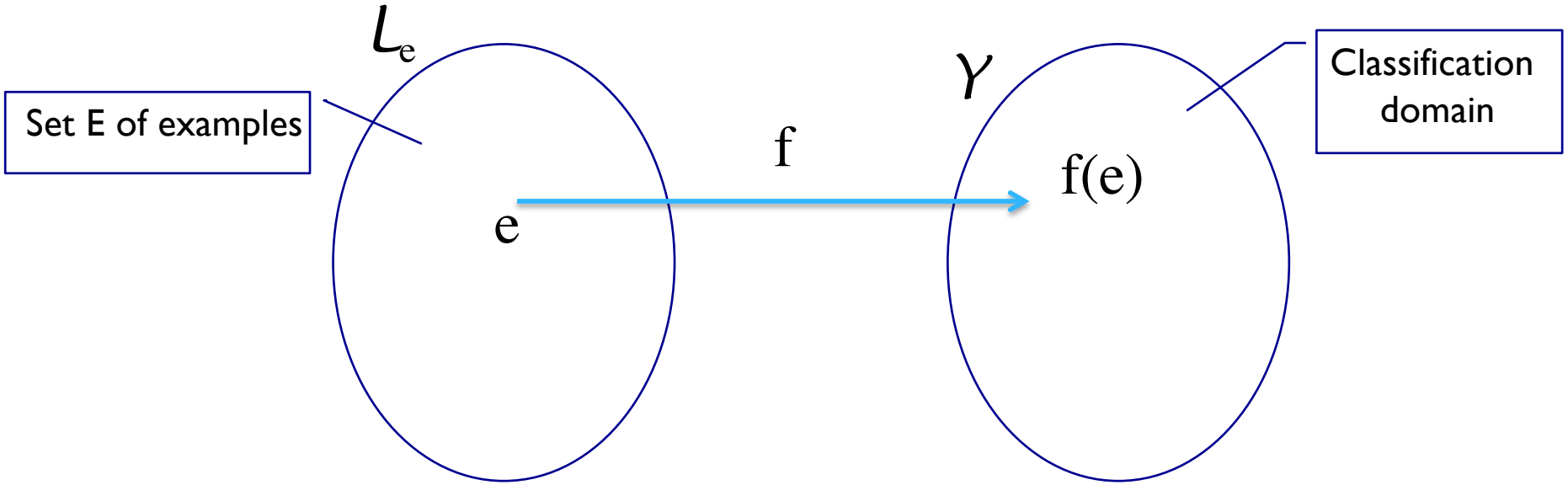
Introducing Inductive Logic Programming

- Inductive Logic Programming for concept learning
  - » *basic definitions*
  - » *ILP as a search*
  - » *version space and generality relation*
- Computational approaches
  - » *top-down (general to specific)*
  - » *bottom-up (specific to general)*
- Learning paradigms
  - » *learning from interpretation*
  - » *learning from entailment*

A Machine Learning Task



Compute approximation of the unknown function f, expressed in a language L_h, that provides information about a set E of examples.

Given

- A language of examples L_e
- A language of hypotheses L_h
- unknown target function $f: L_e \rightarrow Y$
- A set E of training examples

