Inductive Logic Programming
Basic Approaches

By Chongbing Liu

Mar. 15, 2004

Outlines

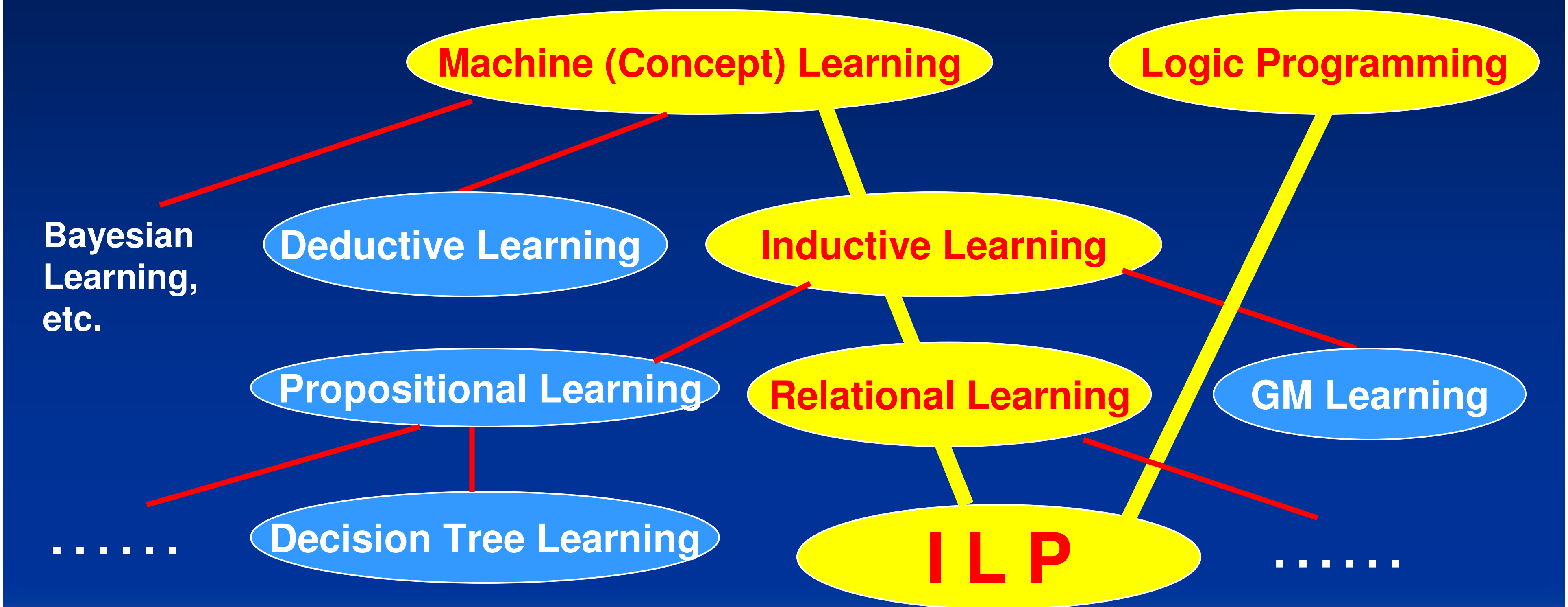
- Introduction
- Basic Approaches
- A Simple Demo
- ILP In The Future

Introduction

- ILP and Machine Learning
- Problem Specification
- An Simple Example
- Different ILP Learners
- ILP Search Space

Introduction:

ILP and Machine Learning (1)



Learning concepts inductively and directly from given examples and background knowledge, with first-order logic as the only representation

Introduction:

ILP and Machine Learning (2)

- is the intersection of inductive Machine Learning and Logic Programming
- inductively and directly learns concepts
- examples, background knowledge and target concepts are all represented in form of first-order logic
- more powerful than propositional learning (because ……)

Introduction:

ILP Problem Specification

Given

- training examples $\mathcal{E}$, $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$
- background knowledge $\mathcal{B}$, $\mathcal{B} \not\models \mathcal{E}$

