A Categorical Framework for Testing Generalised Tree automata

Bálint Kocsis

Radboud University

CALCO, June 17, 2025

Summary

- Motivation: equivalence query in **automata learning**

Summary

- Motivation: equivalence query in **automata learning**
- Provide a **categorical framework** for **testing black-box systems**

Summary

- Motivation: equivalence query in **automata learning**
- Provide a **categorical framework** for **testing black-box systems**
- Generalise the gist of the completeness proof for DFAs/Mealy machines to a categorical level

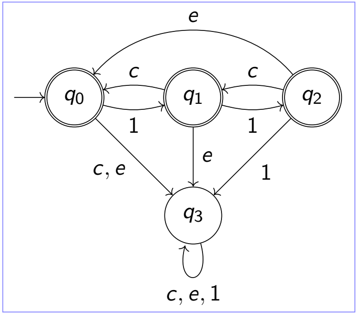
Summary

- Motivation: equivalence query in **automata learning**
- Provide a **categorical framework** for **testing black-box systems**
- Generalise the gist of the completeness proof for DFAs/Mealy machines to a categorical level
- Based on recent work [KR25]

Outline

1. Introduction
   - Conformance Testing
   - The W-method
2. Further generalisation
   - Automata
   - Main result
3. Conclusion

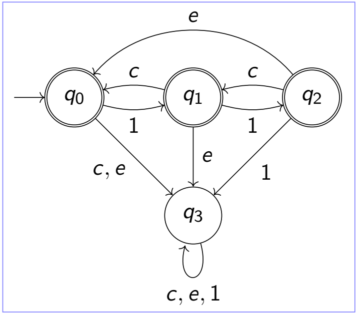
Idea of conformance testing



(a) Specification

(b) Implementation

Idea of conformance testing



(a) Specification

(b) Implementation

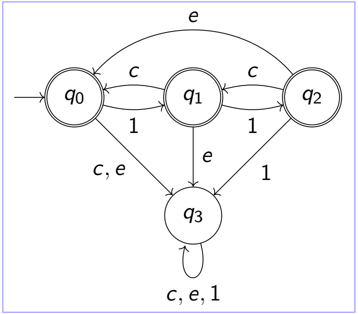
Are they equivalent?

Testing

To decide equivalence, **the only thing we can do is testing**!

Testing

To decide equivalence, **the only thing we can do is testing**!

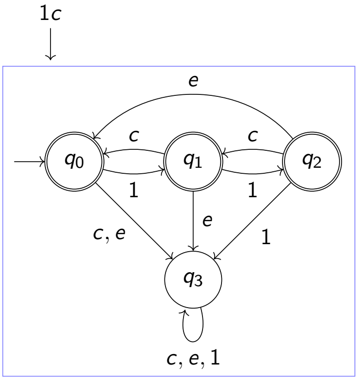


(a) Specification

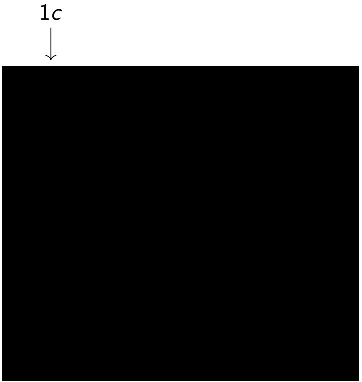
(b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!



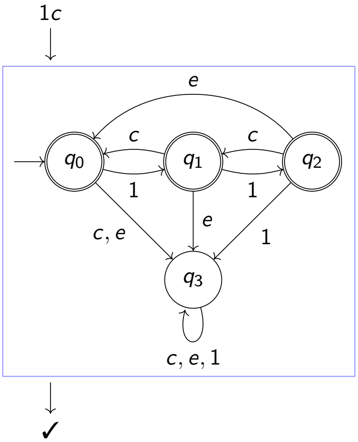
(a) Specification



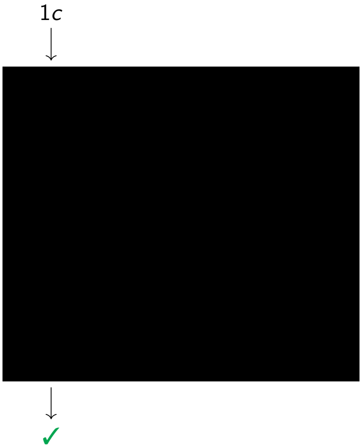
(b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!



