$ $s \not\approx_{L_0,T} s'$

$S \subseteq \Sigma^*$ is T-complete
if $\forall s \in S$ $\forall a \in \Sigma$
$\exists s_a \in S$ $s_a \approx_{L_0,T} s\,a$

Learning as a killer app of Myhill-Nerode theorem

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; ut \in L_0 \leftrightarrow vt \in L_0$

Learner-strategy

```
S = T = {ε}   // S is T-minimal, possibly not T-complete
loop          // this will loop at most index(~L0) times
  while S NOT T-complete
    let s ∈ S and a ∈ Σ such that
        ∀s'∈S ∃t∈T Membership(s a t) ≠ Membership(s' t)
    S = S ∪ {s a}
  A = DFA with state set S, transitions δ(s,a) = sa
      initial state ε, final states s' s.t. Membership(s')
  if Equivalence(A)   // this surely happens
    return A          // when |S| = index(~L0)
  else
    let w be the counter-example of equivalence
    T = T ∪ {suffixes of w}  // S becomes T-incomplete
                             // and will grow at next iteration...
```

Proof by contradiction:

Assume:

- $w = a_1 \dots a_n$ counter-example
- $t_i = a_{i+1} \dots a_n$ suffixes of w
- s_i state reached by A after reading prefix $a_1 \dots a_i$ of w
- S is $(T \cup \{t_0,\dots,t_n\})$-complete

Verify by induction on i that

$$w \in L_0 \text{ iff } s_i t_i \in L_0$$

Conclude $w \in L_0$ iff $w \in L(A)$

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $u t \in L_0 \leftrightarrow v t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $u t \in L_0 \leftrightarrow v t \in L_0$

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Hankel matrix of L_0:

$H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$

$H(s,t) = \text{Membership}(s\,t)$

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $ut \in L_0 \leftrightarrow vt \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	0	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

Hankel matrix of L_0:

$H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$

$H(s,t) = \text{Membership}(s\,t)$

(infinite, highly redundant, but nicely structured)

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	**0**	**1**	**0**	**0**	**0**	**...**	**0**	**...**
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

Hankel matrix of L_0:

$H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$

$H(s,t) = \text{Membership}(s\,t)$

(infinite, highly redundant, but nicely structured)

a row = a $\sim_{L_0}$-class

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; u\,t \in L_0 \leftrightarrow v\,t \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	**1**	0	...	0	...
a	0	1	0	**0**	0	...	0	...
b	0	1	0	**0**	0	...	0	...
aa	1	0	0	**1**	0	...	0	...
ab	0	0	0	**0**	0	...	0	...
...	...	...	...	**...**	...	...	...	...
aba	0	0	0	**0**	0	...	0	...
...	...	...	...	...	...	...	...	...

Hankel matrix of L_0:

$H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$

$H(s,t) = \text{Membership}(s\,t)$

(infinite, highly redundant, but nicely structured)

a row = a $\sim_{L_0}$-class

a column = a test word t

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	**1**	**0**	0	**1**	0	...	0	...
a	**0**	**1**	0	**0**	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	**1**	**0**	0	**1**	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

Hankel matrix of L_0:

$H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$

$H(s,t) = \text{Membership}(s\,t)$

(infinite, highly redundant, but nicely structured)

a row = a $\sim_{L_0}$-class

a column = a test word t

a submatrix = a pair (S,T)

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \;\; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \;\; ut \in L_0 \leftrightarrow vt \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	0	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; ut \in L_0 \leftrightarrow vt \in L_0$

S = {ε} T = {ε}

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	0	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u t \in L_0 \leftrightarrow v t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u t \in L_0 \leftrightarrow v t \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	0	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; ut \in L_0 \leftrightarrow vt \in L_0$

	ε	a	b	aa	ab	...	aba	...
ε	**1**	0	0	1	0	...	0	...
a	**0**	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

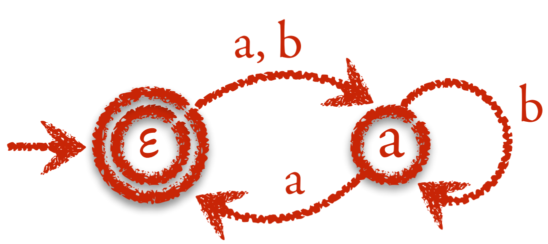
	ε	a	b	aa	ab	...	aba	...
ε	1	0	0	1	0	...	0	...
a	0	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

build candidate automaton...



