# Machine Learning: Inductive Logic Programming

Forest Agostinelli
University of South Carolina

# Topics Covered in This Class

- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

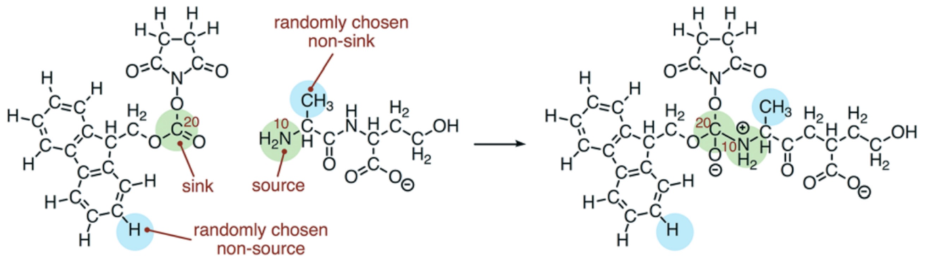
# Outline

- Machine learning
- Logical inference
- Inductive logic programming
- Theta-subsumption
- Inductive bias

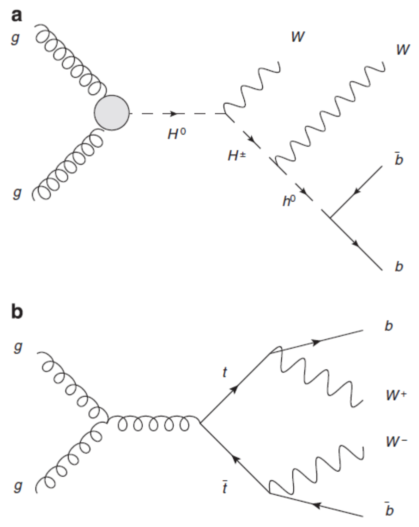
## Machine Learning

- An agent improves its performance after making observations about the world
- Supervised learning
  - Learn a function that maps inputs to outputs from input and output pairs. Outputs are often referred to as labels.
- Reinforcement learning
  - The agent learns from feedback from the environment in the form of rewards. Rewards are often delayed and the agent must determine how to modify its actions to obtain more reward.
- Unsupervised learning
  - The agent learns patterns in the input without any explicit feedback

# Machine Learning: Supervised Learning



Chemical reaction prediction



Exotic particle detection

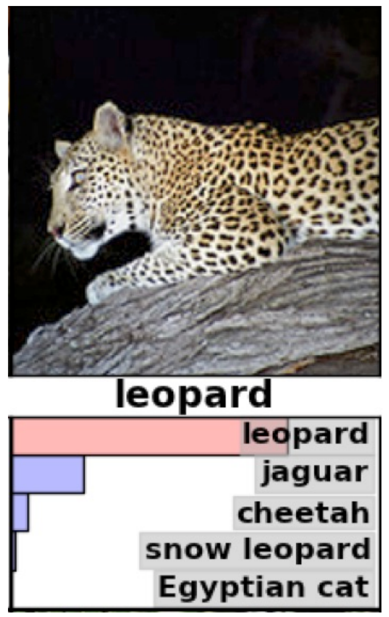
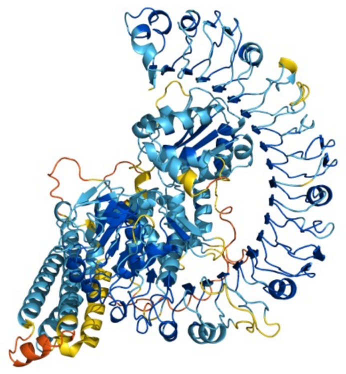


Image classification

**Rule 2 (lambda repressor)** *The protein is between 53 and 88 residues long. Helix A at position 3 is followed by helix B. The coil between A and B is about 6 residues long.*

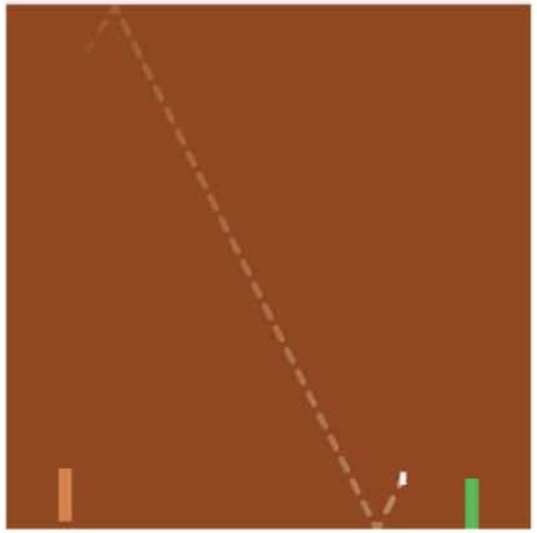
```
fold('lambda repressor', X) :-
    len_interval(53 =< X =< 88),
    adjacent(X, A, B, 3, h, h),
    coil(A, B, 6).
```

