Minimization via Duality

Prakash Panangaden
School of Computer Science
McGill University

Verification, Model Checking and Abstract Interpretation
San Diego
January 2014

Relevant papers

- Canonical regular expressions and minimal state graphs for definite events, by Jan Brzozowski in *Mathematical Theory of Automata*, 1962.
- Brzozowski's algorithm co-algebraically, by Bonchi, Bonsangue, Rutten and Silva, in Kozen Festschrift, Lecture Notes in Computer Science 7230, 2012.
- Minimization via duality: Bezahanishvili, Kupke and P.; proceedings of WoLLIC 2012, Lecture Notes in Computer Science 7456.
- Algebra-coalgebra duality in Brzozowski's minimization algorithm: Bonchi, Bonsangue, Hansen, P., Rutten and Silva, ACM Transactions on Computational Logic 2013.
- Longer paper with above authors plus Bezhanishvili, Kozen and Kupke in preparation.

Brzozowski's algorithm

- Given a DFA M recognizing L:

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,
- flip the roles of initial and final states,

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,
- flip the roles of initial and final states,
- determinize it,

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,
- flip the roles of initial and final states,
- determinize it,
- take the reachable part of the reversed determinized machine to obtain M',

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,
- flip the roles of initial and final states,
- determinize it,
- take the reachable part of the reversed determinized machine to obtain M',
- repeat all the steps on M' to obtain M''.

Brzozowski's algorithm

- Given a DFA M recognizing L:
- reverse the arrows of M,
- flip the roles of initial and final states,
- determinize it,
- take the reachable part of the reversed determinized machine to obtain M',
- repeat all the steps on M' to obtain M''.
- M'' is the minimal automaton accepting L!

Examples of duality principles

- “and” vs “or” in propositional logic

Examples of duality principles

- "and" vs "or" in propositional logic
- Linear programming

Examples of duality principles

- "and" vs "or" in propositional logic
- Linear programming
- Electric and magnetic fields

Examples of duality principles

- “and” vs “or” in propositional logic
- Linear programming
- Electric and magnetic fields
- Controllability and observability in control theory: Kalman

Examples of duality principles

- “and” vs “or” in propositional logic
- Linear programming
- Electric and magnetic fields
- Controllability and observability in control theory: Kalman
- State-transformer and weakest-precondition semantics

Examples of duality principles

- "and" vs "or" in propositional logic
- Linear programming
- Electric and magnetic fields
- Controllability and observability in control theory: Kalman
- State-transformer and weakest-precondition semantics
- Forward and backward dataflow analyses

Examples of duality principles

- "and" vs "or" in propositional logic
- Linear programming
- Electric and magnetic fields
- Controllability and observability in control theory: Kalman
- State-transformer and weakest-precondition semantics
- Forward and backward dataflow analyses
- Induction and co-induction

What is Stone-type duality?

- Two types of structures: Foo and Bar.

What is Stone-type duality?

- Two types of structures: Foo and Bar.
- Every Foo has an associated Bar and vice versa.

What is Stone-type duality?

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

What is Stone-type duality?

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

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:

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

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 $\int f \mathrm{d}\mu$

Brzozowski's Algorithm 1962

- Start with DFA.

Brzozowski's Algorithm 1962

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

Brzozowski's Algorithm 1962

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

Brzozowski's Algorithm 1962

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

Brzozowski's Algorithm 1962

- 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 1962

- 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 1962

- 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.

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

$$\begin{array}{c} 1 \\ \downarrow{\scriptstyle \varepsilon} \\ A^* \\ \downarrow{\scriptstyle \alpha} \\ (A^*)^A \end{array}$$

- 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 construction

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 in terms of power-set

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

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

Reverse in terms of power-set

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

- The power-set construction produces the reversed determinized automaton.
- Initial becomes final under power-set. 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 in terms of power-set

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

- The power-set construction produces the reversed determinized automaton.
- Initial becomes final under power-set. 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?

- A standard reaction:

Abstract nonsense?

- A standard reaction:
- “Surely, this is abstract nonsense; 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.

Weighted automata

- These are essentially automata over vector spaces.

Weighted automata

- These are essentially automata over vector spaces.
- There are n states but the automaton can be in *any* linear combination of states: $\sum r_i s_i$, where r's are real numbers and s's are states. Transitions are matrices.

Weighted automata

- These are essentially automata over vector spaces.
- There are n states but the automaton can be in *any* linear combination of states: $\sum r_i s_i$, where r's are real numbers and s's are states. Transitions are matrices.
- $(V, \alpha, \{T_a\}_{a\in\Sigma}, \eta)$: V an n-dimensional vector space, T_a is a transition matrix for each $a \in \Sigma$, the alphabet.

Weighted automata

- These are essentially automata over vector spaces.
- There are n states but the automaton can be in *any* linear combination of states: $\sum r_i s_i$, where r's are real numbers and s's are states. Transitions are matrices.
- $(V, \alpha, \{T_a\}_{a\in\Sigma}, \eta)$: V an n-dimensional vector space, T_a is a transition matrix for each $a \in \Sigma$, the alphabet.
- $\alpha \in \mathbf{R}^n$ is an *initial* "state" and

Weighted automata

- These are essentially automata over vector spaces.
- There are n states but the automaton can be in *any* linear combination of states: $\sum r_i s_i$, where r's are real numbers and s's are states. Transitions are matrices.
- $(V, \alpha, \{T_a\}_{a \in \Sigma}, \eta)$: V an n-dimensional vector space, T_a is a transition matrix for each $a \in \Sigma$, the alphabet.
- $\alpha \in \mathbf{R}^n$ is an *initial* "state" and
- η is an observation function: an element of V^*, the dual space.

