simula

Crash Course
Machine Learning

An Artificial Intelligence Perspective on Symbolic Machine Learning (ML)

24 Aug. 2016

Arnaud Gotlieb
Simula Research Laboratory
arnaud@simula.no

Certus

sfi Centre for Research-based Innovation

Agenda

1. Introduction
2. Basics in Logics
3. Programming in Logic (Prolog)
4. Decision Tree Induction
5. Generalization/Specialization
6. Version Space Search
7. Conclusions

INTRODUCTION

Artificial Learning

First learning algorithms appeared at the very beginning of Artificial Intelligence, 60 years ago

Herbert Simon (co-founder of Artificial Intelligence):
"Learning is any process by which a system improves performance from experience"

- Self-adaptation, self-modification to improve performances

- Discover regularities in a **knowledge basis** in order to understand or predict unknown rules

Knowledge Acquisition

- Learning involves **languages to represent knowledge** and **tools to manage knowledge basis**, including:
  - **Definitions:** *"Certus is a research-based innovation center dedicated to Software Validation and Verification"*
  - **Categories:** *"Engineers"   "Researchers"*
  - **Assertions:** *"When it rains, an umbrella is useful"*
  - **Exceptions:** *"Some students know more than the teacher"*
  - **Quantifiers (∃, ∀):** *"In class, all students lesson and there is a teacher"*
  - **Beliefs, Possibilities**, ...

- How to **infer new knowledge**? How to **deduce non-explicit knowledge?**

(BTW: What's the difference between deduction and induction?)

Deduction and Inference

- Socrates is a man, all men are mortal → Socrates is mortal **(Deduce)**
- Socrates is mortal, Plato is mortal, Socrates is a man, Plato is a man → All men are mortals (or All mortals are men) **(Infer)**

Learning is mostly concerned with inference, using knowledge representation methods developed in automated deduction

(But, how to learn using inference ?)

Learning by Inference

- An **observation** is a couple (X, Y) where where X can be a vector of data, a proposition expressed in a given Logic, or a formula and Y is a label

- A **training set** is a set of observations $S = \{(X_1, Y_1), \ldots, (x_N, Y_N)\}$

- The goal of **supervised learning** is:

  - either to predict the label Y_{N+1} of a new observation X_{N+1}
  - or to find the **most general hypothesis** h which matches all the observations

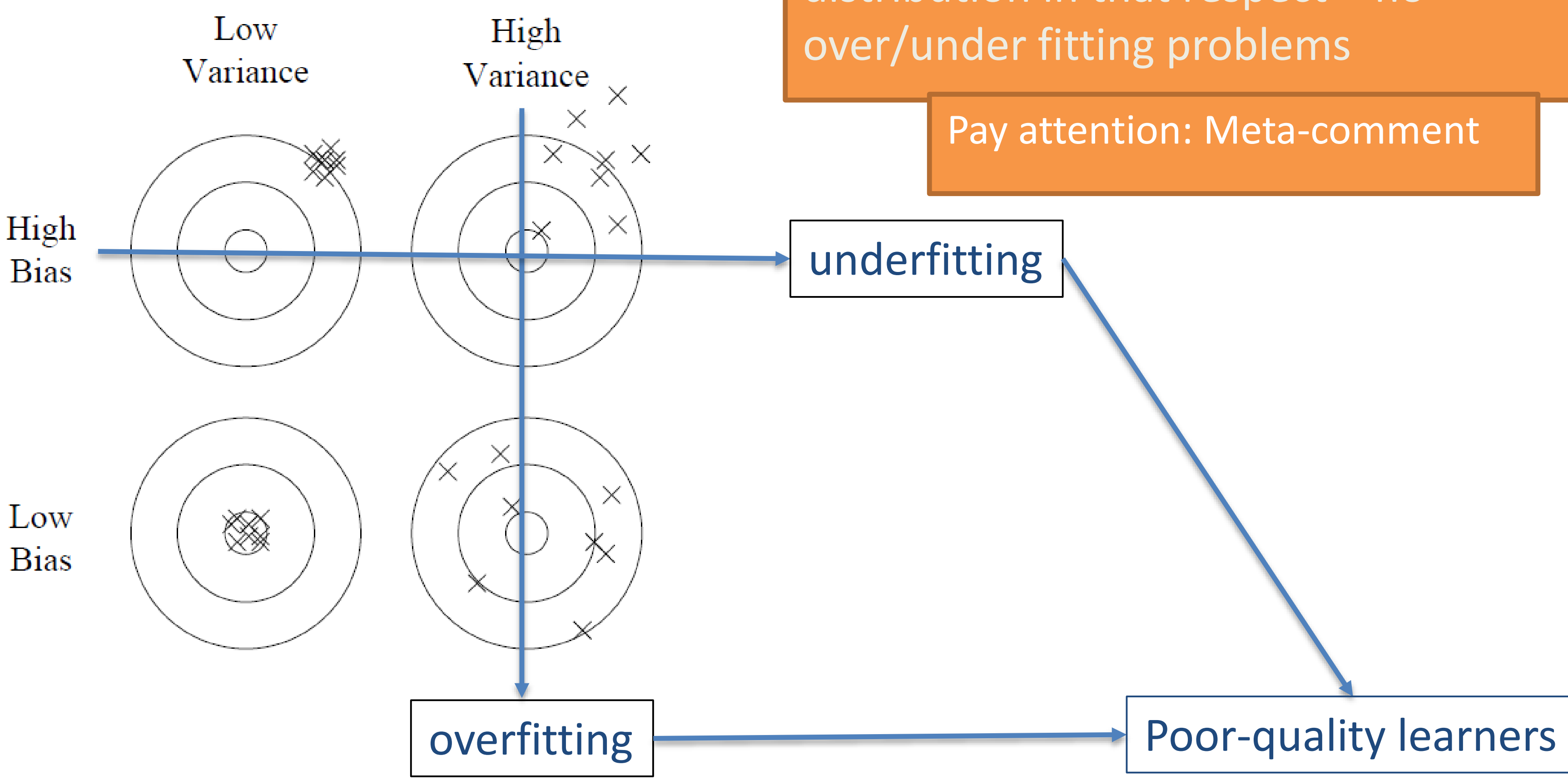
- Any learning algorithm can be seen as a function (deterministic or not), which maps S to h, where h belongs to an **Hypothesis Space**

Some Notes on Data Quality (1)

- The distribution of data $\{(X_1, Y_1), \ldots, (x_N, Y_N)\}$ is crucial for any learning algorithm!

In today's course, we will assume data distribution in that respect – no over/under fitting problems

Pay attention: Meta-comment



P. Domingos: *"A Few Useful Things to Know about ML"* CACM 2012

Some Notes on Data Quality (2)

- Data quality of $S = \{(X_1, Y_1), \ldots, (x_N, Y_N)\}$ is also related to:

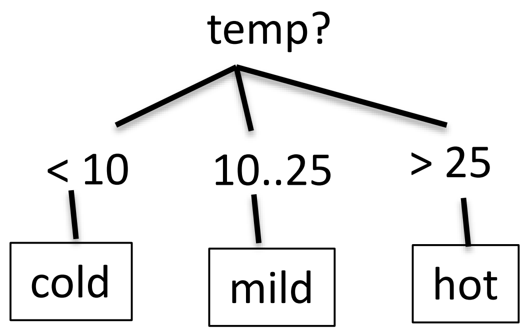
  - **Incorrectness:** when Y_i is incorrectly assigned to Xi
  - **Imprecision:** when X_i is only known with some error measurements,
    e.g., $X_i = v_i \pm \varepsilon$ or X_i in a..b
  - **Incompleteness**: when only some parts of X_i is known
  - **Time-dependency:** when Y_i is a function of t,
    e.g., Y_i = a on Day1 and Y_i=b on Day2

In today's course, we will consider methods that can deal with imprecision and incompleteness

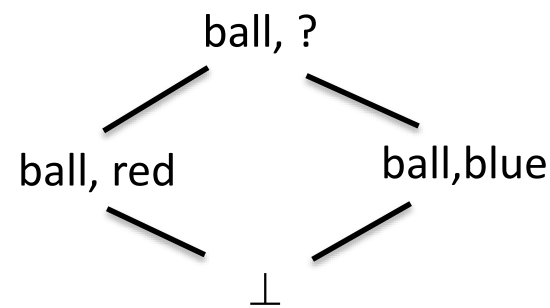
Hypothesis Space

Knowledge acquisition through numercial or symbol-based data structures:

- **$\Re$, $\Re^n$**
- **Decision trees**



- **Lattices** (partially ordered set with unique sup. and inf.)



- **Association rules**

$\{$butter, bread$\} \Rightarrow \{$milk$\}$

- **Constraint networks**

$\{x \text{ in } 3..7,\ y \text{ in } 5..18,\ x+1 = 2*y,\ x \neq y\}$

- ….

Hypothesis Space: Language

Requirements:

- Appropriate knowledge representation
- Facilitate revision and update
- Enable convenient navigation, be operational

Candidates:

Propositional Logic

First-Order Logic

Horn Logic

…

Selected for the course: The **Prolog** programming language

Why Prolog ?
(<u>Pro</u>gramming in <u>Log</u>ic)

Popular in AI, Declarative and operational programming language

Used in many areas of Artificial Intelligence teaching and research
(Natural Language Processing, Knowledge Representation, Machine Learning, etc.)

Availability of several free and commercial compilers (SWI-Prolog, Eclipse-PDT, SICStus Prolog, GNU-Prolog, Cosytech-CHIP++, Visual-Prolog, …)

Large commercial applications (e.g., **IBM-Prolog in Watson**, Dassault, Boeing Corp., NASA Clarissa, Ericsson AB,…)

Selected for the lab session: SICStus **Prolog 4.3**

The Symbolic vs Connectionist AI Debate

*"**Symbolic artificial intelligence** is the collective name for all methods in artificial intelligence research that are based on high-level "symbolic" (human-readable) representations of problems, logic and search. Symbolic AI was the dominant paradigm of AI research from the mid-1950s until the late 1980s" (Source- en.wikipedia.org/wiki/Symbolic_artificial_intelligence)*

Symbolic Learning involves symbol manipulation - Explicit representation of the hypothesis space with symbols and language artefacts

Connectionist ML involves artificial neural networks and their generalization

Two competing paradigms – Both have strengths and weaknesses – Time has come to collaborate (e.g., alpha-go)

Ron Sun *"Artificial Intelligence: Connectionist and Symbolic Approaches"* U. Columbia Report, 2000

Logic (of the agenda)

1. Introduction
2. Basics in Logics
3. Programming in Logic (Prolog)
4. Decision Tree Induction
5. Generalization/Specialization
6. Version Space Search
7. Conclusions

Prerequiste

Prerequiste

1st ML algo of the course

Prerequiste

2nd ML algo of the course

BASICS IN LOGICS

Propositional Logic (PL) - Language

- **Propositions** are sentences, either true or false:
  - *"Alice is smarter than Mike"*
  - *"This car is red"*
  - "Socrates is mortal"
- **Literals** are either constants (either **true** or **false**), or Boolean variables (e.g., X_1) or their negation (e.g., $\neg X_1$)
- A **Clause** is a disjunction of literals $(X_1 \vee \neg X_2)$
- **Formulas** are logical combination of clauses
  $\neg F$ "not" (negation)
  $F_1 \wedge F_2$ "and" (conjunction)
  $F_1 \vee F_2$ "or" (disjunction)
  $F1 \rightarrow F2$ "implies"
  …

Propositional Logic - Reasoning

- The fundamental problem in PL (Deduction):

  Given a formula F, is F satisfiable or unsatifiable? (SAT problem)

  Said otherwise:
  Can we find an assignment of all variables to either **true** or **false** which satisfy F?