$$E = \{(e_1, f(e_1)), \ldots, (e_n, f(e_n))\}$$

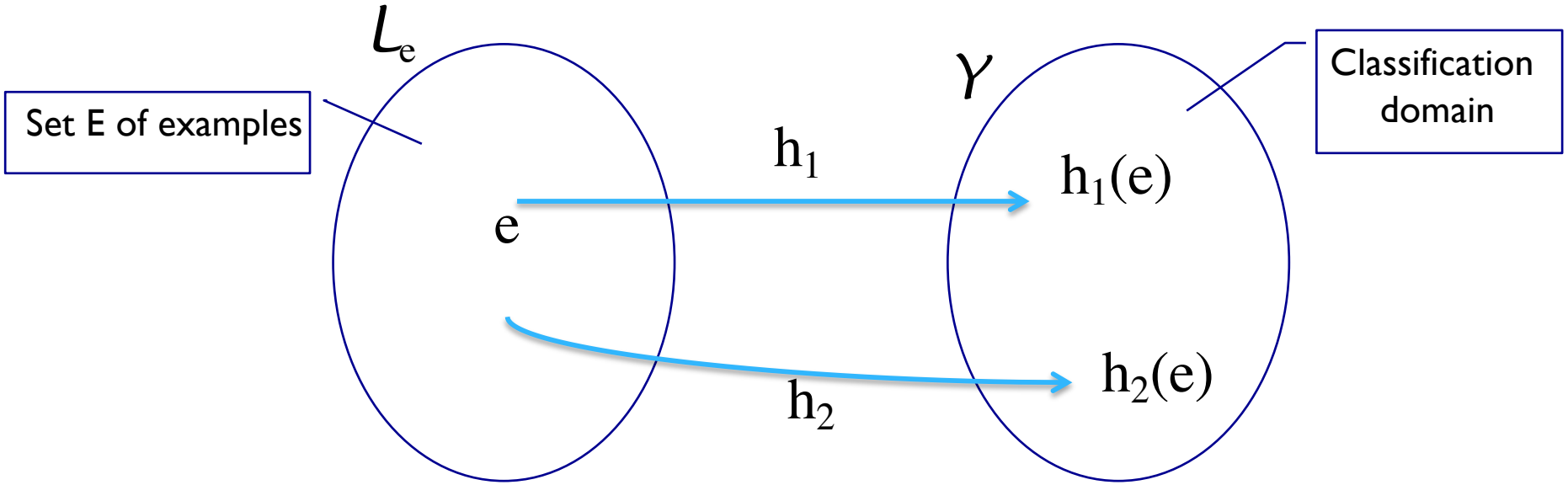
- A loss function

Find

- Hypothesis h in L_h that *minimises loss function*

$$h = \underset{h_j \in L_h}{argmin}\ loss(h_j, E)$$

A Data Mining Task



Compute hypothesis H, expressed in a language L_h, that satisfies a *quality criterion*.

Given

- A language of examples L_e
- A language of hypotheses L_h
- A set $E \subseteq L_e$ of examples
- A quality criterion $Q(h, E)$

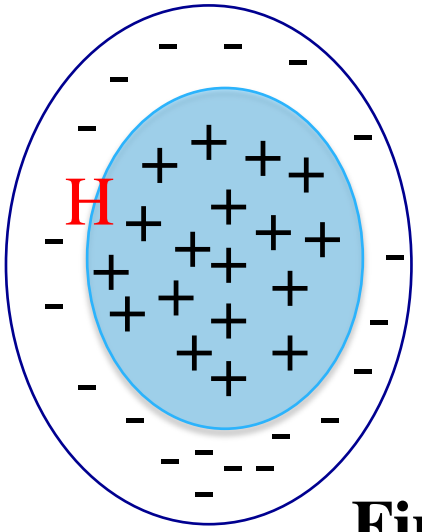
Find

- Hypothesis h in L_h that

$$Q(h_i, E) = \frac{|c(h_i, E)|}{|E|} > \varepsilon$$

Predictive ILP

$H \in L_h$



Given

- a set of observations in L_e
  - positive examples E^+
  - negative examples E^-
- background knowledge B
- hypothesis language L_H
- covers relation

Find

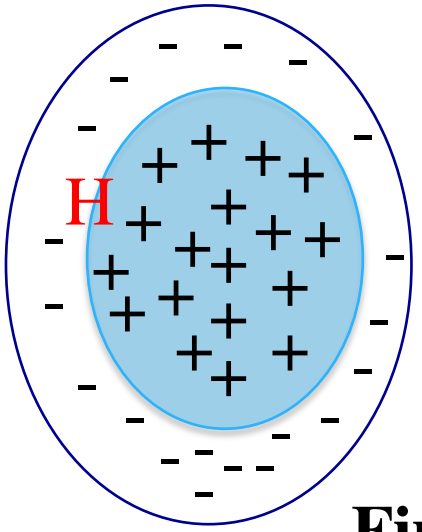
- hypothesis H in L_h

$c(H) = E^+$

Definition of coverage depends on the type of learning task

Predictive ILP

$H \in L_h$



Given

- a set of observations in L_e
  - positive examples E^+
  - negative examples E^-
- background knowledge B
- hypothesis language L_H
- covers relation
- quality criterion

Find

- hypothesis H in L_h

$Q(h_i, E) = \max |c(h_i, E)|$

ILP as search of program clauses

Languages L_e and L_h are logic-based languages.

