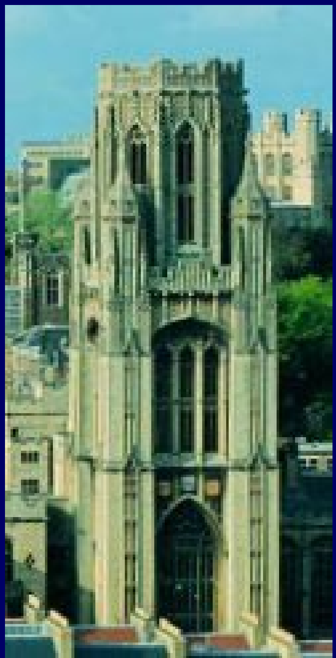
A tutorial on rule induction



Peter A. Flach

Department of Computer Science

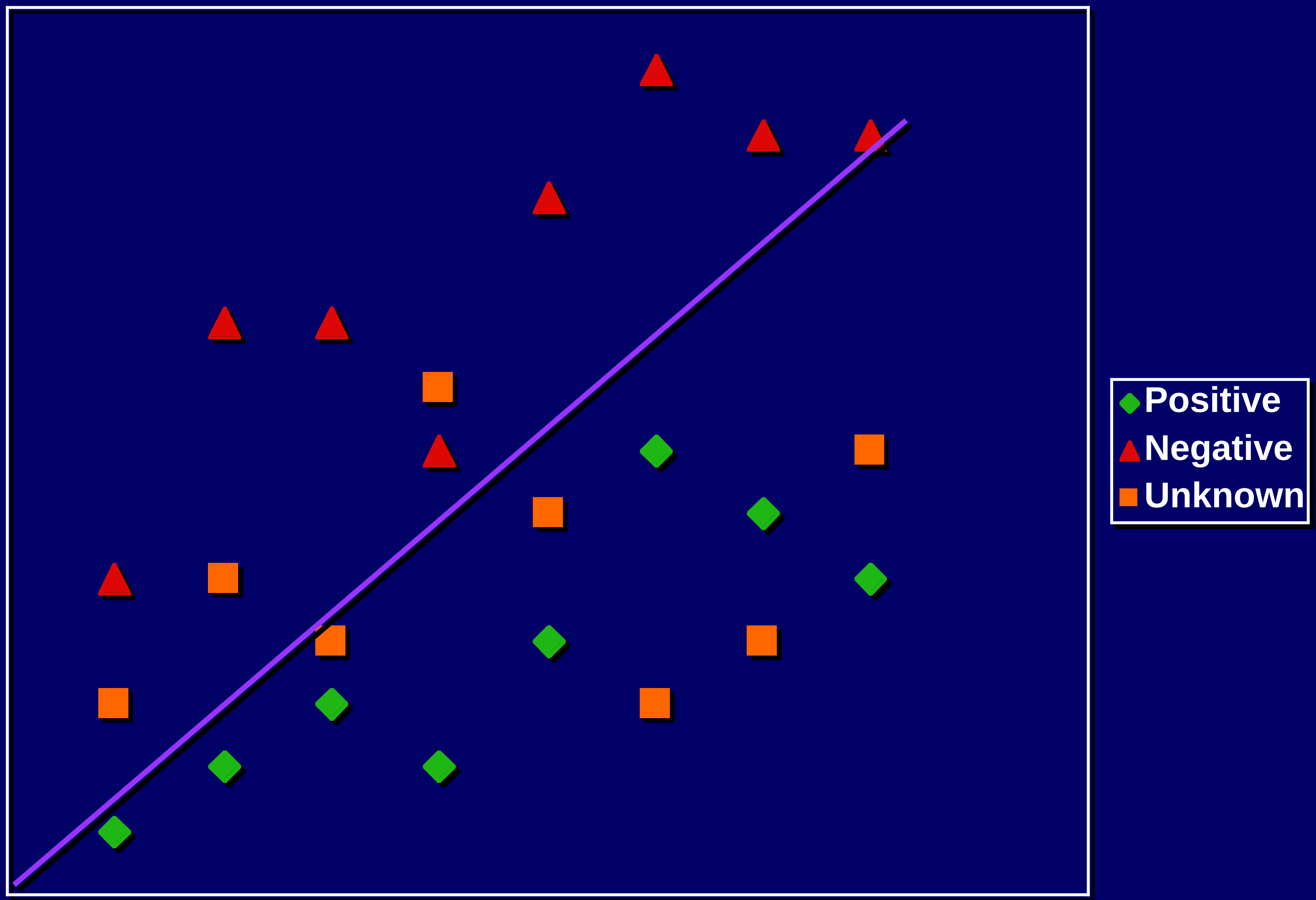
University of Bristol

www.cs.bris.ac.uk/~flach/

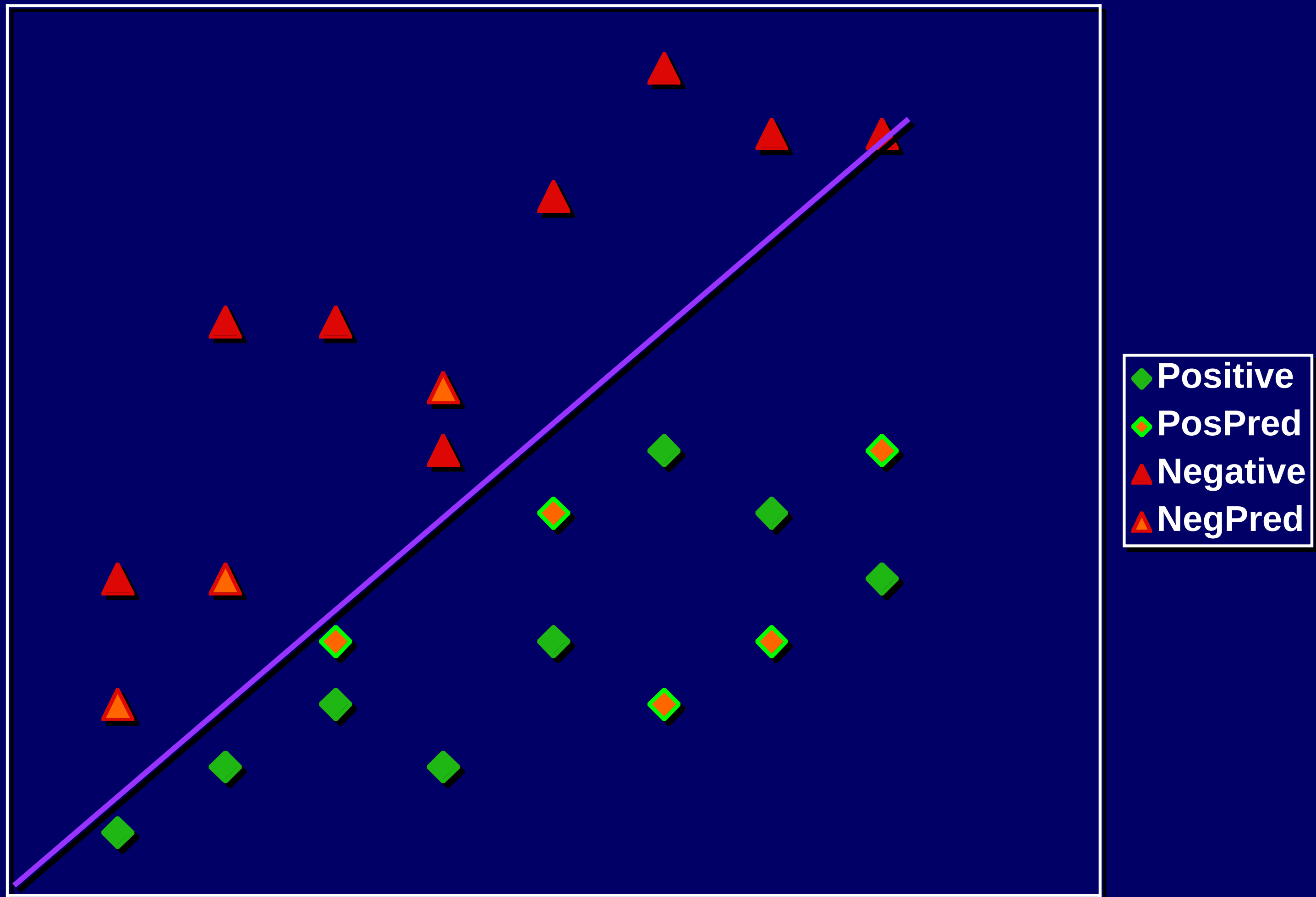
Overview

- **Introduction**
- Learning rules with CN2
- Learning Prolog rules with ILP
- Rule learning with other declarative languages