- Example:

Knowing that Alice's car is not red and, that either Mike's car is either blue or Alice's car is red, can we find the color of Alice and Mike cars?

X_1: Alice's car is red

X_2: Mike's car is blue

Formula F: **(X_1 or X_2) and $\neg X_1$**

Is F satisfiable?

Propositional Logic: Limitations

The following sentences cannot <u>easily</u> be encoded in PL:

- Mike's car color is the same than Alice's car color
- If Mike loves Alice, then Alice loves Mike
- All Mike's cars have a distinct color
  Etc..

No variable equality

No commutativity

No universal quantification

No predicate

➔ First-Order Logic

First-Order Logic (FOL) - Language

Terms

Constants, for ex: 0, 1, 2, mike, alice

Variables of many sorts, for ex: $X, Y, Z, \ldots$

Functions, for ex: $f(t_1,\ldots,t_n)$, father_of, plus, …

Formulas:

Predicates (with an interpretation **true** or **false)**, for ex: $P(t_1,\ldots,t_n)$, less_than, greather_than, color(grass, green), color(grass, yellow),…

Equality, for ex: $t_1 = t_2$

Negation, for ex: $\neg P(t_1,\ldots,t_n)$

Connectors, for ex: $P(X) \rightarrow Q(Y)$, $P(X) \wedge Q(Y) \vee R(Z)$

Quantifiers, for ex: $\forall X, \exists Y, P(X,Y)$

$$\forall X, \; \forall Y, \exists Z, P(X) \wedge Q(Y) \rightarrow R(f(X,Y), Z) \wedge Q(g(Z))$$

It is called first-order logic because quantification is ok only on variables, not on functions or predicates

First-Order Logic - Reasoning

- Fundamental problems in FOL (Deduction):

  Given a formula F, find an interpretation where F is true?
  Is F a « tautology » (always true) ?

- Some deduction rules in FOL (including PL deduction rules):

  Universal Elimination
  If $\forall x\ P(x)$ is true, then $P(c)$ is true, where c is a constant in the domain of x

  Existential Introduction or Elimination
  If $P(c)$ is true, then $\exists x\ P(x)$ is inferred. From $\exists x\ P(x)$ infer $P(c)$

  Paramodulation Ex: From $P(a)$ and $a=b$ derive $P(b)$

  Modus Ponens If A is true and A implies B, then B is true
  (generalized with quantifiers)

First-Order Logic: Example

In natural language:

All cats like fish,
cats eat everything they like,
and tom is a cat.

In FOL:

1. $\forall$**X cat(X) => likes(X, fish)**
2. $\forall$**X** $\forall$**Y (cat(X)** $\wedge$ **likes(X, Y))** $\Rightarrow$ **eats(X, Y)**
3. **cat(tom)**

Question: Does tom eat fish?

Q.E.D.

First-Order Logic: Limitations

- **Great expressive power**, but coming with **high computational costs**

  FOL is not decidable in general, some FOL formulae require exponential-size proofs

- In FOL, automatically deduced formula can be **quite general**

  For ex: 6. $\forall$**X cat(X)** $\Rightarrow$ **eats(X, fish)** is also a valid deduction
  but with little practical value in operational systems

- Many other Logic do exist, but only little is of practical value for knowledge representation in Machine Learning

This course uses only a subset of first-order logic which has great practical value: Horn Logic!

PROGRAMMING IN LOGIC

The Prolog Programming Language

[Colmerauer, Roussel 72]

- It is all about <u>programming</u> in <u>logic</u>
- Links with Logic: Syllogisms in Horn Logic

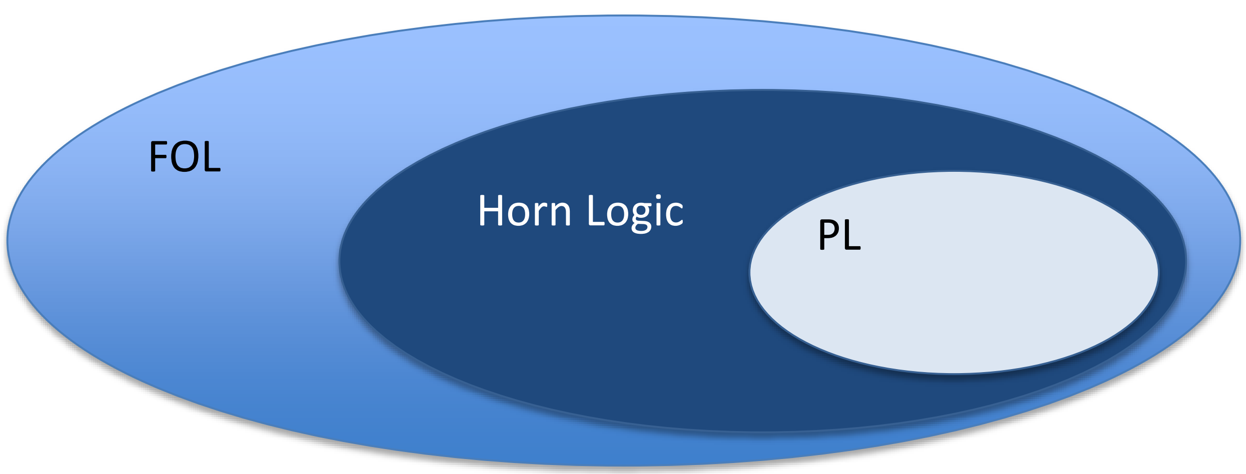
<u>Syllogism</u>	<u>Prolog</u>
Socrates is a man.	`man(socrates).`
All men are mortal.	`mortal(X) :- man(X).`
Is Socrates mortal?	`?- mortal(socrates).`

Prolog as Horn Logic
(A Restricted Form of FOL)

A **Horn clause** is a clause (a disjunction of literals) with at most one positive literal.

$\forall X \; man(X) \Rightarrow mortal(X)$ which is written `mortal(X) :- man(X).` in Prolog

Only a single positive literal



Prolog Notions

- **Forward reasoning**: reasoning from premises to conclusions.
  Inefficient when there are lots of premises

- Instead, Prolog uses **backward reasoning** -- from (potential) conclusions to facts

Facts, Rules, Questions

Fact:	`man(socrates).`
Rule:	`mortal(X) :- man(X).`
Question:	`?- mortal(socrates).`

Atoms, Variables

Atoms:	`socrates, a, aBIZ, …`	(starting with lowercase)
Variables:	`X, Xbiz, _, _XXX …`	(starting with uppercase or _)

Prolog answers questions!

example1.pl

```
man(socrates).
mortal(X) :- man(X).
```

- Prolog's "`Yes`" means "*I can prove it*",

  ```
  ?- mortal(socrates).
  Yes
  ```

- Prolog's "`No`" means "*I cannot prove it*"

  ```
  ?- mortal(plato).
  No
  ```

  (*Closed World Assumption*)

- Prolog supplies values for variables (substitutions) when possible

  ```
  ?- mortal(X).
  X = socrates
  ```

Terms

- A **Prolog term** is either an **atom,** a **variable,** or a **structure consisting of a name (functor) and several sub-terms**

Examples: `joe, man(joe), man(X), path(X,Y), t(X, a, p(b, Y))`

Functors

Pay attention: Term and functors are non-interpreted!

Arity 0, 1, 2, 3

Ground term: term with no variable

Term arguments

Term Unification

[Robinson 1965]

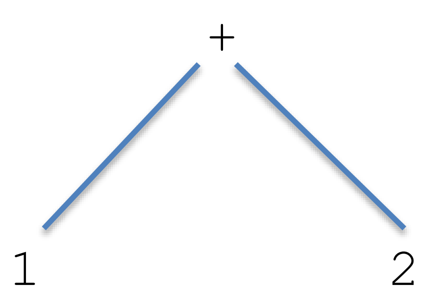
- **Unification (=)** is THE fundamental operation over terms (which can succeed or fail)

  Ex: X = a, a = b, X = Y, X = t(a), t(X) = t(Y), t(a) = t(b), t(X,a) = t(b, Y)

- Unification is bi-directional:

  Ex: X = t(a) is equivalent to t(a) = X

- **Pay attention**: by default, `1 + 2` is a term with non-interpreted functor +




```
?- X = 1+2.
X=+(1,2)

?- X is 1+2.
X=3
```

Use “`is`” for interpretation

Unification results in a set of substitutions!

```
t(X,a,p(b,Y)) = t(Z,R,S)
            Z = X
            R = a
            S = p(b, Y)
```

Unification may succeed

```
t(X,a,p(b,Y)) = t(q(3),a,p(Z,b))
         X = q(3)
         Y = b
         Z = b
```

Unification may fail

```
t(X,a,p(b,Y)) = t(q(3),b,p(Z,b))
```

```
no
```

Occur Check ?

`t(X) = X`

To be examined in the exercises of the lab. session !

Rules are made of Predicates and Clauses

example2.pl

```
man(socrates).
man(plato).
woman(judith).
greek(socrates).

mortal(X) :- man(X).
mortal(X) :- woman(X).

philosopher(X) :- mortal(X), greek(X).
```

Read these predicates by using ”or” between clauses

Read by using “and”

- This program contains 5 **predicates** (man, woman, greek, ..) and 7 **clauses** Predicates can be in any order, clauses are used as they appear

- A **predicate** is defined as a set of clauses with same functor and same arity

Declarative Reading

example2.pl

```
man(socrates).
man(plato).
woman(judith).
greek(socrates).

mortal(X) :- man(X).
mortal(X) :- woman(X).

philosopher(X) :- mortal(X), greek(X).
```

- **If any** X is mortal **and** greek **then** X is a philosopher
- **If any** X is man **or** a woman **then** X is mortal
- socrates **is a** greek, judith **is a** woman, plato **is a** man, socrates **is a** man.

