Duality in Logic and Computation

Prakash Panangaden[1]

[1]School of Computer Science
McGill University

IEEE Symposium On Logic In Computer Science, June 2013

Examples of duality principles

- Linear programming.

Examples of duality principles

- Linear programming.
- Electric and magnetic fields,

Examples of duality principles

- Linear programming.
- Electric and magnetic fields,
- now vastly generalized to geometric Langlands duality.

Examples of duality principles

- Linear programming.
- Electric and magnetic fields,
- now vastly generalized to geometric Langlands duality.
- Controllability and observability in control theory, Kalman.

Examples of duality principles

- Linear programming.
- Electric and magnetic fields,
- now vastly generalized to geometric Langlands duality.
- Controllability and observability in control theory, Kalman.
- Many examples from semantics and logic.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.
- Two – apparently – different structures are actually two different descriptions of the same thing.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.
- Two – apparently – different structures are actually two different descriptions of the same thing.
- More importantly given a map: $f : S_1 \to S_2$ we get a map $\hat{f} : V_2 \to V_1$ and vice versa;

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.
- Two – apparently – different structures are actually two different descriptions of the same thing.
- More importantly given a map: $f : S_1 \to S_2$ we get a map $\hat{f} : V_2 \to V_1$ and vice versa;
- note the *reversal* in the direction of the arrows.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.
- Two – apparently – different structures are actually two different descriptions of the same thing.
- More importantly given a map: $f : S_1 \to S_2$ we get a map $\hat{f} : V_2 \to V_1$ and vice versa;
- note the *reversal* in the direction of the arrows.
- The two mathematical universes are *mirror images* of each other.

What is Stone-type duality?

- Two types of structures: Shiv and Vish.
- Every Shiv has an associated Vish and vice versa.
- $V \to S$, $S \to V'$; V and V' are isomorphic.
- Two – apparently – different structures are actually two different descriptions of the same thing.
- More importantly given a map: $f : S_1 \to S_2$ we get a map $\hat{f} : V_2 \to V_1$ and vice versa;
- note the *reversal* in the direction of the arrows.
- The two mathematical universes are *mirror images* of each other.
- Two completely different sets of theorems that one can use.

Examples of Stone-type dualities

- Vector spaces and vector spaces.

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]
- State transformer semantics and weakest precondition semantics. [DeBakker,Plotkin,Smyth]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]
- State transformer semantics and weakest precondition semantics. [DeBakker,Plotkin,Smyth]
- Logics and Transition systems. [Bonsangue, Kurz,...]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]
- State transformer semantics and weakest precondition semantics. [DeBakker,Plotkin,Smyth]
- Logics and Transition systems. [Bonsangue, Kurz,...]
- Measures and random variables. [Kozen]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]
- State transformer semantics and weakest precondition semantics. [DeBakker,Plotkin,Smyth]
- Logics and Transition systems. [Bonsangue, Kurz,...]
- Measures and random variables. [Kozen]
- Commutative unital C*-algebras and compact Hausdorff spaces. [Gelfand, Stone]

Examples of Stone-type dualities

- Vector spaces and vector spaces.
- Boolean algebras and Stone spaces. [Stone]
- Modal logics and boolean algebras with operators. [Jonsson, Tarski]
- State transformer semantics and weakest precondition semantics. [DeBakker,Plotkin,Smyth]
- Logics and Transition systems. [Bonsangue, Kurz,...]
- Measures and random variables. [Kozen]
- Commutative unital C*-algebras and compact Hausdorff spaces. [Gelfand, Stone]
- Labelled Markov processes and C^*-algebras with operators. [Mislove, Ouaknine, Pavlovic, Worrell]

Maps matter!

- An essential aspect of mathematics: *structure-preserving maps between objects*.

Maps matter!

- An essential aspect of mathematics: *structure-preserving maps between objects*.
- Interesting constructions on objects (usually) have corresponding constructions on the maps.

Maps matter!

- An essential aspect of mathematics: *structure-preserving maps between objects*.
- Interesting constructions on objects (usually) have corresponding constructions on the maps.
- Compositions are *preserved* or *reversed*.

Maps matter!

- An essential aspect of mathematics: *structure-preserving maps between objects*.
- Interesting constructions on objects (usually) have corresponding constructions on the maps.
- Compositions are *preserved* or *reversed*.
- This is *functoriality*.

Maps matter!

- An essential aspect of mathematics: *structure-preserving maps between objects*.
- Interesting constructions on objects (usually) have corresponding constructions on the maps.
- Compositions are *preserved* or *reversed*.
- This is *functoriality*.
- From this one can often conclude *invariance properties*.

Duality categorically

$$\begin{tikzcd}
\mathcal{C} \arrow[rr, bend left, "F"] & & \mathcal{D}^{op} \arrow[ll, bend left, "G"]
\end{tikzcd}$$

Duality categorically

Given

$$\begin{tikzcd} A \in \mathcal{C} \arrow[d, "f"] \\ B \in \mathcal{C} \end{tikzcd}$$

Duality categorically

We get

$$\begin{array}{ccc} A \in \mathcal{C} & \qquad & F(A) \in \mathcal{D} \\ \Big\downarrow{\scriptstyle f} & & \\ B \in \mathcal{C} & & F(B) \in \mathcal{D} \end{array}$$

Duality categorically

and

$$
\begin{tikzcd}
A \in \mathcal{C} \arrow[d, "f"] & F(A) \in \mathcal{D} \\
B \in \mathcal{C} & F(B) \in \mathcal{D}. \arrow[u, "F(f)"]
\end{tikzcd}
$$

Duality categorically

Similarly, given

$$
\begin{tikzcd}
C \in \mathcal{D} \arrow[d, "g"] \\
D \in \mathcal{D}
\end{tikzcd}
$$

Duality categorically

We get

$$\begin{array}{ccc} G(C) \in \mathcal{C} & & C \in \mathcal{D} \\ & & \downarrow{\scriptstyle g} \\ G(D) \in \mathcal{C} & & D \in \mathcal{D} \end{array}$$

Duality categorically

and

$$
\begin{tikzcd}
G(C) \in \mathcal{C} & C \in \mathcal{D} \arrow[d, "g"] \\
G(D) \in \mathcal{C} \arrow[u, "G(g)"] & D \in \mathcal{D}.
\end{tikzcd}
$$

Isomorphisms

We have isomorphisms

$$A \simeq G(F(A)) \text{ and } C \simeq F(G(C)).$$

Duality categorically

Stone-type Duality

We have a (contravariant) adjunction between categories $\mathcal{C}$ and $\mathcal{D}$, which is an *equivalence* of categories.

Often obtained by looking at maps into an object living in both categories: a schizophrenic object.