Protein folding



Protein folding

# Machine Learning Examples: Reinforcement Learning



Play Video Games

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO

Beat humans in Go

Self-Taught AI Masters Rubik's Cube in Just 44 Hours

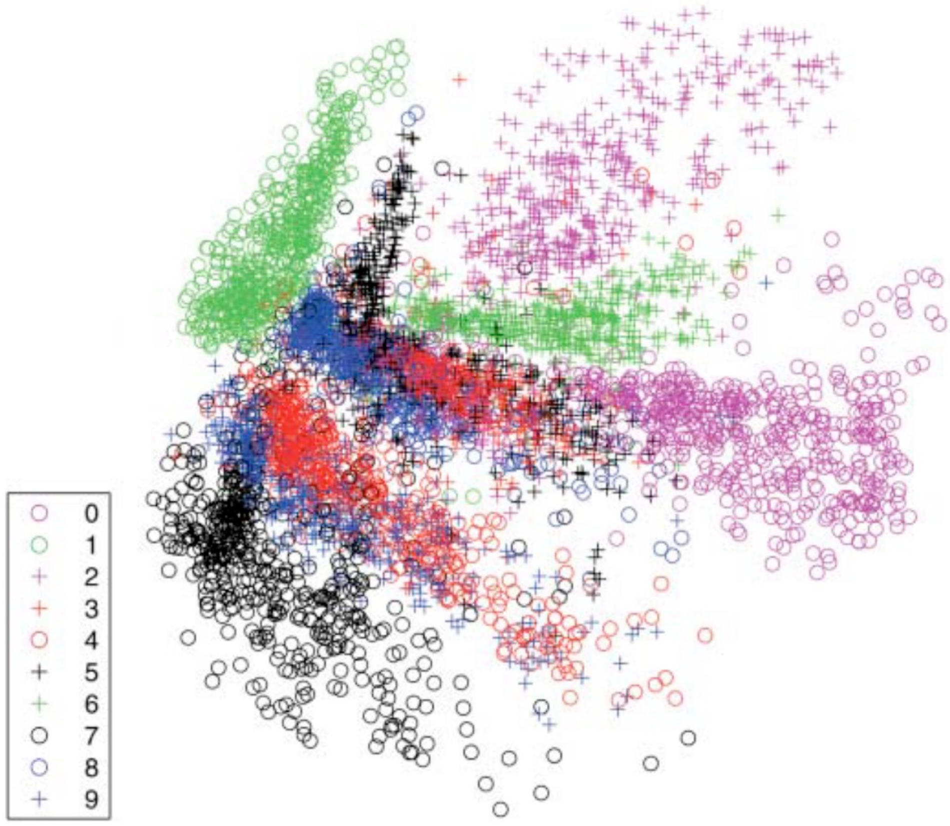
George Dvorsky
6/18/18 1:00pm • Filed to: ARTIFICIAL INTELLIGENCE