Backtracking

- Prolog can provide several answers to a unique question!
- **Bactracking** is the process that allows Prolog to implement this behavior
- Bactracking enables **non-determinic search** in Prolog

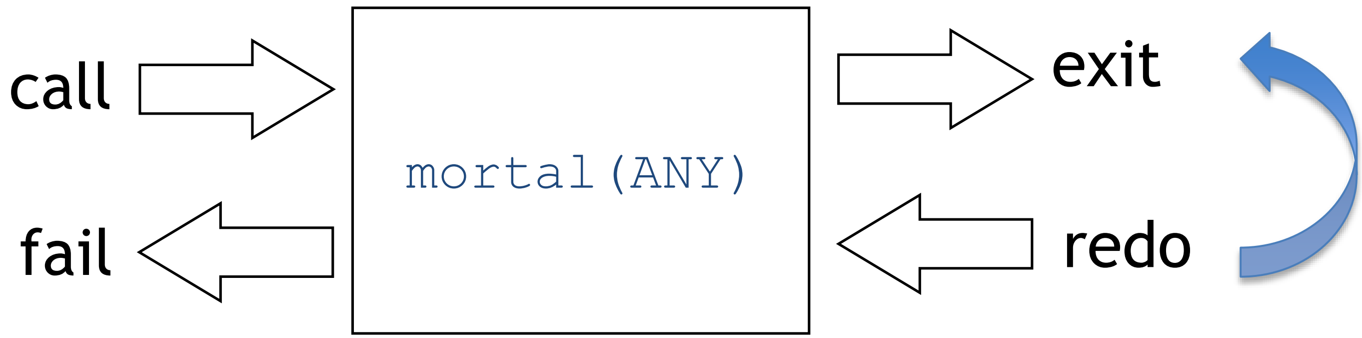
```
man(socrates).
man(plato).
woman(judith).
…
mortal(X) :- man(X).
mortal(X) :- woman(X).
…
```

```
?- mortal(ANY).

        ANY = socrates ? ;
        ANY = plato ? ;
        ANY = judith ? ;
        no
```

How does Bactracking work?

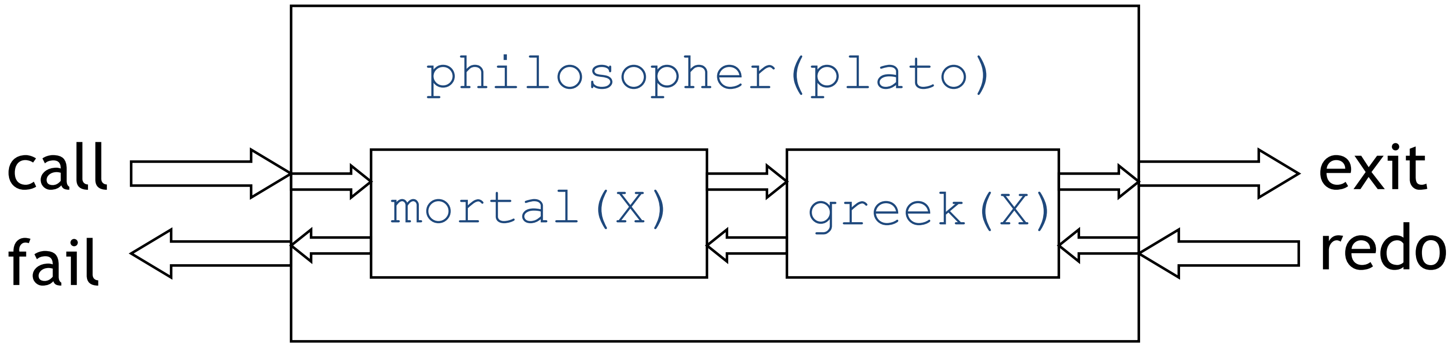
- By using the ***"Byrd Box"*** model




```
ANY = socrates ?  ;
ANY = plato ?     ;
…
no
```

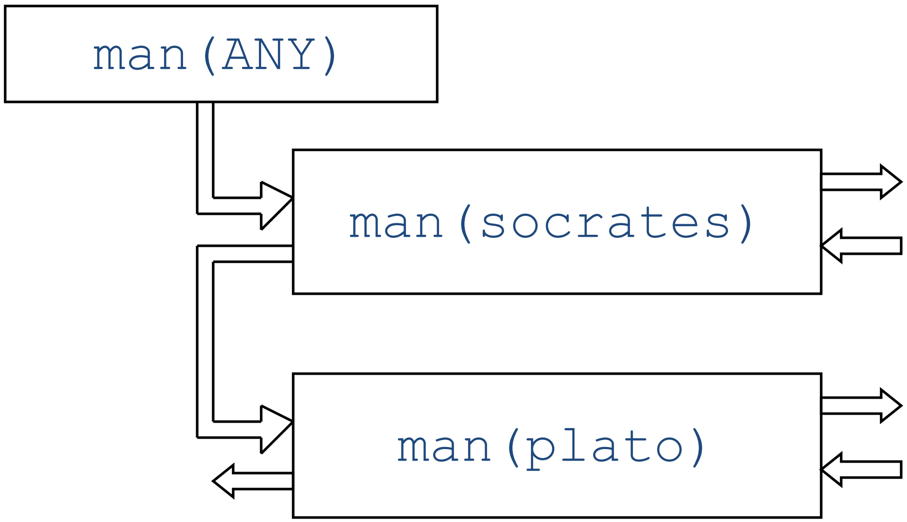
The Byrd box works like nested beads

```
philosopher(X) :- mortal(X), greek(X).
```




```
man(socrates).
man(plato).

?- man(ANY).
   ANY = socrates ;
   ANY = plato ;
   No
```



Tracing Prolog Execution

example2.pl

```
man(socrates).
man(plato).
woman(judith).
greek(socrates).

mortal(X) :- man(X).
mortal(X) :- woman(X).

philosopher(X) :- mortal(X), greek(X).
```

SICStus 4.3.1 (x86_64-win32-nt-4): Mon Dec 1 16:27:00 WEST 2014
Licensed to SP4.3simula.no

| ?- [example2].
% compiling c:/…/lab/example2.pl...
% compiled c:/…/lab/example2.pl in module user, 110 msec 576400 bytes
yes

| ?- trace, philosopher(socrates).
% The debugger will first creep -- showing everything (trace)
Call: philosopher(socrates) ?

$\{ X = socrates \}$

Tracing Prolog Execution

example2.pl

```
man(socrates).
man(plato).
woman(judith).
greek(socrates).

mortal(X) :- man(X).
mortal(X) :- woman(X).

philosopher(X) :- mortal(X), greek(X).
```

Call: mortal(socrates) ?
Call: man(socrates) ?
Exit: man(socrates) ?
Exit: mortal(socrates) ?
Call: greek(socrates) ?
Exit: greek(socrates) ?
Exit: philosopher(socrates) ?

yes

{ *success*}

{ *X = socrates*}

example2.pl

```
man(socrates).
man(plato).
woman(judith).
greek(socrates).

mortal(X) :- man(X).
mortal(X) :- woman(X).

philosopher(X) :- mortal(X), greek(X).
```

```
| ?- trace, philosopher(plato).
```

% The debugger will first creep -- showing everything (trace)

Call: philosopher(plato) ?
Call: mortal(plato) ?
Call: man(plato) ?
Exit: man(plato) ?
Exit: mortal(plato) ?
Call: greek(plato) ?
Fail: greek(plato) ?
Redo: mortal(plato) ?
Call: woman(plato) ?
Fail: woman(plato) ?
Fail: mortal(plato) ?
Fail: philosopher(plato) ?

no

Other possibility for mortal!

{ *failure*}

Does a Prolog Program Always Terminate?

```
p :- p.
```

- No, and there is no automatic procedure able to detect all the non-termination cases (Halting Problem of any program written in a Turing-complete language

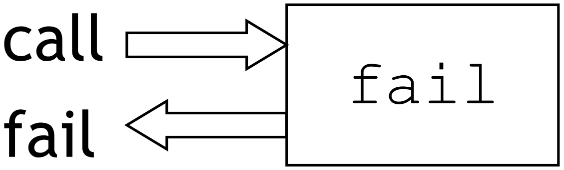
- Is it a problem in practice?

  Not really. Most of the programming languages (C, Java, C++, C#, Python, …) have the same problem

How to enforce backtracking?

- Failure occurs when there is no succesful derivation
- Enforcing backtracking can be done by:

  1. using semi-colon (`;`) at the prompt
  2. using explicitly `fail`




```
go(1) :- fail.
go(2) :- fail.
go(3).
```

using explicit `fail`

```
?- go(X).

  X = 3 ?     ;

  no
```

using semi-colon

How to resist to Backtracking?

- Backtracking undoes the links between variables:

```
?- (X=1 ; X=2 ; X=3 ; X=4 ; X=5), write(X), nl, fail.

1
2
3
4
5
no
```

However,

1. outputs cannot be undone by backtracking (e.g., `write(X)`)

2. `assert/retract` predicates allow us to change/modify/evolve the program at any time

`assert(…)` adds a fact or a clause
`retract(…)` withdraws a fact or a clause

`assert/retract`: Example

Counting the number of solutions:

```
man(socrates).
man(plato).
woman(judith).

mortal(X) :- man(X).
mortal(X) :- woman(X).

count(_) :-
        assert(sol(0)),
        mortal(ANY),
        retract(sol(N)),
        N1 is N+1,
        assert(sol(N1)),
        fail.
count(Nb) :-
        retract(sol(Nb)).
```

- `sol(…)` is said to be **dynamic**
- `assert/retract` allows the program to modify itself, to learn new clauses

→ Machine Learning!

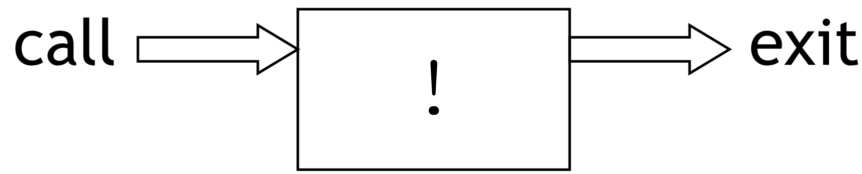
```
?- count(Nb).

Nb = 3 ?
```

Pay attention: N1 = N+1 would not work here !

How to stop backtracking?

