Inductive Logic Programming

Area Presentation & Future Research Directions

Merkouris Papamichail[†]

[†]Computer Science Department, University of Crete
Institute of Computer Science, FORTH

February, 2024

Presentation's Outline

Presentation's Outline

What is Induction?

"Inference of a generalized conclusion from particular instances."

– Marriam-Webster Dictionary

- The primary reasoning process employed in science.
- Scientists compose general *laws* about a *phenomenon*, from a limited set of *observations*.
- Mathematicians, draw conclusions from general, well accepted assumptions (the axioms, or postulates).
- The latter reasoning process is called *deduction*.
- Logic Programming, automates deductive reasoning.
- Inductive Logic Programming, automates inductive reasoning.

On the Certainty of Induction: Refutability Principle

- Deductive reasoning is *certain*.
- Mathematical axioms, are universally quantified:

$$\forall x P(x)$$

- Each observable event can be viewed as an *ground instantiation* of an axiom.
- Mathematical proofs preserve this certainty.
- Induction is *uncertain*.
- Infinite observations must be made for certainty to be guaranteed.

$$P(a) \land P(b) \land P(c) \land \ldots \models \forall x P(x)$$

- Instead, we demand for an induced theory to be *refutable*.
- *A theory is refutable if it can be logically contradicted by empirical observation*.

Query-based Model for Scientific Process

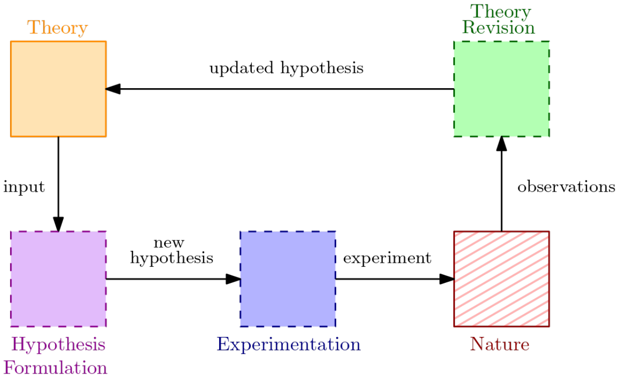


Figure 1: A query-based model for scientific reasoning.

Query-based Model for Scientific Process: Theory

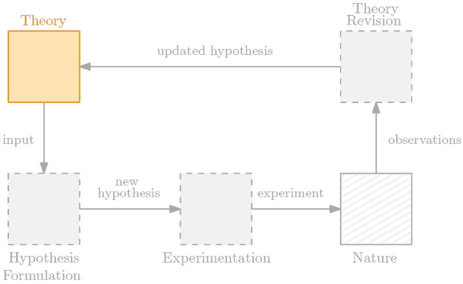


Figure 2: Our current theory.

- Our current theory, as a set of hypotheses.

Query-based Model for Scientific Process: Hypothesis-Formulation

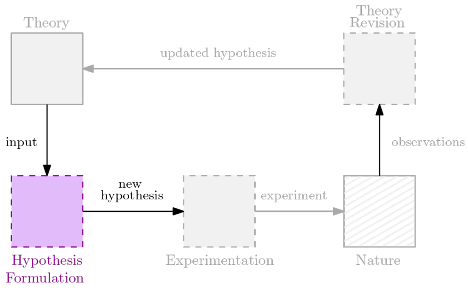


Figure 3: Hypothesis Formulation

- Our current theory, as a set of hypotheses.
- We formulate a new hypothesis to add to our theory.

Query-based Model for Scientific Process: Experimentation

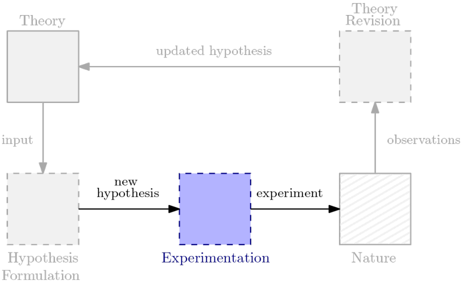


Figure 4: Experimentation

- Our current theory, as a set of hypotheses.
- We formulate a new hypothesis to add to our theory.
- We test our newly formulated hypothesis, by querying the *nature*.

Query-based Model for Scientific Process: Nature

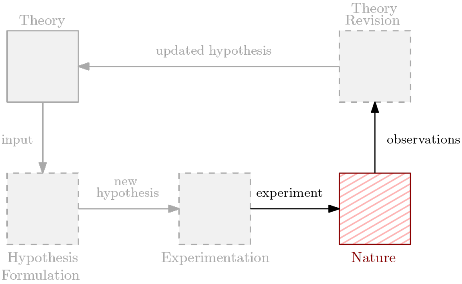


Figure 5: Nature

- Our current theory, as a set of hypotheses.
- We formulate a new hypothesis to add to our theory.
- We test our newly formulated hypothesis, by querying the *nature*.
- Nature is treated as an abstract oracle, providing hints about the validity of a hypothesis.

Query-based Model for Scientific Process: Theory Revision

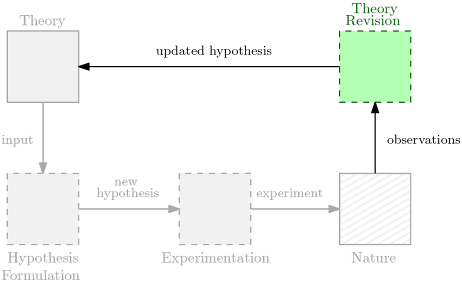


Figure 6: Theory Revision Mechanism

- Our current theory, as a set of hypotheses.
- We formulate a new hypothesis to add to our theory.
- We test our newly formulated hypothesis, by querying the *nature*.
- Nature is treated as an abstract oracle, providing hints about the validity of a hypothesis.
- Depending on the answers provided by nature we revise our hypothesis.

Query-based Model for Scientific Process: Conclusions

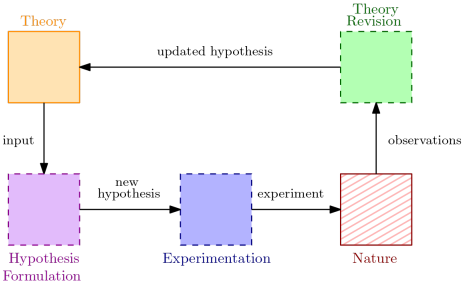


Figure 7: A query-based model for scientific reasoning.