Searching an hypothesis in the hypothesis space means finding hypothesis that satisfies the covers relation and quality criterion (if needed).

Naïve generate-and-text algorithm

For all $H \in L_h$ do
    if H satisfies the covers relation and maintains or improves the quality criterion, then output H

Computationally expensive as the whole search space could in principle be explored.

We need to structure the search process

Concept Learning and Version Space

Infer the general definition of a concept given examples labelled as members or nonmember of that concept.

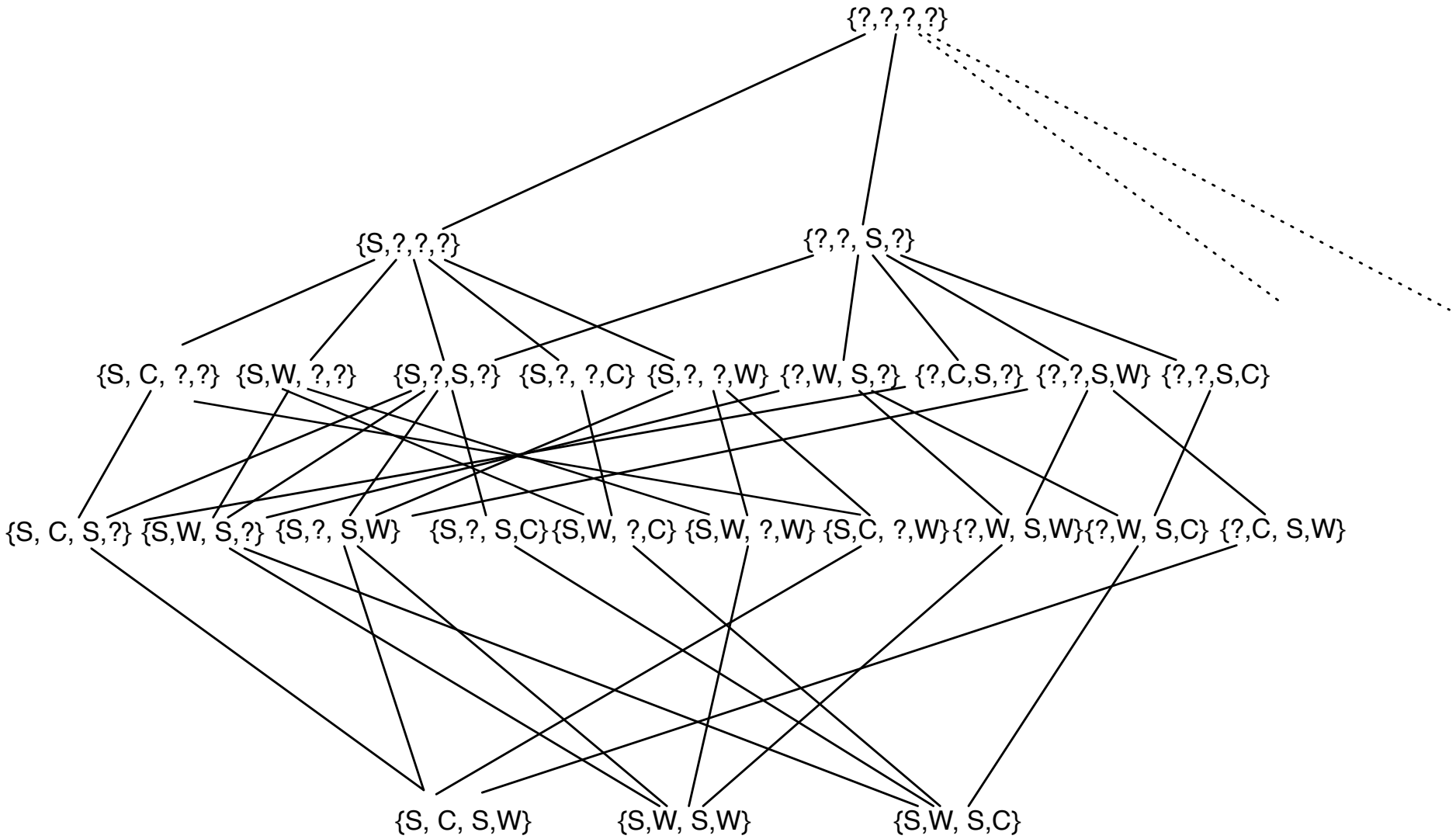
Examples (E)

Sky	Temp	Wind	Water	EnjoySport
Sunny	Warm	Strong	Warm	Yes
Sunny	Warm	Strong	Cold	Yes
Sunny	Cold	Weak	Cold	No
Sunny	Cold	Strong	Warm	Yes
Rainy	Cold	Strong	Cold	No

Concept to learn:

What are the days my friend enjoys his favorite water sports?