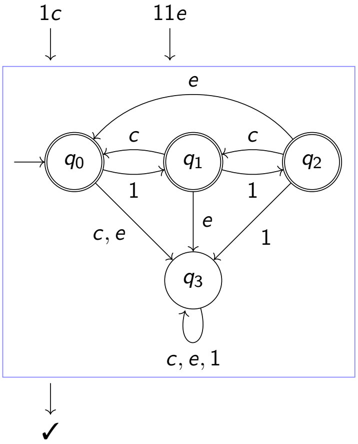
(a) Specification



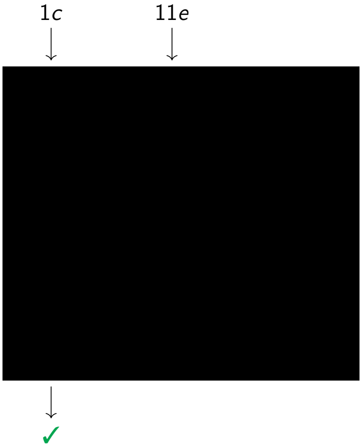
(b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!



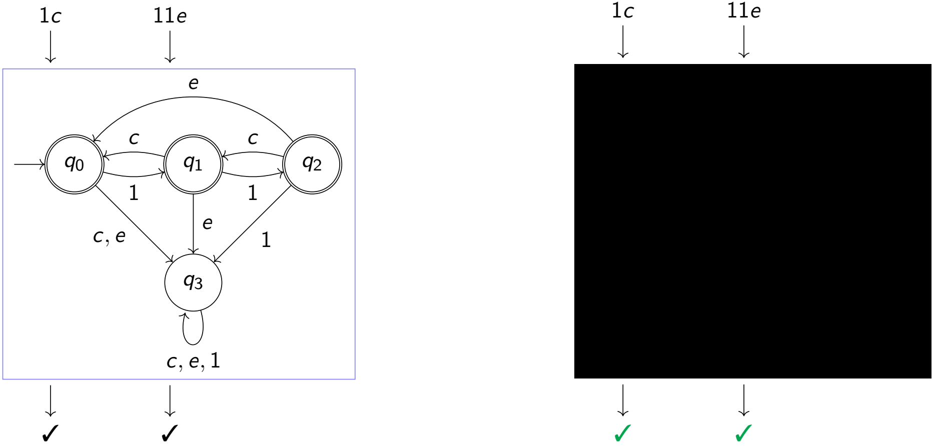
(a) Specification



(b) Implementation

Testing

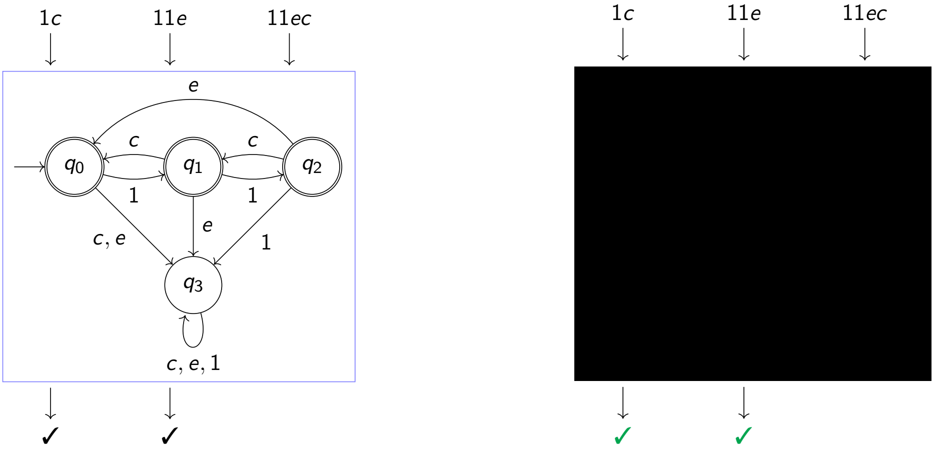
To decide equivalence, **the only thing we can do is testing**!