Find

- target concepts $\mathcal{H}$, which are
  complete, $\mathcal{B} \cup \mathcal{H} \models \mathcal{E}^+$, and
  consistent, $\mathcal{B} \cup \mathcal{H} \cup \mathcal{E}^- \not\models \Box$

with respect to $\mathcal{B}$ and $\mathcal{E}$

Introduction:

An Simple Example

- Training Examples

$\mathcal{E}^{+}$ = { daughter(mary, ann).
daughter(eve, tom). }

$\mathcal{E}^{-}$ = { daughter(tom, ann).
daughter(eve, ann). }

- Background Knowledge

parent(ann, mary).
parent(ann, tom).
parent(tom, eve).
parent(tom,bob).
female(ann).
female(mary).
female(eve).

- Concept learned:

$$daughter(X,Y) \leftarrow female(X), parent(Y,X)$$

Introduction:

Different ILP Learners

- Single concept learners and multiple concepts learners
- Batch learners and incremental learners
- Non-Interactive learners and interactive learners

Empirical ILP Learners:

non-interactive single concept batch learners

Introduction:

ILP Search Space (1)

structure

- ILP problem is a search problem
- ILP search space consists of all syntactically legal hypotheses (clauses) constructed from the predicates provided by the background knowledge
- Very big search space

Introduction:

ILP Search Space (2)

An Example

If target concept : *daughter(X,Y),*

background knowledge : *femal(tom),…, parent(tom,bob)*

then the following hypotheses are in the search space:

$d(X,Y)$

$d(X,Y) \leftarrow f(X)$

$d(X,Y) \leftarrow f(y)$

$d(X,Y) \leftarrow p(X,Y)$

$d(X,Y) \leftarrow p(Y,X)$

$d(X,Y) \leftarrow p(X,X)$

$d(X,Y) \leftarrow p(Y,Y)$

$d(X,Y) \leftarrow f(X), p(X,Y)$

$d(X,Y) \leftarrow f(Y), p(X,Y)$

$d(X,Y) \leftarrow f(X), p(Y,X)$

$d(X,Y) \leftarrow f(Y), p(Y,X)$

$d(X,Y) \leftarrow f(X), p(X,X)$

$d(X,Y) \leftarrow f(Y), p(Y,Y)$

$d(X,Y) \leftarrow f(X), p(X,Z)$

…………

Introduction:

ILP Search Space (4)

Generality relations

- More-general-than
- More-specific-than
- No-more-general-than
- No-more-specific-than

(defined based on Θ-subsumption)

Introduction:

ILP Search Space (5)

Operations

Specialization*s* :

- Add literals into the clause body
- Apply substitutions to the clause

Generalizations :

- Remove literals from the clause body
- Apply inverse-substitutions to the clause

Introduction:

ILP Search Space (6)

Refinement graph

daught(X,Y) ←

……

daught(X,Y) ← X=Y

daught(X,Y) ← female(X)

daught(X,Y) ← parent(Y,X)

daught(X,Y) ← parent(X,Z)

……

daught(X,Y) ← female(X), female(Y)

……

daught(X,Y) ← female(X), parent(Y,X)

Outlines

- Introduction
- ILP Approaches
- A Simple Demo
- ILP In The Future

Basic Approaches

- Top-down Approaches
- Bottom-up Approaches
- Hybrid Approaches
- Summary

Basic Approaches:

Top-down Approaches (1)

Overview

- Use specialization
- Keep a refinement graph
- Search from the most general clause down to less general clauses, i.e., top-down,

(potentially adding literals into the clause body)

Basic Approaches:

Top-down Approaches (2)

Generic Top-down Algorithm (for e.g. FOIL)

specialization

```
E'  := E
H   := Ø
repeat
  c := T ←.
  repeat
     c := refinement(c)
  until some criterion is satisfied
  H := H U {c}
  B := B U {c}
  E' := E' – { positive examples covered by B }
until some criterion is satisfied
```

Basic Approaches:

Bottom-up Approaches (1)

Two main techniques:

- Relative Least General Generalization
- Inverse Resolution

(all use inverse substitution in different ways)