13.1K 28 2

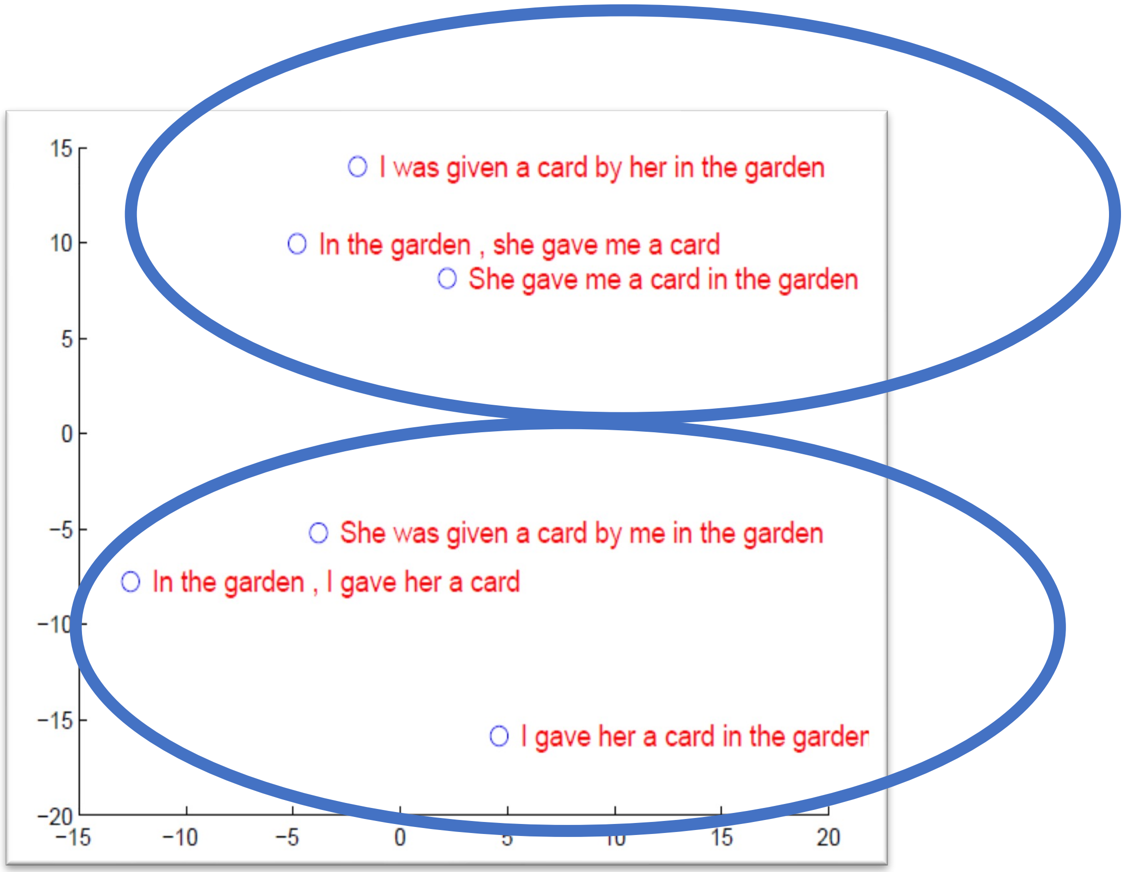
Solve the Rubik's cube

Robotics

# Machine Learning: Unsupervised Learning



Finding patterns in digits



Finding patterns in natural language

# ML Classes at the University of South Carolina

- Machine learning (Prof Jianjun Hu)
- Machine learning systems (Prof. Pooyan Jamshidi)
- Natural language processing (Prof. Biplav Srivastava)
- Deep reinforcement learning (Prof. Forest Agostinelli and Prof. Qi Zhang)
- One day soon: Deep learning

## Supervised Learning

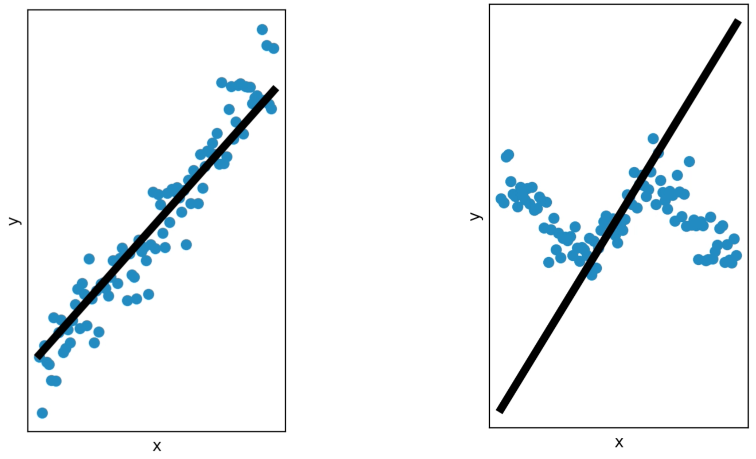
- We would like to come up with a hypothesis to explain some observations
- Each observation can be thought of as having some input and output
    - Images -> labels
    - Protein -> 3D structure
    - Logical statements -> true/false
- We may need to consider many different hypotheses to come up with a good explanation
- We still may not find a hypothesis that explains all the observations
- The types of hypotheses that we can learn are determined by the hypothesis space

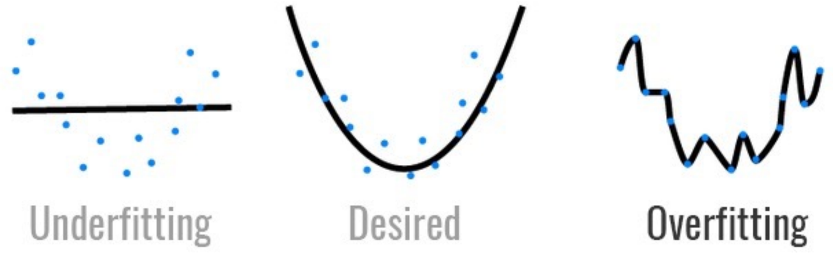
# Hypothesis Space

- The space of all possible hypotheses
- The hypothesis space for inductive logic programming could be all logical statements made up of a given set of predicates
  - This is often further restricted to definite clauses
- The hypothesis space for linear models is all possible lines
- The hypothesis space for neural networks is determined by its architecture and can theoretically approximate any function

# Hypothesis Space

- It is important that the hypothesis space be appropriate for the task at hand
- For example, if the observations are have a linear input/output relationship, it is best to use a linear model
- However, if the observations have a non-linear input/output relationship, then a linear model will provide a poor explanation of the data
- On the other hand, if your hypothesis space is too large, then you may learn unnecessarily complicated hypotheses





# Outline

- Machine learning
- Logical inference
- Inductive logic programming
- Theta-subsumption
- Inductive bias

## Logical Inference: Deduction

- `white(X):- polar_bear(X)`
- `polar_bear(thorton)`

- We can use the aforementioned logical rule that all polar bears are white to deduce
  - `white(thorton)`
- If the given background knowledge is true, then our inferred sentence is guaranteed to be true

## Logical Inference: Abduction

- `white(X):- polar_bear(X)`
- `white(X):- swan(X)`
- `white(thorton)`

- We can use the aforementioned rules in the opposite direction as deduction (from head to body) to abduce
  - `polar_bear(thorton)`
  - `-or-`
  - `swan(thorton)`
- Both of the aforementioned literals explain the observation `white(thorton)`
- This process is **not** truth-preserving like deduction