Concept Learning and Version Space



Generality in Version Space

- Central to search over a version space is the notion of generality:

  h is more general then h' $(h \succcurlyeq h')$ iff $c(h', E) \subseteq c(h, E)$

- Version Space is limited by its
  - top ({all examples covered})
  - bottom ({just positive examples covered})

- Keeping the entire Version Space is not feasible. Most concept learners try to find one hypothesis in the version space.

Generality property can be used for search

- If h covers negative example, then any g $\succcurlyeq$ h also covers negatives
- If h does not cover some positives, then any s $\preccurlyeq$ h does not cover those positive examples either.

ILP as search of program clauses

ILP learner can be described by

- **Structure of the hypothesis space**

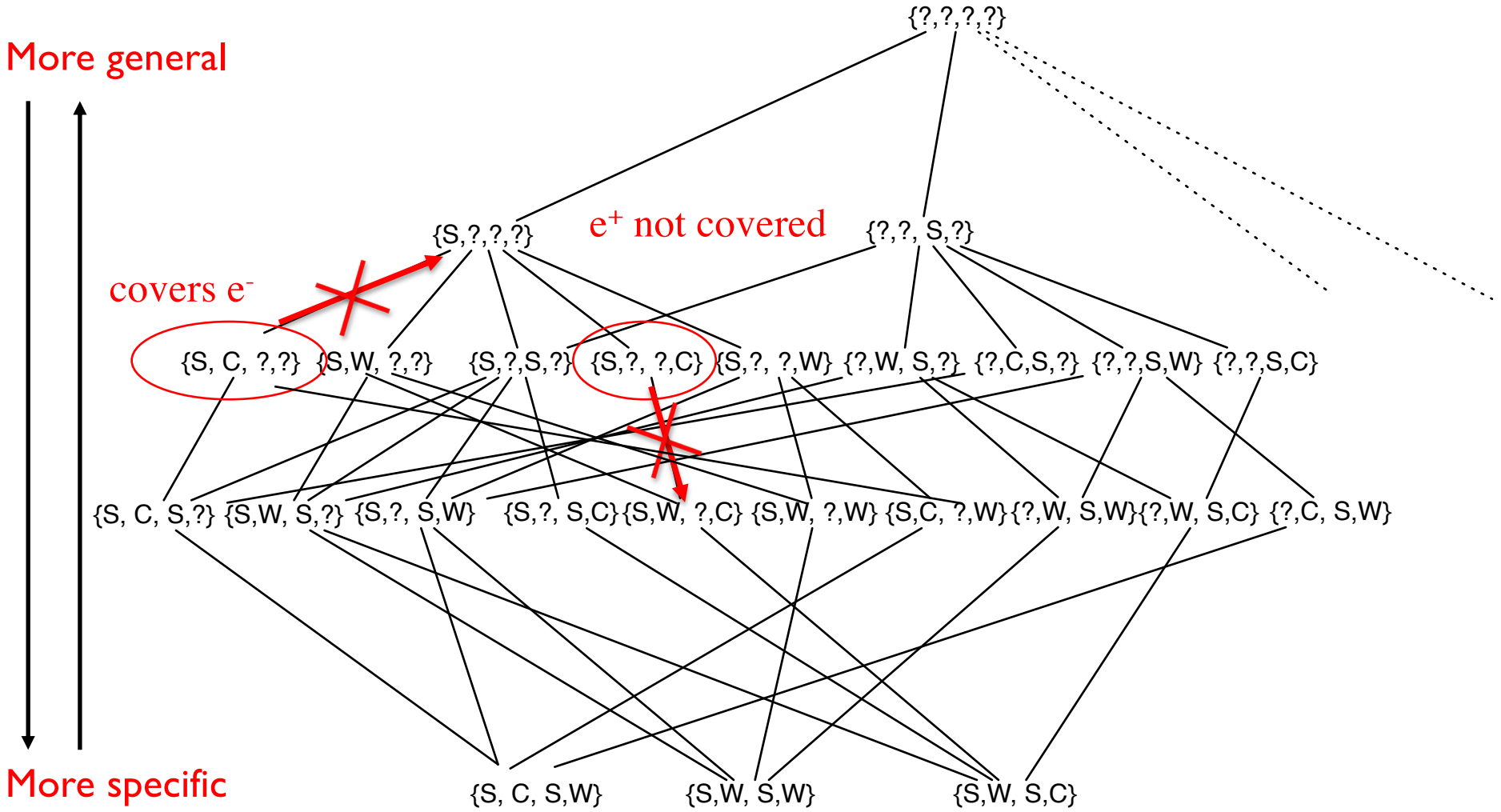
  based on generality relation:
  h is more general then h' iff covers(h') ⊆ covers(h)