Basic Approaches:

Bottom-up Approaches (2)

Relative Least General Generalization (*rlgg*)

- *rlgg* is based on least general generalization *lgg*
- *lgg(structure1,structure2)* is defined between any two structures, e.g.,
  lgg(parent(ann,mary), parent(ann,tom)) = parent(ann,X)
- *rlgg* is *lgg* of two ground atoms *A1* and *A2* with respect to background knowledge *K*,
  rlgg(A1,A2) = lgg((A1←K),(A2←K))

Basic Approaches:

Bottom-up Approaches (3)

rlgg based algorithms (for e.g. Golem)

```
//assume background knowledge is a set of ground facts
E'  := E
H   := Ø
Repeat
    Ep := Ø
    c := a clause which covers no examples
    repeat
       Ep := randomly pick several pairs of examples from E' - Ep
       compute rlggs of the pairs using rlgg(e1,e2) = lgg((e1←B),(e2←B))
       compute rlggs of the rlggs obtained above and c
       c := choose the rlgg with the greatest coverage
       Ep := Ep – { those examples covered by c }
    until no more positive examples are covered by c
    H := H U {c}
    B := B U {c}
    E' := E' – { positive examples covered by H and B }
until some criterion is satisfied
```

generalization

Basic Approaches:

Bottom-up Approaches (4)

Inverse Resolution

Resolution:

Given clause c1 and c2, derive resolvent c

Inverse resolution:

Given clause c1 and resolvent c, derive another clause c2

Two forms:

propositional form

first-order logic form

Basic Approaches:

Bottom-up Approaches (5)

Inverse Resolution

propositional form:

resolution (find L in c1 and ┐L in c2)

$c1 \wedge c2 \vdash c$

$c = (c1 - \{L\}) \cup (c2 - \{\neg L\})$

inverse resolution

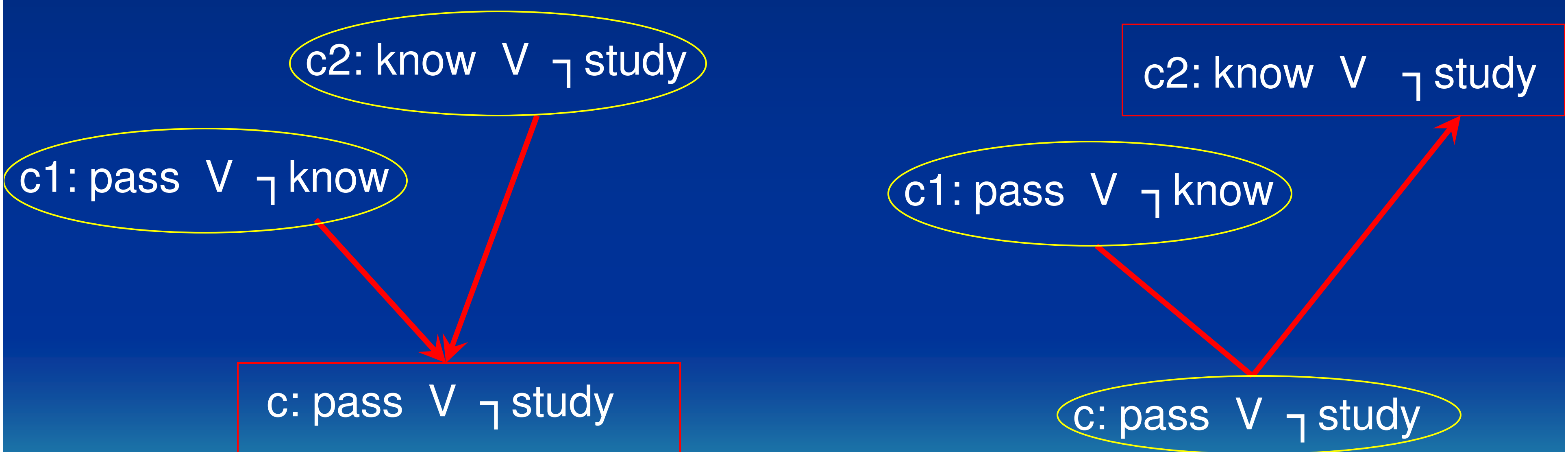
$c2 = (c - (c1 - \{L\})) \cup \{\neg L\}$