An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; u t \in L_0 \leftrightarrow v t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; u t \in L_0 \leftrightarrow v t \in L_0$

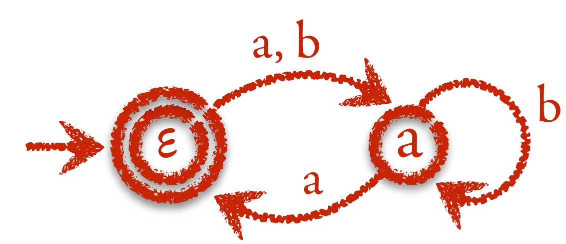
	ε	a	b	aa	ab	...	aba	...
ε	**1**	0	0	1	0	...	0	...
a	**0**	1	0	0	0	...	0	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

build candidate automaton...



expand T by counter-example w = aba

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

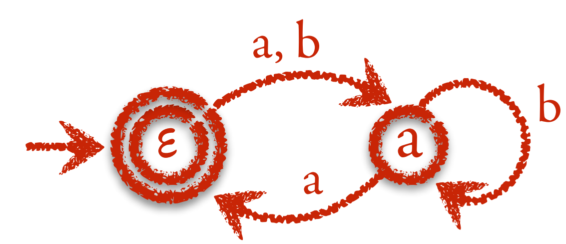
	ε	a	b	aa	ab	...	aba	...
ε	**1**	**0**	0	1	0	...	**0**	...
a	**0**	**1**	0	0	0	...	**0**	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

build candidate automaton...



expand T by counter-example w = aba

S = {ε, a} T = {ε, a, ba, aba}

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $u\,t \in L_0 \leftrightarrow v\,t \in L_0$

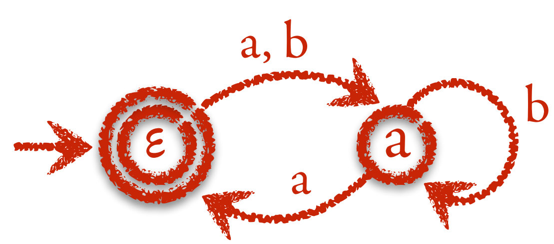
	ε	a	b	aa	ab	...	aba	...
ε	**1**	**0**	0	1	0	...	**0**	...
a	**0**	**1**	0	0	0	...	**0**	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

build candidate automaton...



expand T by counter-example w = aba

S = {ε, a} T = {ε, a, ba, aba}

expand S to make it T-complete...

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^*$ $ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T$ $ut \in L_0 \leftrightarrow vt \in L_0$

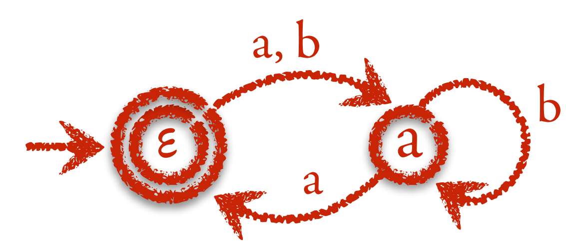
	ε	a	b	aa	ab	...	aba	...
ε	**1**	**0**	0	1	0	...	**0**	...
a	**0**	**1**	0	0	0	...	**0**	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	**0**	**0**	0	0	0	...	**0**	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε} T = {ε}

expand S to make it T-complete...

S = {ε, a} T = {ε}

build candidate automaton...



expand T by counter-example w = aba

S = {ε, a} T = {ε, a, ba, aba}

expand S to make it T-complete...