- The **Cut** operator, written **!**, cuts branches in the Prolog search tree
- When a cut is executed, Prolog forgets all other clauses with same functor and arity than the current clause




```
max(X, Y, X) :-
    X > Y,
    !.
max(X, Y, Y).
```

Don’t explore the other clauses in case of backtracking!

Cut: Example

With cut

```
max(X, Y, X) :-
    X > Y,
    !.
max(X, Y, Y).
```

```
?- max(13, 4, Z), Z < 10.

no
```

Without cut

```
max(X, Y, X) :-
    X > Y.

max(X, Y, Y).
```

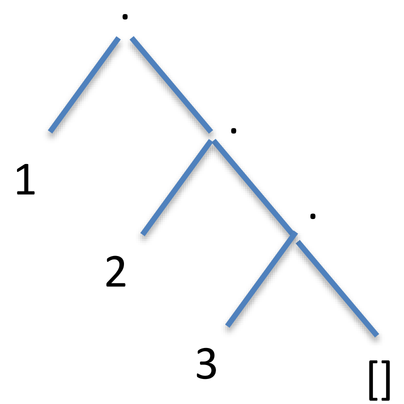
```
?- max(13, 4, Z), Z < 10.

Z = 4  {Success}
```

Meta-Predicates

- Predicates which operate over Prolog entities
- `var(X), nonvar(X)`
- `arg(N, T, X)` For ex: `arg(3,t(X,a,p(Y)),Z)` unifies `Z` to `p(Y)`
- `functor(T,F,N)` extracts the functor and the number of args of a term
  For ex: `functor(t(X,1,p(a)), F, N)`
  ```
  F = t
  N = 3
  ```
- Univ `=..` Term and subterms manipulation
  For ex: `t(X,1,p(a)) =.. [F|L]`
  ```
  F = t
  L = [X,1,p(a)]
  ```

Prolog Lists



- [1,2,3] is syntactic sugar for the term ‘.’(1, ‘.’(2, ‘.’(3, [])))
- [] denotes the void list
- [H|T] where H (resp. T) denotes the head (resp. the tail) of a list can be unified to any non-void(possibly heterogenous) list

```
[H|T]= [a,X,3,b,t(Y)]
  [a]= [H|T]
[H|T]= []
```

BTW: No a-priori limit on integer size or on list size in Prolog!

Prolog Lists: Example (1)

Write a predicate which compute the length of a list

```
lengt([],    N) :- N=0.
lengt([H|T],N) :-
                lengt(T, N1),
                N is N1+1.
```

2 clauses:
- 1 for terminal case
- 1 for recursive case

2 arguments:
- 1 for input (List)
- 1 for output (Integer)

Recursive call with a fresh variable N1

Increment by one

Pay attention to:
- **Dot** at the end of each clause
- Usage of **`is`** for arithm.

```
?- length([a,b,c], N).

?- length(L, 3).

?- length(L, N).
```

Prolog Lists: Example (2)

Write a predicate which reverse a list L **in linear time**

```
reverse([X], [X]).
reverse([X|Xs], RL):-
        reverse(Xs, RLs),
        conc(X, RLs, RL).

conc(X, [], [X]).
conc(X, [Y|S], [Y|R]):-
        conc(X, S, R).
```

This version does the job, but in quadratic time. Why?

Trick: use an accumulator

```
reverse(L, RL) :-
        reverse(L, [], RL).
reverse([], RL, RL).
reverse([X|Xs], RLs, RL):-
        reverse(Xs,[X|RLs], RL).
```

```
?- reverse([a,b,c], L).
```

Prolog: Brief Summary

- Prolog is both a **Declarative and an Operational** programming language
- It comes with two main ingredients: **Unification** and **Backtracking**

  Unification is powerful to construct complex structures

  Backtracking is convenient to explore a search tree

- **assert/retract** can be used to learn new facts/clauses

NB: We will use Prolog Notations in the rest of the course

Limitations of Prolog

(source: en.wikipedia.org/wiki/Prolog)

- Although Prolog is widely used in research and education, it had a **limited impact on the computer industry in general**. Most applications are small by industrial standards, with few exceeding 100,000 lines of code

- **Programming in the large is considered to be complicated** because not all Prolog compilers support modules

- Software developed in Prolog has been criticised for having **a high performance penalty compared to conventional programming languages**, due to Prolog's non-deterministic evaluation strategy

DECISION TREE INDUCTION

Decision Tree Induction

- A **supervised symbol-based machine learning** technique

- Originally coined by **J.R. Quinlan** in 1979, after Hunt's algorithm 1966.

- Data given under the form of attribute-value in a N-dimensional space
  Representation of acquired knowledge under the form of a **decision tree**

  Several algorithms. In chronological order: **ID3,** CART, C4.5, C5.0

- **Application domains:** classification problems, natural language processing, medical diagnosis, chess playing, risk assessment

Example: Credit Risk Assessment

- A large Bank wants **to evaluate the risk of proposing loans** to its customers and prospects
- This bank has access to historical and personal data
- The goal is also to improve the system while using it to make loans

Adopted solution: ML with hierarchical representation of the knowledge, namely, **decision tree**

Training Set from **credit history of loan applications**

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	\$0 to \$15k
2.	high	unknown	high	none	\$15 to \$35k
3.	moderate	unknown	low	none	\$15 to \$35k
4.	high	unknown	low	none	\$0 to \$15k
5.	low	unknown	low	none	over \$35k
6.	low	unknown	low	adequate	over \$35k
7.	high	bad	low	none	\$0 to \$15k
8.	moderate	bad	low	adequate	over \$35k
9.	low	good	low	none	over \$35k
10.	low	good	high	adequate	over \$35k
11.	high	good	high	none	\$0 to \$15k
12.	moderate	good	high	none	\$15 to \$35k
13.	low	good	high	none	over \$35k
14.	high	bad	high	none	\$15 to \$35k

Label Y_i

Example X_i

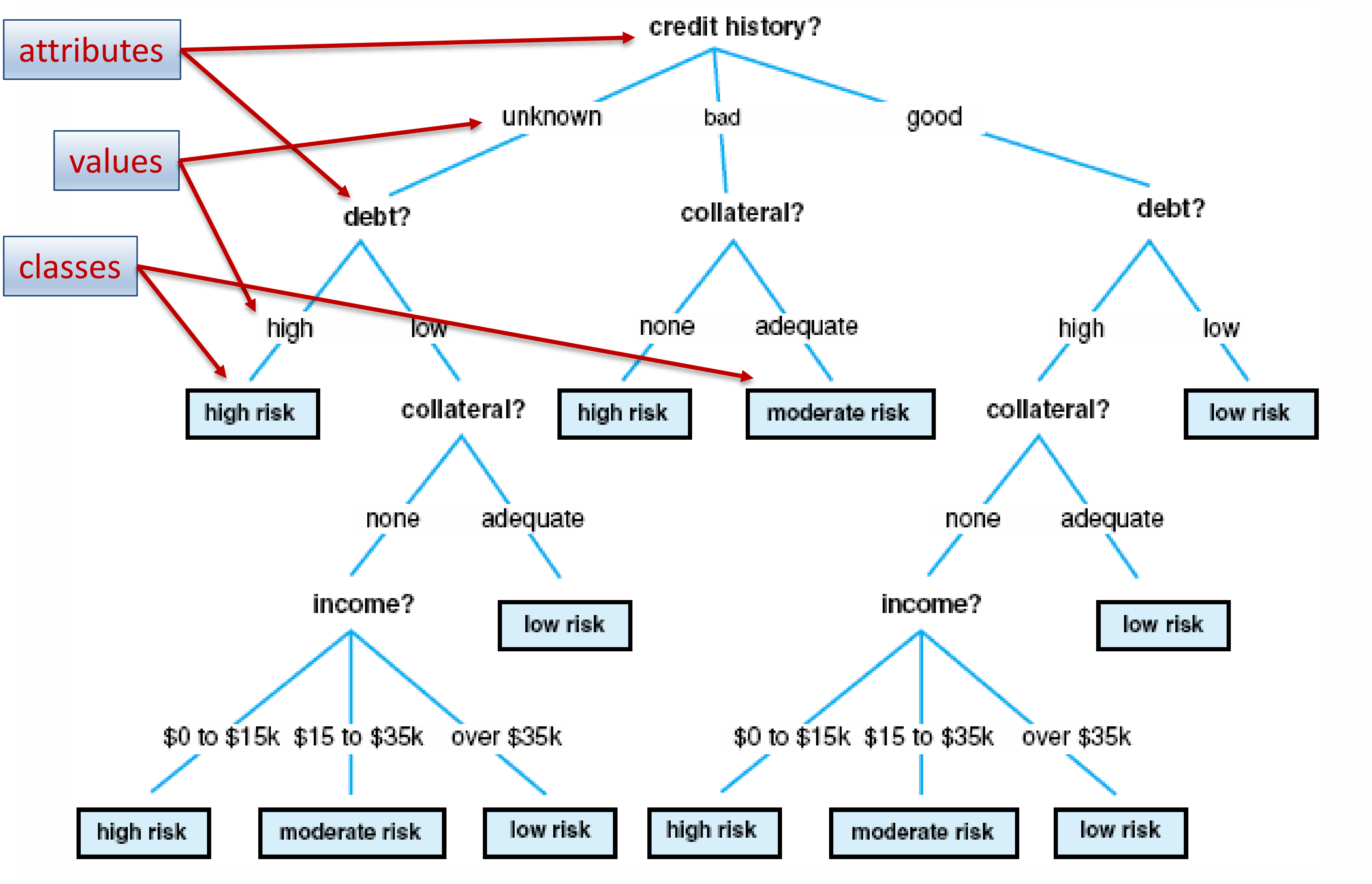
attributes

values

classes

Luger: Artificial Intelligence, 5th edition.