Duality for high school students I

- Finite-dimensional vector space V over, say $\mathbb{R}$,

Duality for high school students I

- Finite-dimensional vector space V over, say $\mathbb{R}$,
- *Dual space* V^* of linear maps from V to $\mathbb{R}$.

Duality for high school students I

- Finite-dimensional vector space V over, say $\mathbb{R}$,
- *Dual space* V^* of linear maps from V to $\mathbb{R}$.
- V^* has the same dimension as V and a (basis-dependent) isomorphism between V and V^*.

Duality for high school students I

- Finite-dimensional vector space V over, say $\mathbb{R}$,
- *Dual space* V^* of linear maps from V to $\mathbb{R}$.
- V^* has the same dimension as V and a (basis-dependent) isomorphism between V and V^*.
- The double dual V^{**} is also isomorphic to V
- with a "nice" canonical isomorphism: $v \in V \mapsto \lambda\sigma \in V^*.\sigma(v)$.

Duality for high school students II

$$U \xrightarrow{\theta} V$$

$$U^* \xleftarrow[\theta^*]{} V^*$$

Given a linear maps θ between vector spaces U and V we get a map θ^* *in the opposite direction* between the dual spaces:

$$\theta^*(\sigma \in V^*)(u \in U) = \sigma(\theta(u)).$$

Boolean algebras

A Boolean algebra is a set A equipped with two constants, $0, 1$, a unary operation $(\cdot)'$ and two binary operations $\vee, \wedge$ which obey the following axioms, p, q, r are arbitrary members of A:

$$\begin{array}{rr} 0' = 1 & 1' = 0 \\ p \wedge 0 = 0 & p \vee 1 = 1 \\ p \wedge 1 = p & p \vee 0 = p \\ p \wedge p' = 0 & p \vee p' = 1 \\ p \wedge p = p & p \vee p = p \end{array}$$

Boolean algebras II

$$
\begin{array}{rcl}
p'' & = & p \\
(p \wedge q)' & = & p' \vee q' \\
(p \vee q)' & = & p' \wedge q' \\
p \wedge q & = & q \wedge p \\
p \vee q & = & q \vee p \\
p \wedge (q \wedge r) & = & (p \wedge q) \wedge r \\
p \vee (q \vee r) & = & (p \vee q) \vee r \\
p \wedge (q \vee r) & = & (p \wedge q) \vee (p \wedge r) \\
p \vee (q \wedge r) & = & (p \vee q) \wedge (p \vee r)
\end{array}
$$

The operation $\vee$ is called *join*, $\wedge$ is called *meet* and $(\cdot)'$ is called *complement*.
Maps are Boolean algebra homomorphisms.

(Toy) Stone duality

$$\begin{tikzcd}
\mathbf{FinSet} \arrow[rr, bend left, "\mathcal{P}"] & & \mathbf{FinBoolAlg}^{op} \arrow[ll, bend left, "\mathcal{A}"]
\end{tikzcd}$$

Here $\mathcal{P}$ is power-set and $\mathcal{A}$ takes the *atoms* of a Boolean algebra.

Stone spaces

- A Stone space is a compact Hausdorff space with a base of *clopen* sets: zero-dimensional space.

Stone spaces

- A Stone space is a compact Hausdorff space with a base of *clopen* sets: zero-dimensional space.
- *Totally disconnected*: the only connected sets are singletons.

Stone spaces

- A Stone space is a compact Hausdorff space with a base of *clopen* sets: zero-dimensional space.
- *Totally disconnected*: the only connected sets are singletons.
- Many, but not all, Stone spaces are Polish.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{Cl}(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.
- Ultrafilters correspond precisely to maps from $\mathcal{B}$ to $\mathbf{2}$:

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.
- Ultrafilters correspond precisely to maps from $\mathcal{B}$ to $\mathbf{2}$:
- $\{b \in \mathcal{B} \mid f(b) = 1\}$ is always an ultrafilter.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.
- Ultrafilters correspond precisely to maps from $\mathcal{B}$ to $\mathbf{2}$:
- $\{b \in \mathcal{B} \mid f(b) = 1\}$ is always an ultrafilter.
- Continuous maps $f : \mathcal{S}_1 \to \mathcal{S}_2$ between Stone spaces give Boolean algebra homomorphisms $f^{-1} : \mathcal{C}l(\mathcal{S}_2) \to \mathcal{C}l(\mathcal{S}_1)$.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{C}l(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.
- Ultrafilters correspond precisely to maps from $\mathcal{B}$ to $\mathbf{2}$:
- $\{b \in \mathcal{B} \mid f(b) = 1\}$ is always an ultrafilter.
- Continuous maps $f : \mathcal{S}_1 \to \mathcal{S}_2$ between Stone spaces give Boolean algebra homomorphisms $f^{-1} : \mathcal{C}l(\mathcal{S}_2) \to \mathcal{C}l(\mathcal{S}_1)$.
- Boolean algebra homomorphisms $h : \mathcal{B}_1 \to \mathcal{B}_2$ give rise to continuous functions $(\cdot) \circ f : \mathcal{U}(\mathcal{B}_2) \to \mathcal{U}(\mathcal{B}_1)$ between the Stone spaces.

Classical Stone duality

- Stone space $\mathcal{S}$, the clopens form a Boolean algebra: $\mathcal{Cl}(\mathcal{S})$.
- Boolean algebra $\mathcal{B}$, Stone space $\mathcal{U}(\mathcal{B})$ is the space $\mathcal{U}$ of maximal filters (ultrafilters).
- $b \in \mathcal{B}$, a *basic open* is defined by $U_b := \{u \in \mathcal{U} \mid b \in u\}$.
- Ultrafilters correspond precisely to maps from $\mathcal{B}$ to $\mathbf{2}$:
- $\{b \in \mathcal{B} \mid f(b) = 1\}$ is always an ultrafilter.
- Continuous maps $f : \mathcal{S}_1 \to \mathcal{S}_2$ between Stone spaces give Boolean algebra homomorphisms $f^{-1} : \mathcal{Cl}(\mathcal{S}_2) \to \mathcal{Cl}(\mathcal{S}_1)$.
- Boolean algebra homomorphisms $h : \mathcal{B}_1 \to \mathcal{B}_2$ give rise to continuous functions $(\cdot) \circ f : \mathcal{U}(\mathcal{B}_2) \to \mathcal{U}(\mathcal{B}_1)$ between the Stone spaces.
- Everything that can, and should be, an isomorphism is an isomorphism.

State-transformer semantics

- Operational semantics: states, transitions. What are the next states?

State-transformer semantics

- Operational semantics: states, transitions. What are the next states?
- Elegant and (almost) compositional version: Plotkin's *structured operational semantics*.

State-transformer semantics

- Operational semantics: states, transitions. What are the next states?
- Elegant and (almost) compositional version: Plotkin's *structured operational semantics*.
- Denotational semantics: compositional, equivalent to operational semantics.