We distinguish the following mechanisms in this process:

Query-based Model for Scientific Process: Conclusions

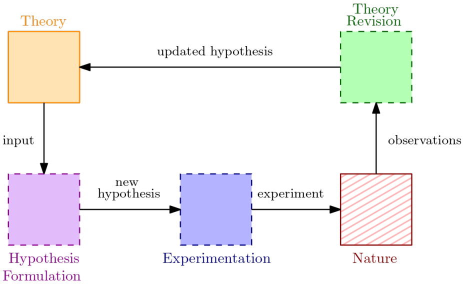


Figure 7: A query-based model for scientific reasoning.

We distinguish the following mechanisms in this process:

- Three reasoning algorithms:
  1. Hypothesis Formulation.
  2. Experimentation.
  3. Theory Revision.

Query-based Model for Scientific Process: Conclusions

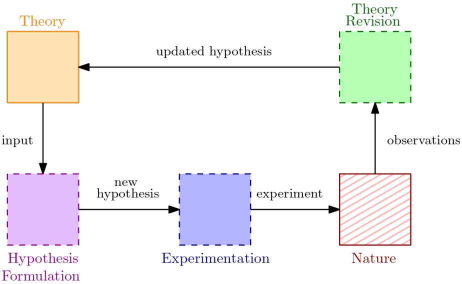


Figure 7: A query-based model for scientific reasoning.

We distinguish the following mechanisms in this process:

- Three reasoning algorithms:
  1. Hypothesis Formulation.
  2. Experimentation.
  3. Theory Revision.
- A storing mechanism, i.e., the "Theory".

Query-based Model for Scientific Process: Conclusions

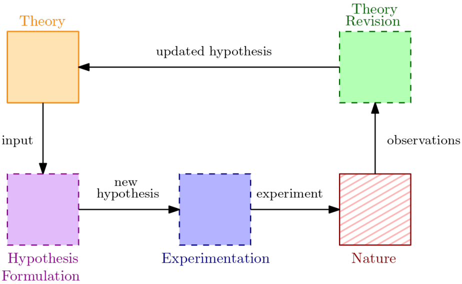


Figure 7: A query-based model for scientific reasoning.

We distinguish the following mechanisms in this process:

- Three reasoning algorithms:
  1. Hypothesis Formulation.
  2. Experimentation.
  3. Theory Revision.
- A storing mechanism, i.e., the "Theory".
- An oracle providing information about the world, i.e., "Nature".

Query-based Model for Scientific Process: Conclusions

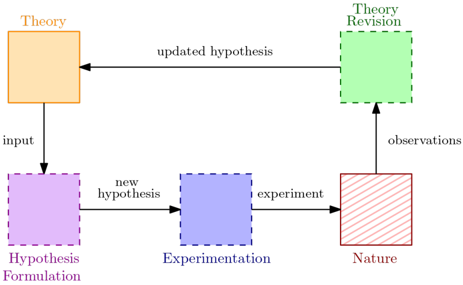


Figure 7: A query-based model for scientific reasoning.

We distinguish the following mechanisms in this process:

- Three reasoning algorithms:
  1. Hypothesis Formulation.
  2. Experimentation.
  3. Theory Revision.
- A storing mechanism, i.e., the "Theory".
- An oracle providing information about the world, i.e., "Nature".
- This (naive) abstract model is followed by both *human* and *automated* scientists.

Presentation's Outline

Partially Ordered Sets

▶ We encode induction & deduction as *traversal* problems in a partially ordered hypothesis space.

Definition

We say that a (finite) set $\mathcal{H}$ is *partially ordered* by a relation $\succeq \subseteq \mathcal{H} \times \mathcal{H}$, when $\succeq$ is *reflective*, *transitive*, and *antisymmetric*. We call the pair $\langle \mathcal{H}, \succeq \rangle$ a poset.

▶ Lattices are of special interest to us. A lattice is a poset where every two elements have both a *least upper bound* (lub) and a *greatest lower bound* (glb).

Lattices

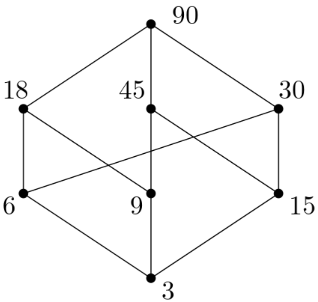


Figure 8: A lattice on natural numbers, organised by the *divides* relation $|$.

Lattices

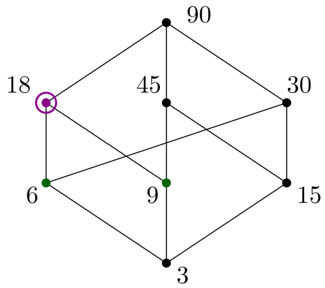


Figure 9: The lub corresponds to the *least common multiplier*.

- Least Upper Bound: $s \wedge t = \min\left[(\mathcal{H}_{s\preceq} \setminus s) \bigcap (\mathcal{H}_{t\preceq} \setminus t)\right]$.

Lattices

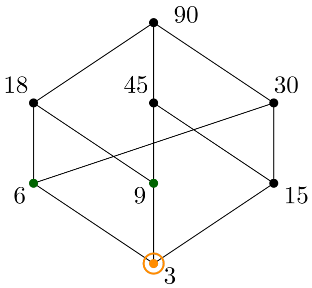


Figure 10: The glb corresponds to the *greatest common divisor*.

- Least Upper Bound: $s \wedge t = \min\left[(\mathcal{H}_{s\preceq} \setminus s) \bigcap (\mathcal{H}_{t\preceq} \setminus t)\right]$.
- Greatest Lower Bound: $s \vee t = \max\left[(\mathcal{H}_{s\succeq} \setminus s) \bigcap (\mathcal{H}_{t\succeq} \setminus t)\right]$

Refinement Operators

- Assume a lattice $\langle \mathcal{H}, \succeq \rangle$, we interpret the relation $\succeq$ as a relation.
- Namely, if $g \succeq s$, we say that g is more general than s, or g *explains* s.
- With $g \sqsupseteq s$ we denote the *immediate* successor g of s, with respect to $\succeq$.
- $\sqsupseteq$ constitutes the *transitive closure* of the $\succeq$ relation.

Refinement Operators

- Assume a lattice $\langle \mathcal{H}, \succeq \rangle$, we interpret the relation $\succeq$ as a relation.
- Namely, if $g \succeq s$, we say that g is more general than s, or g *explains* s.
- With $g \sqsupseteq s$ we denote the *immediate* successor g of s, with respect to $\succeq$.
- $\sqsupseteq$ constitutes the *transitive closure* of the $\succeq$ relation.