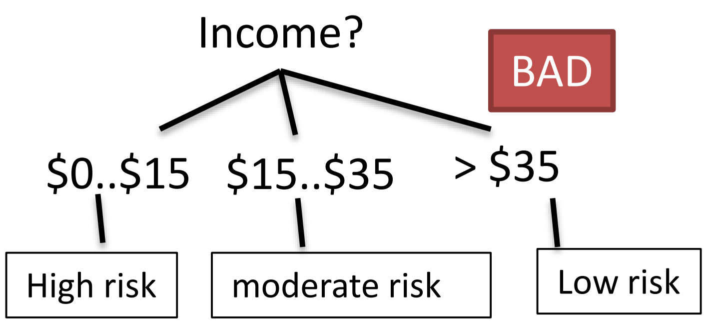
Example: A possible decision tree for credit risk assessment



Luger: Artificial Intelligence, 5th edition. © Pearson Education Limited, 2005

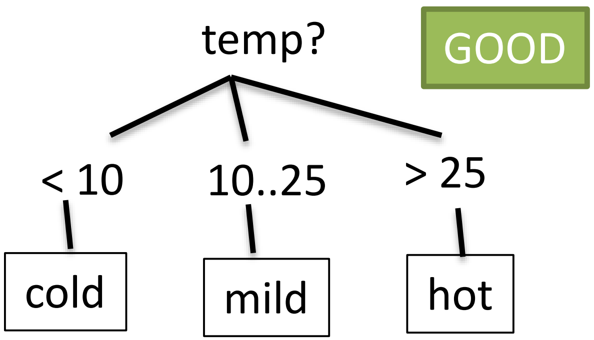
A Note on Data Discretization

Any continuous variable can be discretized in an arbitraty number of branches

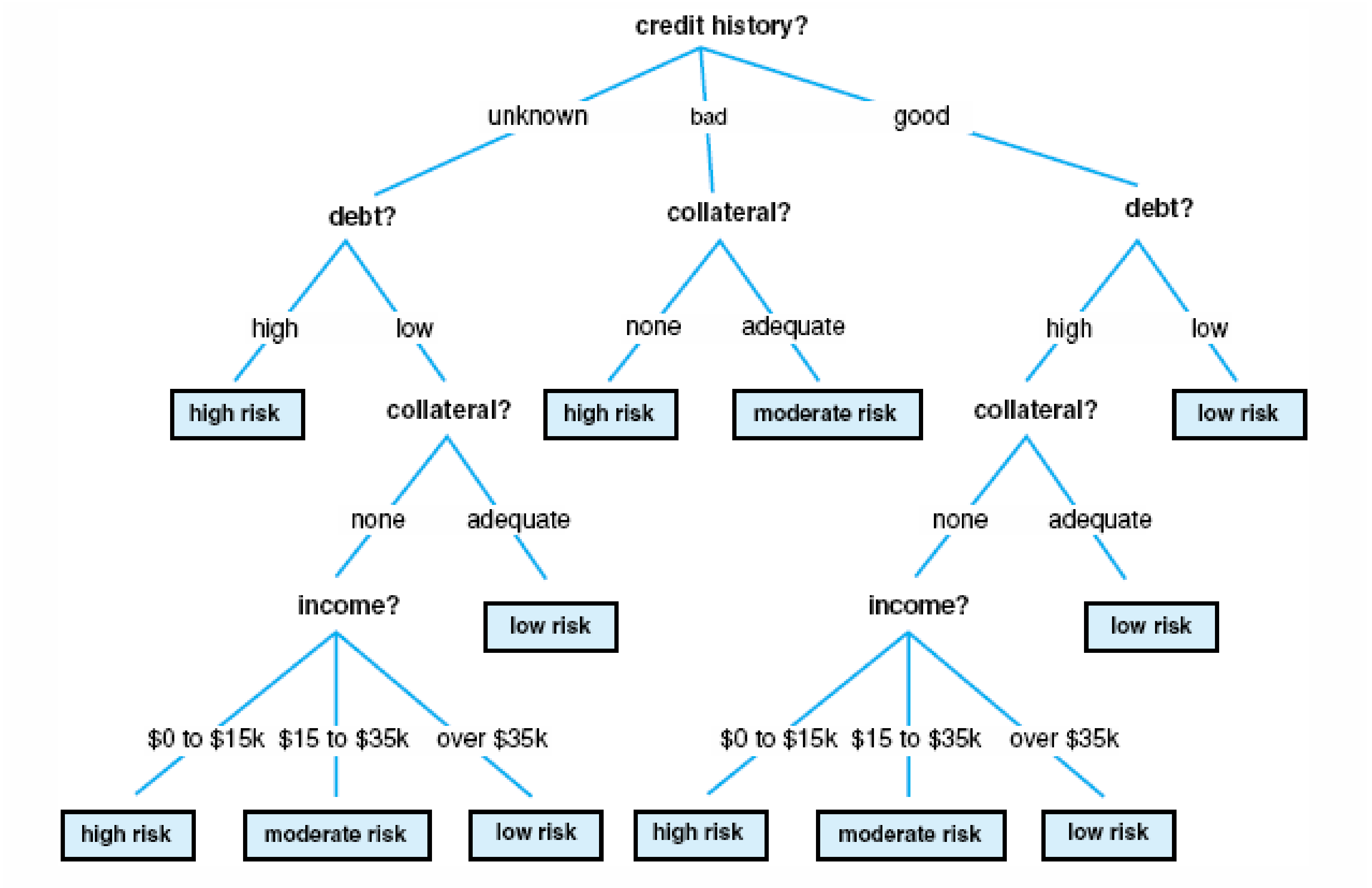


Useful to control the potential combinatorial blow-up at the branch level

But, be careful with the boundaries!

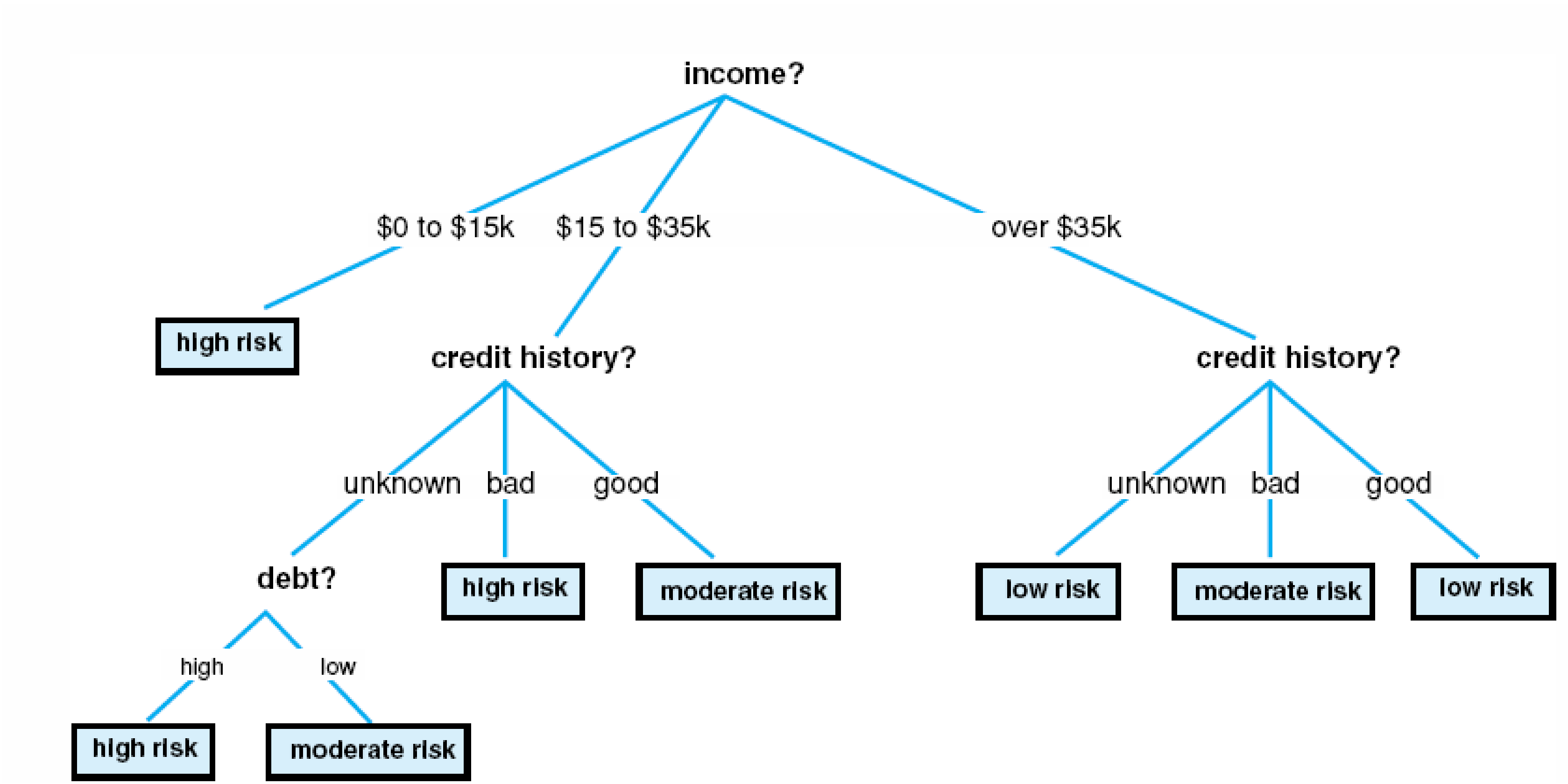


Example: A possible decision tree for credit risk assessment



If depth(T) is the number of attributes on the longest path of T, then **depth(T) = 4**

Example: another possible decision tree for credit risk assessment



If depth(T) is the number of attributes on the longest path of T, then **depth(T) = 3**

➔ **This decision tree is simpler than the the previous one!**

A Note on Attributes

- Selecting the appropriate attributes to consider is crucial!
- Decision trees are appropriate learners only if the right attributes are considered (if a crucial attribute is missing then the decision tree is useless)
- If two examples have the same attribute values, but are classified differently, then the attributes are inadequat (needs revision!)

BAD

	Credit history	Debt	Collateral	Income	Risk
Valeriya	**unknown**	**oui**	**none**	**$0..$15**	**high**
Arnaud	**unknown**	**oui**	**none**	**$0..$15**	**moderate**
.....					

Solution: Find additionalattributes which will discreminate the examples!