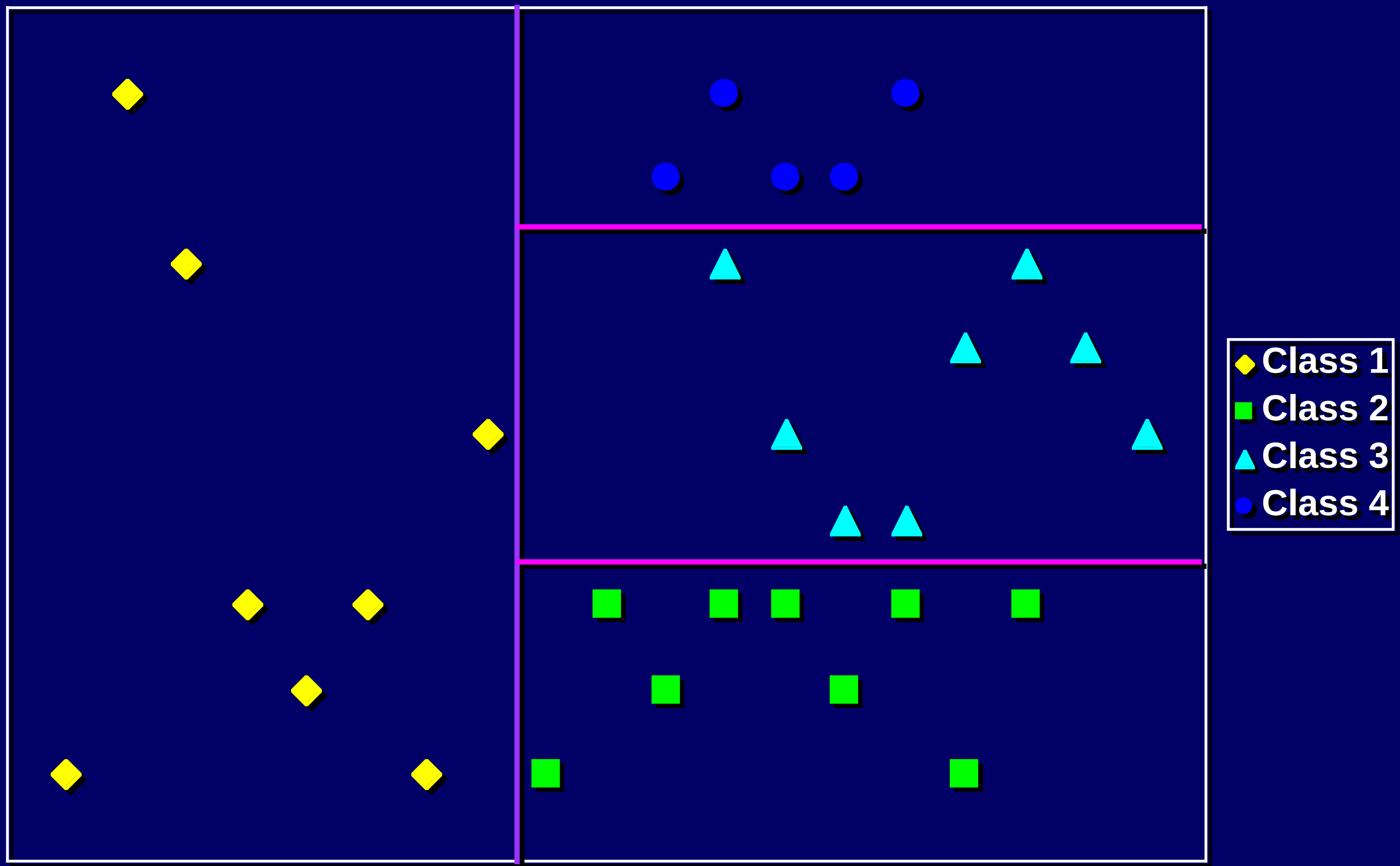
Example 1: linear classification



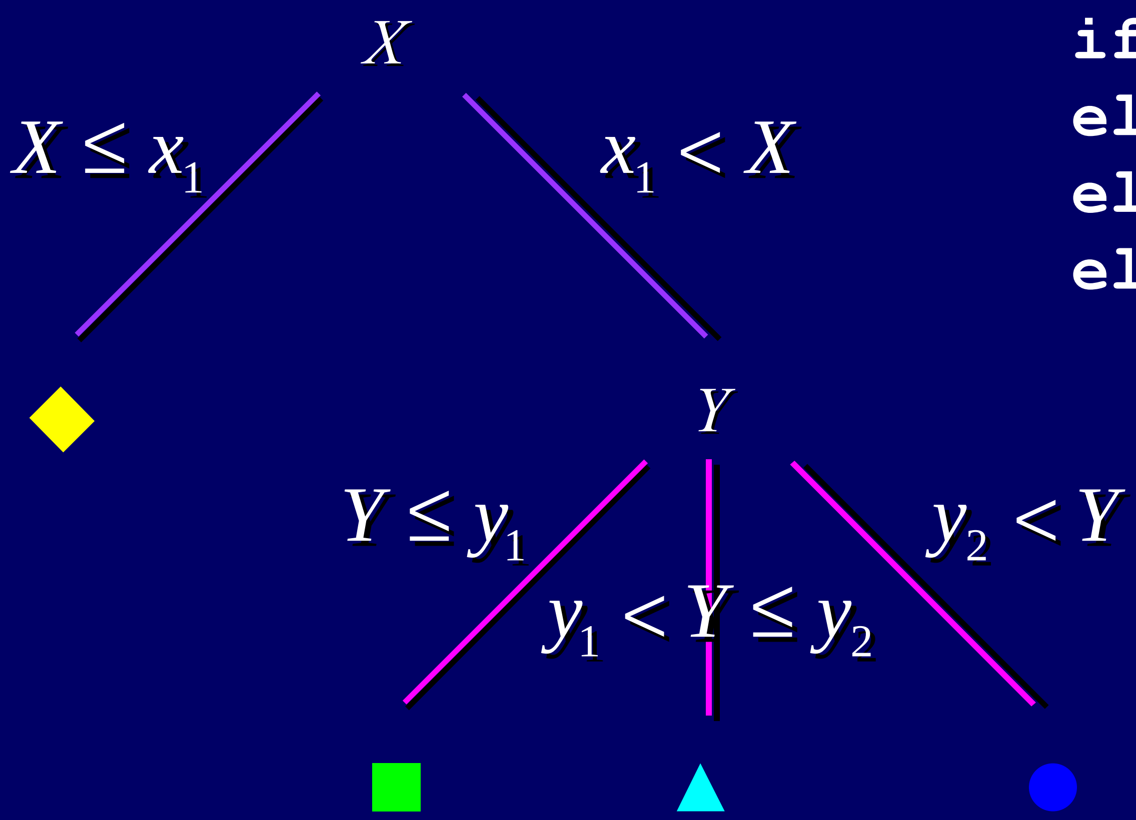
Example 1: linear classification



Example 2: decision tree

