AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald
AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald
Automata over Infinite Data Domains

- Model infinite-state system using a finite model

```java
1: while (true)
2:   pass = readInput;
3:   while (pass \leq 999)
4:     pass = readInput;
5:   pass2 = encrypt(pass);
```
AUTOMATA OVER INFINITE DATA DOMAINS: **LEARNABILITY** AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald
Learnability

1: while (true)
2: pass = readInput;
3: while (pass ≤ 999)
4: pass = readInput;
5: pass2 = encrypt(pass);

Learning symbolic automata (conditions for learning: L* and identification in the limit)

[Fisman, Frenkel, Zilles]
Learnability

Adapting L* algorithm for communicating programs

1: while (true)
2:     pass = readInput;
3:     while (pass ≤ 999)
4:         pass = readInput;
5:     pass2 = encrypt(pass);

Learning symbolic automata (conditions for learning: L* and identification in the limit)

[Frenkel, Grumberg, Pasareanu, Sheinvald 20]

[Fisman, Frenkel, Zilles]
AUTOMATA OVER INFINITE DATA DOMAINS: LEARNABILITY AND APPLICATIONS IN PROGRAM VERIFICATION AND REPAIR

Hadar Frenkel

Advisors: Orna Grumberg & Sarai Sheinvald
Applications in Program Verification and Repair

Bounded model-checking algorithm

1: while (true)
2: pass = readInput;
3: while (pass ≤ 999)
4: pass = readInput;
5: pass2 = encrypt(pass);
Applications in Program Verification and Repair

Compositional verification and repair algorithm

1: while (true)
2:   pass = readInput;
3:   while (pass ≤ 999)
4:     pass = readInput;
5:   pass2 = encrypt(pass);

Bounded model-checking algorithm

[Frenkel, Grumberg, Sheinvald 17, 19]

[Frenkel, Grumberg, Pasareanu, Sheinvald 20]

[Fisman, Frenkel, Zilles]
MODEL CHECKING SYSTEMS OVER INFINITE DATA

Joint work with Orna Grumberg and Sarai Sheinvald

@NFM 2017, @Journal of automated reasoning 2019
Goal

• Develop a Model checking process for systems over infinite data domains
• Using the automata-theoretic approach
Model checking

- System
- Specification
- YES!
- NO! + counter example
Model checking

system

YES!

specification

Given as an LTL formula

NO! + counter example
Verification of Systems over Infinite Data Domains
Verification of Systems over Infinite Data Domains

• LTL cannot express the property “every client is eventually active”
Verification of Systems over Infinite Data Domains

- LTL cannot express the property “every client is eventually active”

Variable LTL (VLTL) [GKS12]
- $\forall x: F active. x$
- $AP$ - finite set of (parameterized) propositions
- $V$ - finite set of quantified variables
$\exists^*\text{VLTL}$ [GKS12]

- VLTL with only existential quantifiers
- $G \exists x: \text{send}. \ x$
- A possible satisfying computation

- We are interested in verifying universal properties, the negation that describes a bad behavior is existential
Model Checking - Infinite Data Domains

- Program automaton
- Emptiness test
- $\exists^* \text{VLTL formula}$
- Formula automaton
Model Checking - Infinite Data Domains

- Reactive systems, Automata over infinite words
- Alternating variable Büchi automaton
- Existential \(\exists^*\) VLTL formula
- Program automaton
- Emptiness test
- Non-Deterministic variable Büchi automaton
Model Checking - Infinite Data Domains

- Program automaton
- Emptiness test
- Easy emptiness test
- Alternating variable Büchi automaton
- Non-Det variable Büchi automaton
- ∃*VLTL formula
- natural translation for LTL formulas
Non-Deterministic Variable Büchi Automata (NVBW) [GKS13]

- $G \exists x: \text{send}.x$
- Alphabet is parameterized propositions
- Ability to reset a variable and to assign it a new value
- As long as there is no reset - the value cannot be changed

Useful for emptiness test
NWBV Cannot Express all $\exists^*\text{VLTL}$

- $G (\exists x: \text{send}. x \land XF \text{receive}. x)$

- Increasing gaps between $\text{send}. x, \text{receive}. x$.
- Not enough variables and states to remember all values
Alternating Variable Büchi Automata (AVBW)

• $G (\exists x: \text{send.}x \land XF \text{receive.}x)$
Alternating Variable Büchi Automata (AVBW)

- $G (\exists x: \text{send}.x \land XF \text{receive}.x)$

Easy construction from $\exists^* \text{VLTL}$
Alternating Variable Büchi Automata (AVBW)

- $G (\exists x: \text{send}. x \land XF \text{receive}. x)$
Alternating Variable Büchi Automata (AVBW)

- $G (\exists x: \text{send}. \ x \land XF \text{receive}. \ x)$

![Diagram of AVBW Automaton]