Predicate transformers: Dijkstra

- Predicate transformers: if *after* the execution of a command a property P holds, what *must have been true* before?

Predicate transformers: Dijkstra

- Predicate transformers: if *after* the execution of a command a property P holds, what *must have been true* before?
- **Weakest precondition** (wp).

Predicate transformers: Dijkstra

- Predicate transformers: if *after* the execution of a command a property P holds, what *must have been true* before?
- **Weakest precondition** (wp).
- Backward flow in **wp** semantics.

Predicate transformers: Dijkstra

- Predicate transformers: if *after* the execution of a command a property P holds, what *must have been true* before?
- **Weakest precondition** (wp).
- Backward flow in **wp** semantics.
- D and E domains, viewed as topological spaces, open sets: $\mathcal{O}_D$ and $\mathcal{O}_E$. A **predicate transformer** is a *strict, continuous and multiplicative* map $p : \mathcal{O}_E \to \mathcal{O}_D$.

Predicate transformers: Dijkstra

- Predicate transformers: if *after* the execution of a command a property P holds, what *must have been true* before?
- **Weakest precondition** (wp).
- Backward flow in **wp** semantics.
- D and E domains, viewed as topological spaces, open sets: $\mathcal{O}_D$ and $\mathcal{O}_E$. A **predicate transformer** is a *strict, continuous and multiplicative* map $p : \mathcal{O}_E \to \mathcal{O}_D$.
- Relate predicate-transformer semantics to state-transformer semantics: Jaco De Bakker (1978).
- Duality: The category of state transformers is equivalent to the (opposite of) the category of predicate transformers: Plotkin (1979).

Duality for probabilistic programs: Kozen

Probabilistic programs and *expectation transformers*: Kozen (1981)

Logic	Probability
States s	Distributions μ
Formulas P	Random variables f
Satisfaction $s \models P$	Integration $\int f \mathrm{d}\mu$

Domain theory in logical form

- Abramsky's program based on Stone duality.

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).
- Usual denotational semantics: types are domains, terms are elements.

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).
- Usual denotational semantics: types are domains, terms are elements.
- DTLF: types are propositional theories, finite elements are propositions.

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).
- Usual denotational semantics: types are domains, terms are elements.
- DTLF: types are propositional theories, finite elements are propositions.
- Terms are described by axiomatizing satisfaction. A modal logic of programs.

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).
- Usual denotational semantics: types are domains, terms are elements.
- DTLF: types are propositional theories, finite elements are propositions.
- Terms are described by axiomatizing satisfaction. A modal logic of programs.
- The two interpretations are Stone duals.

Domain theory in logical form

- Abramsky's program based on Stone duality.
- Metalanguage for types and terms (programs).
- Usual denotational semantics: types are domains, terms are elements.
- DTLF: types are propositional theories, finite elements are propositions.
- Terms are described by axiomatizing satisfaction. A modal logic of programs.
- The two interpretations are Stone duals.
- Ties together semantics, logic and verification.

Chu spaces: Pratt

- Chu spaces: general construction for $*$-autonomous categories.

Chu spaces: Pratt

- Chu spaces: general construction for $*$-autonomous categories.
- Generalized matrices.

Chu spaces: Pratt

- Chu spaces: general construction for $*$-autonomous categories.
- Generalized matrices.
- Pratt's observation: many interesting categories embed in Chu categories.

Chu spaces: Pratt

- Chu spaces: general construction for $*$-autonomous categories.
- Generalized matrices.
- Pratt's observation: many interesting categories embed in Chu categories.
- Stone duality is "transposition" of matrices.

Many other contributors

- Bart Jacobs, ...
- Achim Jung and Drew Moshier
- Mai Gehrke, Jean-Eric Pin, ...
- Bezhanishvilis

Brzozowski's Algorithm 1964

- Start with DFA.

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.
- Determinize the result.

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.
- Determinize the result.
- Take the reachable states.

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.
- Determinize the result.
- Take the reachable states.
- Repeat.
- This gives the minimal DFA recognizing the same language!

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.
- Determinize the result.
- Take the reachable states.
- Repeat.
- This gives the minimal DFA recognizing the same language!
- The intermediate step can blow up the size of the automaton exponentially before minimizing it.

Brzozowski's Algorithm 1964

- Start with DFA.
- Reverse transitions, interchange initial and final states.
- Determinize the result.
- Take the reachable states.
- Repeat.
- This gives the minimal DFA recognizing the same language!
- The intermediate step can blow up the size of the automaton exponentially before minimizing it.
- But experimental results seem to indicate that it often works well in practice.

Deterministic Automata

- $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$: a deterministic finite (Moore) automaton. S is the set of **states**, $\mathcal{A}$ an **input alphabet** (actions), $\mathcal{O}$ is a set of **observations.**
- $\delta : S \times \mathcal{A} \to S$ is the **state transition function.**
- $\gamma : S \to \mathbf{2}^{\mathcal{O}}$ or $\gamma : S \times \mathcal{O} \to \mathbf{2}$ is a labeling function.

Deterministic Automata

- $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$: a deterministic finite (Moore) automaton. S is the set of **states**, $\mathcal{A}$ an **input alphabet** (actions), $\mathcal{O}$ is a set of **observations.**
- $\delta : S \times \mathcal{A} \to S$ is the **state transition function.**
- $\gamma : S \to \mathbf{2}^{\mathcal{O}}$ or $\gamma : S \times \mathcal{O} \to \mathbf{2}$ is a labeling function.
- If $\mathcal{O} = \{\mathbf{accept}\}$ we have ordinary deterministic finite automata,

Deterministic Automata

- $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$: a deterministic finite (Moore) automaton. S is the set of **states**, $\mathcal{A}$ an **input alphabet** (actions), $\mathcal{O}$ is a set of **observations.**
- $\delta : S \times \mathcal{A} \to S$ is the **state transition function.**
- $\gamma : S \to \mathbf{2}^{\mathcal{O}}$ or $\gamma : S \times \mathcal{O} \to \mathbf{2}$ is a labeling function.
- If $\mathcal{O} = \{\mathbf{accept}\}$ we have ordinary deterministic finite automata,
- *except* that we do not have a start state,

Deterministic Automata

- $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$: a deterministic finite (Moore) automaton. S is the set of **states**, $\mathcal{A}$ an **input alphabet** (actions), $\mathcal{O}$ is a set of **observations.**
- $\delta : S \times \mathcal{A} \to S$ is the **state transition function.**
- $\gamma : S \to \mathbf{2}^{\mathcal{O}}$ or $\gamma : S \times \mathcal{O} \to \mathbf{2}$ is a labeling function.
- If $\mathcal{O} = \{\mathbf{accept}\}$ we have ordinary deterministic finite automata,
- *except* that we do not have a start state,
- which means that reachability makes no sense.