S = {ε, a, ab} T = {ε, a, ba, aba}

An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \quad u\,t \in L_0 \leftrightarrow v\,t \in L_0$

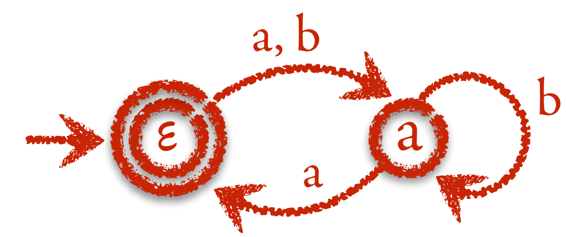
	ε	a	b	aa	ab	…	aba	…
ε	**1**	**0**	0	1	0	…	**0**	…
a	**0**	**1**	0	0	0	…	**0**	…
b	0	1	0	0	0	…	0	…
aa	1	0	0	1	0	…	0	…
ab	**0**	**0**	0	0	0	…	**0**	…
…	…	…	…	…	…	…	…	…
aba	0	0	0	0	0	…	0	…
…	…	…	…	…	…	…	…	…

S = {ε}   T = {ε}

expand S to make it T-complete...

S = {ε, a}   T = {ε}

build candidate automaton...



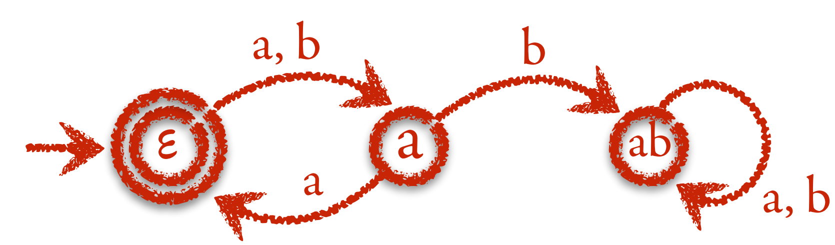
expand T by counter-example w = aba

S = {ε, a}   T = {ε, a, ba, aba}

expand S to make it T-complete...

S = {ε, a, ab}   T = {ε, a, ba, aba}

build candidate automaton...



An alternative view - Hankel matrices

Myhill-Nerode equivalence: $u \sim_{L_0} v$ if $\forall t \in \Sigma^* \; ut \in L_0 \leftrightarrow vt \in L_0$

Relativization to a test set T: $u \approx_{L_0,T} v$ if $\forall t \in T \; ut \in L_0 \leftrightarrow vt \in L_0$

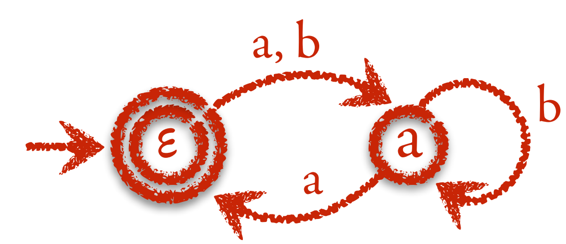
	ε	a	b	aa	ab	...	aba	...
ε	**1**	**0**	0	1	0	...	**0**	...
a	**0**	**1**	0	0	0	...	**0**	...
b	0	1	0	0	0	...	0	...
aa	1	0	0	1	0	...	0	...
ab	**0**	**0**	0	0	0	...	**0**	...
...	...	...	...	...	...	...	...	...
aba	0	0	0	0	0	...	0	...
...	...	...	...	...	...	...	...	...

S = {ε}　　　　T = {ε}

expand S to make it T-complete...

S = {ε, a}　　　　T = {ε}

build candidate automaton...



expand T by counter-example w = aba

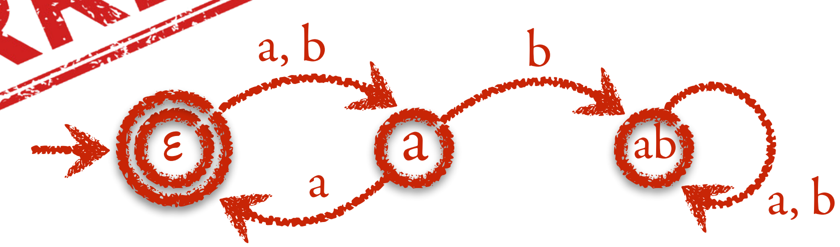
S = {ε, a}　　　　T = {ε, a, ba, aba}

expand S to make it T-complete...

S = {ε, a, ab}　　　　T = {ε, a, ba, aba}

build candidate automaton...