- $q_0$ (reset($x$))
- $q_1$
- $\text{true}$

- Transitions:
  - send.x: $q_0 \rightarrow q_1$
  - send.1: $q_0 \rightarrow q_0$
  - send.2: $q_0 \rightarrow q_1$
  - rec.x: $q_1 \rightarrow q_1$
  - $x = 1$: $q_1 \rightarrow q_0$
  - $x = 2$: $q_1 \rightarrow q_0$

- Initial state: $q_0$
- Final states: $q_0$
Alternating Variable Büchi Automata (AVBW)

- $G (\exists x: \text{send. } x \land XF \text{ receive. } x)$
VLTL to AVBWs

- Similar to [V95]
- Special care of resets
- \( X = \text{vars} (\varphi) \cup \{ x_p | p \in \text{AP} \} \)
- \( Q = \text{sub} (\varphi) \)
- Reset
  - \( x_p \) varaibles
  - variables under \( \exists \)
- \( x \neq y \) for \( \neg a.x \in \text{sub} (\varphi) \)

- \( \delta (a.x, A) = \text{true} \) if \( a.x \in A \) and \( \delta (a.x, A) = \text{false} \), otherwise.
- \( \delta (\neg a.x, A) = \neg \delta (a.x, A).^4 \)
- \( \delta (\eta \land \psi, A) = \delta (\eta, A) \land \delta (\psi, A). \)
- \( \delta (\eta \lor \psi, A) = \delta (\eta, A) \lor \delta (\psi, A) \)
- \( \delta (\varnothing, A) = \text{true} \)
- \( \delta (\eta \cup \psi, A) = \delta (\psi, A) \lor (\delta (\eta, A) \land \eta \cup \psi) \)
- \( \delta (\eta \lor \psi, A) = \delta (\eta \land \psi, A) \lor (\delta (\psi, A) \land \eta \lor \psi) \)
- \( \delta (\exists x \eta, A) = \delta (\eta, A) \)
Model Checking - Infinite Data Domains

∃*VLTL formula

Alternating variable Büchi automaton
Model Checking - Infinite Data Domains

Unlike the finite alphabet case!
Model Checking - Infinite Data Domains

∃*VLTL formula

Program automaton

Emptiness test

 Alternating variable Büchi automaton

Non-Det variable Büchi automaton
Model Checking - Infinite Data Domains

- Emptiness of AVBW is **undecidable**
- Satisfiability problem of $\exists^*\text{VLTL}$ formulas is undecidable [SW14]
- $\exists^*\text{VLTL} \equiv \text{AVBW}$, thus
  - Satisfiability problem $\equiv$ emptiness problem

 Alternating variable
 Büchi automaton

 Emptiness test
Solutions

Model checking $\exists^* \text{VLTL}$

- "easy fragments"
- Reduction to an easy fragment
- Partial translation algorithm AVBW$\rightarrow$NVBW
- Bounded model checking
∃*VLTL Formulas with a Direct Construction to NVBW

• PNF formulas  $\exists x: G\, send\, .\, x\quad (send.\, 7)\omega$
• $X, F$ formulas
• Quantifiers are at the beginning \ next to atomic propositions  $\exists x_1: G\, send\, .\, x_1 \land G\, \exists x_2: rec\, .\, x_2$

“easy fragments”
Flattening

• A formula with no negations has an equisatisfiable formula in PNF

\[ G (\exists x: \text{send}.x \land XF \text{receive}.x) \]

Always holds  \[ \exists x: G (\text{send}.x \land XF \text{receive}.x) \]  No negations

Reduction to an easy fragment
Translation Algorithm

- A partial algorithm for translation
- Based on the Miyano-Hayashi construction [MH84]

AND

- Take care of variables, resets
- Map variables of alternating automaton to variables of non-deterministic automaton

\[
\begin{align*}
(q_0, \emptyset) \\
(q_1, x \rightarrow z_1), \{(q_1, x \rightarrow z_1)\} \\
(q_1, x \rightarrow z_3) \\
\text{reset}(z_2)
\end{align*}
\]
Alternating to Non-Deterministic \cite{MH84}

- $G$ (send $\rightarrow$ XF receive)

\[
\begin{array}{c}
q_0 \\
\downarrow \\
q_1 \\
\downarrow \\
\text{true}
\end{array}
\begin{array}{c}
send.x \\
\downarrow \\
rec.x
\end{array}
\quad
\begin{array}{c}
q_0 \\
\downarrow \\
q_1 \\
\downarrow \\
\text{true}
\end{array}
\begin{array}{c}
q_0 \\
\downarrow \\
q_0 \\
\downarrow \\
q_1
\end{array}
\begin{array}{c}
q_0 \\
\downarrow \\
q_1 \\
\downarrow \\
q_1
\end{array}
\]
AVBW to NVBW