▶ For a single hypothesis space, we define the following operators:

- Specialization Operator: $\rho_s(h) = \{h' \in \mathcal{H} \mid h \sqsupseteq h'\}$
- Generalization Operator: $\rho_g(h) = \{h' \in \mathcal{H} \mid h \sqsubseteq h'\}$

Refinement Operators

- Assume a lattice $\langle \mathcal{H}, \succeq \rangle$, we interpret the relation $\succeq$ as a relation.
- Namely, if $g \succeq s$, we say that g is more general than s, or g *explains* s.
- With $g \sqsupseteq s$ we denote the *immediate* successor g of s, with respect to $\succeq$.
- $\sqsupseteq$ constitutes the *transitive closure* of the $\succeq$ relation.

▶ For a single hypothesis space, we define the following operators:

- Specialization Operator: $\rho_s(h) = \{h' \in \mathcal{H} \mid h \sqsupseteq h'\}$
- Generalization Operator: $\rho_g(h) = \{h' \in \mathcal{H} \mid h \sqsubseteq h'\}$

▶ For sets of hypotheses, we define the following operators:

- Specialization Operator:
  $\gamma_{\rho,s}(H) = \{(H \setminus h) \cup h' \mid h' \in \rho_s(h), h \in H\} \cup \{H \setminus h \mid h \in H\}$
- Generalization Operator:
  $\gamma_{\rho,g}(H) = \{(H \setminus h) \cup h' \mid h' \in \rho_g(h), h \in H\} \cup \{H \cup h \mid h \in \mathcal{H}\}$

Deductive Reasoning

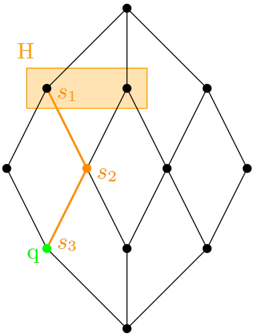


Figure 11: Feasible deductive reasoning in a lattice.

- In Deduction we are given an initial set of *hypotheses* $H \subseteq \mathcal{H}$.
- We are also given a query $q \in \mathcal{Q} \subseteq \mathcal{H}$.
- Our goal is to find, *if possible*, a chain $h = s_1 \succeq s_2 \succeq \cdots \succeq s_n = q$ in $\langle \mathcal{H}, \succeq \rangle$.
- If this is possible, we answer yes.

Deductive Reasoning

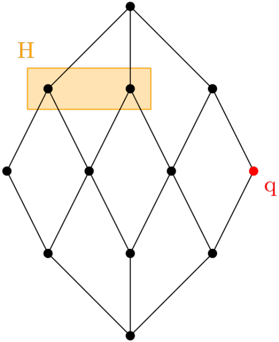


Figure 12: Infeasible deductive reasoning in a lattice.

- In Deduction we are given an initial set of *hypotheses* $H \subseteq \mathcal{H}$.
- We are also given a query $q \in \mathcal{Q} \subseteq \mathcal{H}$.
- Our goal is to find, *if possible*, a chain $h = s_1 \succeq s_2 \succeq \cdots \succeq s_n = q$ in $\langle \mathcal{H}, \succeq \rangle$.
- If this is possible, we answer yes.
- Otherwise, we answer no.

Inductive Reasoning

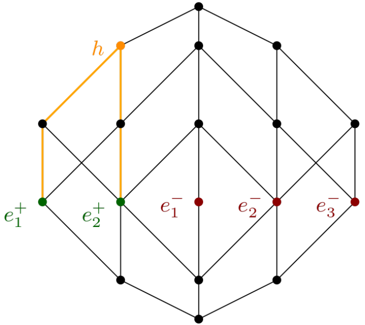


Figure 13: Single hypothesis induction in a lattice.

- In Induction we are given a set of positive $E^+ \subseteq \mathcal{E}$ and negative $E^- \subseteq \mathcal{E}$ examples, where $\mathcal{E} \subseteq \mathcal{H}$.
- We are required to find a hypothesis $h \in \mathcal{H}$, such that:
  1. For each $e^+ \in E^+$, there is a chain $h = s_1 \succeq \cdots \succeq s_n = e^+$.
  2. There is no chain from h to some $e^- \in E^-$.
- If this is possible, we answer yes.
- Otherwise, we answer no.

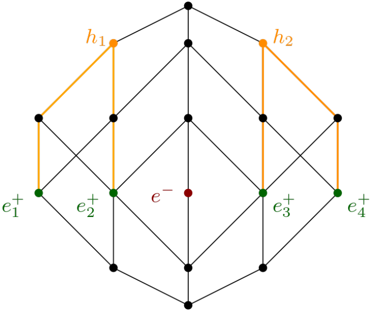


Figure 14: Multiple hypothesis induction in a lattice.

- In Induction we are given a set of positive $E^+ \subseteq \mathcal{E}$ and negative $E^- \subseteq \mathcal{E}$ examples, where $\mathcal{E} \subseteq \mathcal{H}$.
- We are required to find a hypothesis $h \in \mathcal{H}$, such that:
  1. For each $e^+ \in E^+$, there is a chain $h = s_1 \succeq \cdots \succeq s_n = e^+$.
  2. There is no chain from h to some $e^- \in E^-$.
- If this is possible, we answer yes.
- Otherwise, we answer no.
- Note that there are instances that can *only* be multiple hypotheses.

Presentation's Outline

Logic Programming

- Here we review a formalism for *deductive reasoning*, i.e., Logic Programming (LP).

Logic Programming

- Here we review a formalism for *deductive reasoning*, i.e., Logic Programming (LP).
- The most popular logic programming language is Prolog:
  - Declarative Programming
  - Syntax: *First-order Horn clauses*
  - *Turing complete*

Logic Programming

- Here we review a formalism for *deductive reasoning*, i.e., Logic Programming (LP).
- The most popular logic programming language is Prolog:
  - Declarative Programming
  - Syntax: *First-order Horn clauses*
  - *Turing complete*
- LP's importance in our study is twofold:
  - Representation Language, for our hypotheses.
  - A framework for Deductive Reasoning.

Logic Programming

- Here we review a formalism for *deductive reasoning*, i.e., Logic Programming (LP).
- The most popular logic programming language is Prolog:
  - Declarative Programming
  - Syntax: *First-order Horn clauses*
  - *Turing complete*
- LP's importance in our study is twofold:
  - Representation Language, for our hypotheses.
  - A framework for Deductive Reasoning.
- By inverting Prolog's deductive reasoning we construct an inductive system.