CORRECT



From word languages to word functions

From word languages to word functions

Automata can be used to represent not only languages, but also *functions on words*

$$f: \Sigma^* \to \Gamma^*$$

e.g. f(abaab) = abcaacb

From word languages to word functions

Automata can be used to represent not only languages, but also *functions on words*

$$f: \Sigma^* \to \Gamma^*$$

e.g. f(abaab) = abcaacb

Many variants of automata with outputs. Simplest one is <u>sequential transducer</u>
i.e. $A = (\Sigma, \Gamma, Q, q_0, \delta)$ with $\delta: Q \times \Sigma \to Q \times \Gamma^*$ (e.g. $\delta(q, a)=(q', ac)$)

From word languages to word functions

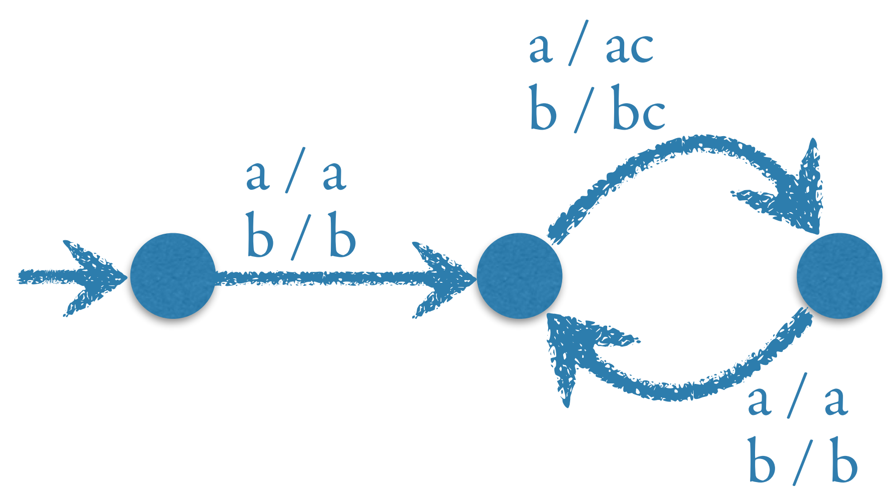
Automata can be used to represent not only languages, but also *functions on words*

$$f:\ \Sigma^* \to \Gamma^*$$

$$\text{e.g.}\quad f(abaab) = abcaacb$$

Many variants of automata with outputs. Simplest one is sequential transducer

i.e. $A = (\Sigma, \Gamma, Q, q_0, \delta)$ with $\delta\colon Q \times \Sigma \to Q \times \Gamma^*$ (e.g. $\delta(q, a)=(q', ac)$)



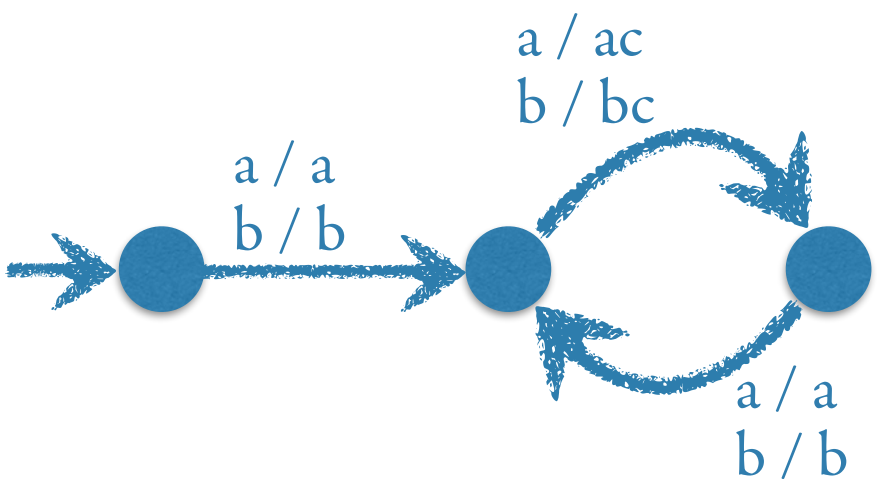
From word languages to word functions

Automata can be used to represent not only languages, but also *functions on words*

$$f: \ \Sigma^* \to \Gamma^*$$

e.g. f(abaab) = abcaacb

Many variants of automata with outputs. Simplest one is <u>sequential transducer</u>
i.e. $A = (\Sigma, \Gamma, Q, q_0, \delta)$ with $\delta: Q \times \Sigma \to Q \times \Gamma^*$ (e.g. $\delta(q, a)=(q', ac)$)