- **Search strategy** (or pruning strategy)
  - Uninformed strategy (depth-first, random walk, etc.)
  - Heuristic search

  If h does not cover e then no specialisation h' covers e
  used with positive examples for pruning
  If h does cover e then all generalisations h' will also cover e
  used with negative examples for pruning

- **Heuristics**
  - for directing search
  - for stopping search (quality criterion)

ILP as search of program clauses



Learning as a search

Given

- a set of observations in L_e
  - positive examples E^+
  - negative examples E^-
- background knowledge B
- hypothesis language L_h
- covers $c(B, L_h, e)$

Find

- A theory H in L_h such that
  - $\forall e \in E^+ : B \cup H \models e$ (H is complete)
  - $\forall e \in E^- : B \cup H \not\models e$ (H is consistent)

How do we find such hypothesis?

- Generality of concepts is a special case of logic-based learning
- ILP is not only useful for learning concepts, but in general for learning theories (e.g., constraints)
- We need generality ordering for theories

Generality of Theories

Let C and D be two definite clauses.

C more general than D if and only if $C \models D$

e.g. "all fruit tastes good" ⊨ "all apples taste good"

Subsumption

Let C and D be two definite clauses.

C subsumes D if and only if there exists a θ such that $C\theta \subseteq D$

e.g.

$p(X, Y) \leftarrow r(Y, X)$ subsumes $p(a, b) \leftarrow r(b, a)$ ($\theta = \{X/a, Y/b\}$)

$p(X, Y) \leftarrow r(Y, X)$ subsumes $p(X, Y) \leftarrow r(Y, X), q(X)$

If C subsumes D then $C \models D$

ILP as search of program clauses

ILP for definite clauses can be described by

- **Structure of the hypothesis space**

  based on generality relation:

  h is more general then h' *iff* h ⊨ h'

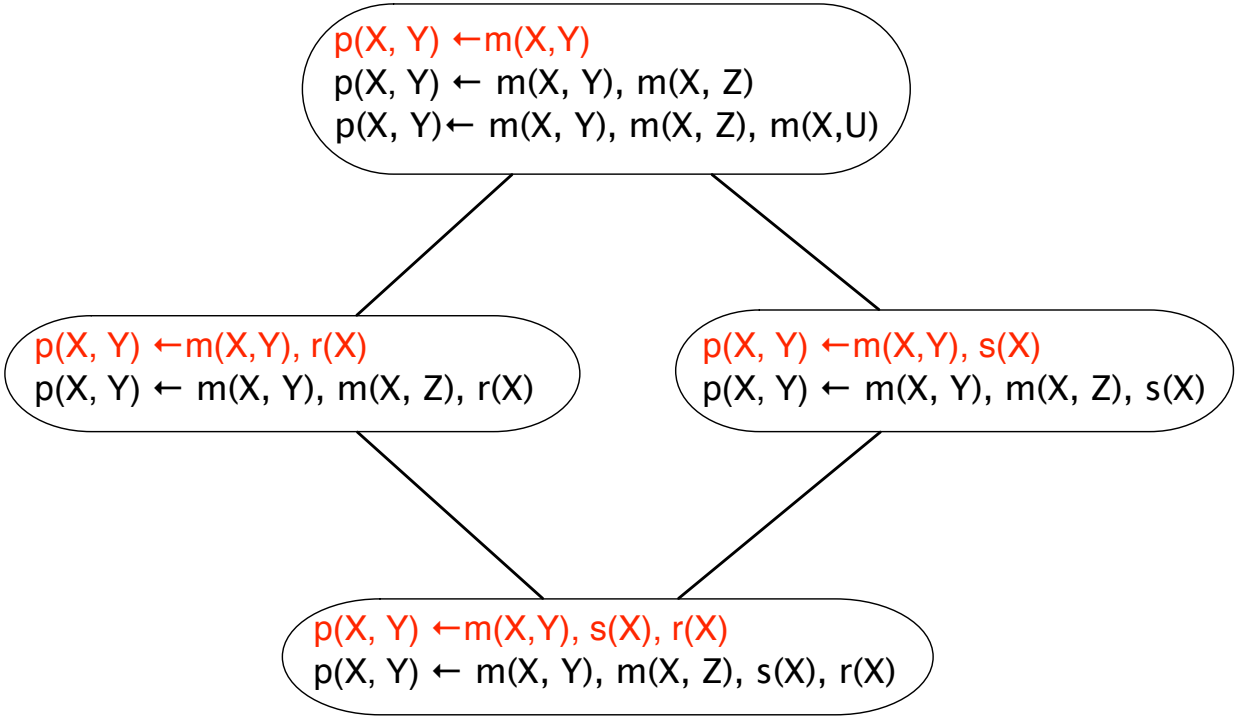
- **Search strategy** (or pruning strategy)
  - θ-subsumption