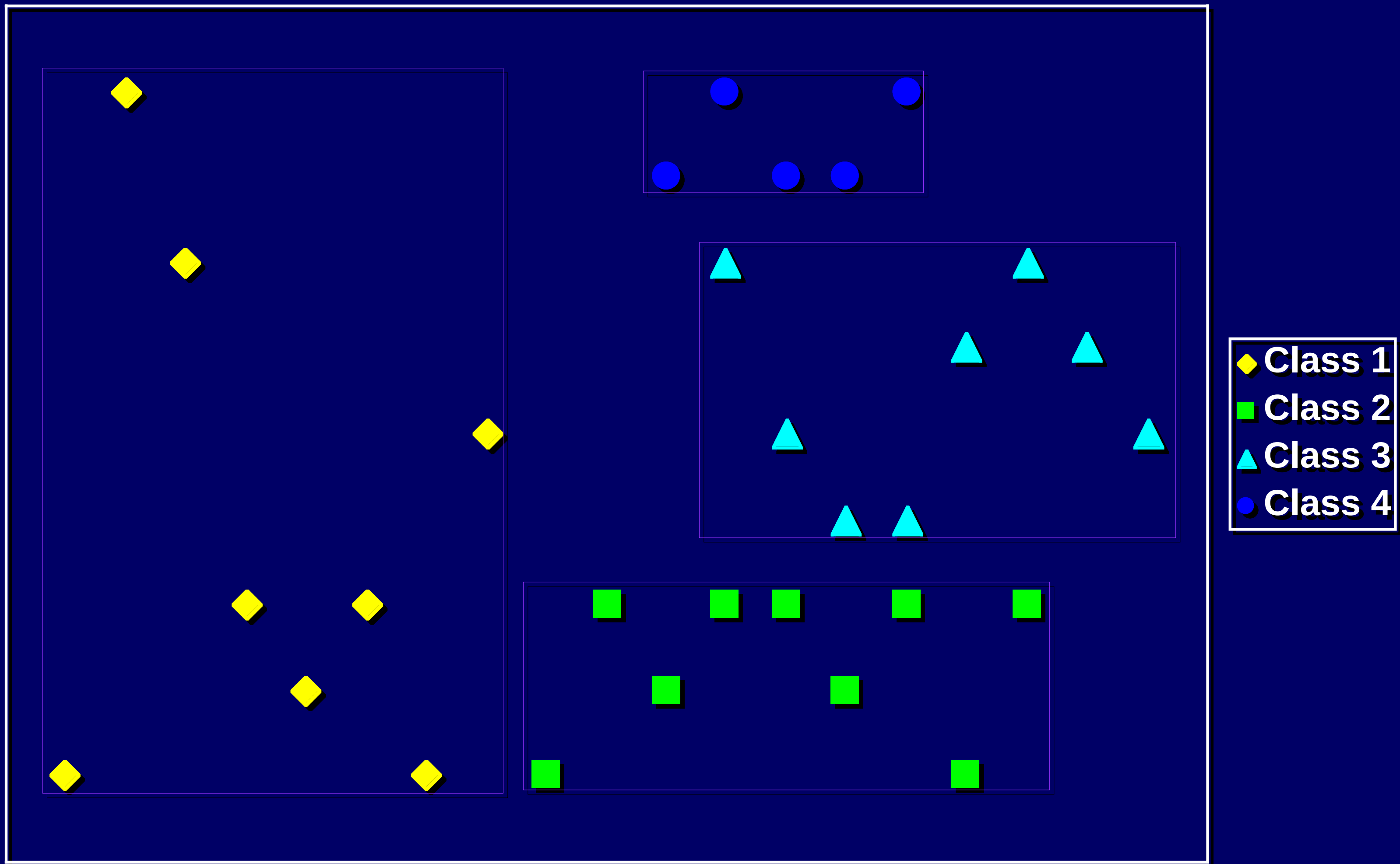


Example 2: decision tree

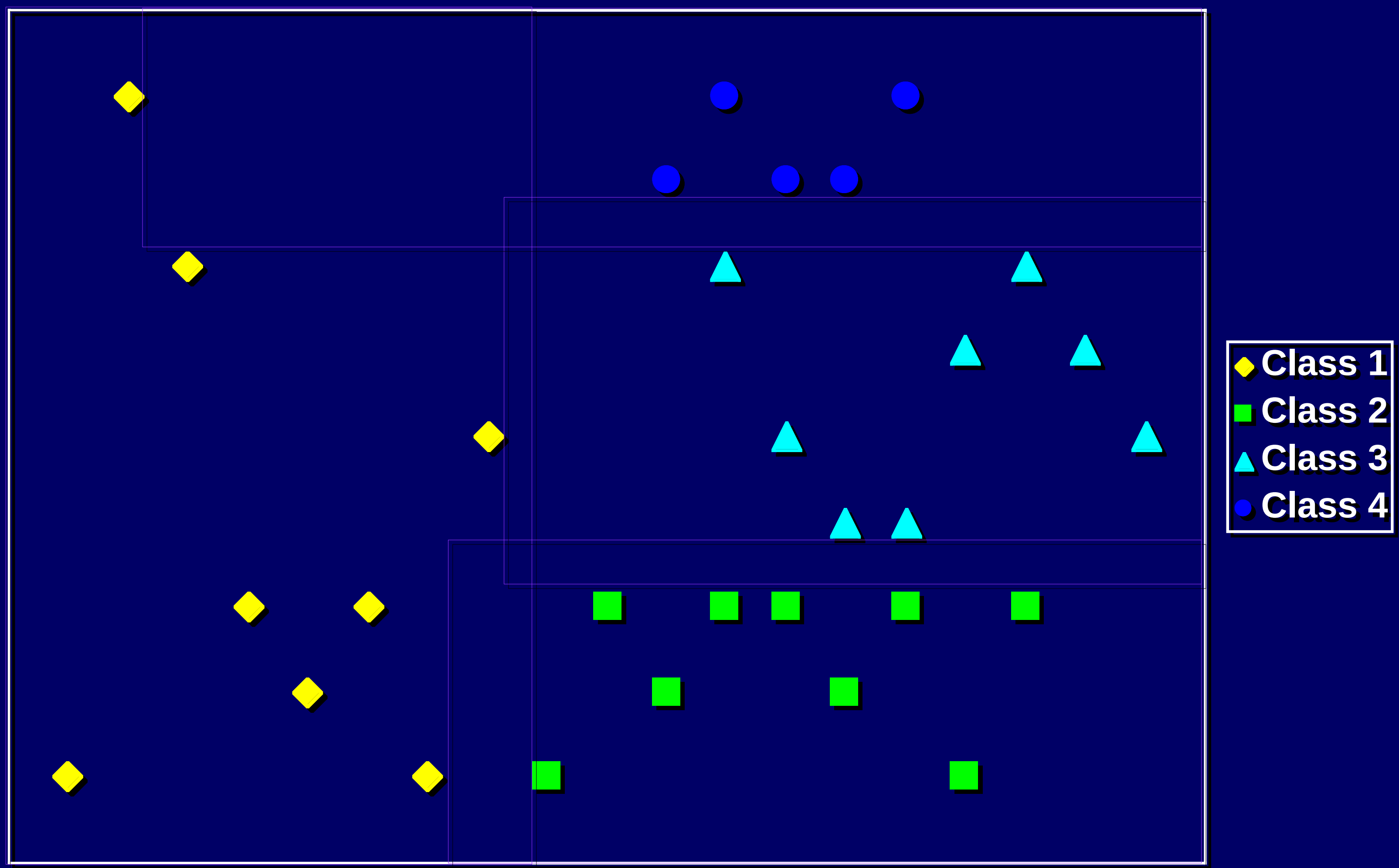


if X≤x1 **then** ◆
else if Y≤y1 **then** ■