- $G \exists x: a. x \land XX b. x$

\[
\begin{align*}
\{ (q_0, \emptyset) \} & \xrightarrow{a.z_1} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1) \} \\
\{ (q_1, x \rightarrow z_1) \} & \xrightarrow{a.z_2} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_1) \} \\
& \xrightarrow{a.z_3, b.z_1} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_2) \} \\
& \xrightarrow{r.e.s.e.t.(z_3)} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_3) \} \\
\& \xrightarrow{r.e.s.e.t.(z_1)} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_3) \} \\
& \xrightarrow{r.e.s.e.t.(z_2)} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_3) \} \\
& \xrightarrow{r.e.s.e.t.(z_1)} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_3) \} \\
& \xrightarrow{r.e.s.e.t.(z_2)} \{ (q_0, \emptyset), (q_1, x \rightarrow z_1), (q_2, x \rightarrow z_3) \} \\
\end{align*}
\]
Incompleteness

- The empty language
- Our algorithm does not halt
BMC Algorithm

- Based on the translation algorithm
- We are looking for a *witness* to non-emptiness
- Test emptiness with a partial NVBW
- Might find “more interesting” witnesses as the algorithm continues
VLTL Summary

• Using alternating variable automata to model VLTL properties
• Translation algorithm from AVBWs to NVBWs
• Bounded model-checking procedure for $\exists^*\text{VLTL}$
• Easy fragments for model-checking
COMPOSITIONAL VERIFICATION AND REPAIR

Joint work with Orna Grumberg, Corina Pasareanu, and Sarai Sheinvald

@TACAS 2020
Model Checking

YES!

Repair!
Model Checking

Number of states in the system model grows exponentially with the number of components in the system.

YES!  NO!  +

Repair!

specification

component

component

component

component
Model Checking

Component

YES!

NO!

Counter example

Component

Number of states in the system model grows exponentially with the number of components in the system.

State Explosion Problem

Repair!
COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

- Model checking and repair algorithm for communicating systems
- Exploit the partition of the system into components
Setting – Communicating Systems

Assume-Guarantee (AG)

\[ M_1 || A \models P \]
\[ M_2 \models A \]
\[ M_1 || M_2 \models P \]

AG rule & Automata Learning

Repair & Results
Communicating Systems

- C-like programs
- Each component is described as a control-flow graph (automaton)
  - Alphabet: program statements & communication channels
- $In?x_1$ – reads a value to $x_1$ through channel $In$
- $enc!x_1$ – sends the value of $x_1$ through channel $enc$

1: while (true)  
2:    pass = readInput;  
3:    while (pass $\leq$ 999)  
4:      pass = readInput;  
5:    pass2 = encrypt(pass);
Example

Synchronization using read-write channels, Interleaving on all other alphabet
Example

Synchronization using read-write channels, Interleaving on all other alphabet

\[ M_1 \]

\[ M_2 \]
Example

Synchronization using read-write channels, Interleaving on all other alphabet
Example

Synchronization using read-write channels, Interleaving on all other alphabet
Example

State Explosion Problem
Specifications

- Safety properties
- Alphabet:
- (Common) communication channels
- Syntactic requirements: program behavior through time
Specifications

- Safety properties
- Alphabet:
- (Common) communication channels
- Syntactic requirements: program behavior through time
- Constraints over local variables
- Semantic requirements:
  - “the entered password is different from the encrypted password”
  - “there is no overflow”
Reasoning About the Smaller Components

Setting – Communicating Systems

Assume-Guarantee (AG)

\[ M_1 \parallel A \models P \]
\[ M_2 \models A \]
\[ M_1 \parallel M_2 \models P \]

AG rule & Automata Learning

Repair & Results
Compositional Verification

• Inputs:
  • composite system $M_1 \parallel M_2$
  • property $P$

• Goal: check if $M_1 \parallel M_2 \models P$

• First attempt: “divide and conquer”
  • Problem: usually impossible to verify each component separately
  • Components are designed to satisfy requirements in specific contexts
Compositional Verification

• **Assume-Guarantee** (AG) paradigm [Pnueli, 1985]:
  • assumptions represent component’s environment

• Under assumption $A$ on its environment, does the component guarantee the property?

\[
M_1 || A \models P = M_2
\]
AG Rule for Safety Properties

1. check if a component $M_1$ guarantees $P$ when it is a part of a system satisfying assumption $A$
AG Rule for Safety Properties

1. check if a component $M_1$ guarantees $P$ when it is a part of a system satisfying assumption $A$

2. discharge assumption: show that the remaining component $M_2$ satisfies $A$

$M_1 || A \models P$

$M_2 \models A$
AG Rule for Safety Properties

1. check if a component $M_1$ guarantees $P$ when it is a part of a system satisfying assumption $A$

2. discharge assumption: show that the remaining component $M_2$ satisfies $A$

3. Conclude that $M_1 || M_2 \models P$

$M_1 || A \models P$

$M_2 \models A$
AG Rule for Safety Properties

1. check if a component $M_1$ guarantees $P$ when it is a part of a system satisfying assumption $A$

2. discharge assumption: show that the remaining component $M_2$ satisfies $A$

3. Conclude that $M_1 || M_2 \models P$

Can we automatically construct $A$?
Automatic Assumption Generation

Setting – Communicating Systems

Assume-Guarantee (AG)