(a) Specification (b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!

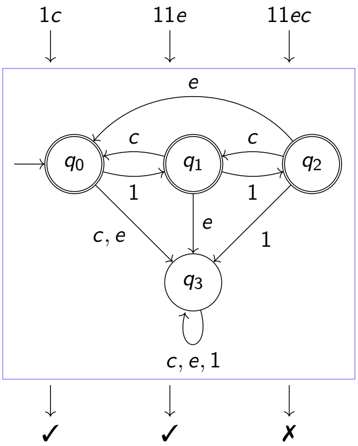


(a) Specification

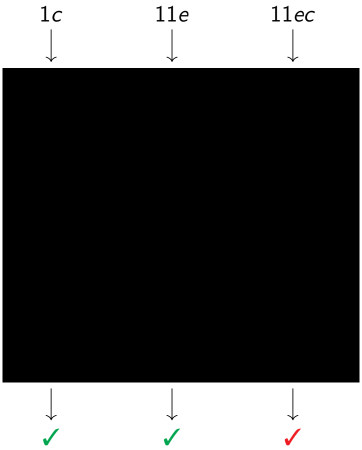
(b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!



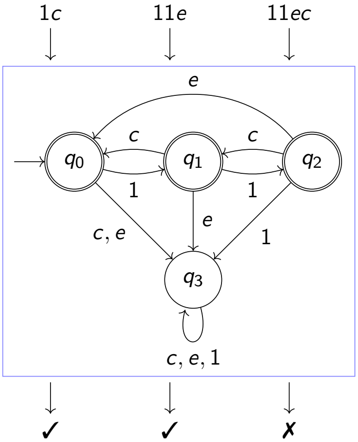
(a) Specification



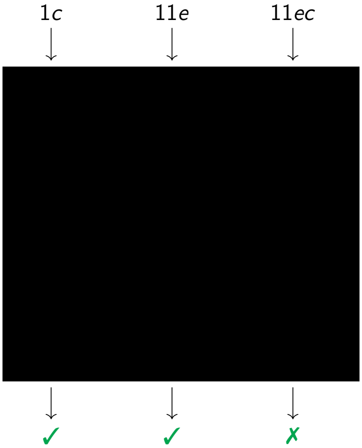
(b) Implementation

Testing

To decide equivalence, **the only thing we can do is testing**!



(a) Specification



(b) Implementation

What if we pass all the tests?

Completeness

- Testing cannot be exhaustive: for any test suite, we can construct a (large) faulty implementation that passes all the tests

Completeness

- Testing cannot be exhaustive: for any test suite, we can construct a (large) faulty implementation that passes all the tests
- More formally, no **complete test suite** exists

Completeness

To obtain completeness, we restrict the space of possible (faulty) implementations

Completeness

To obtain completeness, we restrict the space of possible (faulty) implementations

Definition ([VFM24])

A **fault domain** is a collection of DFAs.

E.g. $\mathcal{U}_m = \{\mathcal{M} \mid \mathcal{M} \text{ has at most } m \text{ states}\}$

Completeness

To obtain completeness, we restrict the space of possible (faulty) implementations

Definition ([VFM24])

A **fault domain** is a collection of DFAs.

E.g. $\mathcal{U}_m = \{\mathcal{M} \mid \mathcal{M} \text{ has at most } m \text{ states}\}$

Fix a specification $\mathcal{S}$

Definition ([VFM24])

A test suite $T \subseteq \Sigma^*$ is **complete** for $\mathcal{S}$ with respect to a fault domain $\mathcal{U}$ if $\mathcal{L}_\mathcal{S} \cap T = \mathcal{L}_\mathcal{M} \cap T$ implies $\mathcal{L}_\mathcal{S} = \mathcal{L}_\mathcal{M}$ for all $\mathcal{M} \in \mathcal{U}$.

Completeness

To obtain completeness, we restrict the space of possible (faulty) implementations

Definition ([VFM24])

A **fault domain** is a collection of DFAs.