First-order Horn Clauses

- A *first-order Horn clauses* is of the form:

$$\underbrace{\mathtt{h}(\mathtt{X}_1, \ldots, \mathtt{X}_\mathtt{n})}_{\text{head}} \leftarrow \underbrace{\mathtt{b}_1(\mathtt{X}_{(1)}, \ldots \mathtt{X}_{(\mathtt{m}_1)}), \ldots \mathtt{b}_k(\mathtt{X}_{(1)}, \ldots \mathtt{X}_{(\mathtt{m}_\mathtt{k})})}_{\text{body}}.$$

- A Horn clause is a disjunction of literals in which, *at most one*, is positive.
- The variables appearing at the head are *universally* quantified.
- The rest of the variables, if any, are *existentially* quantified.
- Definite is a clause with exactly one positive literal.

First-order Horn Clauses: Facts

- A *first-order Horn clauses* is of the form:

$$\overbrace{\underbrace{\mathtt{h}(\mathtt{X}_1,\ldots,\mathtt{X}_n)}_{\text{head}} \leftarrow}^{\text{fact}} \underbrace{\mathtt{b}_1(\mathtt{X}_{(1)},\ldots \mathtt{X}_{(m_1)}),\ldots \mathtt{b}_k(\mathtt{X}_{(1)},\ldots \mathtt{X}_{(m_k)}).}_{\text{body}}$$

- A Horn clause is a disjunction of literals in which, *at most one*, is positive.
- The variables appearing at the head are *universally* quantified.
- The rest of the variables, if any, are *existentially* quantified.
- Definite is a clause with exactly one positive literal.
- A fact is a definite clause, with no body.

First-order Horn Clauses: Queries

- A *first-order Horn clauses* is of the form:

$$\underbrace{h(X_1, \ldots, X_n)}_{\text{head}} \overbrace{\leftarrow \underbrace{b_1(X_{(1)}, \ldots X_{(m_1)}), \ldots b_k(X_{(1)}, \ldots X_{(m_k)}).}_{\text{body}}}^{\text{query}}$$

- A Horn clause is a disjunction of literals in which, *at most one*, is positive.
- The variables appearing at the head are *universally* quantified.
- The rest of the variables, if any, are *existentially* quantified.
- Definite is a clause with exactly one positive literal.
- A fact is a definite clause, with no body.
- A query is an, *existentially quantified*, disjunction of negative clauses.

First-order Horn Clauses: Ground Clause

- A *first-order Horn clauses* is of the form:

$$\underbrace{\mathtt{h}(\mathtt{a}, \ldots, \mathtt{c})}_{\text{head}} \leftarrow \underbrace{\mathtt{b}_1(\mathtt{a}, \ldots \mathtt{c}), \ldots \mathtt{b}_k(\mathtt{d}, \ldots \mathtt{e})}_{\text{body}}.$$

- A Horn clause is a disjunction of literals in which, *at most one*, is positive.
- The variables appearing at the head are *universally* quantified.
- The rest of the variables, if any, are *existentially* quantified.
- Definite is a clause with exactly one positive literal.
- A fact is a definite clause, with no body.
- A query is an, *existentially quantified*, disjunction of negative clauses.
- A ground clause contains *no-variables*.

Reasoning in Prolog: Resolution

▶ Prolog uses an *approximation* of the classical syntactic entailment (modus ponens), *resolution*.

$$\frac{\begin{array}{l} h \leftarrow b_1, \ldots, b_n \\ g \leftarrow c_1, \ldots, c_{i-1}, h, c_i, \ldots, c_m \end{array}}{g \leftarrow c_1, \ldots, c_{i-1}, b_1, \ldots, b_n, c_i, \ldots, c_m}$$

Figure 15: Resolution Operator.

- We denote resolution with $\vdash_{\text{res}}$.
- Resolution is *sound* i.e., $P \vdash_{\text{res}} q \Rightarrow P \models q$.
- Resolution is *not* complete i.e., $P \models q \not\Rightarrow P \vdash_{\text{res}} q$.
- But, resolution is *refutation complete* i.e., $P \models \Box \Leftrightarrow P \vdash_{\text{res}} \Box$.

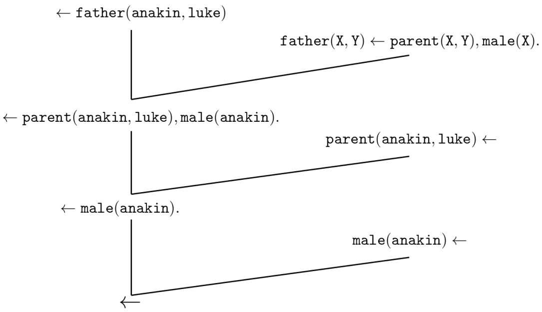


Figure 16: Resolution's derivation process.

- Each derivation begins with a query.
- A *substitution* θ is a mapping from variables, to other variables or constants, e.g. $\theta = \{\mathtt{X} \mapsto \mathtt{a}, \mathtt{Y} \mapsto \mathtt{Z}, \mathtt{W} \mapsto \mathtt{f(a)}\}$.
- The queries are *unified* with the programme's clauses, by a substitution θ.
- We obtain a proof by *contradiction*, i.e., $P \cup \neg q \vdash_{res} \square$.

Abstract Reasoning Framework under Resolution

- Let $\textsc{Horn}$ be the set of all possible Horn clauses.
- $2^{\textsc{Horn}}$ constitutes the set of all Prolog programmes.
- We call the pair $\langle 2^{\textsc{Horn}}, \vdash_{\mathsf{res}} \rangle$ an *abstract reasoning framework*.
- $\langle 2^{\textsc{Horn}}, \vdash_{\mathsf{res}} \rangle$ is a *partial order* on the set of possible Prolog programmes.

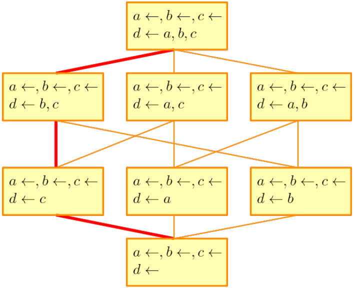


Figure 17: The abstract reasoning framework $\langle 2^{\textsc{Horn}}, \vdash_{\mathsf{res}} \rangle$.

Abstract Reasoning Framework under Resolution

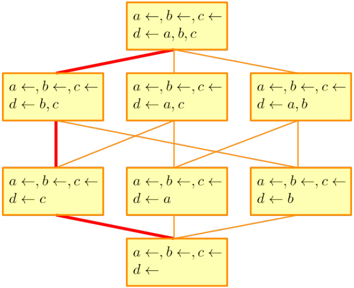


Figure 18: The abstract reasoning framework $\langle 2^{\text{HORN}}, \vdash_{\text{res}} \rangle$.

- *We are able to encode the deduction reasoning problem as traversal problems in a lattice.*

Presentation's Outline

- We begin our analysis from induction on single clauses.
- We want to discover the *definition* of a concept, as a Horn clause.
- Each clause is composed from the same head predicate, the *target*.
- We search from a finite selection of candidate predicates to compose the body.