$M_1||A \models P$

$M_2 \models A$

$M_1||M_2 \models P$

AG rule & Automata Learning

Repair & Results

$\frac{\text{repair size}}{\text{assumption size}}$

$\frac{\text{repair size}}{\text{assumption size}}$

$\frac{\text{repair size}}{\text{assumption size}}$

$\frac{\text{repair size}}{\text{assumption size}}$
L* Algorithm for Learning Regular Languages [Angluin87]

- Learning assumptions for compositional verification [CGP03]
- Given a regular language $L$, we learn a DFA $A$ such that $\mathcal{L}(A) = L$
L* Algorithm for Learning Regular Languages [Angluin87]

- Learning assumptions for compositional verification [CGP03]
- Given a regular language $L$, we learn a DFA $A$ such that $\mathcal{L}(A) = L$
- Membership queries
**L* Algorithm for Learning Regular Languages** [Angluin87]

- Learning assumptions for compositional verification [CGP03]
- Given a regular language $L$, we learn a DFA $A$ such that $\mathcal{L}(A) = L$
- Equivalence queries, for a candidate $A_i$

**Learner**

Is $\mathcal{L}(A_i) = L$?

**Teacher**

Yes – Done!

No $+$

cex $\in \mathcal{L}(A_i) \Delta L$
L* Algorithm for Learning Regular Languages [Angluin87]

- Learning assumptions for compositional verification [CGPo3]
- Given a regular language $L$, we learn a DFA $A$ such that $\mathcal{L}(A) = L$
- Equivalence queries, for a candidate $A_i$
- Try to use intermediate candidates $A_i$ as assumptions for AG rule
- But, the weakest assumption is not regular in our case

\[
M_1 \parallel A_i \models P \\
M_2 \models A_i \\
M_1 \parallel M_2 \models P
\]
Weakest Assumption is not always regular

• By a way of contradiction

• $A_w$ is over $\alpha M_2 = \{x := 0, \ y := 0, \ x := x + 1, \ y := y + 1, \ sync\}$

• Consider $L = \{x := 0\} \cdot \{y := 0\} \cdot \{x := x + 1, y := y + 1\}^* \cdot \{sync\}$
A New Goal for Learning

• The teacher answers queries according to the *syntactic language* of $M_2$
• Regular since it is given as an automaton
A New Goal for Learning

- The teacher answers queries according to the *syntactic language* of $M_2$
- Regular since it is given as an automaton

But I already know $M_2$ ...

You might find a much smaller assumption!

$M_1 || M_2 \models P$

$M_2 \models M_2$

$M_1 || M_2 \models P$
Membership Queries - $T(M_2)$

\[ w \notin T(M_2) \]
\[ \text{NO!} \]

\[ w \in T(M_2) \land M_1 \models w \models P \]
\[ \text{YES!} \]

\[ w \notin T(M_2) \land M_1 \models w \not\models P \]
\[ w \text{ is a real cex!} \]

$M_1 || A \models P$

$M_2 \models A$

$M_1 || M_2 \models P$
Equivalence Queries - $T(M_2)$

- $M_2 \subseteq A_i$
- $M_1 \parallel A_i \models P$
- $M_2 \models A$
- $M_1 \parallel M_2 \models P$

Teacher

- $M_1 \parallel A_i \models P$
- $t \in A_i: M_1 \models t \not\models P$

Learner

- Is $L(A_i) = L$?

$M_2 \not\subseteq A_i$

- $t \in M_2$
  - $t$ is a real cex!

- $t \not\in M_2$
  - NO!
  - take $t$ off $A_{i+1}$

- $t \in M_2 \setminus A_i$
  - NO!
  - add it to $A_{i+1}$
AG rule with learning

Model Checking

1. $A_i \parallel M_1 \models P$

2. $M_2 \subseteq A_i$

Automata Learning $L^*$
AG rule with learning

\[ A_i \models M_1 \models P \]

1. \( A_i \models M_1 \models P \)
2. \( M_2 \subseteq A_i \)

P is violated in \( M_1 \parallel M_2 \)
AG rule with learning

Automata Learning $L^*$

$A_i$

Model Checking

1. $A_i \parallel M_1 \models P$

2. $M_2 \subseteq A_i$

false

real error? $cex \in M_2$?

P is violated in $M_1 \parallel M_2$
AG rule with learning

Automata Learning $L^*$

Model Checking

1. $A_i \parallel M_1 \models P$

2. $M_2 \subseteq A_i$

strengthen assumption

real error? $\text{cex} \in M_2$?

false

No

Yes

P is violated in $M_1 \parallel M_2$
AG rule with learning

Automata Learning $L^*$

$A_i$

Model Checking

1. $A_i || M_1 \models P$

2. $M_2 \subseteq A_i$

strengthen assumption

false

real error? $cex \in M_2$?

No

Yes

$P$ is violated in $M_1 || M_2$
AG rule with learning

Automata Learning $L^*$

$A_i$\

Model Checking

1. $A_i \parallel M_1 \models P$
2. $M_2 \subseteq A_i$

true\

false

strenthen assumption

No

real error? $cex \in M_2$?