The ID3 Algorithm (Iterative Dichotomiser 3)

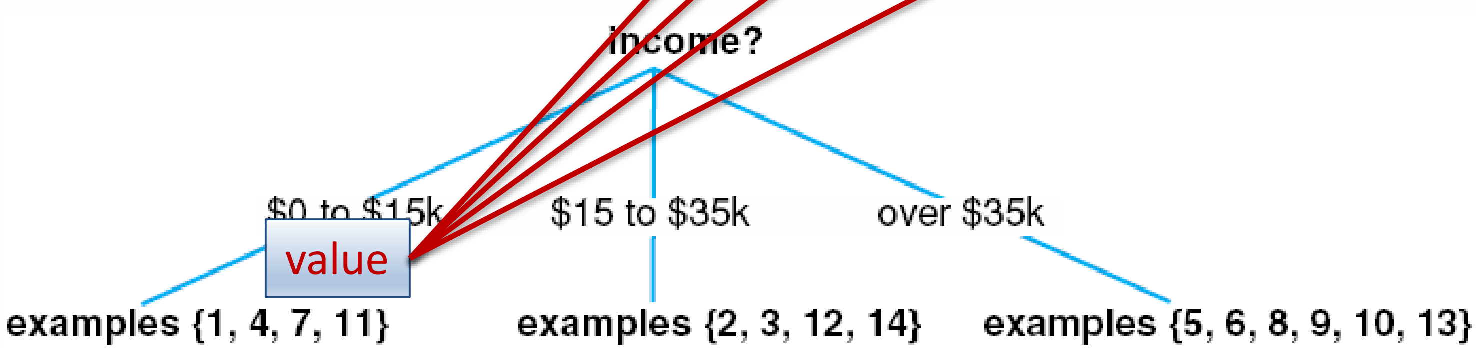
[Quinlan 86]

- A **recursive algorithm** which constructs a decision tree from a Training Set, by successively looking at each attribute
- Dependent from the order in which the attributes are considered
- Labelling all the possible decision tree to find the smallest one is untractable in general
- **Heuristics to select the next attribute**, based on so-called ID3 metrics
- Can easily be implemented in Prolog (Trees represented by Terms)

ID3 in practice (1)

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

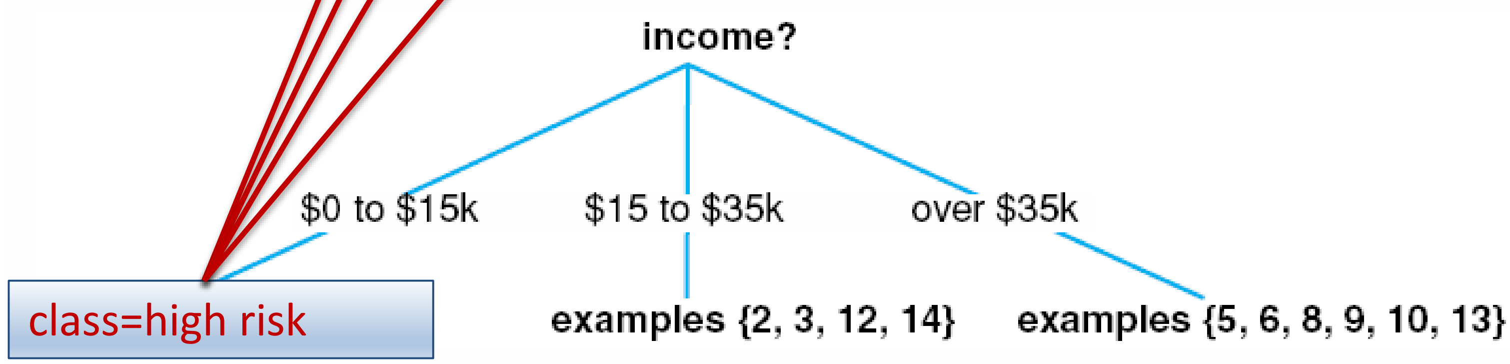
Considering the "INCOME" attribute



ID3 in practice (2)

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

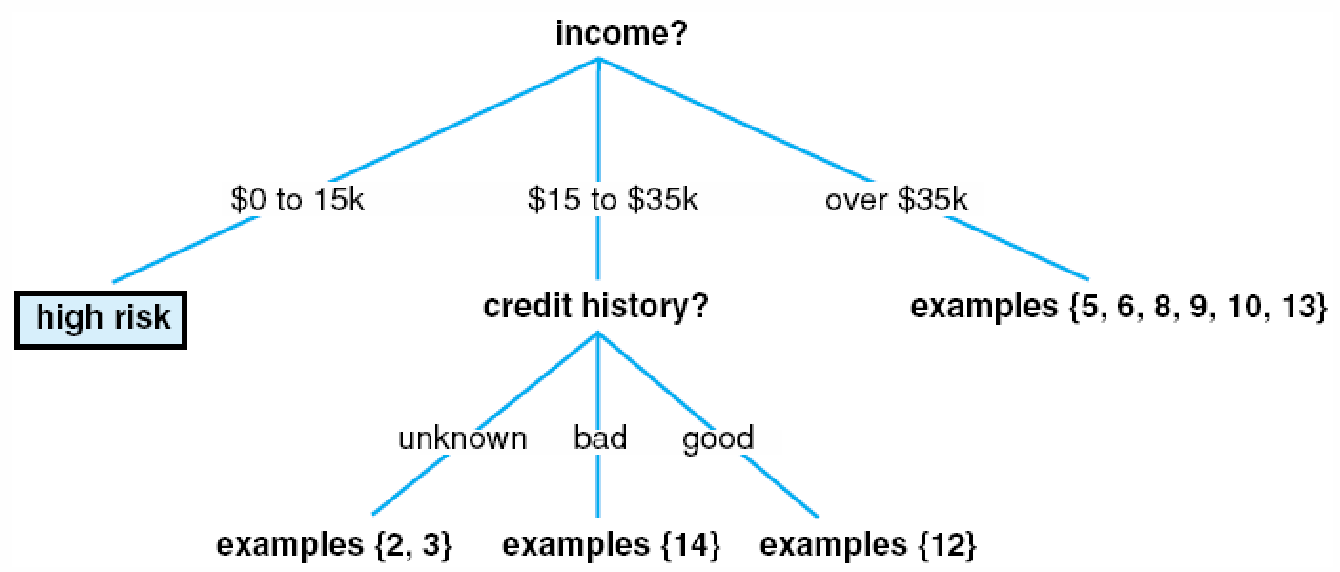
Considering the "INCOME" attribute



ID3 in practice (3)

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

Considering now the attribute "CREDIT HISTORY":



ID3 Algorithm in pseudo-code

[Quinlan 86]

ID3 (Examples, Attributes) → Decision Tree
Begin
If all examples are in the same class c
Then Return a single-node Tree labeled with c

If Attributes is empty
Then Return a Tree with leaf nodes labeled with disjunction of all classes

Heuristic to select the « best » attribute

Otherwise
Extract A from Attributes s.t. A best classifies examples
Create a Tree with Root labelled A
For each possible value v_i of A,
  Add a new branch below Root, corresponding to the test A = v_i
  Let Examples(v_i) be the subset of examples that have the value v_i for A
  Add the subtree **ID3** (Examples(v_i), Attributes – {A}) to each new branch A = v_i
Return Tree
End

ID3 Metrics

- Simple variable orderings can be used, such as, selecting the attribute which classifies (the most) examples as early as possible.

- **Entropy of the training set S with C classes *[Quinlan 86]***

$$I(S) = -\sum_{i=1}^{C} p(c_i) \cdot \log p(c_i)$$

$p(c_i)$: proba. of classe c_i

When there is a single class, I(S)=0

I(S) is maximum when classes are equiprobable (= $log_2(k)$)
Number of necessary bits to express the info (Information Gain)

- **Other metrics**: Gini Index *[Breiman et al. 84]*

$$Gini(S) = 1 - \sum_{i=1}^{C} (p(c_i))2$$

See details in textbooks, e.g., [Luger 05]

ID3 Analysis

- **Occam's razor principle** (*"entities must not be multiplied beyond necessity")* implies the selection of the simplest decision tree

  Highly sensitive to the order in which attributes are considered!

- **Single-label classification algo.** (but multi-class classification)

- Extension to deal with continuous attributes (by discretization)

- At each non-leaf node of the decision tree (each recursive step), **algorithm's time complexity is O(|S|.|A|)**
  where |S| stands for the size of the training set and,
  |A| is the number of attributes.

ID3 Limitations

- Searching among all the possible decision trees is untractable! Even searching in this hypothesis space can be computationaly expensive! (**polynomial but dependent on the size of training set**)

  All the examples are considered at each recursive step!

- **ID3 converges to locally optimal solutions, not necessarily globally optimal** (no backtracking possibility during search)

  No guarantee to reach the "simplest" decision tree,
  growing size of the decision tree

Despite these problems, one of the most popular ML algo.

Avaialble in **WEKA (Open-source ML Java Workbench from U. of Waiko, New Zealand)** - *http://www.cs.waikato.ac.nz/ml/weka*

SPECIALIZATION/GENERALIZATION

Anti-Unification

- Induction in Machine Learning implies **generalization over examples**
- But, term **unification (in Prolog)** goes from **general to specific**,

  e.g., if `T = t(X,3)` $\wedge$ `T = t(a,Y)` **then** `T = t(a,3)`

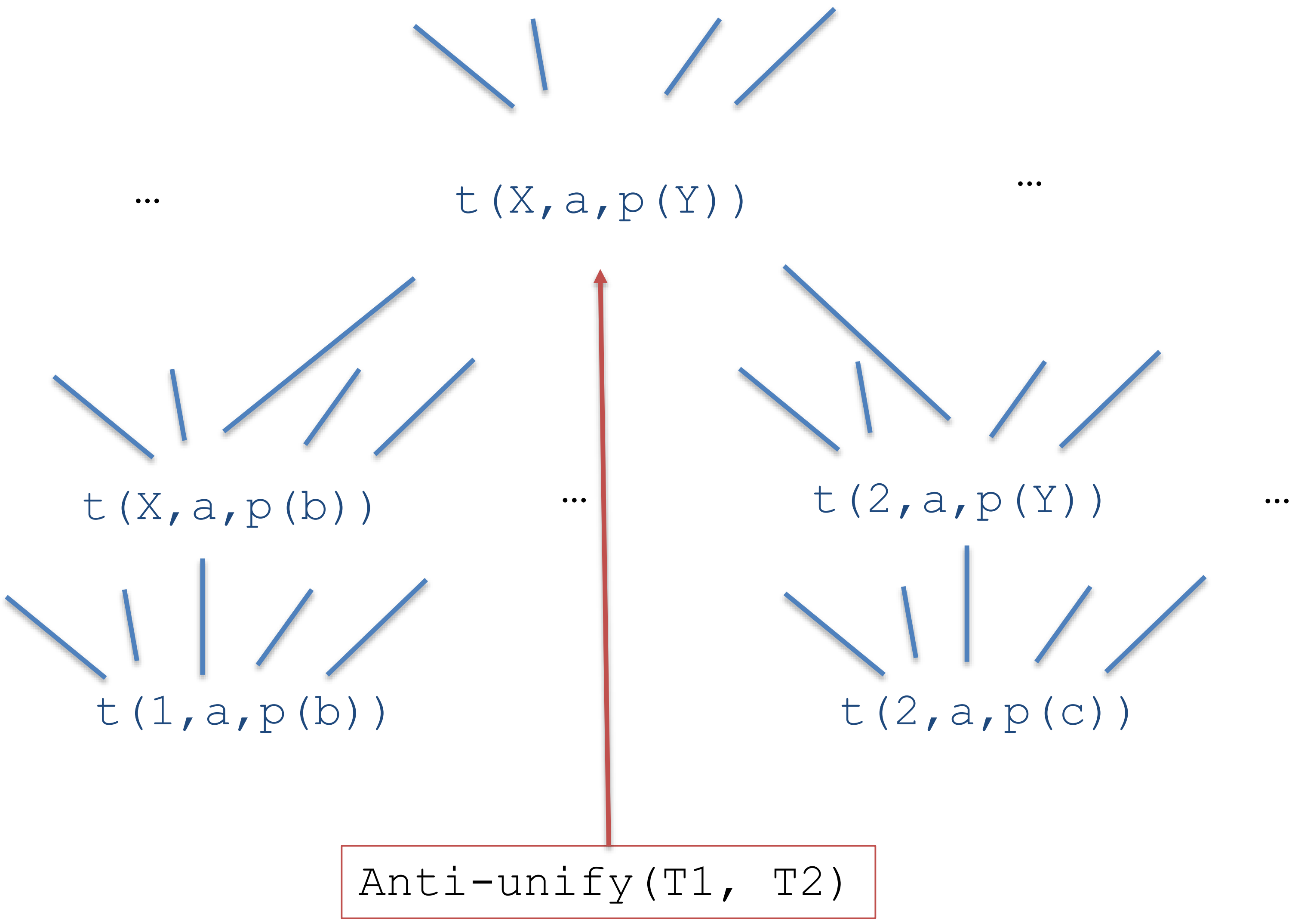
- Hopefully, Prolog also implements **anti-unification,** going from **specific to general**