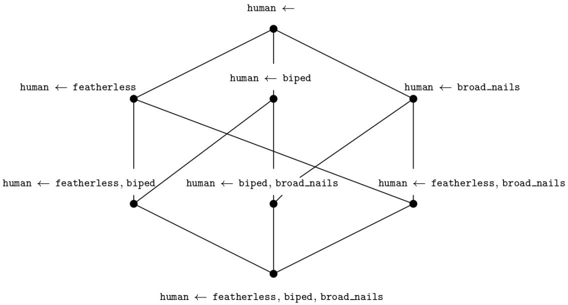


Figure 19: Plato's definition of a human.

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1: a \leftarrow b, c$
  - $\phi_2: a \leftarrow b, c, d$

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1 : a \leftarrow b, c$
  - $\phi_2 : a \leftarrow b, c, d$
- Obviously, $\phi_1 \models \phi_2$, or ϕ_1 is *more general* than ϕ_2 i.e., $\phi_1 \succeq \phi_2$.

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1 : a \leftarrow b, c$
  - $\phi_2 : a \leftarrow b, c, d$
- Obviously, $\phi_1 \models \phi_2$, or ϕ_1 is *more general* than ϕ_2 i.e., $\phi_1 \succeq \phi_2$.
- In *set notation* we write:
  - $c_{\phi_1} : \{a, \neg b, \neg c\}$
  - $c_{\phi_1} : \{a, \neg b, \neg c, \neg d\}$

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1: a \leftarrow b, c$
  - $\phi_2: a \leftarrow b, c, d$
- Obviously, $\phi_1 \models \phi_2$, or ϕ_1 is *more general* than ϕ_2 i.e., $\phi_1 \succeq \phi_2$.
- In *set notation* we write:
  - $c_{\phi_1}: \{a, \neg b, \neg c\}$
  - $c_{\phi_1}: \{a, \neg b, \neg c, \neg d\}$
- Here, $c_{\phi_1} \subseteq c_{\phi_2}$.
- We say that ϕ_1 *subsumes* ϕ_2, iff $c_{\phi_1} \subseteq c_{\phi_2}$. Notation, $\phi_1 \vdash_{\mathrm{sub}} \phi_2$.

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1 : a \leftarrow b, c$
  - $\phi_2 : a \leftarrow b, c, d$
- Obviously, $\phi_1 \models \phi_2$, or ϕ_1 is *more general* than ϕ_2 i.e., $\phi_1 \succeq \phi_2$.
- In *set notation* we write:
  - $c_{\phi_1} : \{a, \neg b, \neg c\}$
  - $c_{\phi_1} : \{a, \neg b, \neg c, \neg d\}$
- Here, $c_{\phi_1} \subseteq c_{\phi_2}$.
- We say that ϕ_1 *subsumes* ϕ_2, iff $c_{\phi_1} \subseteq c_{\phi_2}$. Notation, $\phi_1 \vdash_{\text{sub}} \phi_2$.
- For *first-order* Horn clauses, we need to handle variables.

θ-Subsumption

- We can organise the hypothesis space using θ-subsumption.
- Assume the *propositional* clauses:
  - $\phi_1 : a \leftarrow b, c$
  - $\phi_2 : a \leftarrow b, c, d$
- Obviously, $\phi_1 \models \phi_2$, or ϕ_1 is *more general* than ϕ_2 i.e., $\phi_1 \succeq \phi_2$.
- In *set notation* we write:
  - $c_{\phi_1} : \{a, \neg b, \neg c\}$
  - $c_{\phi_1} : \{a, \neg b, \neg c, \neg d\}$
- Here, $c_{\phi_1} \subseteq c_{\phi_2}$.
- We say that ϕ_1 *subsumes* ϕ_2, iff $c_{\phi_1} \subseteq c_{\phi_2}$. Notation, $\phi_1 \vdash_{\text{sub}} \phi_2$.
- For *first-order* Horn clauses, we need to handle variables.
- We say that ϕ_1 *θ-subsumes* ϕ_2, iff there is substitution θ, s.t. $c_{\phi_1}\theta \subseteq c_{\phi_2}$.
- We demand *all* the literals of c_{ϕ_1} to be *unified* with some of the literals of c_{ϕ_2}.

Lattice Properties of θ-subsumption

▶ The hypothesis space of first-order Horn clauses, under θ-subsumption, forms a *lattice*.

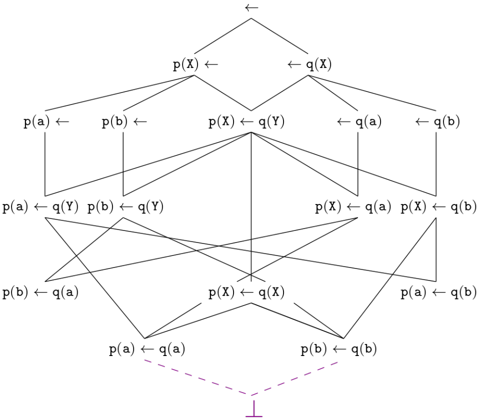


Figure 20: The lattice of first-order Horn clauses, under θ-subsumption.

Least General Generalization

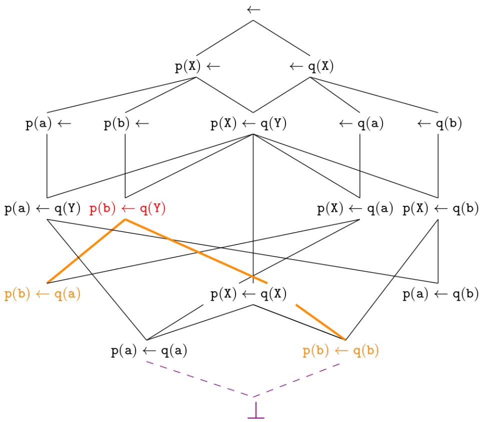


Figure 21: The lattice of first-order Horn clauses, under θ-subsumption.

- The lub operator, coincides with the *least general generalization*.

Least General Generalization

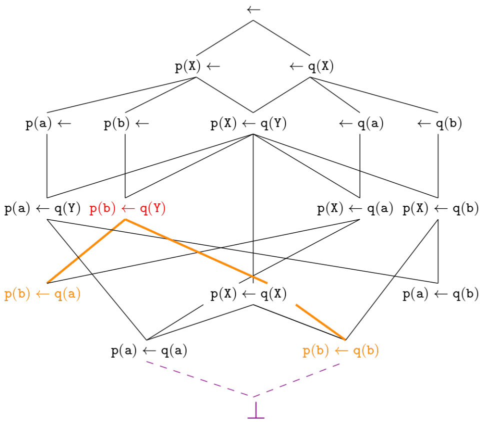


Figure 21: The lattice of first-order Horn clauses, under θ-subsumption.