E.g. $\mathcal{U}_m = \{\mathcal{M} \mid \mathcal{M} \text{ has at most } m \text{ states}\}$

Fix a specification $\mathcal{S}$

Definition ([VFM24])

A test suite $T \subseteq \Sigma^*$ is **complete** for $\mathcal{S}$ with respect to a fault domain $\mathcal{U}$ if $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$ for all $\mathcal{M} \in \mathcal{U}$.

How do we construct complete test suites?

The W-method [Cho78, Vas73]

We need:

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

$$T^k_{P,W} =$$

Tests are constructed from 3 components:

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

$$T^k_{P,W} = P$$

Tests are constructed from 3 components:

- an access sequence (in P) to reach a specific state in $\mathcal{S}$;

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

$$T^k_{P,W} = P \cdot \Sigma^{\leq k+1}$$

Tests are constructed from 3 components:

- an **access sequence** (in P) to reach a specific state in $\mathcal{S}$;
- an **infix** (in $\Sigma^{\leq k+1}$) to cover all states and transitions in $\mathcal{M}$;

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

$$T^k_{P,W} = P \cdot \Sigma^{\leq k+1} \cdot W$$

Tests are constructed from 3 components:

- an **access sequence** (in P) to reach a specific state in $\mathcal{S}$;
- an **infix** (in $\Sigma^{\leq k+1}$) to cover all states and transitions in $\mathcal{M}$;
- a **distinguishing sequence** (in W) for testing the reached states in $\mathcal{S}$ and $\mathcal{M}$.

The W-method [Cho78, Vas73]

We need:

- $P \subseteq \Sigma^*$ **state cover**: words that **cover** the state space
- $W \subseteq \Sigma^*$ **characterisation set**: words that **distinguish states**

$$T^k_{P,W} = P \cdot \Sigma^{\leq k+1} \cdot W$$

Tests are constructed from 3 components:

- an access sequence (in P) to reach a specific state in $\mathcal{S}$;
- an **infix** (in $\Sigma^{\leq k+1}$) to cover all states and transitions in $\mathcal{M}$;
- a distinguishing sequence (in W) for testing the reached states in $\mathcal{S}$ and $\mathcal{M}$.

Theorem

Suppose $\mathcal{S}$ has n states. Then $T^k_{P,W}$ is complete for $\mathcal{S}$ with respect to $\mathcal{U}_{n+k}$.

Proof of completeness

Two steps:

Proof of completeness

Two steps:

- Construct a state cover for the **implementation**(!!!) from the state cover of the specification $\leftarrow$ need assumptions on the implementation

Proof of completeness

Two steps:

- Construct a state cover for the **implementation**(!!!) from the state cover of the specification $\leftarrow$ need assumptions on the implementation
- Prove equivalence from assumption that all tests pass

Proof of completeness

Two steps:

- Construct a state cover for the **implementation**(!!!) from the state cover of the specification $\leftarrow$ need assumptions on the implementation
- Prove equivalence from assumption that all tests pass

Lemma ([KJR24])

Let $\mathcal{S}$ and $\mathcal{M}$ be two DFAs, and suppose C is a state cover for $\mathcal{M}$ and W is a characterisation set for $\mathcal{S}$. Let $T = C \cdot \Sigma^{\leq 1} \cdot W$. Then $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$.

Proof of completeness

Two steps:

- Construct a state cover for the **implementation**(!!!) from the state cover of the specification $\leftarrow$ need assumptions on the implementation
- Prove equivalence from assumption that all tests pass

Lemma ([KJR24])

Let $\mathcal{S}$ and $\mathcal{M}$ be two DFAs, and suppose C is a state cover for $\mathcal{M}$ and W is a characterisation set for $\mathcal{S}$. Let $T = C \cdot \Sigma^{\leq 1} \cdot W$. Then $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$.