  e.g., if `T1 = t(a)` $\wedge$ `T2 = t(b)` **then** `anti-unify(T1,T2) = t(_X)`

```
anti-unify(t(X), t(X)) =  t(_Z)
anti-unify(t(X), t(X, 1)) =  _Z
```
% A new fresh variable

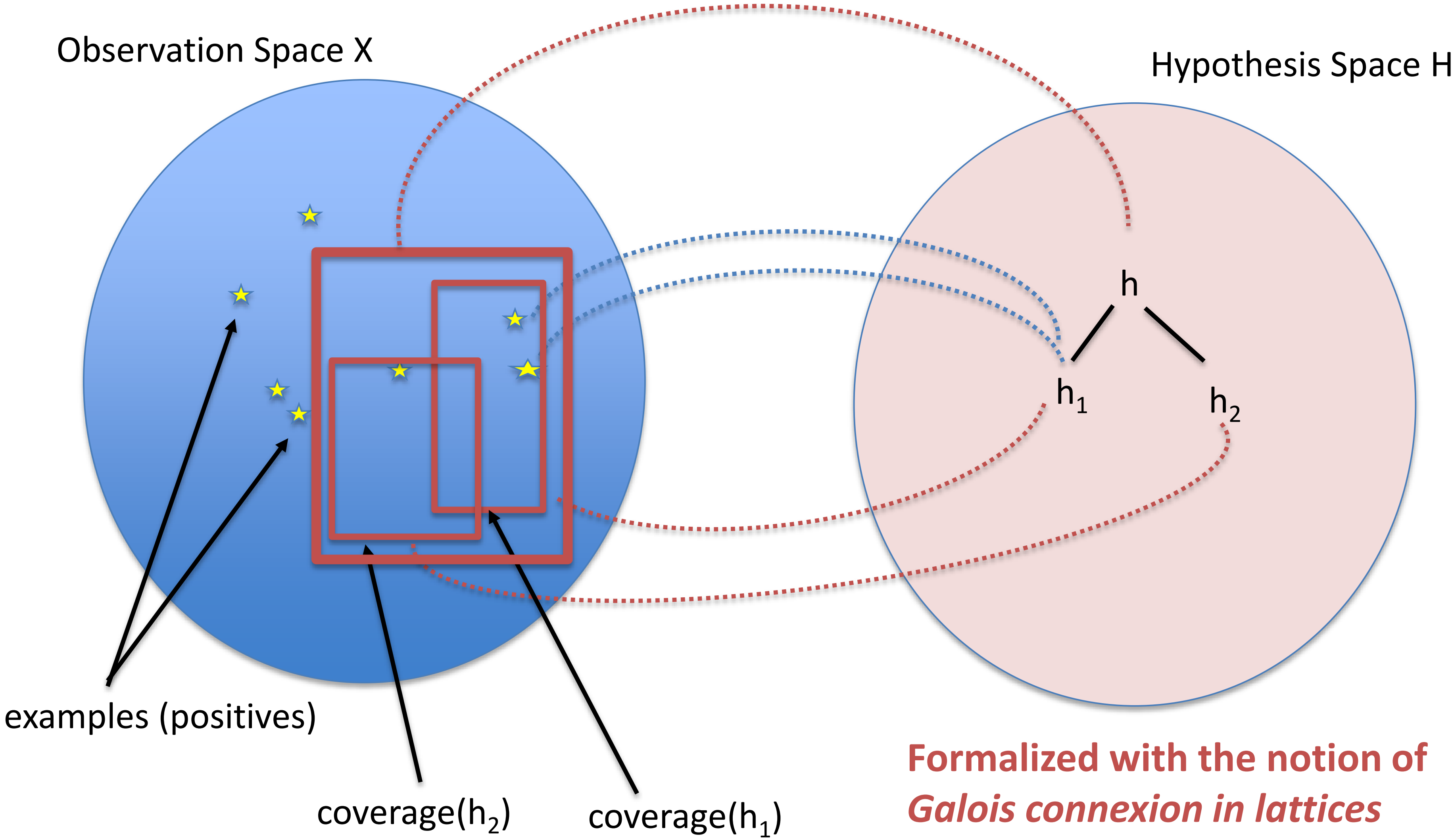
```
anti-unify(t(a,1,p(X)), t(a,2,p(Y)) =  t(a, _Z, _T)
```

Anti-unification as concept exploration



Concept Learning

The most Important property of any generalization op.



Formalized with the notion of *Galois connexion in lattices*

Examples of Generalization Operators (based on Logic)

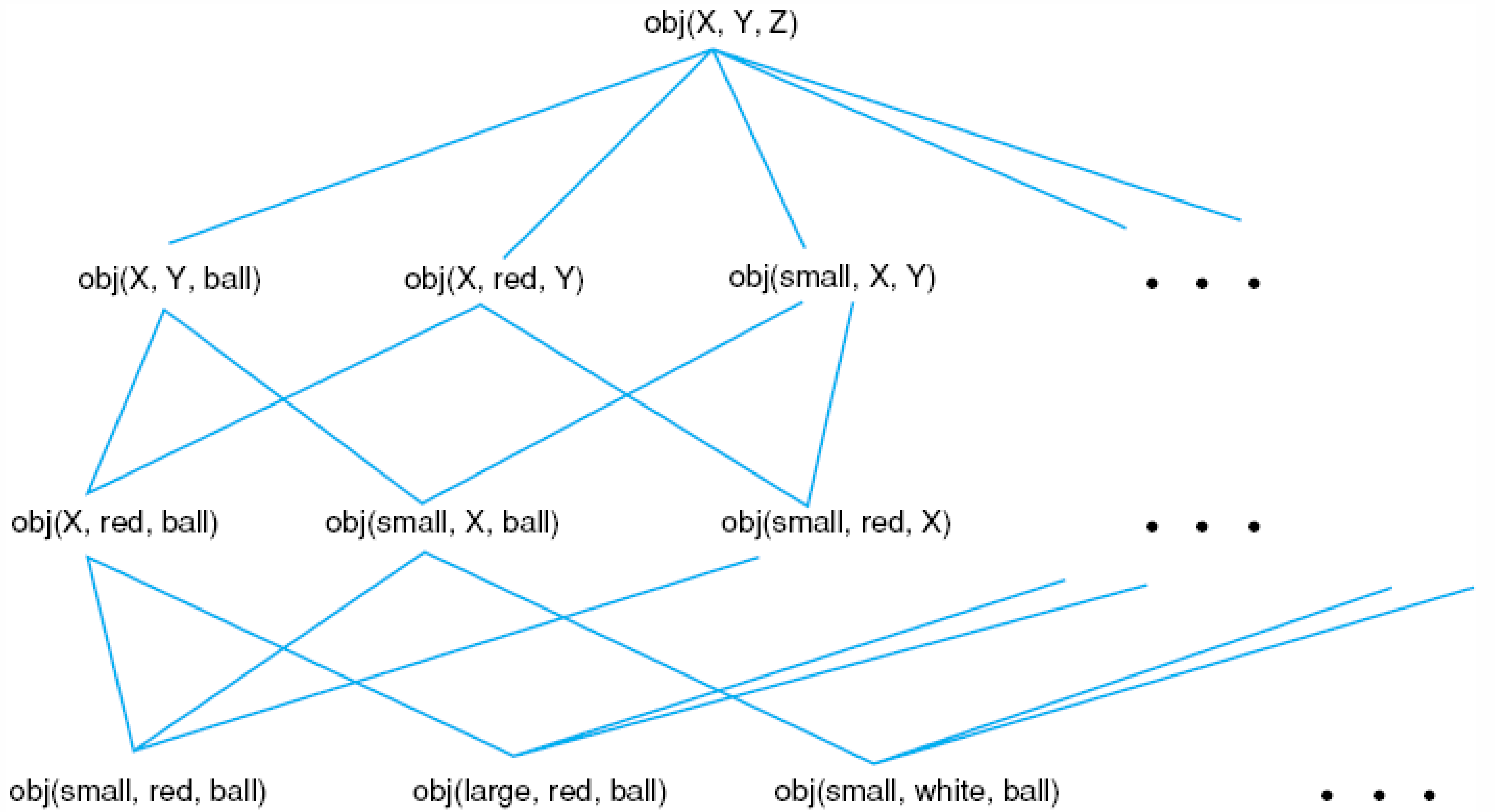
- **Replace constants with variables**
  Ex: color(ball, red) → color(ball,**X**)

- **Remove literals from conjunctions**
  Ex: shape(X, round) ∧ **size(X, small)** ∧ color(X, red)
  → shape(X, round) ∧ color(X, red)

- **Add disjunctions**
  Ex: shape(X, round) ∧ size(X, small) ∧ color(X, red)
  → shape(X, round) ∧ size(X, small) ∧ (**color(X, red)** ∨ **color(X, blue))**

- **Replace an class with the superclass in relations**
  Ex: is-a(tom, cat) → is-a(tom, **animal**)

A concept space example: obj(Size, Color, Type)



VERSION SPACE SEARCH

Version Space Search (VSS)

- A **supervised symbol-based machine learning** technique

- Proposed by **T. Mitchell** in 1978

- Data given as **positive and negative examples of a concept** (binary classification) **-** Representation of acquired knowledge under the form of a **Lattice (**Poset with unique least upper bound and greatest lower bound)