Note: sequential transducers can only compute *total, monotone* functions
(f is <u>monotone</u> if whenever w is prefix of w' then f(w) is prefix of f(w'))

Learning monotone word functions

Learning monotone word functions

1) Teacher has a secret function f_0: $\Sigma^* \to \Gamma^*$, computed by a seq. transducer A_0
Learner initially only knows the input alphabet Σ

Learning monotone word functions

1) Teacher has a secret function f_0: $\Sigma^* \to \Gamma^*$, computed by a seq. transducer A_0
   Learner initially only knows the input alphabet Σ

2) Learner choses a query:
   a) either an evaluation query "What is the value of $f_0(w)$?"
   b) or an equivalence query "Is f_0 computed by seq. transducer A ?"

Learning monotone word functions

1) Teacher has a secret function f_0: $\Sigma^* \rightarrow \Gamma^*$, computed by a seq. transducer A_0
Learner initially only knows the input alphabet Σ

2) Learner choses a query:
   a) either an <u>evaluation query</u> "What is the value of $f_0(w)$?"
   b) or an <u>equivalence query</u> "Is f_0 computed by seq. transducer A ?"

3) Teacher answers accordingly:
   a) gives value of $f_0(w)$
   b) yes if f_0 is computed by A (requires an algorithm for testing equivalence),
   otherwise gives a shortest <u>counter-example</u> w such that $f_0(w) \neq A(w)$

Learning monotone word functions

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^* \quad f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^* \; f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Example

f_0 inserts c between every two positions

$f_0(\varepsilon) = \varepsilon$, $f_0(a) = a$,
$f_0(aa) = aac$, $f_0(aaa) = aaca$, ...

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^* \;\; f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Example

f_0 inserts c between every two positions

$f_0(\varepsilon) = \varepsilon$, $f_0(a) = a$,
$f_0(aa) = aac$, $f_0(aaa) = aaca$, ...

$\sim_{f_0}$ has only two equivalence classes:
$[\varepsilon]_{\sim_{f_0}}$ $[a]_{\sim_{f_0}}$

E.g. $a \sim_{f_0} bba$ because $f(a\,???...) - f(a) = a\,?c??c... - a$
$f(bba\,???...) - f(bba) = bbca\,?c??c... - bbca$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Example

f_0 inserts c between every two positions

$f_0(\varepsilon) = \varepsilon$, $f_0(a) = a$,
$f_0(aa) = aac$, $f_0(aaa) = aaca$, ...

$\sim_{f_0}$ has only two equivalence classes:
$[\varepsilon]_{\sim_{f_0}}$ $[a]_{\sim_{f_0}}$

E.g. $a \sim_{f_0} bba$ because $f(a\,???...) - f(a) = a\,?c??c... - a$
$f(bba\,???...) - f(bba) = bbca\,?c??c... - bbca$

Properties:

- $\sim_{f_0}$ is right-invariant (i.e. $u \sim_{f_0} v \rightarrow u\,a \sim_{f_0} v\,a$)

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Example

f_0 inserts c between every two positions

$f_0(\varepsilon) = \varepsilon$, $f_0(a) = a$,
$f_0(aa) = aac$, $f_0(aaa) = aaca$, ...

$\sim_{f_0}$ has only two equivalence classes:
$[\varepsilon]_{\sim_{f_0}}$ $[a]_{\sim_{f_0}}$

Properties:

- $\sim_{f_0}$ is right-invariant (i.e. $u \sim_{f_0} v \to u\,a \sim_{f_0} v\,a$)
- $\sim_{f_0}$ has finite index iff f is computed by a seq. transducer...