Deterministic Automata

- $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$: a deterministic finite (Moore) automaton. S is the set of **states**, $\mathcal{A}$ an **input alphabet** (actions), $\mathcal{O}$ is a set of **observations.**
- $\delta : S \times \mathcal{A} \to S$ is the **state transition function.**
- $\gamma : S \to \mathbf{2}^{\mathcal{O}}$ or $\gamma : S \times \mathcal{O} \to \mathbf{2}$ is a labeling function.
- If $\mathcal{O} = \{\mathbf{accept}\}$ we have ordinary deterministic finite automata,
- *except* that we do not have a start state,
- which means that reachability makes no sense.
- We will worry about that in a minute.

A Simple Modal Logic

- View $\mathcal{O}$ as propositions, define a simple modal logic. A *formula* φ is:

  $$\varphi ::== \omega \in \mathcal{O} \mid (a)\varphi$$

  where $a \in \mathcal{A}$.

- We say $s \models \omega$, if $\omega \in \gamma(s)$ (or $\gamma(s, \omega) = T$).
  We say $s \models (a)\varphi$ if $\delta(s, a) \models \varphi$.

- Now we define $[\![\varphi]\!]_{\mathcal{M}} = \{s \in S | s \models \varphi\}$.

An Equivalence Relation on Formulas

- We write sa as shorthand for $\delta(s, a)$.
- Define $\sim_{\mathcal{M}}$ between *formulas* as $\varphi \sim_{\mathcal{M}} \psi$ if $[\![\varphi]\!]_{\mathcal{M}} = [\![\psi]\!]_{\mathcal{M}}$.

An Equivalence Relation on Formulas

- We write sa as shorthand for $\delta(s, a)$.
- Define $\sim_{\mathcal{M}}$ between *formulas* as $\varphi \sim_{\mathcal{M}} \psi$ if $[\![\varphi]\!]_{\mathcal{M}} = [\![\psi]\!]_{\mathcal{M}}$.
- Equivalence class for φ same as of states $[\![\varphi]\!]_{\mathcal{M}}$ that satisfy φ.

A Dual Automaton

- Given a finite automaton $\mathcal{M} = (S, \mathcal{A}, \mathcal{O}, \delta, \gamma)$.
  Let T be the set of $\sim_{\mathcal{M}}$-equivalence classes of formulas on $\mathcal{M}$.
- We define $\mathcal{M}' = (S', \mathcal{A}, \mathcal{O}', \delta', \gamma')$ as follows:
- $S' = T = \{[\![\varphi]\!]_{\mathcal{M}}\}$
- $\mathcal{O}' = S$
- $\delta'([\![\varphi]\!]_{\mathcal{M}}, a) = [\![(a)\varphi]\!]_{\mathcal{M}}$
- $\gamma'([\![\varphi]\!]_{\mathcal{M}}) = [\![\varphi]\!]_{\mathcal{M}}$ or $\gamma'([\![\varphi]\!]_{\mathcal{A}}, s) = (s \models \varphi)$.

The intuition

Interchange states and observations.

Minimality Properties

- In general, the double dual is the minimal machine with the same behaviour!

Minimality Properties

- In general, the double dual is the minimal machine with the same behaviour!
- For deterministic machines bisimulation is the same as trace equivalence.

Minimality Properties

- In general, the double dual is the minimal machine with the same behaviour!
- For deterministic machines bisimulation is the same as trace equivalence.
- This gives an intuition for why Brzozowski's algorithm works,

Minimality Properties

- In general, the double dual is the minimal machine with the same behaviour!
- For deterministic machines bisimulation is the same as trace equivalence.
- This gives an intuition for why Brzozowski's algorithm works,
- but it does not really address the role of reachability properly.

Probabilistic systems

- In joint work with Chris Hundt, Joelle Pineau and Doina Precup (AAAI 2006): duals for various kinds of probabilistic transition systems like probabilistic Moore automata and partially observable Markov decision processes.

Probabilistic systems

- In joint work with Chris Hundt, Joelle Pineau and Doina Precup (AAAI 2006): duals for various kinds of probabilistic transition systems like probabilistic Moore automata and partially observable Markov decision processes.
- Dual automaton from tests: probabilistic analogues of modal formulas.

Probabilistic systems

- In joint work with Chris Hundt, Joelle Pineau and Doina Precup (AAAI 2006): duals for various kinds of probabilistic transition systems like probabilistic Moore automata and partially observable Markov decision processes.
- Dual automaton from tests: probabilistic analogues of modal formulas.
- Main point: not minimization, but can learn systems from data even when the state is not directly observable

Probabilistic systems

- In joint work with Chris Hundt, Joelle Pineau and Doina Precup (AAAI 2006): duals for various kinds of probabilistic transition systems like probabilistic Moore automata and partially observable Markov decision processes.
- Dual automaton from tests: probabilistic analogues of modal formulas.
- Main point: not minimization, but can learn systems from data even when the state is not directly observable
- because the double-dual serves as a substitute for the original machine.

Application to learning

- One can plan when one has the model: value iteration etc.,

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.
- In the absence of a model, one is forced to learn from data.

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.
- In the absence of a model, one is forced to learn from data.
- Learning is hopeless when one has no idea what the state space is.

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.
- In the absence of a model, one is forced to learn from data.
- Learning is hopeless when one has no idea what the state space is.
- There should be no such thing as absolute state!

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.
- In the absence of a model, one is forced to learn from data.
- Learning is hopeless when one has no idea what the state space is.
- There should be no such thing as absolute state!
- State is just a summary of past observations that can be used to make predictions.

Application to learning

- One can plan when one has the model: value iteration etc.,
- but quite often one does not have the model.
- In the absence of a model, one is forced to learn from data.
- Learning is hopeless when one has no idea what the state space is.
- There should be no such thing as absolute state!
- State is just a summary of past observations that can be used to make predictions.
- Double dual: state can be regarded as the summary of the outcomes of experiments.

What is the right categorical description?

- Is this is any kind of familiar Stone-type duality?

What is the right categorical description?

- Is this is any kind of familiar Stone-type duality?
- We know that machines are co-algebras and logics are algebras but

What is the right categorical description?

- Is this is any kind of familiar Stone-type duality?
- We know that machines are co-algebras and logics are algebras but
- why is the dual another automaton?

Automata as Coalgebras

- Our automata are coalgebras of the following functor:

$$F(S) = S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}}, \;\; F(f : S \to S') = \lambda\langle \alpha : \mathcal{A} \to S,\; O \subseteq \mathcal{O}\rangle.\langle f \circ \alpha,\; O\rangle.$$

Automata as Coalgebras

- Our automata are coalgebras of the following functor:

$$F(S) = S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}}, \;\; F(f : S \to S') = \lambda\langle \alpha : \mathcal{A} \to S, \; O \subseteq \mathcal{O}\rangle.\langle f \circ \alpha, \; O\rangle.$$

- The category of these coalgebras is called **PODFA**.