## Logical Inference: Induction

- `white(thorton)`
- `polar_bear(thorton)`

- We can use the aforementioned observations to induce a rule
  - `polar_bear(X):- white(X)`
  - `white(X):- polar_bear(X)`
- Both of these rules explain our observations
- This process is **not** truth-preserving like deduction

# Logical Inference: Induction

- `white(thorton)`
- `polar_bear(thorton)`
- `white(sam)`
- `swan(sam)`

- Given theses additional observations, this would eliminate the possible rule
  - `polar_bear(X):- white(X)`
- However, this rule still explains our observations about thorton
  - `white(X):- polar_bear(X)`

## Outline

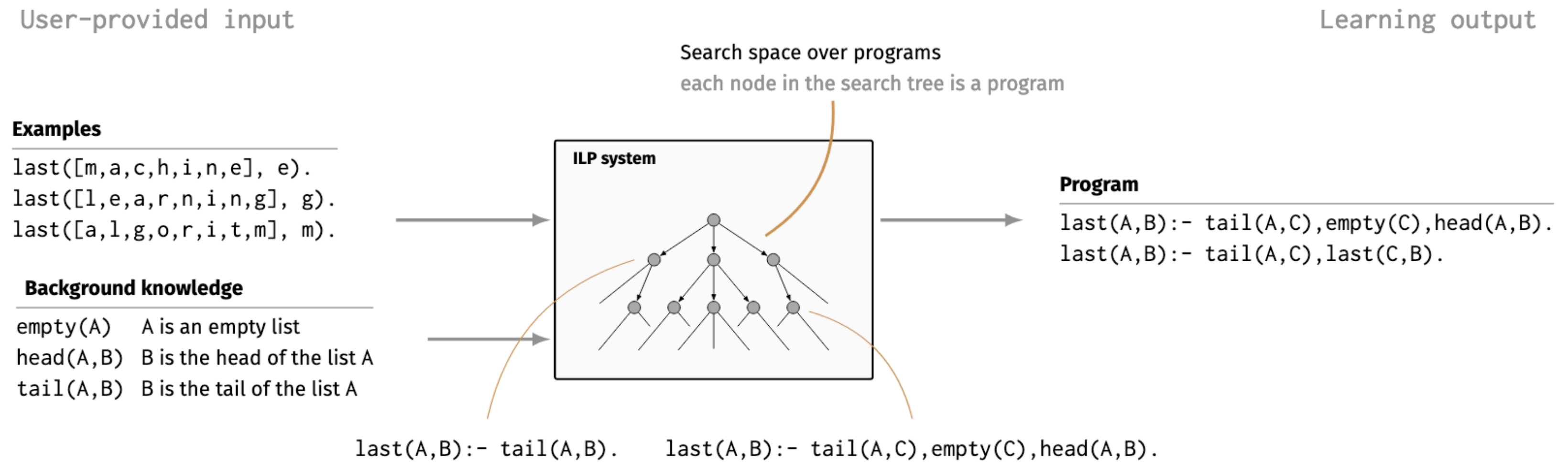
- Machine learning
- Logical inference
- Inductive logic programming
- Theta-subsumption
- Inductive bias

## Inductive Logic Programming

- In inductive logic programming, we seek to learn a hypothesis in the form of an implication. Example in prolog:
  - h(X) :- b1(X), b2(X), b3(X).
  - h(X) :- b4(X), \+ b5(X).
- The predicate defining the hypothesis is in the head (consequent) and the hypothesis space determines what can go in the body (antecedent)
- For example, if we want to learn about chemical reactions, we would need predicates related to atoms, bonds, charge, etc.

# Inductive Logic Programming (ILP)

- Given some examples (can be positive or negative) and some background knowledge, find a logic program (hypothesis) that explains the examples
- The hypothesis should entail all positive examples and should not entail any negative examples


User-provided input
Learning output
Search space over programs
each node in the search tree is a program
ILP system

**Examples**

```
last([m,a,c,h,i,n,e], e).
last([l,e,a,r,n,i,n,g], g).
last([a,l,g,o,r,i,t,m], m).
```

**Background knowledge**

| | |
|---|---|
| `empty(A)` | A is an empty list |
| `head(A,B)` | B is the head of the list A |
| `tail(A,B)` | B is the tail of the list A |

**Program**

```
last(A,B):- tail(A,C),empty(C),head(A,B).
last(A,B):- tail(A,C),last(C,B).
```

```
last(A,B):- tail(A,B).
```

```
last(A,B):- tail(A,C),empty(C),head(A,B).
```

# ILP: Example

- Given this background knowledge, what is a hypothesis that, together with the background knowledge, entails all positive examples and no negative examples?

$$\forall e \in E^+,\ H \cup B \models e \text{ (i.e. } H \text{ is } \textit{complete}\text{)}$$
$$\forall e \in E^-,\ H \cup B \not\models e \text{ (i.e. } H \text{ is } \textit{consistent}\text{)}$$

- Available predicates
  - happy
  - lego_builder
  - estate_agent
  - enjoys_lego

```
B = { lego_builder(alice).
      lego_builder(bob).
      estate_agent(claire).
      estate_agent(dave).
      enjoys_lego(alice).
      enjoys_lego(claire). }
```

```
E+ = { happy(alice). }   E- = { happy(bob).
                                happy(claire).
                                happy(dave). }
```

```
H = { happy(A):- lego_builder(A),enjoys_lego(A). }
```