  Related approaches: Inductive Logic Programming, Constraint Acquisition

- **Application domains:** symbolic integration, planning, concept and constraint model acquisition, intelligent robotics

Version Space Search (VSS): Basic Idea

- Given **a representation language** and a set of **positive and negative examples**, find the **most general/specific concept description** consistent with all examples
- The concept description has to **match all positive examples**, it has to **reject all negative examples**
- VSS uses generalization/specialization operators (as seen before) to impose an ordering on the concepts → lattice representation
- 3 combined processes to reduce the version space based on learning examples
  - From specific to general
  - From general to specific
  - From both side: the **candidate elimination algorithm**

Hypotheses

VSS maintains two sets of hypotheses:

G – the **most general hypotheses** that match the training data

S – the **most specific hypotheses** that match the training data

Each hypothesis is represented as a **term** of the known attributes

e.g. **obj(small, X, Y)** in the lattice **obj(SIZE, COLOR, TYPE)** represents any small objects, whatever be the color or the type

Example of concept learning in version space

Consider the learning task to obtain a description of the concept "**Japanese Economic Car"** from examples (positive and negative)

The **attributes** under consideration are:

Origin, Manufacturer, Color, Decade, Type

Training set:

positive ex: `(japan, honda,  blue, 1980, economic)`

positive ex: `(japan, honda,  white, 1980, economic)`

negative ex: `(japan, toyota, green, 1970, sport)`

Courtesy to Prof. **Lydia Sinapova** for the example
[Ref: E. Rich, K. Knight, "Artificial Intelligence", McGraw Hill, Second Edition]

Most General/Specific Hypothesis

positive ex: `(japan, honda,  blue, 1980, economic)`

positive ex: `(japan, honda,  white, 1980, economic)`

negative ex: `(japan, toyota, green, 1970, sport)`

The **most general hypothesis** covers all the positive examples and none of the negative examples AND is **the least instantiated**:

(_, honda, _, _, _) ∨ (_, _, _, 1980, _) ∨ (_, _, _, _, economic)

where the symbol '_' means that the attribute may take any value (as in Prolog)

The **most specific hypothesis** covers all the positive examples and none of the negative example AND is **the most instantiated**:

(japan, honda, _, 1980, economic)

The Candidate elimination Algo. In VSS

4 parts

1. Initialization
2. Process a positive example
3. Process a negative example
4. Termination conditions

1. Initialization

- Initialize G with the most general concept description (top of the lattice)
- Initialize S to empty
- Process a new training example (positive or negative)

The Candidate elimination Algo. In VSS

4 parts

1. Initialization
2. Process a positive example
3. Process a negative example
4. Termination conditions

2. Process a positive example

- **Remove from G** any description that does not cover the positive example
- **Generalize S** as little as possible so that the new training example is covered
- **Remove from S** all elements that cover negative examples.

The Candidate elimination Algo. In VSS

4 parts

1. Initialization
2. Process a positive example
3. Process a negative example
4. Termination conditions

3. Process a negative example

- **Remove from S** any descriptions that cover the negative example
- **Specialize G** as little as possible so that the negative example is not covered
- **Remove from G** all elements that do not cover the positive examples

The Candidate elimination Algo. In VSS

4 parts

1. Initialization
2. Process a positive example
3. Process a negative example
4. Termination conditions

4. Termination conditions

Continue processing new training examples, until one of the following occurs:

- Either **S** or **G** become empty, there are no consistent hypotheses in the space {*Failure*}
- **S** and **G** are both singleton sets.
  - **if** they are **identical**, output their value and stop {*Success*}
  - **if** they are **different**, the training examples are inconsistent {*Failure*}

Example: Learning the concept of "*Japanese economic car*"

- Attributes: **(Origin, Manufacturer, Color, Decade, Type)**
- Positive ex: **`(japan, honda, blue, 1980, economic)`**
- **Initialize G** to the most general concept description
- **Initialize S** to a singleton that includes the first positive example

G = { (_, _, _, _, _) }
S = { (japan, honda, blue, 1980, economic)}

Example: Learning the concept of "*Japanese economic car*"

- Negative ex: **`(japan, toyota, green, 1970, sport)`**

- **Specialize G** to exclude the negative example

G = {(_, honda, _, _, _), (_, _, blue, _, _) , (_, _, _, 1980, _), (_, _, _, _, economic)}
S = {(japan, honda, blue, 1980, economic)}

To be read as a disjunction of concept descr.

Example: Learning the concept of "*Japanese economic car*"

- Positive ex: `(japan, toyota, blue, 1990, economic)`
- **Remove from G** descriptions inconsistent with positive example
- **Generalize S** to include the positive example

  G = { (_, _, blue, _, _), (_, _, _, _, economic)}

  S = {(japan, _, blue, _, economic)}

Remove disjunct operator

Anti-unification on positive ex.

Example: Learning the concept of "*Japanese economic car*"

- Negative ex: **`(usa, chrysler, red, 1980, economic)`**
- **Specialize G** to exclude the negative example (but staying within version space, i.e., **staying consistent with S**)

  G = {(_, _, blue, _, _), (japan, _, _, _, economic)}

  S = {(japan, _, blue, _, economic)}

Is everything economic a « Japanese economic car »? : No

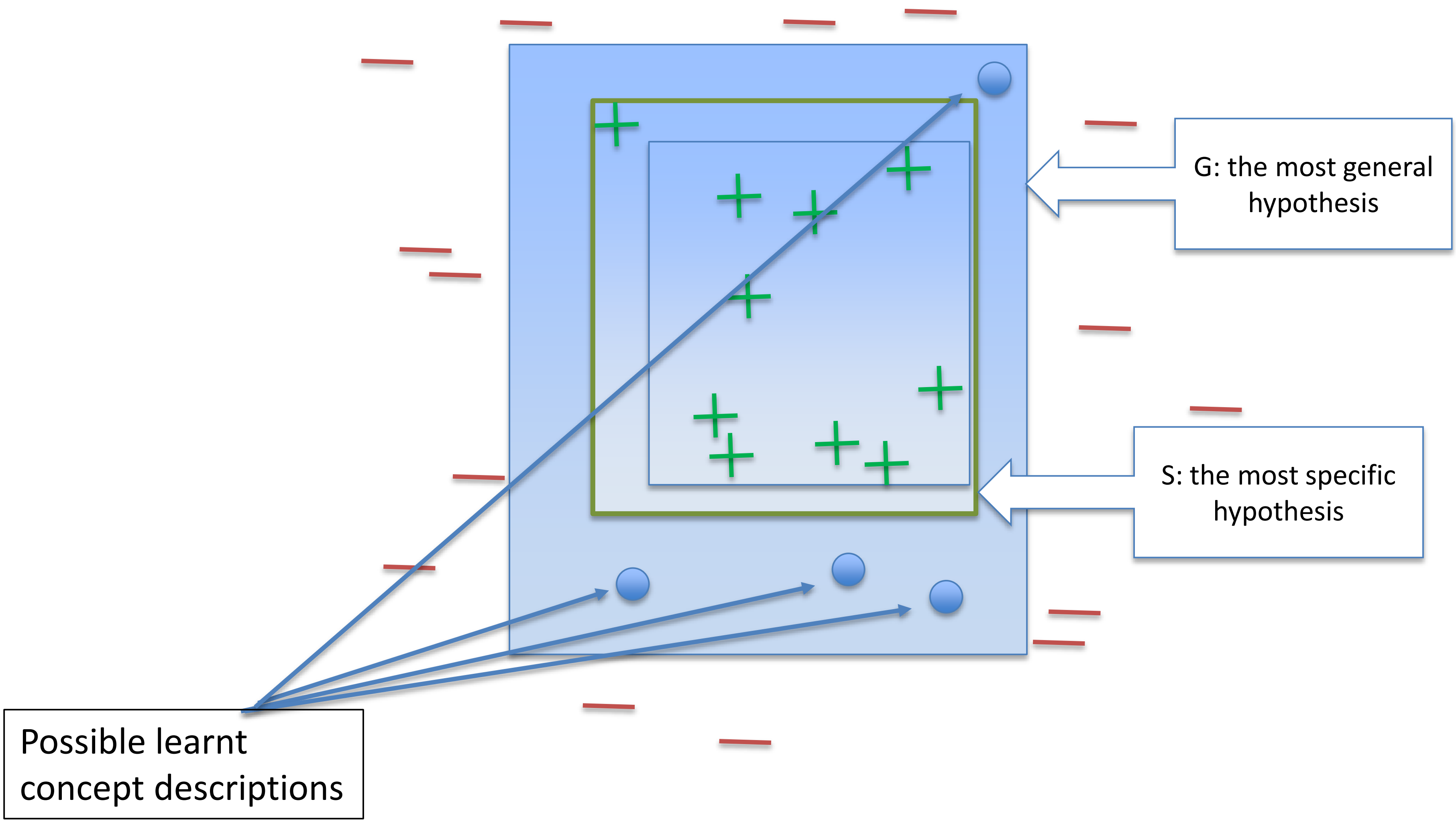
Example: Learning the concept of "*Japanese economic car*"

- Positive ex.: **`(japan, honda, white, 1980, economic)`**
- **Remove from G** descriptions inconsistent with positive example
- **Generalize S** to include the positive example

  G = {(japan, _, _, _, economic)}
  S = {(japan, _, _, _, economic)}

- S = G, then {Success} !

Visualization of G and S in VSS



VSS: Properties and Limitations

- Powerful **concept learning** technique – Structured knowledge (lattice) Interesting for Active Learning (**can deal with incomplete examples**)
- A major drawback of version space learning is its **inability to deal with noise**:
- Can collapse in case of inconsistency, but also propose many ways to backtrack (through non-deterministic search)

CONCLUSIONS

Conclusions

- **Symbol-based Machine Learning** is an important AI topic, with more than 4 decades of active research works
- **Programming in Logic** offers the programmer an ideal playground for implementing Symbol-based ML techniques
- **Decision Tree Induction** (multiclass supervised learning)
  Version Space Search (binary supervised learning)
- **Many other approaches in Symbolic ML,** including unsupervised learning, explanation-based learning, reinforcement learning, etc.
- Pay attention to not fall into the next AI Winter which will happen
  (due to too-high expectations in ML)
- Hope you will participate and enjoy the Lab. Session in Prolog!

Acknowledgements

- **Valeriya Naumova** for her immense expertise in ML, patience and understanding For having given to me the oppportunity to talk of this topic

- **Carlo Ieva** for his help in the Lab. session

- **Certus Board** and **Certus Partners** support of this event

Centre for Research-based Innovation