Automata as Coalgebras

- Our automata are coalgebras of the following functor:

$$F(S) = S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}}, \;\; F(f : S \to S') = \lambda\langle \alpha : \mathcal{A} \to S, \; O \subseteq \mathcal{O}\rangle.\langle f \circ \alpha, \; O\rangle.$$

- The category of these coalgebras is called **PODFA**.
- In ordinary language they are quintuples

$$(S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \to S, \gamma : S \to \mathcal{O})$$

Automata as Coalgebras

- Our automata are coalgebras of the following functor:

$$F(S) = S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}}, \;\; F(f : S \to S') = \lambda\langle\alpha : \mathcal{A} \to S, \; O \subseteq \mathcal{O}\rangle.\langle f \circ \alpha, \; O\rangle.$$

- The category of these coalgebras is called **PODFA**.
- In ordinary language they are quintuples

$$(S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \to S, \gamma : S \to \mathcal{O})$$

- with S: states, $\mathcal{A}$: actions, $\mathcal{O}$: observations, δ is a transition function and γ is an observation function.

Automata as Coalgebras

- Our automata are coalgebras of the following functor:

$$F(S) = S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}}, \quad F(f : S \to S') = \lambda\langle \alpha : \mathcal{A} \to S,\ O \subseteq \mathcal{O}\rangle.\langle f \circ \alpha,\ O\rangle.$$

- The category of these coalgebras is called **PODFA**.
- In ordinary language they are quintuples

$$(S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \to S, \gamma : S \to \mathcal{O})$$

- with S: states, $\mathcal{A}$: actions, $\mathcal{O}$: observations, δ is a transition function and γ is an observation function.
- They are well known as Moore machines.

Homomorphisms

A homomorphism for these coalgebras is a function $f : S \to S'$ such that the following diagram commutes:

$$\begin{tikzcd} S \arrow[r, "f"] \arrow[d, "\langle \delta,\ \gamma \rangle"'] & S' \arrow[d, "\langle \delta',\ \gamma' \rangle"] \\ S^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}} \arrow[r, "f^{\mathcal{A}} \times \mathrm{id}"'] & S'^{\mathcal{A}} \times \mathbf{2}^{\mathcal{O}} \end{tikzcd}$$

where $f^{\mathcal{A}}(\alpha) = f \circ \alpha$.

This translates to the following conditions:

$$\forall s \in S, \omega \in \mathcal{O},\ \omega \in \gamma(s) \iff \omega \in \gamma'(f(s)) \tag{1}$$

and

$$\forall s \in S, a \in \mathcal{A},\ f(\delta(s,a)) = \delta'(f(s), a). \tag{2}$$

The Dual Category

- The category of **finite boolean algebras with operators** ($\mathbf{FBAO}$) has as objects finite boolean algebras B with

The Dual Category

- The category of **finite boolean algebras with operators** ($\mathbf{FBAO}$) has as objects finite boolean algebras B with
- the usual operations $\wedge$, $\neg$ and constants T and $\perp$ and, in addition,

The Dual Category

- The category of **finite boolean algebras with operators** ($\mathbf{FBAO}$) has as objects finite boolean algebras B with
- the usual operations $\wedge$, $\neg$ and constants T and $\perp$ and, in addition,
- together with unary operators (a) and constants $\underline{\omega}$.

The Dual Category

- The category of **finite boolean algebras with operators** ($\mathbf{FBAO}$) has as objects finite boolean algebras B with
- the usual operations $\wedge$, $\neg$ and constants T and $\perp$ and, in addition,
- together with unary operators (a) and constants $\underline{\omega}$.
- We denote an object by $\mathcal{B} = (B, \{(a)|a \in \mathcal{A}\}, \{\underline{\omega}|\omega \in \mathcal{O}\}, \mathsf{T}, \wedge, \neg)$.

Equations

The following three equations hold:

$$(a)(b_1 \wedge b_2) = (a)b_1 \wedge (a)b_2, \tag{3a}$$

$$(a)\mathsf{T} = \mathsf{T}, \tag{3b}$$

$$\neg(a)\neg b = (a)b. \tag{3c}$$

Morphisms

The morphisms are the usual boolean homomorphisms preserving, in addition, the constants and the unary operators.

Duality Theorem

There is a dual equivalence of categories

$$\mathbf{PODFA}^{op} \cong \mathbf{FBAO}.$$

One functor $\mathcal{P}$ is just the contravariant power set functor and the other one $\mathcal{H}$ maps a boolean algebra to its set of atoms.

Minimization?

- Obviously, if we have an equivalence of categories we get the same machine back when we go back and forth.

Minimization?

- Obviously, if we have an equivalence of categories we get the same machine back when we go back and forth.
- So how do we explain the minimization?

Definable Subsets

Define a logic $\mathcal{L}$ by

$$\phi ::== \mathsf{T} | \bot | \phi_1 \wedge \phi_2 | \neg\phi | (a)\phi | \underline{\omega}$$

and define the **definable subsets** $\mathcal{D}(S)$ of a machine $\mathcal{M} = (S, \delta, \gamma)$ as sets of the form $[\![\phi]\!]$.

- $\mathcal{D}(S)$ is a subobject of $\mathcal{P}(\mathcal{M})$

- $\mathcal{D}(S)$ is a subobject of $\mathcal{P}(\mathcal{M})$
- in fact it is the *smallest* possible subalgebra and

- $\mathcal{D}(S)$ is a subobject of $\mathcal{P}(\mathcal{M})$
- in fact it is the *smallest* possible subalgebra and
- any other subalgebra must contain $\mathcal{D}(S)$.