Yes

P is violated in $M_1 \parallel M_2$

P holds in $M_1 \parallel M_2$
AG rule with learning

Automata Learning \( L^* \) \( \rightarrow \) \( A_i \)

Model Checking

1. \( A_i \parallel M_1 \models P \)

2. \( M_2 \subseteq A_i \)

strenthen assumption

\( M_2 \subseteq A_i \) ?

real error? \( \text{cex} \in M_2 \) ?

false

true

false

true

yes

P holds in \( M_1 \parallel M_2 \)

no

Yes

P is violated in \( M_1 \parallel M_2 \)

weaken assumption
Repair

Setting – Communicating Systems

Assume-Guarantee (AG)

\[ M_1 \parallel A \models P \]
\[ M_2 \models A \]
\[ M_1 \parallel M_2 \models P \]

AG rule & Automata Learning

Repair & Results
AG rule with learning

Automata Learning $L^*$ \rightarrow Automata Learning $A_i$

Model Checking

1. $A_i \parallel M_1 \models P$

- true

2. $M_2 \subseteq A_i$

- false

- true

- false

strengthen assumption

false

true

true

real error? cex$\in M_2$?

false

true

P holds in $M_1 \parallel M_2$

Return to verification with the repaired $M_2$

P is violated in $M_1 \parallel M_2$

false

true

false

true

false

true

Repair $M_2$

82
Assume Guarantee or Repair

• Repair by elimination of error traces

• Two types of repair
  • Syntactic repair
  • Semantic repair
Assume Guarantee or Repair

Syntactic repair – counterexample does not contain constraints
Syntactic Repair

• Implemented 3 methods to removing the trace $t$:
  • **Exact**
    remove exactly $t$ from $M_2$
  • **Approximate**
    add an intermediate state and use it to direct some traces off the accepting state, including $t$
  • **Aggressive**
    make the accepting state that $t$ reaches not-accepting
Assume Guarantee or Repair

Semantic repair – counterexample contains violated constraints of the specification
Semantic Repair

• AGR returns a counterexample $t$, for input $x_1 = 2^{63}$

• Goal: make $t$ infeasible by adding a new constraint $C$ such that
  • $(\varphi_t \land C \rightarrow false)$

• Applying abduction, quantifier elimination and simplification results in $C = (x_1 < 2^{63})$
while (true)

pass = readInput;

while (pass ≤ 999)

pass = readInput;

pass2 = encrypt(pass);

assume pass<2^{63};
AG rule with learning

Again, where $M_2 := \text{Repaired } M_2$

Return to verification with the repaired $M_2$.
Termination

- In case $M_1 || M_2 \models P$
- $M_2$ is a correct assumption for the AG rule
- $M_2$ is regular, therefore $L^*$ terminates
  → In the case of verification, termination is guaranteed

- In case $M_1 || M_2 \not\models P$
- Every iteration with an erroneous $M_2$ will result in a cex
  → In the case of an error, progress is guaranteed
Correctness and Termination

• Correctness of Repair
• All questions relate to language containment
• Repair only eliminates traces

• Incremental
• Previous answers to the learner’s questions are still correct
• Can use the same table for $L^*$
Comparing Repair Methods (logarithmic scale)

#15, #16, #18, #19 apply also abduction
AGR Summary

• Modular verification for communicating systems

• Adjusting automata learning to systems with data

• Iterative and incremental verification and repair to prove correctness of repaired system
LEARNING SYMBOLIC AUTOMATA

Joint work with Dana Fisman and Sandra Zilles
Symbolic Finite-State Automata (SFAs)

• Finite state automata
• Defined with respect to a Boolean algebra
• The transition relation is over predicates from the Boolean algebra
Monotonic Algebras

- Predicates correspond to a total order over the domain elements
- $[\psi] = \{ d \mid a \leq d \leq b \}$
- Interval algebra over $\mathbb{N}, \mathbb{Z}, \mathbb{R}$
Identification in the Limit using polynomial time and data \[ G_{78}, dlH_{97} \]

- Passive learning (vs. active learning in L*)
Identification in the Limit using polynomial time and data \[ [G78, dH97] \]

- Passive learning (vs. active learning in L*)
- Given a set \( S \) of labeled words, build an automaton that agrees with \( S \)
Identification in the Limit using polynomial time and data \cite{G78, dH97}

- Given an automaton $A$, **build a characteristic sample $S$**
Identification in the Limit using polynomial time and data \[\text{[G78, dH97]}\]

- Given an automaton \(A\), build a characteristic sample \(S\)
- For every sample \(S' \supseteq S\) that agrees with \(A\), infer an equivalent automaton to \(A\) in polynomial time
Identification in the Limit for DFAs \cite{OG92}

- Constructing a characteristic sample
- Every state is represented by an access word
Identification in the Limit for DFAs \cite{OG92}

- Constructing a characteristic sample
- Every state is represented by an access word
Identification in the Limit for DFAs [OG92]