else if Y≤y2 **then** ▲
else ●

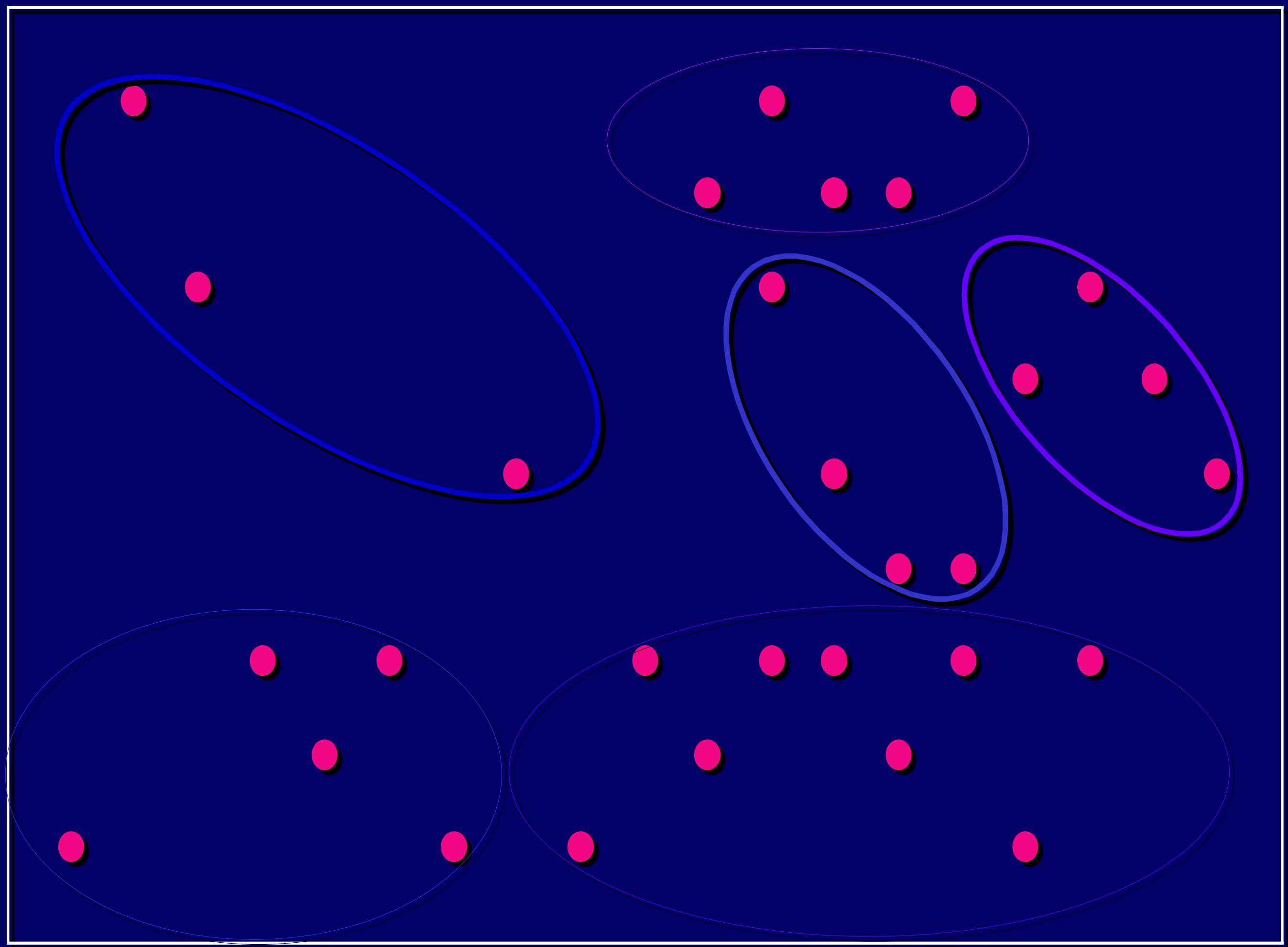
Example 3: rules



Example 3: rules

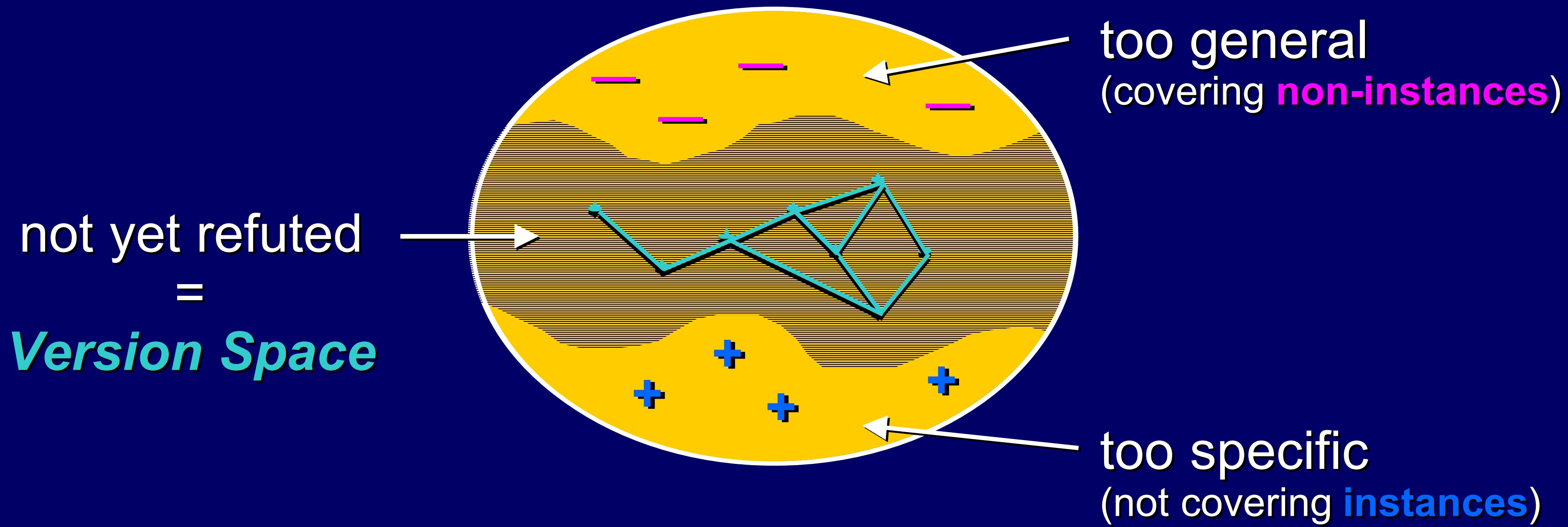


Example 4: clusters



Inductive concept learning

- **Given**: descriptions of **instances** and **non-instances**
- **Find**: a **concept covering all instances** and **no non-instances**