## ILP: Example

- Background knowledge
  - `animal(X):- dog(X)`
  - `animal(X):- cat(X)`
  - `dog(fido). dog(spot). dog(rover).`
  - `cat(kitty). cat(kelly).`
  - `duck(donald). duck(daffy). duck(huey).`
- Positive examples
  - `barks(fido)`
- Negative examples
  - None
- Hypothesis
  - `barks(X):- animal(X)`

# ILP: Example

- Background knowledge
  - `animal(X):- dog(X)`
  - `animal(X):- cat(X)`
  - `dog(fido). dog(spot). dog(rover).`
  - `cat(kitty). cat(kelly).`
  - `duck(donald). duck(daffy). duck(huey).`
- Positive examples
  - `barks(fido). barks(spot).`
- Negative examples
  - `barks(kitty). barks(donald).`
- Hypothesis
  - `barks(X):- dog(X)`

# ILP: Example

- Background knowledge
  - `animal(X):- dog(X)`
  - `animal(X):- cat(X)`
  - `dog(fido). dog(spot). dog(rover).`
  - `cat(kitty). cat(kelly).`
  - `duck(donald). duck(daffy). duck(huey).`
- Positive examples
  - `four_legged(fido). four_legged(kitty).`
- Negative examples
  - `four_legged(daffy). four_legged(tree17).`
- Hypothesis
  - `four_legged(X):- animal(X), \+ duck(X).`
- Hypothesis (if only definite clauses are allowed)
  - `four_legged(X):- dog(X).`
  - `four_legged(X):- cat(X).`

## Automatic Induction of Logic Programs

- We would like a domain-independent method of learning a logic program given background knowledge, positive examples, and negative examples
- Furthermore, we are interested in finding optimal programs
  - In this context, a program is optimal if it uses the fewest literals possible
- For simplicity, let us assume all programs are definite clauses
  - h(X) :- b1(X), b2(X), b3(X).
  - Only positive literals in the body
- Also, let us assume that we need only a single clause
- What is an algorithm that is guaranteed to find an optimal program?
  - We can do a breadth first search and, for each program, check whether or not it meets our criteria

## Outline

- Machine learning
- Logical inference
- Inductive logic programming
- Theta-subsumption
- Inductive bias

## The Generality Relation

- We can order hypotheses according to their generality
- Suppose we have hypotheses H1 and H2 with a single predicate, h, in the head
    - H1 = h(Y) :- b1(Y)
    - H2 = h(X) :- b1(X), b2(X)
- Then, H1 is more general than H2 if, for every object o, if H1U B entails h(o), then H2UB entails h(o)
    - In other words, the objects entailed by H1 is a superset of the objects entailed by H2
- Using this relationship, we can prune our search space
- If H entails a negative example, then all generalizations of H will also entail that negative example
- If H does not entail a positive example, then no specializations of H will entail that positive example

## Theta-Subsumption

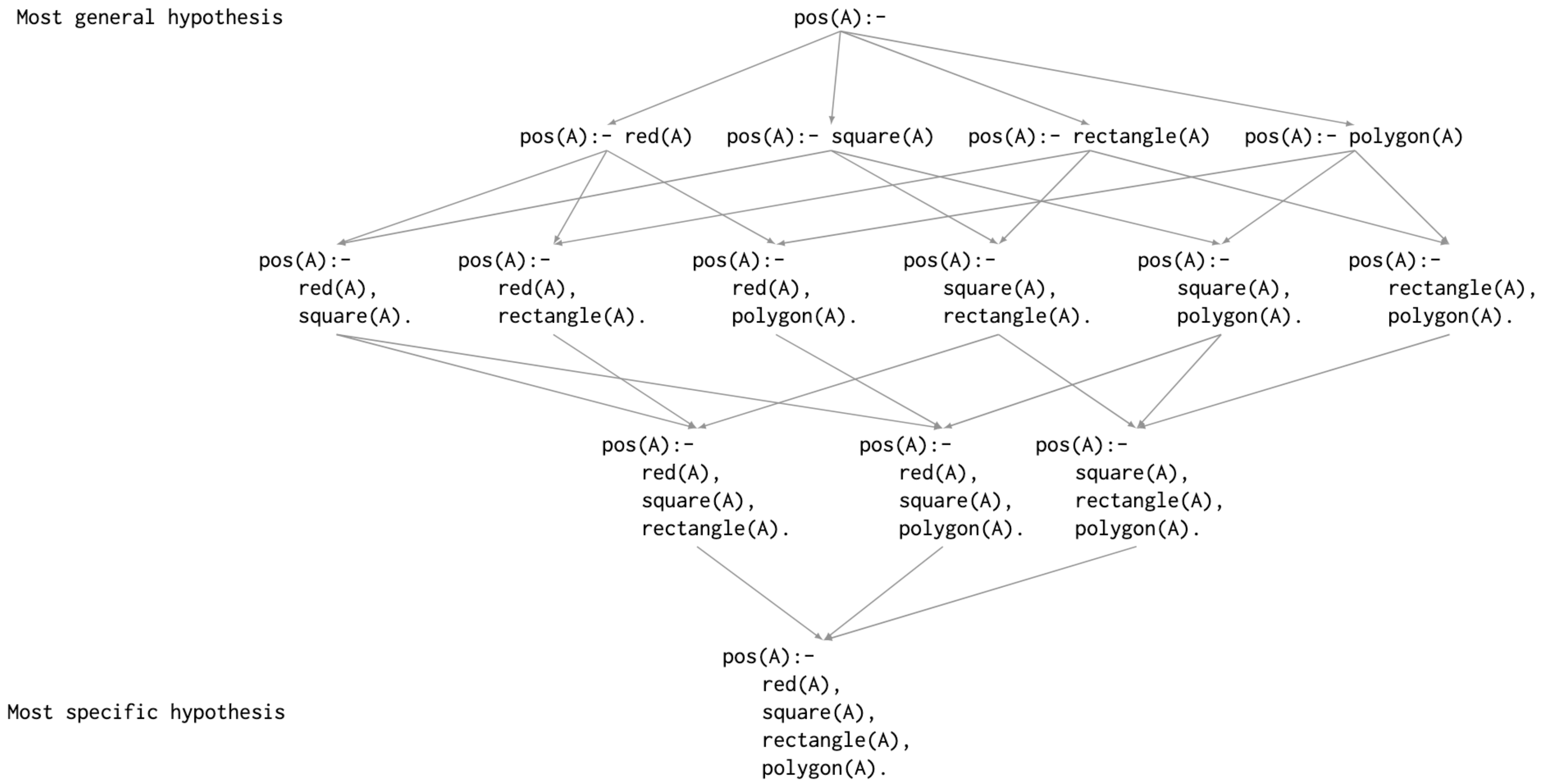
- Using entailment to compute the generality relation can present problems in practice
  - Undecidable
  - Even if decidable, can be computationally expensive
- We can use a weaker generality relation, namely, theta-subsumption, to compute generality
  - Decidable
  - Computationally faster than entailment
  - Still NP-complete
- If H1 theta-subsumes H2, then H1 entails H2
- However, if H1 entails H2, then it does not necessarily follow that H1 theta-subsumes H2