- Constructing a characteristic sample
- Every state is represented by an access word

\[ q_0, q_1, a, b, \epsilon, 0, 1, \langle \epsilon, 0 \rangle, \langle a, 1 \rangle \]
Identification in the Limit for DFAs [OG92]

- Constructing a characteristic sample
- Distinctive suffixes between states:
  - If $\delta(q_0, w) \neq \delta(q_0, u)$
  - there exists a suffix $z$ such that $w \cdot z \in L(A)$, $u \cdot z \notin L(A)$
  - Add $w \cdot z, u \cdot z$
Identification in the Limit for DFAs \cite{OG92}

- Constructing a characteristic sample
- Representing the transition relation
Identification in the Limit for DFAs [OG92]

- Constructing a characteristic sample
- Representing the transition relation

\[ q_0 \xrightarrow{a} q_1 \]
\[ q_1 \xrightarrow{b} q_0 \]
\[ q_0 \xrightarrow{b} q_1 \]

\[ \langle \varepsilon, 0 \rangle \]
\[ \langle a, 1 \rangle \]
\[ \langle b, 0 \rangle \]
Identification in the Limit for DFAs \cite{OG92}

- Constructing a characteristic sample
- Representing the transition relation
Identification in the Limit for DFAs \cite{OG92}

- Constructing a characteristic sample
- Representing the transition relation

![Diagram of a DFA with states $q_0$ and $q_1$, transitions labeled with $a$ and $b$, and accepting states marked with orange arrows.]

- $\{\varepsilon, 0\}$
- $\{a, 1\}$
- $\{b, 0\}$
- $\{aa, 0\}$
- $\{ab, 1\}$
Identification in the Limit for DFAs \[\text{[OG}92\text{]}\]

- Constructing a DFA

\[
\begin{align*}
\langle \epsilon, 0 \rangle \\
\langle a, 1 \rangle \\
\langle b, 0 \rangle \\
\langle aa, 0 \rangle \\
\langle ab, 1 \rangle \\
\langle aba, 0 \rangle \\
\langle abb, 1 \rangle 
\end{align*}
\]
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton

\[
\begin{align*}
\langle \epsilon, 0 \rangle \\
\langle a, 1 \rangle \\
\langle b, 0 \rangle \\
\langle aa, 0 \rangle \\
\langle ab, 1 \rangle \\
\langle aba, 0 \rangle \\
\langle abb, 1 \rangle 
\end{align*}
\]
Identification in the Limit for DFAs \cite{OG92}

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

\[
\begin{align*}
\langle \epsilon, 0 \rangle \\
\langle a, 1 \rangle \\
\langle b, 0 \rangle \\
\langle aa, 0 \rangle \\
\langle ab, 1 \rangle \\
\langle aba, 0 \rangle \\
\langle abb, 1 \rangle 
\end{align*}
\]
Identification in the Limit for DFAs \cite{OG92}

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

$\langle \epsilon, 0 \rangle$
$\langle a, 1 \rangle$
$\langle b, 0 \rangle$
$\langle aa, 0 \rangle$
$\langle ab, 1 \rangle$
$\langle aba, 0 \rangle$
$\langle abb, 1 \rangle$
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

$\langle \epsilon, 0 \rangle$
$\langle a, 1 \rangle$
$\langle b, 0 \rangle$
$\langle aa, 0 \rangle$
$\langle ab, 1 \rangle$
$\langle aba, 0 \rangle$
$\langle abb, 1 \rangle$
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

$\langle \varepsilon, 0 \rangle$
$\langle a, 1 \rangle$
$\langle b, 0 \rangle$
$\langle aa, 0 \rangle$
$\langle ab, 1 \rangle$
$\langle aba, 0 \rangle$
$\langle abb, 1 \rangle$
Identification in the Limit for DFAs \[\text{[OG92]}\]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

\[
\begin{align*}
&\langle \epsilon, 0 \rangle \\
&\langle a, 1 \rangle \\
&\langle b, 0 \rangle \\
&\langle aa, 0 \rangle \\
&\langle ab, 1 \rangle \\
&\langle aba, 0 \rangle \\
&\langle abb, 1 \rangle
\end{align*}
\]
Identification in the Limit for DFAs [OG92]

- Constructing a DFA
- Prefix-tree automaton
- Join states according to $S'$

\[
\begin{align*}
\langle \epsilon, 0 \rangle \\
\langle a, 1 \rangle \\
\langle b, 0 \rangle \\
\langle aa, 0 \rangle \\
\langle ab, 1 \rangle \\
\langle aba, 0 \rangle \\
\langle abb, 1 \rangle
\end{align*}
\]
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- Creating a set of concrete words
  - concretize \((\langle \psi_1, \ldots, \psi_n \rangle) = \langle \Gamma_1, \ldots, \Gamma_n \rangle\)
  - concretize \(([0,100), [100, \infty)) = \langle \{0\}, \{100\} \rangle\)
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- Creating a set of concrete words
- \( \text{concretize } (\langle \psi_1, ..., \psi_n \rangle) = \langle \Gamma_1, ..., \Gamma_n \rangle \)
- \( \text{concretize } ([0,100), [100, \infty)) = \{\{0\}, \{100\}\} \)
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- Creating a set of concrete words
Identification in the Limit for SFAs