E.g. $a \sim_{f_0} bba$ because f(a ???...) - f(a) = a ?c??c... - a
f(bba ???...) - f(bba) = bbca ?c??c... - bbca

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall\, t \in \Sigma^* \quad f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t)$ - $f_0(u) = f_0(v\,t)$ - $f_0(v)$

Hankel matrix of f_0:

$H \in (\Gamma^*)^{\Sigma^* \times \Sigma^*}$

$H(s,t) = f_0(s\,t) - f_0(s)$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	ε	a	b	aac	abc	...	aaca	...
a	ε	ac	bc	aca	acb	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

Hankel matrix of f_0:

$H \in (\Gamma^*)^{\Sigma^* \times \Sigma^*}$

$H(s,t) = f_0(s\,t) - f_0(s)$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	ε	a	b	aac	abc	…	aaca	…
a	ε	ac	bc	aca	acb	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

Hankel matrix of f_0:

$H \in (\Gamma^*)^{\Sigma^* \times \Sigma^*}$

$H(s,t) = f_0(s\,t) - f_0(s)$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	ε	a	b	aac	abc	...	aaca	...
a	ε	ac	bc	aca	acb	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

Hankel matrix of f_0:

$H \in (\Gamma^*)^{\Sigma^* \times \Sigma^*}$

$H(s,t) = f_0(s\,t) - f_0(s)$

Learner maintains (S,T)

S T-minimal if $\forall s \neq s' \in S$
$s \not\approx_{f_0,T} s'$

S T-complete if $\forall s \in S\ \forall a \in \Sigma$
$\exists s_a \in S \quad s_a \approx_{f_0,T} s\,a$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	ε	a	b	aac	abc	...	aaca	...
a	ε	ac	bc	aca	acb	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

Learner maintains (S,T)

S T-minimal if $\forall s \neq s' \in S$ $s \not\approx_{f_0,T} s'$

S T-complete if $\forall s \in S\ \forall a \in \Sigma$ $\exists s_a \in S$ $s_a \approx_{f_0,T} s\,a$

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	ε	a	b	aac	abc	…	aaca	…
a	ε	ac	bc	aca	acb	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

Learner maintains (S,T)

S T-minimal if $\forall s \neq s' \in S$
$s \not\approx_{f_0,T} s'$

S T-complete if $\forall s \in S\ \forall a \in \Sigma$
$\exists s_a \in S \quad s_a \approx_{f_0,T} s\,a$

start with

S = {ε} T = {ε}

Learning monotone word functions

Myhill-Nerode <u>equivalence</u>: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	ε	a	b	aac	abc	…	aaca	…
a	ε	ac	bc	aca	acb	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

Learner maintains (S,T)

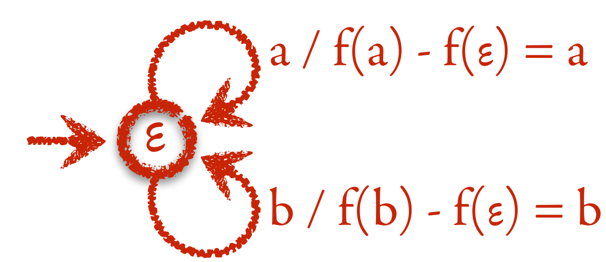
S <u>T-minimal</u> if $\forall s \neq s' \in S$ $s \not\approx_{f_0,T} s'$

S <u>T-complete</u> if $\forall s \in S\ \forall a \in \Sigma$ $\exists s_a \in S$ $s_a \approx_{f_0,T} s\,a$

start with

S = {ε} T = {ε}

build candidate transducer…



Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

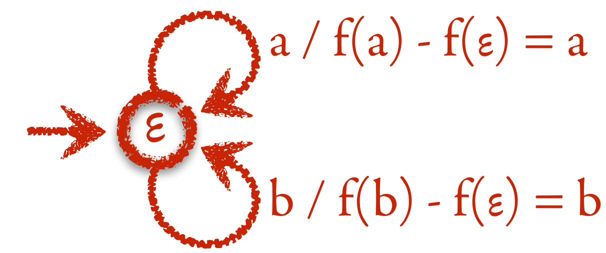
Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	ε	a	b	aac	abc	...	aaca	...
a	ε	ac	bc	aca	acb	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

start with

S = {ε} T = {ε}

build candidate transducer...



Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

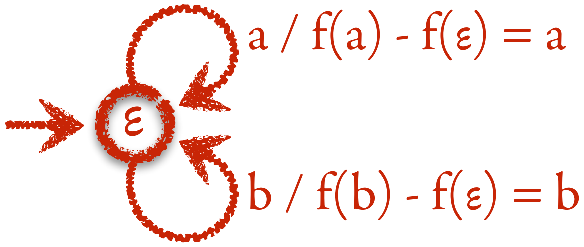
Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	ε	a	b	aac	abc	...	aaca	...
a	ε	ac	bc	aca	acb	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

start with

S = {ε} T = {ε}

build candidate transducer...



expand T by counter-example w = ab

Learning monotone word functions

Myhill-Nerode <u>equivalence</u>: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

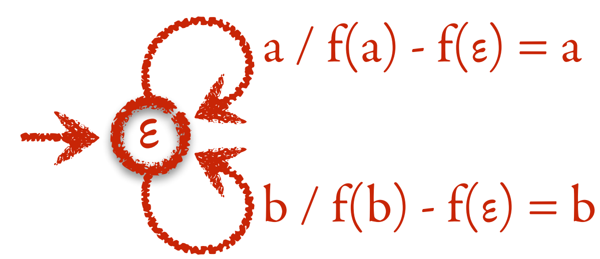
Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	**ε**	a	**b**	aac	**abc**	…	aaca	…
a	ε	ac	bc	aca	acb	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

start with

S = {ε} T = {ε}

build candidate transducer…



expand T by counter-example w = ab

S = {ε} T = {ε, b, ab}

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^* \quad f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

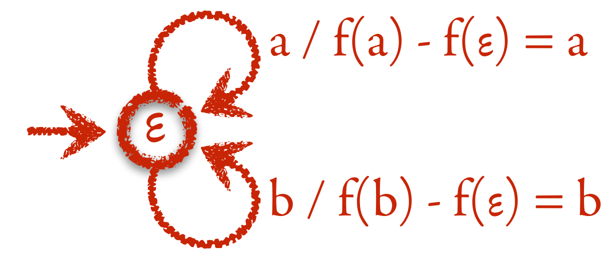
Relativization to test set T: $u \approx_{f_0, T} v$ if $\forall t \in T \quad f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	**ε**	a	**b**	aac	**abc**	…	aaca	…
a	ε	ac	bc	aca	acb	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

start with

S = {ε} T = {ε}

build candidate transducer…



expand T by counter-example w = ab

S = {ε} T = {ε, b, ab}

expand S to make it T-complete…

Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

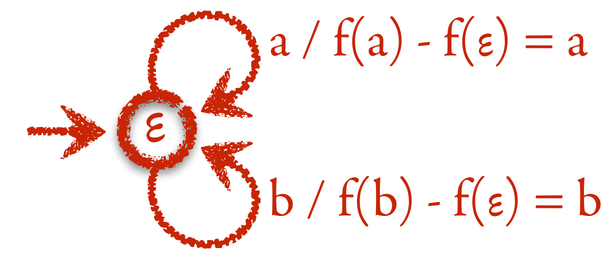
Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	**ε**	a	**b**	aac	**abc**	…	aaca	…
a	**ε**	ac	**bc**	aca	**acb**	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

start with

S = {ε} T = {ε}

build candidate transducer…



expand T by counter-example w = ab

S = {ε} T = {ε, b, ab}

expand S to make it T-complete…

S = {ε, a} T = {ε, b, ab}

Learning monotone word functions

Myhill-Nerode <u>equivalence</u>: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

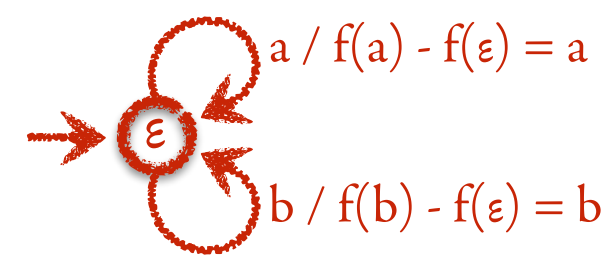
Relativization to test set T: $u \approx_{f_0, T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	…	aaa	…
ε	**ε**	a	**b**	aac	**abc**	…	aaca	…
a	**ε**	ac	**bc**	aca	**acb**	…	acaac	…
b	ε	ac	bc	aca	acb	…	acaac	…
aa	ε	a	b	aac	abc	…	aaca	…
ab	ε	a	b	aac	abc	…	aaca	…
…	…	…	…	…	…	…	…	…
aaa	ε	ac	bc	aca	acb	…	acaac	…
…	…	…	…	…	…	…	…	…