This lemma was generalised to a categorical setting in [KR25]; we improve on that work

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}} \colon T_F I \to Q$: maps an input to the state reached upon processing it

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}}\colon T_F I \to Q$: maps an input to the state reached upon processing it
- **behaviour map** $B_{\mathcal{A}}\colon T_F I \to O$: maps in an input to an output

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}} \colon T_F I \to Q$: maps an input to the state reached upon processing it
- **behaviour map** $B_{\mathcal{A}} \colon T_F I \to O$: maps in an input to an output
- **observability map** $o_{\mathcal{A}} \colon Q \to [T_F I, O]$: maps a state to its behaviour

Automata in categories [AM75]

$$\begin{tikzcd} & FQ \arrow[d, "\delta"] & \\ I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O \end{tikzcd}$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}} \colon T_F I \to Q$: maps an input to the state reached upon processing it
- **behaviour map** $B_{\mathcal{A}} \colon T_F I \to O$: maps in an input to an output
- **observability map** $o_{\mathcal{A}} \colon Q \to [T_F I, O]$: maps a state to its behaviour

Examples:

Automata in categories [AM75]

$$
\begin{tikzcd}
 & FQ \arrow[d, "\delta"] & \\
I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O
\end{tikzcd}
$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}} \colon T_F I \to Q$: maps an input to the state reached upon processing it
- **behaviour map** $B_{\mathcal{A}} \colon T_F I \to O$: maps in an input to an output
- **observability map** $o_{\mathcal{A}} \colon Q \to [T_F I, O]$: maps a state to its behaviour

Examples:

- DFAs, NFAs, weighted automata, deterministic nominal automata: $FX = X \otimes \Sigma$ for an alphabet Σ

Automata in categories [AM75]

$$
\begin{tikzcd}
 & FQ \arrow[d, "\delta"] & \\
I \arrow[r, "i"'] & Q \arrow[r, "f"'] & O
\end{tikzcd}
$$

- We assume that free F-algebras $(T_F X, \gamma_X, \eta_X)$ exist and that T_F is strong
- $T_F I$ is the object of **inputs**, γ_I is an **extension** operation, η_I is the trivial input

For any automaton $\mathcal{A}$, we get three maps:

- **reachability map** $r_{\mathcal{A}} \colon T_F I \to Q$: maps an input to the state reached upon processing it
- **behaviour map** $B_{\mathcal{A}} \colon T_F I \to O$: maps in an input to an output
- **observability map** $o_{\mathcal{A}} \colon Q \to [T_F I, O]$: maps a state to its behaviour

Examples:

- DFAs, NFAs, weighted automata, deterministic nominal automata: $FX = X \otimes \Sigma$ for an alphabet Σ
- **tree automata**: $FX = \sum_{f \in \Sigma} X^{\mathrm{ar}(f)}$ for an algebraic signature Σ

State covers and characterisation morphisms

'Sets of inputs' $\sim$ morphisms into $T_F I$

State covers and characterisation morphisms

'Sets of inputs' $\sim$ morphisms into $T_F I$

Definition

A morphism $c\colon C \to T_F I$ is called a **state cover** for $\mathcal{A}$ if $C \xrightarrow{c} T_F I \xrightarrow{r_{\mathcal{A}}} Q$ is a split epi (and η_I factors through c).

State covers and characterisation morphisms

'Sets of inputs' $\sim$ morphisms into $T_F I$

Definition

A morphism $c \colon C \to T_F I$ is called a **state cover** for $\mathcal{A}$ if $C \xrightarrow{c} T_F I \xrightarrow{r_{\mathcal{A}}} Q$ is a split epi (and η_I factors through c).

This notion is too strong (existence not guaranteed) $\Rightarrow$ we need **weak state covers**

State covers and characterisation morphisms

'Sets of inputs' $\sim$ morphisms into $T_F I$

Definition

A morphism $c \colon C \to T_F I$ is called a **state cover** for $\mathcal{A}$ if $C \xrightarrow{c} T_F I \xrightarrow{r_{\mathcal{A}}} Q$ is a split epi (and η_I factors through c).

This notion is too strong (existence not guaranteed) $\Rightarrow$ we need **weak state covers**

Definition

A morphism $w \colon W \to T_F I$ is called a **characterisation morphism** for $\mathcal{A}$ if $Q \xrightarrow{o_{\mathcal{A}}} [T_F I, O] \xrightarrow{[w,\mathrm{id}]} [W, O]$ is a mono
(and η_I factors through w).