    Set of clauses H subsumes another set H' of clauses if and only if *for all clauses c'∈ H' there exists c∈ H such that* cθ-subsumes c'

Two types of search traversal over the lattice of hypotheses:

- specialisation
- generalisation

Lattice of clauses

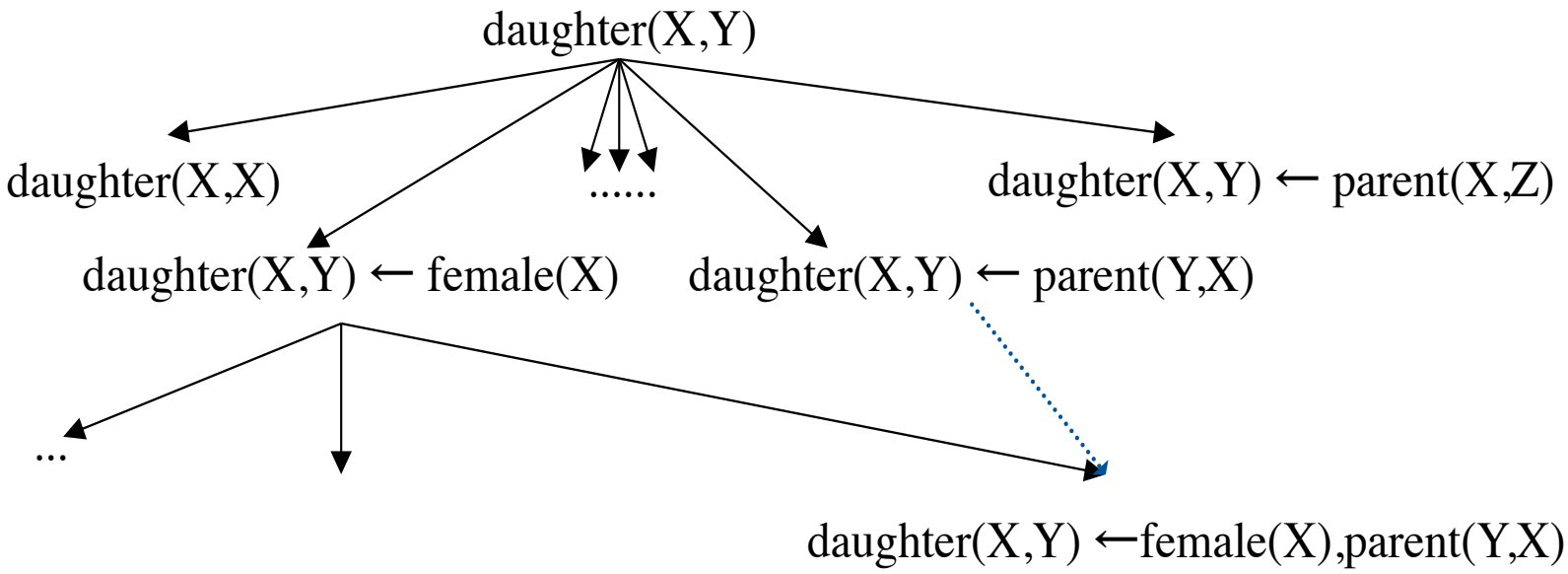


General to specific traversal

Refinement or specialisation operator **ρ** (Shapiro):