start with

S = {ε} T = {ε}

build candidate transducer…



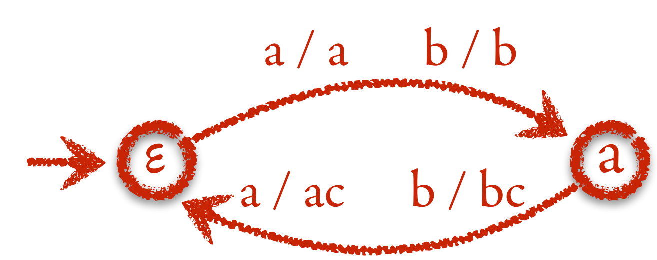
expand T by counter-example w = ab

S = {ε} T = {ε, b, ab}

expand S to make it T-complete…

S = {ε, a} T = {ε, b, ab}

build candidate transducer…



Learning monotone word functions

Myhill-Nerode equivalence: $u \sim_{f_0} v$ if $\forall t \in \Sigma^*$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

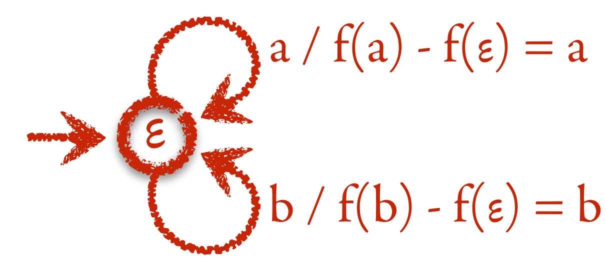
Relativization to test set T: $u \approx_{f_0,T} v$ if $\forall t \in T$ $f_0(u\,t) - f_0(u) = f_0(v\,t) - f_0(v)$

	ε	a	b	aa	ab	...	aaa	...
ε	**ε**	a	**b**	aac	**abc**	...	aaca	...
a	**ε**	ac	**bc**	aca	**acb**	...	acaac	...
b	ε	ac	bc	aca	acb	...	acaac	...
aa	ε	a	b	aac	abc	...	aaca	...
ab	ε	a	b	aac	abc	...	aaca	...
...	...	...	...	...	...	...	...	...
aaa	ε	ac	bc	aca	acb	...	acaac	...
...	...	...	...	...	...	...	...	...

start with

S = {ε} T = {ε}

build candidate transducer...



expand T by counter-example w = ab

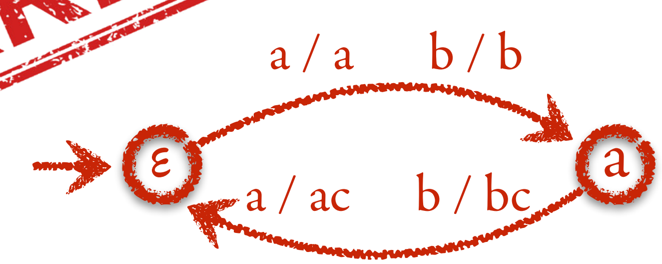
S = {ε} T = {ε, b, ab}

expand S to make it T-complete...

S = {ε, a} T = {ε, b, ab}

CORRECT

build candidate transducer...



From word languages to word functions… and beyond

This learning technique with Hankel matrices has been successfully applied to:

- Weighted automata (i.e. outputs given by products of weights along transitions) and, partially, to probabilistic automata
- Büchi automata
- Tree transducers (e.g. for learning XSLT transformations of XML documents)
- Timed & register automata (e.g. for processing strings with timestamps/data)

A simple yet useful real application (surprisingly, not yet done :/)

Implementing a learning algorithm for automata/transducers

would allow to automatically derive *RegEx* expressions like

```
/^(0?[1-9]|[12][0-9]|3[01])([ \/\-])(0?[1-9]|1[012])
\2([0-9][0-9][0-9][0-9])(([ -])([0-1]?[0-9]|2[0-3]):
[0-5]?[0-9]:[0-5]?[0-9])?$/
```

from positive and negative examples examples like

01/01/2000	ab/01/2000
18/10/1985	1/01/2000
…	18/10/200x
	…