- The lub operator, coincides with the *least general generalization*.
- There we perform *anti-unification* substituting *constants* with variables.

Induction for Sets of Clauses

- We discuss induction for sets of clauses, i.e., programmes.
- We *invert* the resolution operator.
- There are two ways of inverting resolution, i.e.:
  - `V`-operators.
  - `W`-operators.
- We want to reconstruct the premises, form the conclusion.

V-operator

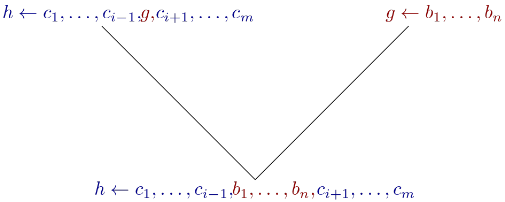


Figure 22: The V-operator for inverse resolution.

- The V-operator *inverts* a resolution step.
- From the conclusions:
  - $g \leftarrow b_1, \ldots, b_n$
  - $h \leftarrow c_1, \ldots, c_{i-1}, b_1, \ldots, b_n, c_{i+1}, \ldots, c_m$
- We reconstruct the premise:
  - $h \leftarrow c_1, \ldots, c_{i-1}, h, c_{i+1}, \ldots, c_m$

W-operator

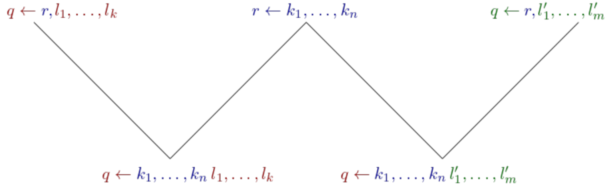


Figure 23: The W-operator for inverse resolution & predicate invention.

- W-operator combines two V-operators.
- From the conclusions:
  - $q \leftarrow k_1, \ldots, k_n, l_1, \ldots, l_k$
  - $q \leftarrow k_1, \ldots, k_n, l'_1, \ldots, l'_m$
- We reconstruct the premises:
  - $q \leftarrow r, l_1, \ldots, l_m$
  - $q \leftarrow r, l'_1, \ldots, l'_m$
  - $r \leftarrow k_1, \ldots, k_n$
- The predicate r does not appear in the conclusions.
- It is automatically *invented*!

Inferring Missing Facts with Abduction

$$\frac{\begin{array}{l} q_1\theta \wedge \cdots \wedge q_n\theta \\ p \leftarrow q_1, \ldots, q_n \end{array}}{\begin{array}{l} p\theta \\ p \leftarrow q_1, \ldots, q_n \end{array}}$$

Figure 24: The Abduction Operator for inferring missing facts.

- Assume that $p\theta$ is a ground *positive* example.
- Also, assume that $p \leftarrow q_1, \ldots, q_n$ is a known rule.
- In order for $p\theta$ to be covered by the current theory.
- The current theory also needs to cover $q_1\theta, \ldots, q_n\theta$.

Presentation's Outline

An Induction Problem: Defining the “Bird”

- Assume we want to define the notion of a `bird`.
- Form experience, we know that some birds `fly`, others are `feathered`, and poses the ability to `mimic` the voices they hear.
- Our theory must cover instances that embody all characteristics:

$$e^{+} : \texttt{bird} \leftarrow \texttt{fly}, \texttt{feathered}, \texttt{mimic}.$$

- We know that not all birds fly, or can mimic voices, thus:

$$e^{-} : \texttt{bird} \leftarrow \texttt{fly}, \texttt{mimic}.$$

- We are looking for a theory $h \in \mathcal{H}$, s.t.:
  1. $h \succeq e^{+}$
  2. $h \not\succeq e^{-}$

Solving the Induction Problem: Hypothesis Space

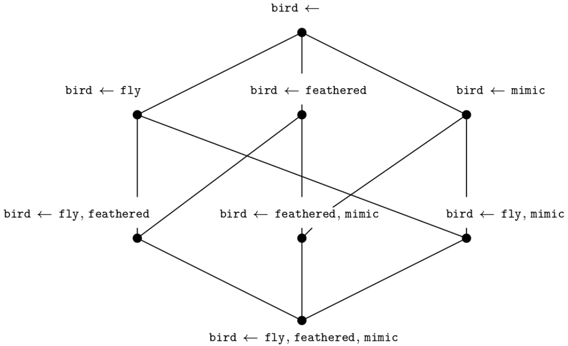


Figure 25: The hypothesis space for “bird” definitions.

Solving the Induction Problem: Hypothesis Space

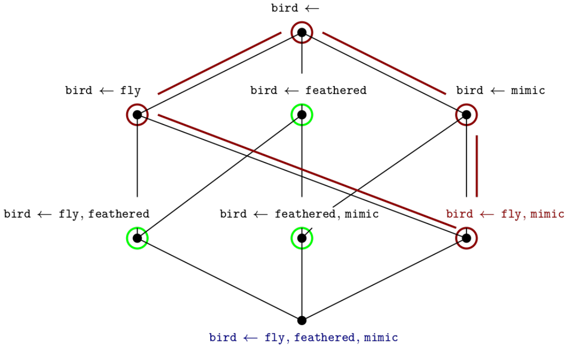


Figure 26: The hypothesis space for "bird" definitions.

1. The positive example e^+ poses lower bounds.
2. The negative example e^- poses upper bounds.
3. Both examples circumscribe a feasible area.

Solving the Induction Problem: Traversal

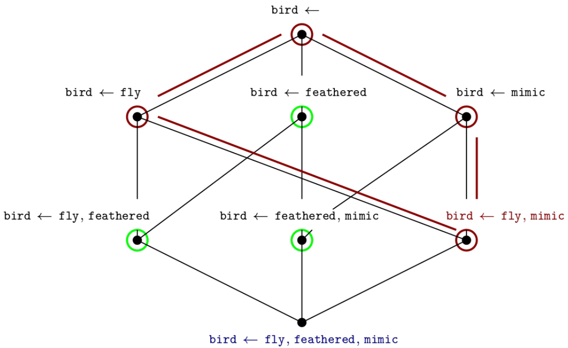


Figure 27: The hypothesis space for "bird" definitions.