Basic Approaches:

Bottom-up Approaches (6)

Inverse Resolution

propositional form: (an example)



Basic Approaches:

Bottom-up Approaches (7)

Inverse Resolution

first-order logic form:

resolution

(*find* L_1 *in c1 and* L_2 *in c2 s.t.* $L_1\Theta_1 = L_2\Theta_2$)

$c1 \wedge c2 \vdash c$

$c = (c1 - \{L_1\})\,\Theta_1 \cup (c2 - \{L_2\})\,\Theta_2$

inverse resolution

$c - (c1 - \{L_1\})\,\Theta_1 = (c2 - \{L_2\})\,\Theta_2$

$L_2 = L_1\,\Theta_1\,\Theta_2^{-1}$

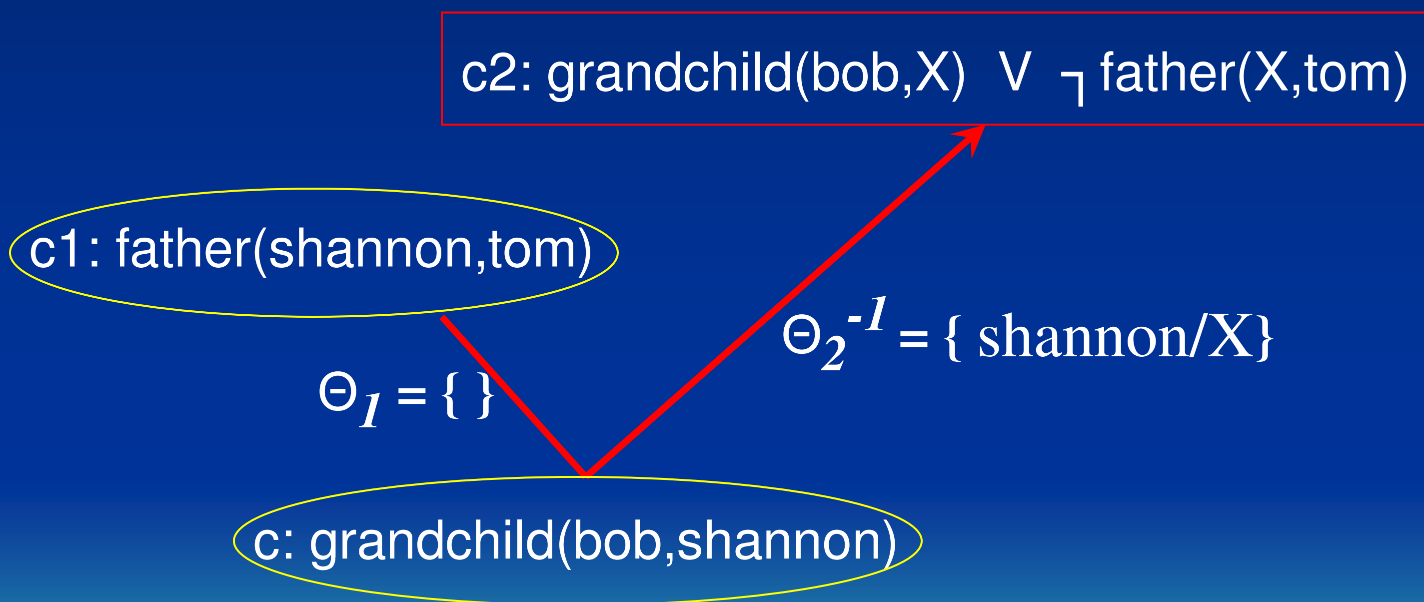
$c2 = (\ c - (c1 - \{L_1\})\,\Theta_1\ )\,\Theta_2^{-1} \cup \{\neg L_1\,\Theta_1\,\Theta_2^{-1}\}$