In Pictures

$$\mathcal{M} \dashrightarrow \mathcal{P}(\mathcal{M})$$

$$
\begin{tikzcd}
\mathcal{M} \arrow[r, dotted] & \mathcal{P}(\mathcal{M}) \\
& D(S) \arrow[u, hook, "1"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathcal{M} \arrow[r, dotted] \arrow[d, two heads, "2"'] & \mathcal{P}(\mathcal{M}) \\
\mathcal{H}(D(S)) & D(S) \arrow[l, dotted] \arrow[u, hook, "1"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \mathcal{M} \arrow[r, dotted] \arrow[d, two heads, "2"'] \arrow[ddl, "3"'] & \mathcal{P}(\mathcal{M}) & \\
 & \mathcal{H}(D(S)) & D(S) \arrow[l, dotted] \arrow[u, hook, "1"'] & \\
\mathcal{S}' \arrow[rrr, dotted, "4"'] & & & \mathcal{P}(\mathcal{S}')
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \mathcal{M} \arrow[r, dotted] \arrow[d, two heads, "2"'] \arrow[ddl, "3"'] & \mathcal{P}(\mathcal{M}) & \\
 & \mathcal{H}(D(S)) & D(S) \arrow[l, dotted] \arrow[u, hook, "1"'] & \\
\mathcal{S}' \arrow[rrr, dotted, "4"'] & & & \mathcal{P}(\mathcal{S}') \arrow[uul, hook, "5"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \mathcal{M} \arrow[r, dotted] \arrow[d, two heads, "2"] \arrow[ddl, "3"'] & \mathcal{P}(\mathcal{M}) & \\
 & \mathcal{H}(D(S)) & D(S) \arrow[l, dotted] \arrow[u, hook, "1"'] \arrow[dr, hook, "6"'] & \\
\mathcal{S}' \arrow[rrr, dotted, "4"'] & & & \mathcal{P}(\mathcal{S}') \arrow[uul, hook, "5"']
\end{tikzcd}
$$

The Secret of Minimization

$$
\begin{tikzcd}
 & \mathcal{M} \arrow[rr, dotted] \arrow[d, two heads, "2"'] \arrow[dl, two heads, "3"'] & & \mathcal{P}(\mathcal{M}) & \\
 & \mathcal{H}(D(S)) & & D(S) \arrow[ll, dotted] \arrow[u, hook, "1"'] \arrow[dr, hook, "6"'] & \\
\mathcal{S}' \arrow[ur, two heads, "7"'] \arrow[rrrr, dotted, "4"'] & & & & \mathcal{P}(\mathcal{S}') \arrow[uul, hook, "5"']
\end{tikzcd}
$$

A Simpler Logic

- Why did the minimization work with just the logic

$$\phi ::== \underline{\omega} | (a)\phi?$$

A Simpler Logic

- Why did the minimization work with just the logic

$$\phi ::== \underline{\omega} | (a)\phi?$$

- With this logic the definable subsets $E(S)$ do not form a boolean algebra,

A Simpler Logic

- Why did the minimization work with just the logic

$$\phi ::== \underline{\omega} | (a)\phi ?$$

- With this logic the definable subsets $E(S)$ do not form a boolean algebra,
- it is just a "set with operations"

A Simpler Logic

- Why did the minimization work with just the logic

$$\phi ::== \underline{\omega} | (a)\phi?$$

- With this logic the definable subsets $E(S)$ do not form a boolean algebra,
- it is just a "set with operations"
- in other words, it can be viewed as an automaton!

Why the simpler logic works

- For deterministic automata we can flatten formulas like $(a)(\omega_1 \wedge (b)\omega_2)$ to $(a)\omega_1 \wedge (a)(b)\omega_2$.

Why the simpler logic works

- For deterministic automata we can flatten formulas like $(a)(\omega_1 \wedge (b)\omega_2)$ to $(a)\omega_1 \wedge (a)(b)\omega_2$.
- Thus for **deterministic** automata the boolean algebra generated by $E(S)$ is just the same as $D(S)$ so the minimization picture works with boolean algebra generated by $E(S)$.

Why the simpler logic works

- For deterministic automata we can flatten formulas like $(a)(\omega_1 \wedge (b)\omega_2)$ to $(a)\omega_1 \wedge (a)(b)\omega_2$.
- Thus for **deterministic** automata the boolean algebra generated by $E(S)$ is just the same as $D(S)$ so the minimization picture works with boolean algebra generated by $E(S)$.
- For nondeterministic automata the story is different.

Minimality = fewest states?

- The minimal machines defined above are really the “most quotiented versions” of a system.

Minimality = fewest states?

- The minimal machines defined above are really the “most quotiented versions” of a system.
- To really get the system with the fewest states one needs to deal with reachability.

Minimality = fewest states?

- The minimal machines defined above are really the “most quotiented versions” of a system.
- To really get the system with the fewest states one needs to deal with reachability.
- The following discussion is a rapid version of what Jan Rutten discussed in his beautiful MFPS talk on Monday.

An automaton in diagrams

$$\begin{tikzcd} 1 \arrow[dr, "i"] & & 2 \\ & S \arrow[ur, "f"] \arrow[d, "\delta"] & \\ & S^A & \end{tikzcd}$$

- Here S is the state space, A is the set of actions, 1 is the one-element set and 2 is a two-element set.
- The map i defines an initial state and f defines a set of final states. I will write i for the map and for the initial state itself.
- the transition function $\delta : S \times A \to S$ has been written as $\delta : S \to S^A$.
- There is a natural extension $\delta^* : S \to S^{A^*}$.

A very special (infinite) automaton

$$
\begin{tikzcd}
1 \arrow[d, "\varepsilon"] \\
A^* \arrow[d, "\alpha"] \\
(A^*)^A
\end{tikzcd}
$$

- This automaton has all words as its state space.
- The initial state is the empty word ε.
- The transition function α acts by $\alpha(w) = \lambda a : A.w \cdot a$.
- We do not bother to define "final" states in this machine.

Reachability

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"] \arrow[dr, "i"] & \\
A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, dashed, "r^A"] & S^A
\end{tikzcd}$$

- Given any function between sets $f : V \to W$, we have a map $f^A : V^A \to W^A$, given by $f^A(\phi) = \lambda a : A.f(\phi(a)) = f \circ \phi$.

Reachability

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"] \arrow[dr, "i"] & \\
A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, dashed, "r^A"] & S^A
\end{tikzcd}$$

- Given any function between sets $f : V \to W$, we have a map $f^A : V^A \to W^A$, given by $f^A(\phi) = \lambda a : A.f(\phi(a)) = f \circ \phi$.
- There is a *unique* map $r : A^* \to S$ such that $r(\varepsilon) = i$ and $\delta(r(w))(a) = r(w \cdot a)$, which can easily be defined inductively.

Reachability

$$\begin{tikzcd} 1 \arrow[d, "\varepsilon"] \arrow[dr, "i"] & \\ A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"] & S \arrow[d, "\delta"] \\ (A^*)^A \arrow[r, dashed, "r^A"] & S^A \end{tikzcd}$$

- Given any function between sets $f : V \to W$, we have a map $f^A : V^A \to W^A$, given by $f^A(\phi) = \lambda a : A.f(\phi(a)) = f \circ \phi$.
- There is a *unique* map $r : A^* \to S$ such that $r(\varepsilon) = i$ and $\delta(r(w))(a) = r(w \cdot a)$, which can easily be defined inductively.
- The image of A^* under r is exactly the reachable subset of S.