Coverage and subsumption

- (Semi-)propositional languages such as ***attribute-value languages*** cannot distinguish between instances and concepts.
- Consequently, testing **coverage** of an instance by a concept becomes equivalent to testing **subsumption** of one concept by another.
  - `(size=medium or large) and (colour=red)`

    covers / subsumes
  - `(size=large) and (colour=red) and (shape=square)`

Generalisation and specialisation

- *Generalising* a concept involves enlarging its extension in order to cover a given instance or subsume another concept.
- *Specialising* a concept involves restricting its extension in order to avoid covering a given instance or subsuming another concept.
- **LGG** = Least General Generalisation
- **MGS** = Most General Specialisation

Overview

- Introduction
- **Learning rules with CN2**
- Learning Prolog rules with ILP
- Rule learning with other declarative languages

The CN2 algorithm

- Combine AQ (Michalski) with decision tree learning (search as for AQ, criteria as for decision trees)
  - AQ depends on a seed example
  - AQ has difficulties with noise handling
- CN2 learns unordered or ordered rule sets of the form: {R1, R2, R3, …, D}
  - covering approach (but stopping criteria relaxed)
  - unordered rules: rule `Class IF Conditions` is learned by first determining `Class` and then `Conditions`
  - ordered rules: rule `Class IF Conditions` is learned by first determining `Conditions` and then `Class`

CN2 rule set representation

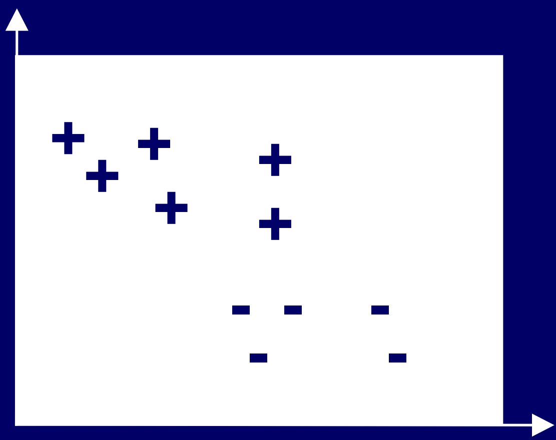
- Form of CN2 rules:
  IF Conditions THEN MajClass [ClassDistr]

- Sample CN2 rule for an 8-class problem 'early diagnosis of rheumatic diseases':

```
IF Sex = male AND Age > 46 AND
      Number_of_painful_joints > 3 AND
      Skin_manifestations = psoriasis
THEN Diagnosis = Crystal_induced_synovitis
          [0 1 0 1 0 12 0 0 ]
```

- CN2 rule base: **{R1, R2, R3, …, DefaultRule}**

Original AQ covering algorithm



- **for** each class C_i **do**
  - $E_i := P_i \cup N_i$ (P_i positive, N_i negative)
  - RuleSet(C_i) := empty
  - **repeat** {find-set-of-rules}
    - find-one-rule R covering some positive examples and no negatives
    - add R to RuleSet(C_i)
    - delete from P_i all positive examples covered by R
  - **until** P_i = empty

Learning unordered set of rules

- **for** each class C_i **do**
  - E_i := P_i U N_i, RuleSet(C_i) := empty
  - **repeat** {find-set-of-rules}
    - R := `Class = C`$_i$ `IF Conditions`, `Conditions` := true
    - **repeat** {learn-one-rule}
      R' := `Class = C`$_i$ `IF Conditions AND Cond`
      (general-to-specific beam search of Best R')
    - **until** stopping criterion is satisfied
      (no negatives covered or Performance(R') < ThresholdR)
    - add R' to RuleSet(C_i)
    - delete from P_i all positive examples covered by R'
  - **until** stopping criterion is satisfied (all positives covered or Performance(RuleSet(C_i)) < ThresholdRS)