Main theorem

Recall:

Main theorem

Recall:

Lemma ([KJR24])

Let $\mathcal{S}$ and $\mathcal{M}$ be two DFAs, and suppose C is a state cover for $\mathcal{M}$ and W is a characterisation set for $\mathcal{S}$. Let $T = C \cdot \Sigma^{\leq 1} \cdot W$. Then $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$.

Main theorem

Recall:

Lemma ([KJR24])

Let $\mathcal{S}$ and $\mathcal{M}$ be two DFAs, and suppose C is a state cover for $\mathcal{M}$ and W is a characterisation set for $\mathcal{S}$. Let $T = C \cdot \Sigma^{\leq 1} \cdot W$. Then $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$.

Theorem

Let $\mathcal{S}$ and $\mathcal{M}$ be two automata. Suppose c is a weak state cover for $\mathcal{M}$ and w is a characterisation morphism for $\mathcal{S}$. Let $t = c^{+} \cdot w$. Then $B_{\mathcal{S}} \circ t = B_{\mathcal{M}} \circ t$ implies $B_{\mathcal{S}} = B_{\mathcal{M}}$.

Main theorem

Recall:

Lemma ([KJR24])

Let $\mathcal{S}$ and $\mathcal{M}$ be two DFAs, and suppose C is a state cover for $\mathcal{M}$ and W is a characterisation set for $\mathcal{S}$. Let $T = C \cdot \Sigma^{\leq 1} \cdot W$. Then $\mathcal{L}_{\mathcal{S}} \cap T = \mathcal{L}_{\mathcal{M}} \cap T$ implies $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{M}}$.

Theorem

Let $\mathcal{S}$ and $\mathcal{M}$ be two automata. Suppose c is a weak state cover for $\mathcal{M}$ and w is a characterisation morphism for $\mathcal{S}$. Let $t = c^{+} \cdot w$. Then $B_{\mathcal{S}} \circ t = B_{\mathcal{M}} \circ t$ implies $B_{\mathcal{S}} = B_{\mathcal{M}}$.

- $c^{+} = [c, \gamma_I \circ Fc]$: obtained by extending inputs in c with one symbol
- $a \cdot b = m \circ (a \otimes b)$: concatenation of the inputs in a and in b

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner
- New applications: all **adjoint automata** and **tree automata**

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner
- New applications: all **adjoint automata** and **tree automata**

Future work:

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner
- New applications: all **adjoint automata** and **tree automata**

Future work:

- Find abstract conditions on the implementation that allow us to construct state covers for the implementation from state covers of the specification

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner
- New applications: all **adjoint automata** and **tree automata**

Future work:

- Find abstract conditions on the implementation that allow us to construct state covers for the implementation from state covers of the specification
- Create abstract algorithms for constructing state covers and characterisation sets

Conclusion

- We provided a categorical framework for proving completeness of test suites for a wide range of automaton models
- We generalised the results of [KR25], but the framework is more abstract and cleaner
- New applications: all **adjoint automata** and **tree automata**

Future work:

- Find abstract conditions on the implementation that allow us to construct state covers for the implementation from state covers of the specification
- Create abstract algorithms for constructing state covers and characterisation sets
- Implement algorithms for concrete models

References I

Michael A. Arbib and Ernest G. Manes.
Adjoint machines, state-behavior machines, and duality.
Journal of Pure and Applied Algebra, 6(3):313–344, 1975.

Tsun S. Chow.
Testing software design modeled by finite-state machines.
IEEE Transactions on Software Engineering, 4(3):178–187, 1978.

Loes Kruger, Sebastian Junges, and Jurriaan Rot.
Small test suites for active automata learning.
In *TACAS (2)*, volume 14571 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2024.

References II

Bálint Kocsis and Jurriaan Rot.
Complete test suites for automata in monoidal closed categories.
In *FoSSaCS*, volume 15691 of *Lecture Notes in Computer Science*, pages 198–219. Springer, 2025.

M. P. Vasilevskii.
Failure diagnosis of automata.
Cybernetics, 9(4):653–665, 1973.

Frits W. Vaandrager, Paul Fiterau-Brostean, and Ivo Melse.
Completeness of FSM test suites reconsidered.
CoRR, abs/2410.19405, 2024.