## Theta-Subsumption

- When limiting hypotheses to a single clause, H1 theta-subsumes H2 if and only if there exists some substitution $\boldsymbol{\theta}$ to H1 such that H1$\boldsymbol{\theta} \subseteq$ H2
- Example
  - H1 = h(Y) :- b1(Y)
  - H2 = h(X) :- b1(X), b2(X)
  - H1$\boldsymbol{\theta} \subseteq$ H2 with $\boldsymbol{\theta}$={Y/X}
- The theta-subsumption relationship forms a lattice
- This lattice can then be used to direct search and prune the search space

# Theta-Subsumption Lattice

- Breadth-first search: “expand” returns all children in the lattice
- We can prune the search space once we fail to entail a positive example

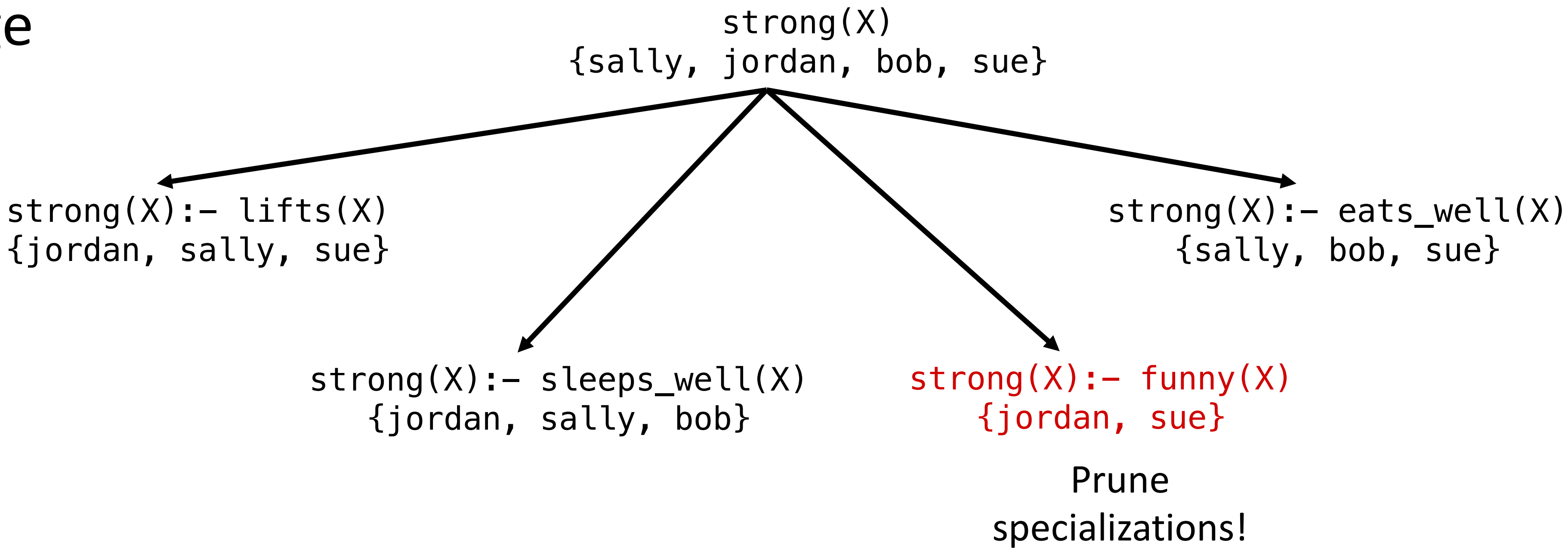

Most general hypothesis
pos(A):-
pos(A):- red(A)
pos(A):- square(A)
pos(A):- rectangle(A)
pos(A):- polygon(A)
pos(A):- red(A), square(A).
pos(A):- red(A), rectangle(A).
pos(A):- red(A), polygon(A).
pos(A):- square(A), rectangle(A).
pos(A):- square(A), polygon(A).
pos(A):- rectangle(A), polygon(A).
pos(A):- red(A), square(A), rectangle(A).
pos(A):- red(A), square(A), polygon(A).