• Learn the SFA out of a set of concrete words
• Construct an SFA
• generalize($\langle \Gamma_1, ..., \Gamma_n \rangle$) = $\langle \psi_1, ..., \psi_n \rangle$
• generalize($\{0\}, \{100\}$) = $\langle [0,100), [100, \infty) \rangle$

Monotonic algebra!
Identification in the Limit for SFAs

• Learn the SFA out of a set of concrete words
• Construct an SFA
• generalize($\Gamma_1, ..., \Gamma_n$) = $\langle \psi_1, ..., \psi_n \rangle$
• generalize($\{0\}, \{100\}$) = $\langle [0,100), [100, \infty) \rangle$
Identification in the Limit for SFAs

- generalize

\[ \Gamma_1 = \{0, 50, 400\} \quad \Gamma_2 = \{100, 800\} \quad \Gamma_3 = \{2048\} \]

- generalize \( (\Gamma_1, \Gamma_2, \Gamma_3) = (\{0,100\} \lor [400,800), \ [100, 400) \lor [800,2048), [2048, \infty)) \)
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- Construct an SFA
Identification in the Limit for SFAs

- Learn the SFA out of a set of concrete words
- **Construct an SFA**
  - $\text{decontaminate}(\Sigma) = \Sigma'$
  - $\Sigma' \subseteq \Sigma$ and contains exactly the alphabet of concretizations
Identification in the Limit for SFAs

- Travers words by lexicographic order
- Add letters that are needed for access words and for transitions relation
Identification in the Limit for SFAs

- Travers words by lexicographic order
- Add letters that are needed for access words and for transitions relation

Monotonic algebra!
Identification in the Limit for SFAs

DFA $D_A$

concretize

Black box characteristic sample for DFAs

SFA $A$

Sample $S_A$

Sample $S \supseteq S_A$

generalize

DFA $D_{S'}$

Black box infer a DFA

SFA $A_{S'}$

Sample $S \supseteq S' \supseteq S_A$
Necessary Condition

\[ \langle \psi_1, ..., \psi_n \rangle \xrightarrow{\text{concretize}} \langle \Gamma_1, ..., \Gamma_n \rangle \]

\[ \langle \varphi_1, ..., \varphi_n \rangle \xrightarrow{\text{generalize}} \langle \Delta_1, ..., \Delta_n \rangle \]
Necessary Condition

\[ \langle \psi_1, \ldots, \psi_n \rangle \xrightarrow{\text{concretize}} \langle \Gamma_1, \ldots, \Gamma_n \rangle \]

Poly time and data

If \([\Delta_i] \supseteq [\Gamma_i]\)

Then \([\varphi_i] = [\psi_i]\)

\[ \langle \varphi_1, \ldots, \varphi_n \rangle \xleftarrow{\text{generalize}} \langle \Delta_1, \ldots, \Delta_n \rangle \]
Necessary Condition

\[ \langle \psi_1, ..., \psi_n \rangle \] concretize \[ \langle \Gamma_1, ..., \Gamma_n \rangle \]

Poly time and data

If \[ [\Delta_i] \supseteq [\Gamma_i] \]
Then \[ [\varphi_i] = [\psi_i] \]

\[ \langle \varphi_1, ..., \varphi_n \rangle \] generalize \[ \langle \Delta_1, ..., \Delta_n \rangle \]

Otherwise, we cannot learn outgoing transitions of a single state

Diagram:
- \( q_0 \) with transitions to \( [0,100) \) and \( [100, \infty) \)
Propositional Algebra

• Predicates are defined over \{p_1, \ldots, p_k\}
• Examples: \( p_1 \lor p_2, (p_1 \land p_2) \lor p_3 \)
• Looking for efficient concretize and generalize
Propositional Algebra

- Predicates are defined over \( \{p_1, \ldots, p_k\} \)
- Examples: \( p_1 \lor p_2, (p_1 \land p_2) \lor p_3 \)
- Looking for efficient concretize and generalize

\[ \gamma \]
\[ |\gamma| = 2^{2^k} \]
set of semantic Boolean functions over \( k \) propositions

\[ \mathcal{P} \]
set of concrete partitions of polynomial size in \( k \)

\[ |\mathcal{P}| < |\gamma| \]

No one to one function from \( \gamma \) to \( \mathcal{P} \)

Every function defines a set of sets of propositions satisfying the function
Query Learning of SFAs

• L* - style learning of SFA

• Goal: learn an SFA over a Boolean algebra, while asking queries over concrete letters

• [AD18] suggest MAT* for learning SFAs
Query Learning of SFAs

• Learnability of the underlying algebra is a necessary condition
• Membership
Query Learning of SFAs