Reachability

$$
\begin{tikzcd}
1 \arrow[d, "\varepsilon"] \arrow[dr, "i"] & \\
A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, dashed, "r^A"] & S^A
\end{tikzcd}
$$

- Given any function between sets $f : V \to W$, we have a map $f^A : V^A \to W^A$, given by $f^A(\phi) = \lambda a : A.f(\phi(a)) = f \circ \phi$.
- There is a *unique* map $r : A^* \to S$ such that $r(\varepsilon) = i$ and $\delta(r(w))(a) = r(w \cdot a)$, which can easily be defined inductively.
- The image of A^* under r is exactly the reachable subset of S.
- The entire state space is *reachable* exactly when r is a surjection.

Reachability

$$
\begin{tikzcd}
1 \arrow[d, "\varepsilon"] \arrow[dr, "i"] & \\
A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, dashed, "r^A"] & S^A
\end{tikzcd}
$$

- Given any function between sets $f : V \to W$, we have a map $f^A : V^A \to W^A$, given by $f^A(\phi) = \lambda a : A.f(\phi(a)) = f \circ \phi$.
- There is a *unique* map $r : A^* \to S$ such that $r(\varepsilon) = i$ and $\delta(r(w))(a) = r(w \cdot a)$, which can easily be defined inductively.
- The image of A^* under r is exactly the reachable subset of S.
- The entire state space is *reachable* exactly when r is a surjection.
- Note, final states play no role.

Another very special infinite automaton

$$\begin{tikzcd} 2 \\ 2^{A^*} \arrow[u, "\varepsilon?"] \arrow[d, "\beta"'] \\ (2^{A^*})^A \end{tikzcd}$$

- This automaton has all *languages* as its state space.

Another very special infinite automaton

$$\begin{tikzcd} 2 \\ 2^{A^*} \arrow[u, "\varepsilon?"] \arrow[d, "\beta"'] \\ (2^{A^*})^A \end{tikzcd}$$

- This automaton has all *languages* as its state space.
- The final states *contain* the empty word ε.

Another very special infinite automaton

$$\begin{tikzcd} 2 \\ 2^{A^*} \arrow[u, "\varepsilon?"] \arrow[d, "\beta"'] \\ (2^{A^*})^A \end{tikzcd}$$

- This automaton has all *languages* as its state space.
- The final states *contain* the empty word ε.
- The transition function β acts by $\beta(L)(a) = \{w \in A^* \mid a \cdot w \in L\}$; the (left) a-derivative of L.

Another very special infinite automaton

$$\begin{tikzcd} 2 \\ 2^{A^*} \arrow[u, "\varepsilon?"] \arrow[d, "\beta"'] \\ (2^{A^*})^A \end{tikzcd}$$

- This automaton has all *languages* as its state space.
- The final states *contain* the empty word ε.
- The transition function β acts by $\beta(L)(a) = \{w \in A^* \mid a \cdot w \in L\}$; the (left) a-derivative of L.
- We do not bother to define an "initial" state in this machine.

Observability

$$
\begin{tikzcd}
 & 2 \\
S \arrow[ur, "f"] \arrow[r, dashed, "o"] \arrow[d, "\delta"'] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

- Here o is the map that takes a state to the language recognized starting from that state.

Observability

$$\begin{tikzcd}
 & 2 \\
S \arrow[ur, "f"] \arrow[r, dashed, "o"] \arrow[d, "\delta"'] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

- Here o is the map that takes a state to the language recognized starting from that state.
- It is the unique map making the upper triangle and the lower square commute.

Observability

$$
\begin{tikzcd}
 & 2 \\
S \arrow[ur, "f"] \arrow[r, dashed, "o"] \arrow[d, "\delta"'] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

- Here o is the map that takes a state to the language recognized starting from that state.
- It is the unique map making the upper triangle and the lower square commute.
- Think of o as giving the observable behaviour of a state.

Observability

$$
\begin{tikzcd}
 & 2 \\
S \arrow[ur, "f"] \arrow[r, dashed, "o"] \arrow[d, "\delta"'] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

- Here o is the map that takes a state to the language recognized starting from that state.
- It is the unique map making the upper triangle and the lower square commute.
- Think of o as giving the observable behaviour of a state.
- A machine is *observable* exactly when distinct states recognize different languages, i.e. when o is an injection.

The butterfly

$$
\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"] \arrow[d, "\alpha"'] & S \arrow[r, dashed, "o"] \arrow[d, "\delta"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & S^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

A deterministic automaton (S, δ, i, f) is minimal if it is both reachable and observable.

The power-set functor

Given sets U, V and a function $f : U \to V$ we define

$$\mathcal{P}(f) : \mathcal{P}(V) \to \mathcal{P}(U)$$

by

$$\mathcal{P}(f)(P \subseteq V) = f^{-1}(P).$$

Reverse functorially

$$
\begin{array}{c||c|c||c}
S & S \times A & 2^{S\times A} & (2^S)^A \\
\delta \big\downarrow & \big\downarrow & \big\uparrow & \big\uparrow 2^{\delta} \\
S^A & S & 2^S & 2^S
\end{array}
$$

- The power-set functor produces the reversed determinized automaton.

Reverse functorially

$$
\begin{array}{c|c|c|c}
S & S \times A & 2^{S\times A} & (2^S)^A \\
\big\downarrow{\scriptstyle \delta} & \big\downarrow & \big\uparrow & \big\uparrow{\scriptstyle 2^{\delta}} \\
S^A & S & 2^S & 2^S
\end{array}
$$

- The power-set functor produces the reversed determinized automaton.
- Initial becomes final under powerset. The final state $S \to \mathbf{2}$ becomes the new initial state by observing that such a function is the same thing as a subset.

Reverse functorially

$$\begin{array}{c||c|c||c}
S & S \times A & 2^{S \times A} & (2^S)^A \\
{\scriptstyle\delta}\downarrow & \downarrow & \uparrow & \uparrow{\scriptstyle 2^{\delta}} \\
S^A & S & 2^S & 2^S
\end{array}$$

- The power-set functor produces the reversed determinized automaton.
- Initial becomes final under powerset. The final state $S \to \mathbf{2}$ becomes the new initial state by observing that such a function is the same thing as a subset.
- It makes reachable into observable, *but not vice versa*.

Why Brzozowski's algorithm works

Theorem

If (S, δ, i, f) is a reachable deterministic automaton accepting L, then $(2^S, 2^\delta, f, 2^i)$ is an observable deterministic automaton accepting $rev(L)$.

If, we take its reachable part again and reverse it again we again get an observable automaton this time recognizing L. If we take the reachable part we get a minimal automaton recognizing L.

Abstract nonsense?

- Channeling my inner Moshe:

Abstract nonsense?

- Channeling my inner Moshe:
- "Surely, this is categorical mumbo-jumbo for something that can be explained simply!"

No, it is generalized abstract nonsense!

- Exactly the same construction can be used in other settings by just changing the duality at work.

No, it is generalized abstract nonsense!

- Exactly the same construction can be used in other settings by just changing the duality at work.
- Moore automata work by changing the functor slightly.