pos(A):- square(A), rectangle(A), polygon(A).
pos(A):- red(A), square(A), rectangle(A), polygon(A).
Most specific hypothesis

## Breadth-First Search (General to Specific)

- Background Knowledge
  - `lifts(jordan). sleeps_well(jordan). funny(jordan).`
  - `lifts(sally). sleeps_well(sally). eats_well(sally).`
  - `sleeps_well(bob). eats_wells(bob).`
  - `lifts(sue). eats_well(sue). funny(sue).`
- Positive Examples
  - `strong(sally).`
- Negative Examples
  - `strong(jordan). strong(bob). strong(sue).`

# Breadth-First Search (General to Specific)

- Background Knowledge
  - `lifts(jordan).`
    `sleeps_well(jordan).`
    `funny(jordan).`
  - `lifts(sally).`
    `sleeps_well(sally).`
    `eats_well(sally).`
  - `sleeps_well(bob).`
    `eats_wells(bob).`
  - `lifts(sue).`
    `eats_well(sue).`
    `funny(sue).`
- Positive Examples
  - `strong(sally).`
- Negative Examples
  - `strong(jordan).`
    `strong(bob).`
    `strong(sue).`



# Heuristic Search

- Background Knowledge

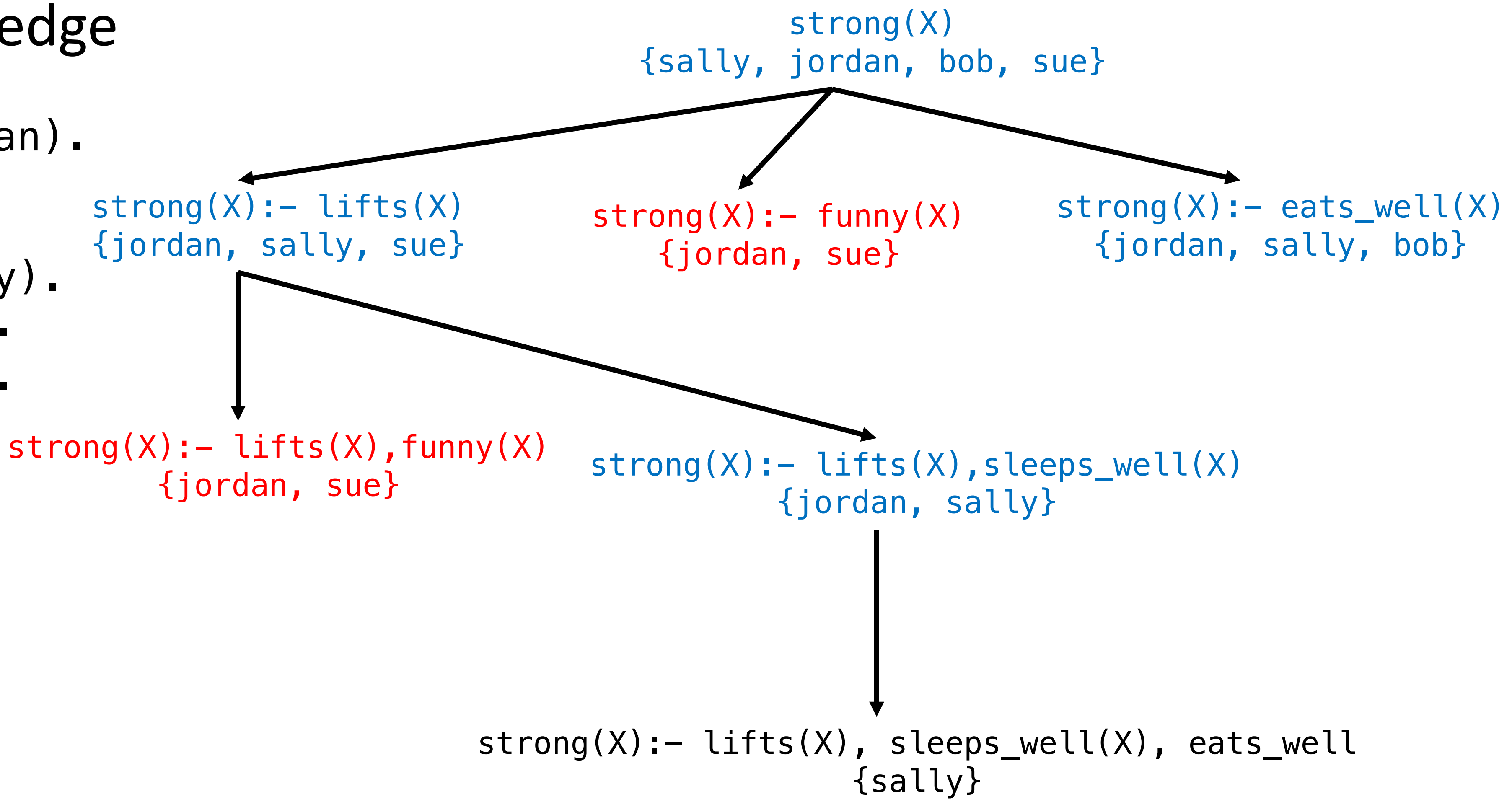
  -
    ```
    lifts(jordan).
    sleeps_well(jordan).
    funny(jordan).
    ```