- Learnability of the underlying algebra is a necessary condition
- Equivalence

\[ \text{Is } \llbracket \psi \rrbracket = \llbracket \varphi \rrbracket? \]

Teacher

\( \varphi \)

Yes / No + cex

Learner
Query Learning of SFAs

- Learnability of the underlying algebra is a necessary condition
- Assume that we can learn SFA, then we can learn the algebra
Query Learning of SFAs

• Concise SFA over the propositional algebra cannot be polynomially learned using MQ and EQ
• The teacher can force the learner to ask $2^k - 1$ queries
• Membership

Teacher: No

Is $\langle 0,1,0,\ldots,1 \rangle \in \llbracket \varphi \rrbracket$?
Query Learning of SFAs

- Concise SFA over the propositional algebra cannot be polynomially learned using MQ and EQ
- The teacher can force the learner to ask $2^k - 1$ queries
- Equivalence

Is $\llbracket \psi \rrbracket = \llbracket \varphi \rrbracket$?

Teacher: No, $b \notin \llbracket \psi \rrbracket$.
Complexity of SFAs

- Usually, the size of DFA is measured by its number of states.
- For SFAs, we need to consider:

\[ \langle n, m, l \rangle \]

\[ \text{number of states} \quad \text{out-degree} \quad \text{size of the most complex predicate} \]
Complexity of SFAs

**Normalized SFA**
- One transition between each pair of states
- Predicates labeling the transitions can be very complex

**Neat SFA**
- Only basic transitions
- Predicates labeling transitions are simple
- Can cause an exponential blowup in the number of transitions
Complexity of SFAs

- Converting to normalized
- Disjunction between all transition predicates

\[(n, m, l)\]
Complexity of SFAs

• Converting to neat
• Splitting into basic transitions, using DNF

\[(p_1 \lor p_3) \land (p_2 \lor p_4)\]

\(\langle n, m, l \rangle\)

\[(p_1 \land p_2) \lor (p_1 \land p_4) \lor (p_3 \land p_2) \lor (p_3 \land p_4)\]

\(\langle n, m \cdot 2^l, l \rangle\)
Complexity of SFAs

- **For monotonic algebras**, transforming to DNF is polynomial in the size of the original formula

\[
([0, 100) \lor [200, 500)) \land ([0, 300) \lor [400, 600)) = \\
([0, 100) \land [0, 300)) \lor ([0, 100) \land [400, 600)) \lor \\
([200, 500) \land [0, 300)) \lor ([200, 500) \land [400, 600)) = \\
[0, 100) \lor [200, 300) \lor [400, 500)
\]

- Then, over monotonic algebras, transforming to neat is polynomial
### Complexity of SFAs – Automata Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>$\langle n, m, l \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>product construction $M_1, M_2$</td>
<td>$\langle n_1 \times n_2, m_1 \times m_2, \size_{\land}(l_1, l_2) \rangle$</td>
</tr>
<tr>
<td>complementation of deterministic $M_1^1$</td>
<td>$\langle n_1 + 1, m_1 + 1, \size_{\lor}(l_1) \rangle$</td>
</tr>
<tr>
<td>determinization of $M_1$</td>
<td>$\langle 2^{n_1}, 2^{m_1}, \size_{\land}(n_1 \times m_1, l_1) \rangle$</td>
</tr>
<tr>
<td>minimization of $M_1$</td>
<td>$\langle n_1, m_1, \size_{\land}(l_1) \rangle$</td>
</tr>
</tbody>
</table>

Table 5.1: Analysis of standard automata procedures on SFAs.
Complexity of SFAs – Decision Procedures

<table>
<thead>
<tr>
<th>Decision Procedures</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>emptiness</td>
<td>linear in $n, m$</td>
</tr>
<tr>
<td>emptiness + feasibility</td>
<td>$n \times m \times \text{sat}^\text{P}(l)$</td>
</tr>
<tr>
<td>membership of $\gamma_1 \cdots \gamma_t \in \mathbb{D}^*$</td>
<td>$\sum_{i=1}^{t} \text{sat}^\text{P}(\text{size}^\text{P}(l,</td>
</tr>
<tr>
<td>inclusion $\mathcal{M}_1 \subseteq \mathcal{M}_2$</td>
<td>$((n_1 \times n_2) \times (m_1 \times m_2) \times \text{sat}^\text{P}(\text{size}^\text{P}(l_1, l_2)))$</td>
</tr>
</tbody>
</table>

Table 5.2: Analysis of times complexity of decision procedures for SFAs
SFA Summary

- Identification in the limit of SFA
  - Necessary and sufficient conditions
  - Algorithm for identification of SFAs over monotonic algebras

- Necessary condition for query learning of SFAs
  - SFAs over the propositional algebra are not efficiently learnable

- Complexity of automata algorithms in terms of \( \langle n, m, l \rangle \)
Thank you!
Questions?

1: while (true)
2:     pass = readInput;
3:     while (pass ≤ 999)
4:         pass = readInput;
5:     pass2 = encrypt(pass);