Basic Approaches:

Bottom-up Approaches (8)

Inverse Resolution

first-order logic form: (an example)



Basic Approaches:

Bottom-up Approaches (9)

Inverse resolution based algorithms (for e.g. Cigol)

generalization

E’ := E
H := Ø
while E’ ≠ Ø do
    e := the next positive example
    invs := all the inverse resolutions of e and B
    c := choose the one with the highest accuracy
    H := H U {c}
    B := B U {c}
    E’ := E’ – { positive examples covered by B }

Basic Approaches:

Hybrid Approaches (1)

- Use most specific boundary (Progol)
- Use general and specific boundaries
- Translate into propositional learning problem

Basic Approaches:

Hybrid Approaches (2)

Use most specific boundary (Progol)

inverse resolution

E' := E
H := Ø
while E' ≠ Ø do
    e := the next positive example
    ⊥ := the most specific clause from e and B
    c := top-down search a best clause between T←. and ⊥
    H := H U {c}
    B := B U {c}
    E' := E' – { positive examples covered by B }

Basic Approaches:

Hybrid Approaches (3)

Use most specific boundary (Progol)

Consider examples one by one, using resolution, sounds *bottom-up*

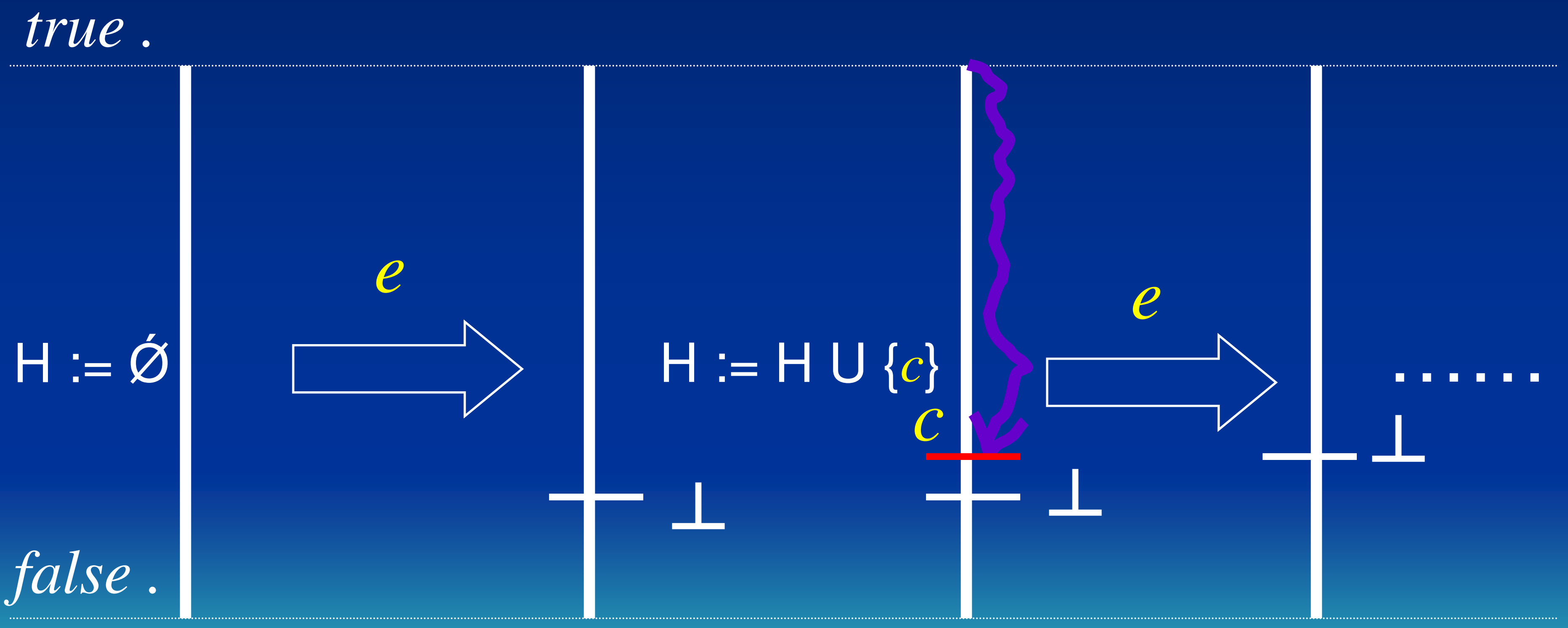
Use resolution to construct a lower bound $\bot$ from B and e , instead of a hypothesis directly