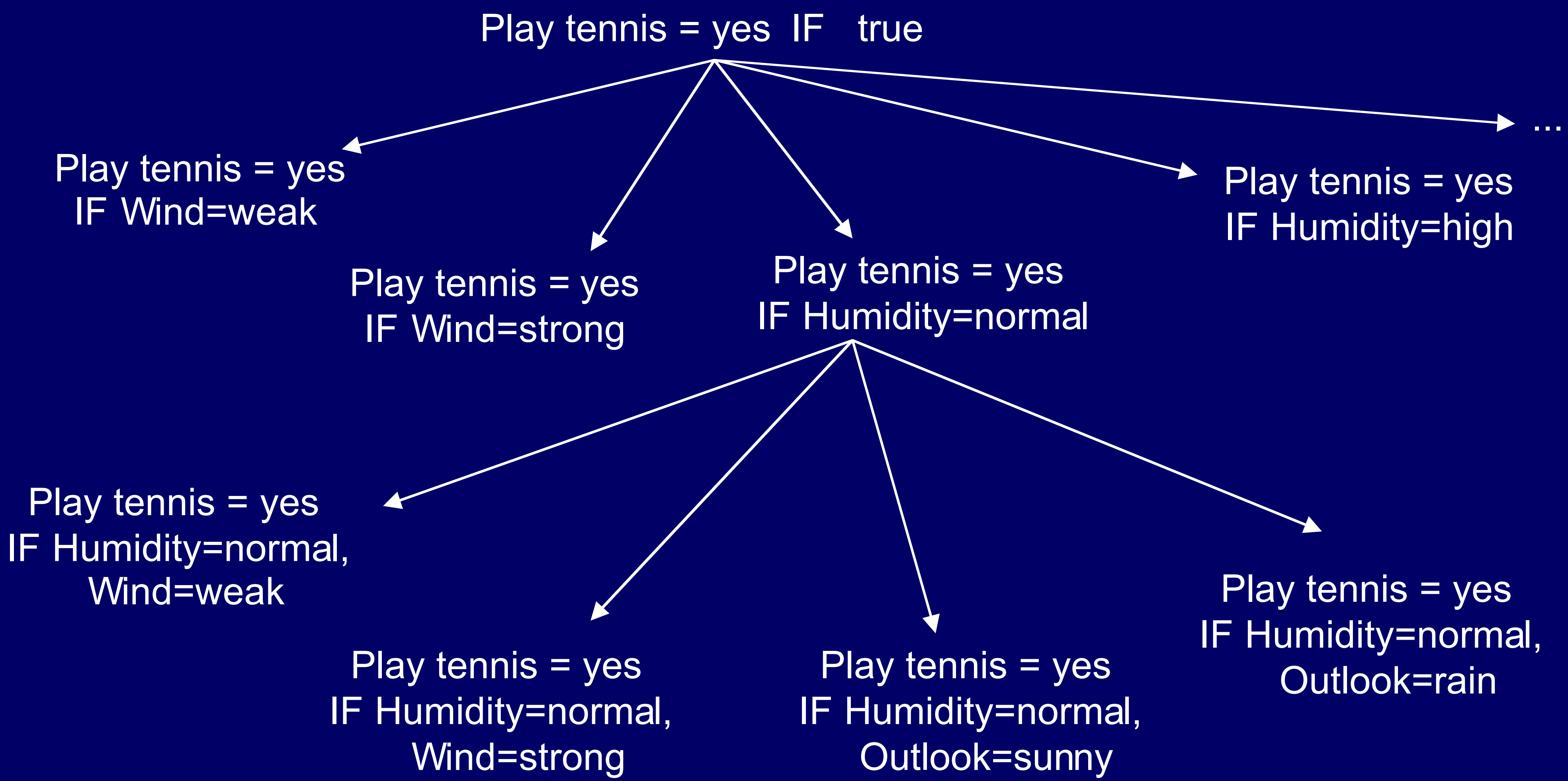
Unordered rulesets

- rule `Class IF Conditions` is learned by first determining `Class` and then `Conditions`
  - NB: **ordered** sequence of classes C_1, …, C_n in RuleSet
  - But: **unordered** (independent) execution of rules when classifying a new instance: all rules are tried and predictions of those covering the example are collected; voting is used to obtain the final classification
- if no rule fires, then `DefaultClass` (majority class in E)

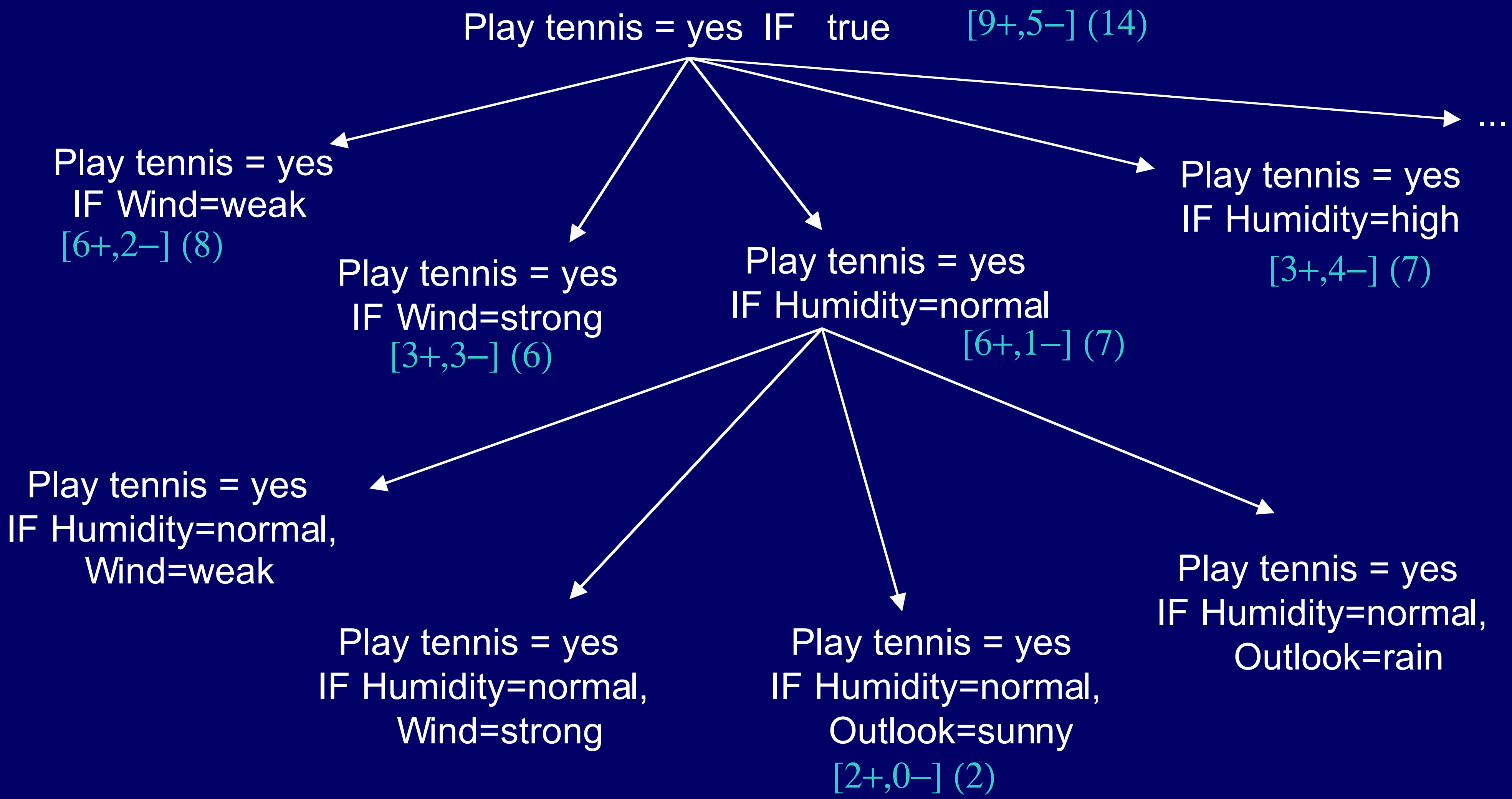
PlayTennis training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Weak	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Learn-one-rule as search