- We traverse the hypothesis space, using refinement operators:
  1. Top-Down
  2. Bottom-Up

Solving the Induction Problem: Heuristics

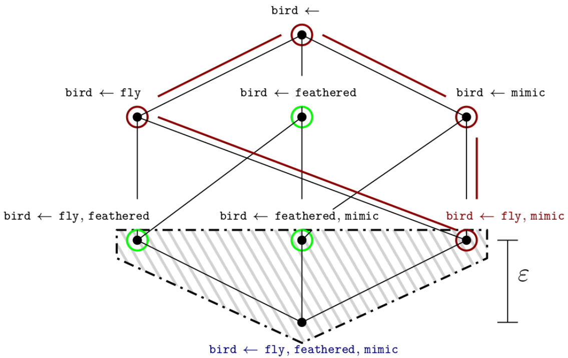


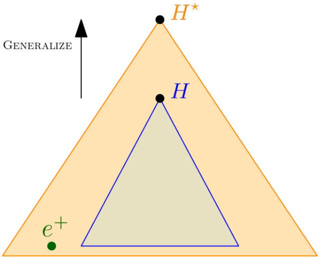
Figure 28: The hypothesis space for "bird" definitions.

- If needed we employ the heuristic parameter ε.
- This parameter bounds the specificity of a hypothesis.
- Thus opting for more general, feasible solutions.

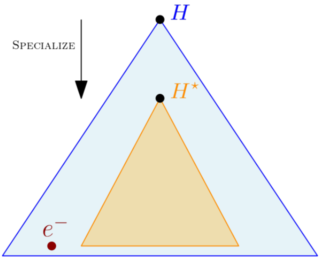
Query-based Induction (1/2)

- Previously, we assumed we are given a set of positive E^+ and negative E^- examples.
- Alternatively, we may assume that we have access to an *oracle function* $\texttt{comp}\colon \mathcal{H} \to \{\succeq, \prec, \not\sim\}$
- At each step we *compare* our current hypothesis H to nature's (unknown) hypothesis $H^\star$.
- If $H \succeq H^\star$ we *specialize*.
- If $H \prec H^\star$ we *generalize*.
- If $H \not\sim H^\star$ we *backtrack* choosing an alternative specialization of generalization.

Query-based Induction (2/2)



(a) If $H \prec H^\star$ we *generalize*.



(b) If $H \succeq H^\star$ we *specialize*.

Figure 29: The effect of oracle calls to a query-based algorithm's behavior.

- Each oracle call provides additional information:
  - If $H \prec H^\star$, there is a *positive* example e^+, *not* covered by our hypothesis.
  - If $H \succeq H^\star$, there is a *negative* example e^-, covered by our hypothesis.

Query-Based Models in Practice

- In literature query-based methods do *not* use *comparison* queries.
- HORN [Angluin et al., 1992] system & MODEL INFERENCE SYSTEM [Shapiro, 1991] utilize, membership and equivalence queries.
- Membership Queries:
  - Assume $\mathcal{M}(H^\star)$ be the (unknown) models satisfying natures hypothesis.
  - Assume u being an interpretation satisfying our current hypothesis H.
  - A membership query asks whether $u \in \mathcal{M}(H^\star)$.
- Equivalence Queries:
  - An equivalence query asks whether our current hypothesis equals, the (unknown) nature's hypothesis i.e., $H \models\!\!\!\!\!\dashv H^\star$.
  - If not, provides a *counterexample*.
- Remains an *open question* whether comparison-query setting is equivalent to membership, equivalence-queries model.

Presentation's Outline

Meta-level Programming

- In meta-level programming we *reduce* the ILP problem as an constraint problem in Answer Set Programming (ASP).
- The reduced problem is solved by an ASP solver.
- Thus, an induction problem is *transformed* to a deduction problem.
- Related Work: ILSAP [Law et al., 2014], ASPAL [Corapi et al., 2011], etc.

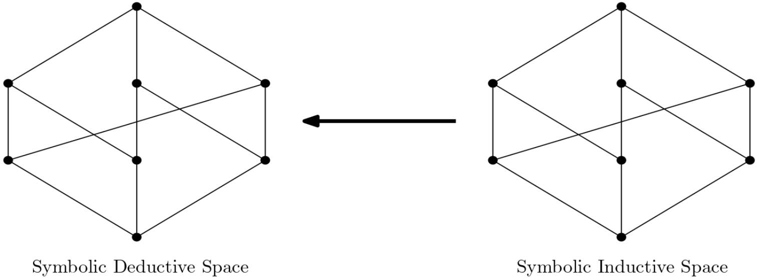


Figure 30: The reduction in meta-level programming.

Neurosymbolism

- An induction problem is reduced to a deduction problem.
- The deduction problem is further relaxed to a continuous space.
- The continuous problem is solved by neural networks.
- This relaxation opts for noise handling, of mislabeled training data.
- Related Work: Differential ILP (∂ILP) [Evans and Grefenstette, 2018], etc.

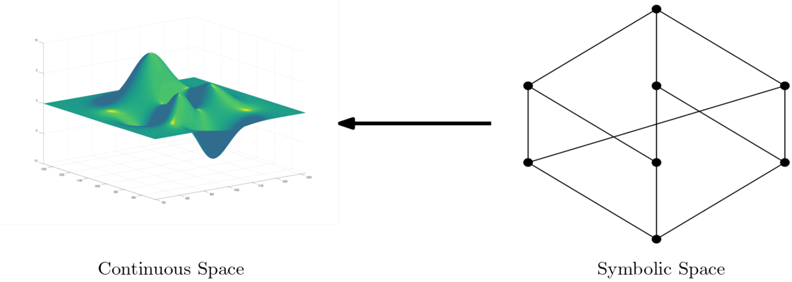
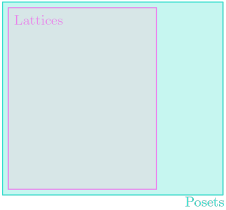


Figure 31: The reduction in neurosymbolism.