Then search from the very top down $\bot$ to find a clause which covers e and does not cover any of the negative examples, sounds *top-down*

Basic Approaches:

Hybrid Approaches (4)

Use most specific boundary (Progol)



Basic Approaches:

Hybrid Approaches (5)

Maintain a upper bound and a lower bound

E' := E

Bound_{upper} = true ← .

Bound_{lower} = fase ← **true .**

while E' ≠ Ø do

- e := the next positive example
- if e is positive
  - Bound_{lower} = generalize(Bound_{lower}) which covers e but none of E^-
- if e is negative
  - Bound_{upper} = **specialize**(Bound_{upper}) which does not cover e
- E' := E' – {e}

H := { those hypotheses in between Bound_{upper} and Bound_{lower} }

Basic Approaches:

Hybrid Approaches (6)

Maintain a upper bound and a lower bound

- _Positive examples

  raise the lower bound up

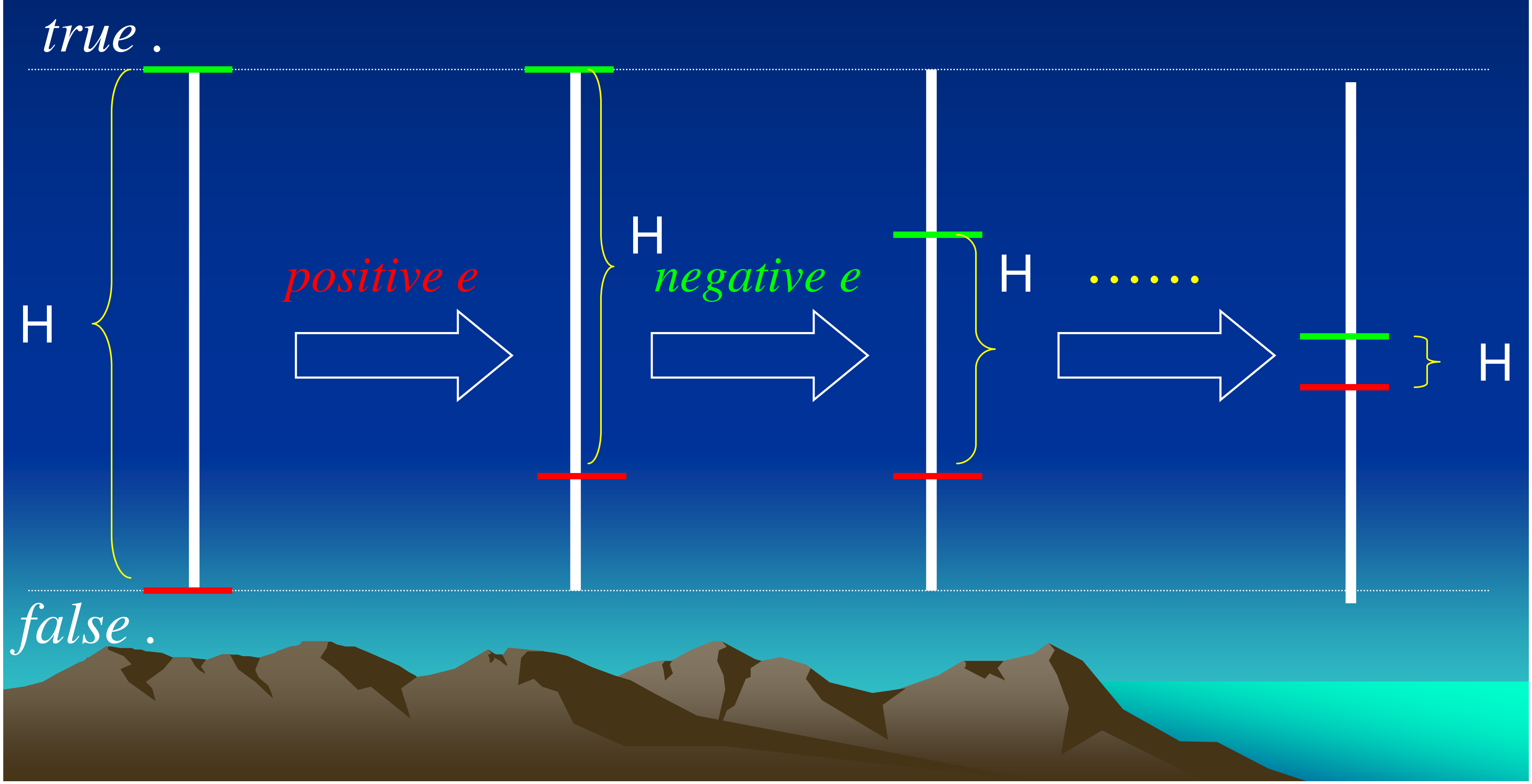
- Negative examples

  push the upper bound down

Basic Approaches:

Hybrid Approaches (7)

Use upper and lower boundaries



Basic Approaches:

Summary

Top-down approaches:

- perform specialization operations
- search the refinement graph top-down in brute-force, unique starting point
- successor hypotheses generated based only on the syntax of the current hypothesis representation, independent of the coming data
- generate-and-test fashion
- the impact of noisy data is minimized
- batch mode: all examples are considered simultaneously

Bottom-up approaches:

- perform generalization operations
- search is guided bottom-up by inverse resolution, multiple starting points
- hypotheses generated based on analysis of an individual example
- example-driven fashion
- more easily misled by noisy data