Weighted languages

- Given a string $w = ab\ldots c$ the final state is $\beta = T_c(\ldots T_b(T_a\alpha)\ldots)$.

Weighted languages

- Given a string $w = ab \ldots c$ the final state is $\beta = T_c(\ldots T_b(T_a\alpha)\ldots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.

Weighted languages

- Given a string $w = ab \dots c$ the final state is $\beta = T_c(\dots T_b(T_a\alpha)\dots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.
- $L(w) = \eta(\beta)$.

Weighted languages

- Given a string $w = ab\ldots c$ the final state is $\beta = T_c(\ldots T_b(T_a\alpha)\ldots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.
- $L(w) = \eta(\beta)$.
- What is the minimal automaton recognizing the same language?

Weighted languages

- Given a string $w = ab \dots c$ the final state is $\beta = T_c(\dots T_b(T_a \alpha) \dots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.
- $L(w) = \eta(\beta)$.
- What is the minimal automaton recognizing the same language?
- Computed in exactly the same way except that reversal here means, "take the dual automaton".

Weighted languages

- Given a string $w = ab \ldots c$ the final state is $\beta = T_c(\ldots T_b(T_a\alpha) \ldots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.
- $L(w) = \eta(\beta)$.
- What is the minimal automaton recognizing the same language?
- Computed in exactly the same way except that reversal here means, “take the dual automaton”.
- Natural matrix: $H[x, y] = L(xy)$: a Hankel matrix.

Weighted languages

- Given a string $w = ab\ldots c$ the final state is $\beta = T_c(\ldots T_b(T_a\alpha)\ldots)$.
- The automaton associates the number $\eta(\beta)$ with the string w.
- $L(w) = \eta(\beta)$.
- What is the minimal automaton recognizing the same language?
- Computed in exactly the same way except that reversal here means, "take the dual automaton".
- Natural matrix: $H[x, y] = L(xy)$: a Hankel matrix.
- The minimal automaton has size equal to the rank of the Hankel matrix.

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.
- $\mathcal{A}^* = (V^*, \eta, \{T_a^*\}, \alpha)$.

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.
- $\mathcal{A}^* = (V^*, \eta, \{T_a^*\}, \alpha)$.
- What about reachability?

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.
- $\mathcal{A}^* = (V^*, \eta, \{T_a^*\}, \alpha)$.
- What about reachability?
- we define the *forward space* as the span of $\{T_w(\alpha) \mid w \in \Sigma^*\}$: a subspace of V.

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.
- $\mathcal{A}^* = (V^*, \eta, \{T_a^*\}, \alpha)$.
- What about reachability?
- we define the *forward space* as the span of $\{T_w(\alpha) \mid w \in \Sigma^*\}$: a subspace of V.
- We define the *backward space* as the span of $\{T_w^*(\eta) \mid w \in \Sigma^*\}$: a subspace of V^*.

Dualizing a weighted automaton

- Given a weighted automaton $\mathcal{A} = (V = \mathbf{R}^n, \alpha, \{T_a\}, \eta)$,
- We construct the dual automaton $\mathcal{A}^*$ by using linear algebra duality.
- $\mathcal{A}^* = (V^*, \eta, \{T_a^*\}, \alpha)$.
- What about reachability?
- we define the *forward space* as the span of $\{T_w(\alpha) \mid w \in \Sigma^*\}$: a subspace of V.
- We define the *backward space* as the span of $\{T_w^*(\eta) \mid w \in \Sigma^*\}$: a subspace of V^*.

Forward and backward reduction

- Define $\vec{\mathcal{A}}$ as $\mathcal{A}$ projected onto the forward space.

Forward and backward reduction

- Define $\vec{\mathcal{A}}$ as $\mathcal{A}$ projected onto the forward space.
- Define $\overleftarrow{\mathcal{A}}$ as $\mathcal{A}^*$ projected onto the backward space.

Forward and backward reduction

- Define $\vec{\mathcal{A}}$ as $\mathcal{A}$ projected onto the forward space.
- Define $\overleftarrow{\mathcal{A}}$ as $\mathcal{A}^*$ projected onto the backward space.
- One can show $L_{\mathcal{A}} = L_{\vec{\mathcal{A}}} = L_{\overleftarrow{\mathcal{A}}}$.

Forward and backward reduction

- Define $\vec{\mathcal{A}}$ as $\mathcal{A}$ projected onto the forward space.
- Define $\overleftarrow{\mathcal{A}}$ as $\mathcal{A}^*$ projected onto the backward space.
- One can show $L_{\mathcal{A}} = L_{\vec{\mathcal{A}}} = L_{\overleftarrow{\mathcal{A}}}$.
- One can also show that $\overleftarrow{\vec{\mathcal{A}}}$ is the minimal automaton with the same (weighted) language as $\mathcal{A}$ by using the rank criterion.

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

- Metric analogue of Stone duality: Mardare and Kozen.

Ongoing and Future Work

- Metric analogue of Stone duality: Mardare and Kozen.
- Understand why the Brzozowski algorithm is often efficient.

Ongoing and Future Work

- Metric analogue of Stone duality: Mardare and Kozen.
- Understand why the Brzozowski algorithm is often efficient.
- Convex automata: exploit convex duality.

Thank you!