- add a literal in the body of the clause c
- apply a substitution θ

Training examples		Background knowledge	
daughter(mary,ann)	+	parent(ann, mary)	female(ann)
daughter(eve,tom)	+	parent(ann, tom)	female(mary)
daughter(tom,ann)	-	parent(tom, eve)	female(eve)
daughter(eve,ann)	-	parent(tom, ian)	



Specific to general traversal

For bottom-up search:

- Plotkin's least general generalisation (lgg) of two clauses
- Muggleton's inverse resolution

Specific to general traversal

For bottom-up search:

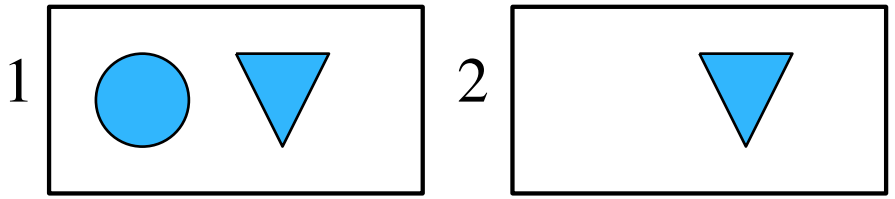
- Plotkin's least general generalisation (lgg) of two clauses
- Muggleton's inverse resolution

Consider 2 clauses, compute the most specific clause that is more general than both of the given clauses.

- lgg of terms: lgg(a, b) = X, lgg(f(X), g(Z)) = W,
  lgg(f(a,b,a), f(c,c,c)) = f(X, Y, X).
- lgg of literals: lgg(p(s1,…sn), p(t1, …, tn)) =
  p(lgg(s1,t1),…,lgg(sn, tn))
- lgg of clauses: $lgg(c_1,c_2) = \{lgg(l_1, l_2) \mid l_1 \in c_1, l_2 \in c_2$ and $lgg(l_1, l_2)$ is defined$\}$.

Specific to general traversal

Example:



pos(1).	pos(2).
contains(1, o1).	contains(2, o3).
contains(1, o2).	
triangle(o1).	triangle(o3).
points(o1, down).	points(o3, down).
circle(o2).	

Represent examples by clauses:

pos(1) ← contains(1,o1),
contains(1, o2), triangle(o1),
points(o1, down), circle(o2).
pos(2) ← contains(2, o3), triangle(o3),
points(o3, down).

Most specific clauses at the bottom of the lattice

lgg((pos(1) ← contains(1, o1), contains(1, o2), triangle(o1),
points(o1, down), circle(o2)) ,
(pos(2) ← contains(2, o3), triangle(o3), points(o3, down))

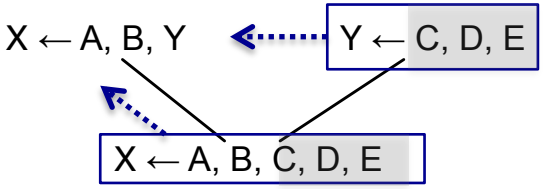
= pos(X) ← contains(X,Y), contain(X, Z), triangle(Y), points(Y, down)

Specific to general traversal

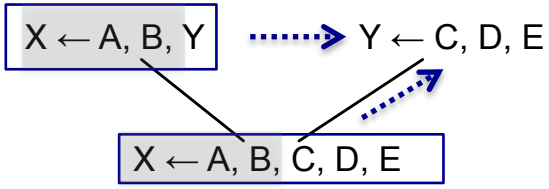
For bottom-up search:

- Plotkin's least general generalisation (lgg) of two clauses
- Muggleton's inverse resolution