No, it is generalized abstract nonsense!

- Exactly the same construction can be used in other settings by just changing the duality at work.
- Moore automata work by changing the functor slightly.
- Kleene algebra with tests.

No, it is generalized abstract nonsense!

- Exactly the same construction can be used in other settings by just changing the duality at work.
- Moore automata work by changing the functor slightly.
- Kleene algebra with tests.
- Weighted automata (i.e. automata over vector spaces) can be minimized by using the same idea with the self duality of vector spaces.

No, it is generalized abstract nonsense!

- Exactly the same construction can be used in other settings by just changing the duality at work.
- Moore automata work by changing the functor slightly.
- Kleene algebra with tests.
- Weighted automata (i.e. automata over vector spaces) can be minimized by using the same idea with the self duality of vector spaces.
- Belief automata can be minimized using Gelfand duality.

What is Gelfand duality?

- Problem in undergrad algebra courses: Given the ring of continuous real-valued functions defined on a compact Hausdorff space X, call this $C(X)$, show that for every maximal ideal M there is an $x \in X$ such that

$$M = \{f \in C(X) \mid f(x) = 0\}.$$

What is Gelfand duality?

- Problem in undergrad algebra courses: Given the ring of continuous real-valued functions defined on a compact Hausdorff space X, call this $C(X)$, show that for every maximal ideal M there is an $x \in X$ such that

$$M = \{f \in C(X) \mid f(x) = 0\}.$$

- In short, the points of the space can be reconstructed from the algebraic structure of the ring.

What is Gelfand duality?

- Problem in undergrad algebra courses: Given the ring of continuous real-valued functions defined on a compact Hausdorff space X, call this $C(X)$, show that for every maximal ideal M there is an $x \in X$ such that

$$M = \{f \in C(X) \mid f(x) = 0\}.$$

- In short, the points of the space can be reconstructed from the algebraic structure of the ring.
- In fact, one can even get the topology.

What is Gelfand duality?

- Problem in undergrad algebra courses: Given the ring of continuous real-valued functions defined on a compact Hausdorff space X, call this $C(X)$, show that for every maximal ideal M there is an $x \in X$ such that
$$M = \{f \in C(X) \mid f(x) = 0\}.$$
- In short, the points of the space can be reconstructed from the algebraic structure of the ring.
- In fact, one can even get the topology.
- $C(X)$ is more than a ring it is a *commutative, unital* C^*-algebra.

What is Gelfand duality?

- Problem in undergrad algebra courses: Given the ring of continuous real-valued functions defined on a compact Hausdorff space X, call this $C(X)$, show that for every maximal ideal M there is an $x \in X$ such that

$$M = \{f \in C(X) \mid f(x) = 0\}.$$

- In short, the points of the space can be reconstructed from the algebraic structure of the ring.
- In fact, one can even get the topology.
- $C(X)$ is more than a ring it is a *commutative, unital* C^*-algebra.
- Incidentally, this works just as well with complex-valued functions.

C^*-algebras

- A (complex) C^*-algebra is a (complex) vector space with
- an associative multiplication (satisfying obvious laws)
- and a norm $\|\cdot\|$ with respect to which it is complete (hence a Banach space).
- The norm satisfies: $\|a \cdot b\| \leq \|a\| \cdot \|b\|$.
- There is also an involution $*$ satisfying $(ab)^* = b^* a^*$ and $(\alpha a)^* = \overline{\alpha} a^*$.
- The crucial property is:
- $\|a^* a\| = \|a\|^2$.
- Morphisms are homomorphisms preserving the $*$.
- We say that A is *unital* if there is a unit element for the multiplication.

Gelfand duality

The category of commutative unital C^*-algebras is dually equivalent to the category of compact Hausdorff spaces.

It does not matter if the C^* algebras are complex (Gelfand) or real (Stone); though the proofs are very different.

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0,1], \gamma : S \times \mathcal{O} \to [0,1]).$$

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0,1], \gamma : S \times \mathcal{O} \to [0,1]).$$

- Given such an automaton one often works with an automaton whose state space is the set of distributions over S: the so-called *belief automaton*.

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0, 1], \gamma : S \times \mathcal{O} \to [0, 1]).$$

- Given such an automaton one often works with an automaton whose state space is the set of distributions over S: the so-called *belief automaton*.
- It is a deterministic automaton with probabilistic observations.

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0,1], \gamma : S \times \mathcal{O} \to [0,1]).$$

- Given such an automaton one often works with an automaton whose state space is the set of distributions over S: the so-called *belief automaton*.
- It is a deterministic automaton with probabilistic observations.
- If S is finite then the space is distributions is compact Hausdorff.

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0,1], \gamma : S \times \mathcal{O} \to [0,1]).$$

- Given such an automaton one often works with an automaton whose state space is the set of distributions over S: the so-called *belief automaton*.
- It is a deterministic automaton with probabilistic observations.
- If S is finite then the space is distributions is compact Hausdorff.
- So we are dealing with automata over compact Hausdorff spaces.

Belief automata

- A **probabilistic automaton with observations** is

$$\mathcal{F} = (S, \mathcal{A}, \mathcal{O}, \delta : S \times \mathcal{A} \times S \to [0,1], \gamma : S \times \mathcal{O} \to [0,1]).$$

- Given such an automaton one often works with an automaton whose state space is the set of distributions over S: the so-called *belief automaton*.
- It is a deterministic automaton with probabilistic observations.
- If S is finite then the space is distributions is compact Hausdorff.
- So we are dealing with automata over compact Hausdorff spaces.
- Minimization via Stone duality $\longrightarrow$ minimization via Gelfand duality.

Conclusions

- Duality tells one how to move between logics and transition systems.

Conclusions

- Duality tells one how to move between logics and transition systems.
- Completeness theorems, which typically work by constructing transition systems from consistent sets of formulas embody a key aspect of duality results *but*,

Conclusions

- Duality tells one how to move between logics and transition systems.
- Completeness theorems, which typically work by constructing transition systems from consistent sets of formulas embody a key aspect of duality results *but*,
- the arrow part of the duality is crucial for proving our minimization results.

Ongoing and Future Work

- Unify the BKP picture with the BBHPRS picture: BBBHKKPRS unified picture in progress.

Ongoing and Future Work

- Unify the BKP picture with the BBHPRS picture: BBBHKKPRS unified picture in progress.
- Metric analogue of Stone duality: Mardare and Kozen.

Ongoing and Future Work

- Unify the BKP picture with the BBHPRS picture: BBBHKKPRS unified picture in progress.
- Metric analogue of Stone duality: Mardare and Kozen.
- Pressing research topic of great interest in quantum information theory: what is the duality theory for *non-commutative* C^*-algebras?: Tobias Fritz.

Thank you!