- incremental mode: examples are considered one at a time

Outlines

- Introduction
- ILP Approaches
- A Simple Demo
- ILP In The Future

A Simple Demo:

Sample Problem

Target concept:

daughter(X,Y)

Background Knowledge:

parent(ann,mary).	*female(ann).*
parent(ann,tom).	*female(mary).*
parent(tom,eve).	*female(eve).*
parent(tom,ian).	

Examples:

% Positive examples	*% Negative examples*
daughter(mary,ann).	*daughter(tom,ann).*
daughter(eve,tom).	*daughter(eve,ann).*

A Simple Demo:

Progol Output (1)

CProgol Version 4.4
[Testing for contradictions]
[No contradictions found]

[Generalising daughter(mary,ann).]
[Most specific clause is]

daughter(A,B) :- parent(B,A), female(A), female(B).

A Simple Demo:

Progol Output (2)

[Learning daughter/2 from positive examples]

[C:-9993,8,10000,0 daughter(A,B).]

[C:-9994,8,10000,0 daughter(A,B) :- parent(B,A).]

[C:-9994,8,10000,0 daughter(A,B) :- female(A).]

[C:-19996,4,10000,0 daughter(A,B) :- female(B).]

[C:3,8,2,0 daughter(A,B) :- parent(B,A), female(A).]

[C:-19998,4,10000,0 daughter(A,B) :- parent(B,A), female(B).]

[C:-19998,4,10000,0 daughter(A,B) :- female(A), female(B).]

[7 explored search nodes]

f=3,p=8,n=2,h=0

A Simple Demo:

Progol Output (3)

[Result of search is]

daughter(A,B) :- parent(B,A), female(A).

[2 redundant clauses retracted]

daughter(A,B) :- parent(B,A), female(A).

[Total number of clauses = 1]

[Time taken 0.00s]

Outlines

- Introduction
- ILP Approaches
- A Simple Demo
- ILP In The Future

ILP In The Future

- *Novel search methods*
- *Incorporation of explicit probabilities*
- *Special-purpose reasoners*
- *Parallel implementations (PILP)*
- *Enhanced human interaction*

(handle huge data sets in the future)

Thank You !