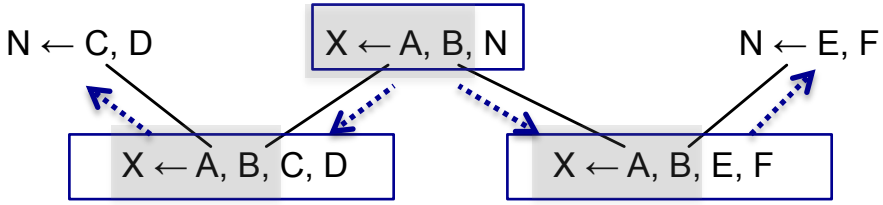
Absorption



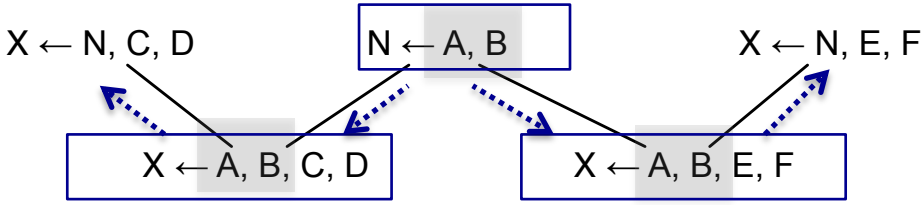
Identification



Intra-construction



Inter-construction



Learning as a search

Given

- a set of observations in L_e
  - positive examples E^+
  - negative examples E^-
- background knowledge B
- hypothesis language L_H
- covers $c(B, L_h, e)$

Find

- A theory H in L_h that covers
- $\forall e \in E^+ : B \cup H \models e$ (H is complete)
- $\forall e \in E^- : B \cup H \not\models e$ (H is consistent)

Two basic types of search operators

- θ-subsumption-based
  - specialisation operator (**ρ** refinement) — FOIL
  - generalisation operator (lgg of 2 clauses) — MIS, DUNE
- inverse resolution — CIGOL

Learning Settings

Different formalisation of concept learning in logic, depending of different representation of examples:

- Learning from interpretations
- Learning from entailment

Learning from interpretation

Different formalisation of concept learning in logic, depending of different representation of examples:

- Learning from interpretations
  - Example = interpretation (*set of* facts) e
    - contains a full description of the example
    - all information that intuitively belongs to the example, is represented in the example, not in background knowledge
  - Background = *domain* knowledge
    - general information concerning the domain, not concerning specific examples

A definite clause theory *H is a solution* if and only if e *is a model of H*

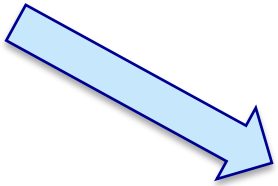
Learning from interpretation

Examples

pos: {contains(1, o1), contains(1, o2), triangle(o1), points(o1, down), circle(o2), pos(1)}
pos: {contains(2, o3), triangle(o3), points(o3, down), pos(2)}

Background knowledge

polygon(X) ← triangle(X).
polygon(X) ← square(X).



Hypothesis

pos(X) ← contains(X,Y),
triangle(Y),
points(Y, down).

Learning from entailment

Different formalisation of concept learning in logic, depending of different representation of examples:

- Learning from entailment
  - Examples E = < E^+, E^-> *(single example e is a ground fact)*
  - Background B = definite clause theory
  - FIND hypothesis H (definite clause theory) such that
    - $B \cup H \models e^+$ for all $e^+ \in E^+$
    - $B \cup H \not\models e^-$ for all $e^- \in E^-$

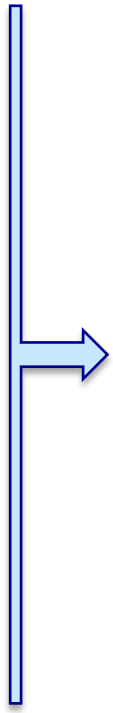
Learning from entailment

Examples

pos(1).
pos(2).
¬ pos(3).

Background knowledge

contains(1, o1).
contains(1, o2).
contains(2, o3).
triangle(o1).
triangle(o3).
points(o1, down).
points(o3, down).
circle(o2).
contains(3, o4).
circle(o4).



Hypothesis

pos(X) ← contains(X,Y),
triangle(Y),
points(Y, down).

Summary

- Introduced ILP as concept learning
- Defined a notion of generality relation
- Generality in terms of entailment and then θ-subsumption
- Search operators
  - Specialisation operators for top-down search
    - Shapiro's ρ refinement operator
  - Generalisation operators for bottom-up search
    - Plotkin's least general generalisation (lgg)
    - Muggleton's inverse resolution
- Two learning paradigms
  - Learning from interpretation
  - Learning from entailment