  -
    ```
    lifts(sally).
    sleeps_well(sally).
    eats_well(sally).
    ```

  -
    ```
    sleeps_well(bob).
    eats_wells(bob).
    ```

  -
    ```
    lifts(sue).
    eats_well(sue).
    funny(sue).
    ```
- Positive Examples
  - `strong(sally).`
- Negative Examples

  -
    ```
    strong(jordan).
    strong(bob).
    strong(sue).
    ```



# Outline

- Machine learning
- Logical inference
- Inductive logic programming
- Theta-subsumption
- Inductive bias

## Inductive Bias

- We can significantly reduce the search space by encoding knowledge about the form we think a hypothesis should or should not take
- This can greatly speed up search and result in hypotheses that are easier for humans to understand
- However, if we are not careful, we can accidentally remove valid hypotheses from consideration and, perhaps, this may result in us not finding a solution at all
- One can think of convolution and weight regularization as a form of an inductive bias

# Inductive Bias

- Background knowledge
  - `animal(X):- dog(X)`
  - `animal(X):- cat(X)`
  - `animal(X):- duck(X)`
  - `animal(X):- cheetah(X)`
  - `animal(X):- lion(X)`
  - `animal(X):- duck(X)`
  - `dog(fido). dog(spot). dog(rover).`
  - `cat(kitty). bony_hyoid(kitty).`
  - `cheetah(speedy). bony_hyoid(speedy)`
  - `duck(donald). duck(daffy). duck(huey).`
  - `lion(simba)`
- Positive examples
  - `purrs(speedy). purrs(kitty).`
- Negative examples
  - `purrs(simba). purrs(rover). purrs(spot).`

- There are many hypotheses that we already know are not going to work
- We can explicitly ban these by requiring no hypothesis be theta-subsumed by these programs
  - `h(X):- dog(X),cat(X)`
  - `h(X):- duck(X),cat(X)`
  - `h(X):- cheetah(X),cat(X)`
  - …
- If we have access to higher-order logic, we can ban all combinations in one higher-order constraint
  - `h(X):- P1(X),P2(X),type(P1, animal), type(P2,animal), P1 \= P2`

# Inductive Bias

- There are systems that propositionalize the ILP problem and solve it using a SAT solver
  - Popper (Cropper et. al)
  - ILASP (Law et. al)
- This formulation often allows for very robust inductive biases

```
head_var(Clause,Var):- head_literal(Clause,_,_,Vars), var_member(Var,Vars).

body_var(Clause,Var):- body_literal(Clause,_,_,Vars), var_member(Var,Vars).

% prevent singleton variables
:- clause_var(Clause,Var), #count{P,Vars: var_in_literal(Clause,P,Vars,Var)} == 1.

% head vars must appear in the body
:- head_var(Clause,Var), not body_var(Clause,Var).

%% type matching
var_type(Clause,Var,Type):-
    var_in_literal(Clause,P,Vars,Var),
    var_pos(Var,Vars,Pos),
    type(P,Pos,Type).

:- clause_var(Clause,Var), #count{Type : var_type(Clause,Var,Type)} > 1.
```

## Summary

- We wish to find a hypothesis that, along with the background knowledge, entails all positive examples and does not entail any negative examples
- Using theta-subsumption, we can order the hypothesis space to guide our search for a hypothesis
- Inductive biases can be used to prune the hypothesis space before the induction process begins

# References

- Cropper, Andrew, and Rolf Morel. "Learning programs by learning from failures." *Machine Learning* 110.4 (2021): 801-856.
- Cropper, Andrew, and Sebastijan Dumančić. "Inductive logic programming at 30: a new introduction." *Journal of Artificial Intelligence Research* 74 (2022): 765-850.
- Law, Mark, Alessandra Russo, and Krysia Broda. "Inductive learning of answer set programs." *European Workshop on Logics in Artificial Intelligence*. Springer, Cham, 2014.
- Turcotte, Marcel, Stephen H. Muggleton, and Michael JE Sternberg. "The effect of relational background knowledge on learning of protein three-dimensional fold signatures." *Machine Learning* 43.1 (2001): 81-95.