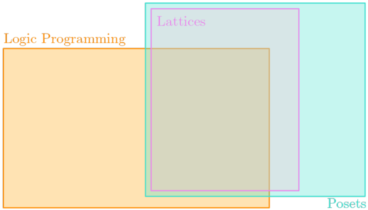
Presentation's Outline

Conclusions: Lattices & Posets



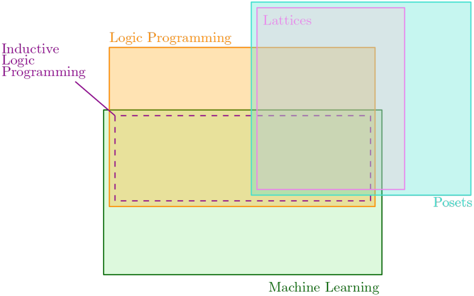
- Lattices & Posets:
  - Formal Deduction
  - Formal Induction
  - Reduction to Lattice Traversal

Conclusions: Logic Programming



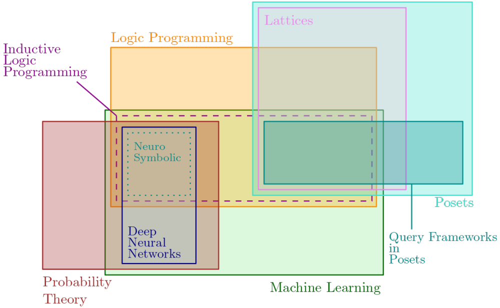
- Lattices & Posets:
  - Formal Deduction
  - Formal Induction
  - Reduction to Lattice Traversal
- Logic Programming:
  - Horn Clauses
  - Resolution
  - Deduction in a Lattice

Conclusions: Inductive Logic Programming



- Lattices & Posets:
  - Formal Deduction
  - Formal Induction
  - Reduction to Lattice Traversal
- Logic Programming:
  - Horn Clauses
  - Resolution
  - Deduction in a Lattice
- Inductive Logic Programming:
  - θ-Subsumption
  - Inverse Resolution
  - Abduction

Conclusions: Discussion



- Lattices & Posets:
  - Formal Deduction
  - Formal Induction
  - Reduction to Lattice Traversal
- Logic Programming:
  - Horn Clauses
  - Resolution
  - Deduction in a Lattice
- Inductive Logic Programming:
  - θ-Subsumption
  - Inverse Resolution
  - Abduction
- Discussion:
  - Meta-level programming
  - Neurosymbolism

Research Directions

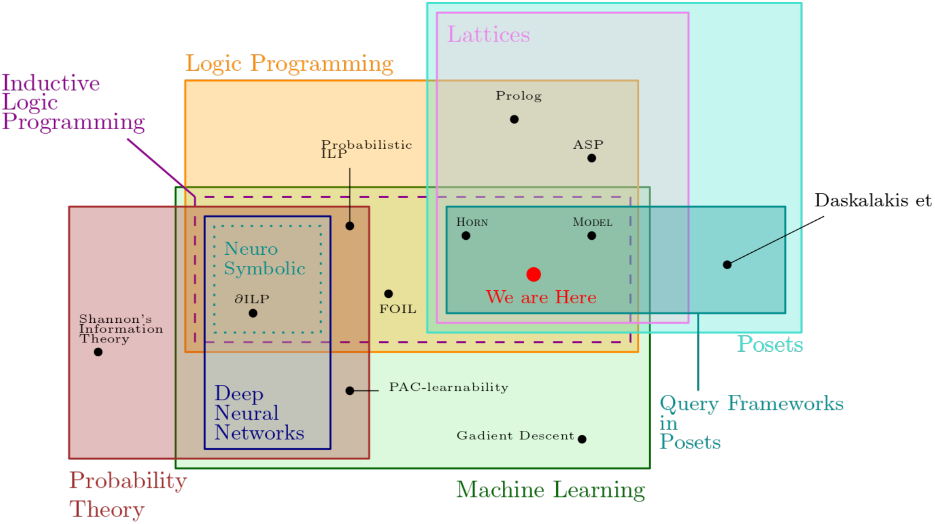
- ILP reduces the learning problem to a search problem in a lattice.
- The lattice is organised by a generality relation $\succeq$.
- Understandable effect of the training data to the resulting hypothesis.
- Clear relation between the data observed and the accumulated knowledge.

Research Directions

- ILP reduces the learning problem to a search problem in a lattice.
- The lattice is organised by a generality relation $\succeq$.
- Understandable effect of the training data to the resulting hypothesis.
- Clear relation between the data observed and the accumulated knowledge.

▶ Studying the lattices in induction, will help us:

1. Utilize the work on query-optimal poset search [Daskalakis et al., 2011].
2. *Data-efficient as Query Optimality*.
3. Formal treatment of explainability based on generality.
4. *Explainability as debuggability*.



Thank you for your time!

Presentation's Outline

Bibliography I

Angluin, D., Frazier, M., and Pitt, L. (1992).
Learning conjunctions of horn clauses.
Mach. Learn., 9:147–164.

Corapi, D., Russo, A., and Lupu, E. (2011).
Inductive logic programming in answer set programming.
In Muggleton, S. H., Tamaddoni-Nezhad, A., and Lisi, F. A., editors, *Inductive Logic Programming - 21st International Conference, ILP 2011, Windsor Great Park, UK, July 31 - August 3, 2011, Revised Selected Papers*, volume 7207 of *Lecture Notes in Computer Science*, pages 91–97. Springer.

Daskalakis, C., Karp, R. M., Mossel, E., Riesenfeld, S. J., and Verbin, E. (2011).
Sorting and selection in posets.
SIAM J. Comput., 40(3):597–622.

Evans, R. and Grefenstette, E. (2018).
Learning explanatory rules from noisy data.
J. Artif. Intell. Res., 61:1–64.

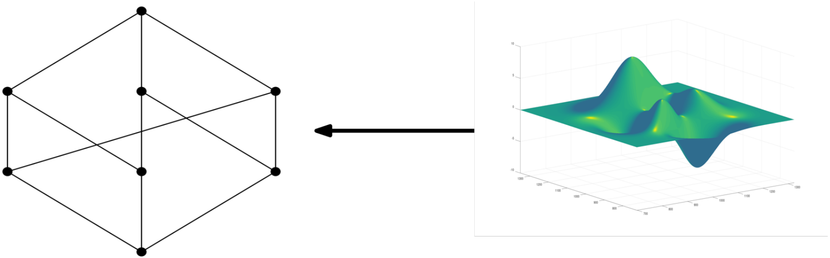
Bibliography II

Law, M., Russo, A., and Broda, K. (2014).
Inductive learning of answer set programs.
In Fermé, E. and Leite, J., editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 311–325. Springer.

Shapiro, E. (1991).
Inductive inference of theories from facts.
In Lassez, J. and Plotkin, G. D., editors, *Computational Logic - Essays in Honor of Alan Robinson*, pages 199–254. The MIT Press.

Presentation's Outline

Continuous Hypotheses as Symbolic Algebraic Expressions



Symbolic Space

Continuous Space

Figure 32: In traditional learning the hypotheses space, often, consists of a family of real functions $\mathcal{F}$ providing *scores* of certainty. A generality relation on $\mathcal{F}$ could be defined as follows, $f_1 \succeq f_2$ iff $f_1(e) \geq f_2(e)$ for all $e \in \mathcal{E}$.