Learn-one-rule as heuristic search



Heuristics for learn-one-rule

- Evaluating accuracy of a rule:
  - $A(C_i \text{ IF Conditions}) = p(C_i \mid \text{Conditions})$
- Estimating probability with relative frequency:
  - covered positives / covered examples
  - [6+,1-] (7) = 6/7, [2+,0-] (2) = 2/2 = 1

Probability estimates

- Relative frequency of covered positives:
  - problems with small samples

$$p(+|R) = \frac{n^+(R)}{n(R)}$$

- Laplace estimate :
  - assumes uniform prior distribution of k classes

$$p(+|R) = \frac{n^+(R)+1}{n(R)+k}$$

- m-estimate :
  - special case: $p_a(+)=1/k$, m=k
  - takes into account prior probabilities $p_a(C)$ instead of uniform distribution
  - independent of the number of classes k
  - m is domain dependent (more noise, larger m)

$$p(+|R) = \frac{n^+(R)+m \cdot p_a(+)}{n(R)+m}$$

Other search heuristics

- Expected accuracy on positives
  - $A(R) = p(+|R)$
- Informativity (#bits needed to specify that example covered by R is +)
  - $I(R) = -\log_2 p(+|R)$
- Accuracy gain (increase in expected accuracy):
  - $AG(R',R) = p(+|R') - p(+|R)$
- Information gain (decrease in the information needed):
  - $IG(R',R) = \log_2 p(+|R') - \log_2 p(+|R)$
- Weighted measures in order to favour more general rules:
  - $WAG(R',R) = n(+R')/n(+R) * (p(+|R') - p(+|R)$
  - $WIG(R',R) = n(+R')/n(+R) * (\log_2 p(+|R') - \log_2 p(+|R))$

Ordered rulesets

- rule `Class IF Conditions` is learned by first determining `Conditions` and then `Class`
  - NB: **mixed** sequence of classes C_1, …, C_n in RuleSet
  - But: **ordered** execution when classifying a new instance: rules are sequentially tried and the first rule that 'fires' (covers the example) is used for classification
- if no rule fires, then `DefaultClass` (majority class in E)

Learning ordered set of rules

- RuleList := empty; E_{cur}:= E
- **repeat**
  - learn-one-rule R
  - RuleList := RuleList ++ R
  - E_{cur} := E_{cur} - {all examples covered by R}
- **until** performance(R, E_{cur}) < ThresholdR
- RuleList := sort RuleList by performance(R,E)
- RuleList := RuleList ++ DefaultRule(E_{cur})

Overview

- Introduction
- Learning rules with CN2
- **Learning Prolog rules with ILP**
- Rule learning with other declarative languages

First-order representations

- **Propositional** representations:
  - datacase is ***fixed-size vector of values***
  - features are those given in the dataset
- **First-order** representations:
  - datacase is ***flexible-size, structured object***
    - sequence, set, graph
    - hierarchical: e.g. set of sequences
  - features need to be **selected** from potentially infinite set

Predicting carcinogenicity

- A molecular compound is carcinogenic if:
  - (1) it tests positive in the Salmonella assay; or
  - (2) it tests positive for sex-linked recessive lethal mutation in Drosophila; or
  - (3) it tests negative for chromosome aberration; or
  - (4) it has a carbon in a six-membered aromatic ring with a partial charge of -0.13; or
  - (5) it has a primary amine group and no secondary or tertiary amines; or
  - (6) it has an aromatic (or resonant) hydrogen with partial charge ≥ 0.168; or
  - (7) it has an hydroxy oxygen with a partial charge ≥ -0.616 and an aromatic (or resonant) hydrogen; or
  - (8) it has a bromine; or
  - (9) it has a tetrahedral carbon with a partial charge ≤ -0.144 and tests positive on Progol's mutagenicity rules.

Concept learning in logic

- **Given**:
  - ***positive examples P***: ground facts to be entailed,
  - ***negative examples N***: ground facts not to be entailed,
  - ***background theory B***: a set of predicate definitions;
- **Find**: a ***hypothesis H*** (one or more predicate definitions) such that
  - for every $p \in P$: $B \cup H \models p$ (***completeness***),
  - for every $n \in N$: $B \cup H \not\models n$ (***consistency***).

Clausal logic

- **predicate logic:**

  $\forall X$: bachelor(X) $\leftrightarrow$ male(X)$\wedge$adult(X)$\wedge\neg$married(X)

- **clausal logic:**

```
bachelor(X);married(X):-male(X),adult(X).

male(X):-bachelor(X).
adult(X):-bachelor(X).

:-bachelor(X),married(X).
```

indefinite clause

definite (Horn) clauses

denial

Prolog

- Ancestors:
  - `ancestor(X,Y):-parent(X,Y).`
  - `ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).`
- Lists:
  - `member(X,[X|Z]).`
  - `member(X,[Y|Z]):-member(X,Z).`

  - `append([],X,X).`
  - `append([X|Xs],Ys,[X|Zs]):-append(Xs,Ys,Zs).`

ILP methods

- **bottom-up**:
  - **data-driven** approach
  - start with **long, specific clause**
  - **generalise** by applying inverse substitutions and/or removing literals
- **top-down**:
  - **generate-then-test** approach
  - start with **short, general clause**
  - **specialise** by applying substitutions and/or adding literals

Top-down induction: example

example	*action*	*hypothesis*
`+p(b,[b])`	add clause	`p(X,Y).`
`-p(x,[])`	specialise	`p(X,[V|W]).`
`-p(x,[a,b])`	specialise	`p(X,[X|W]).`
`+p(b,[a,b])`	add clause	`p(X,[X|W]).` `p(X,[V|W]):-p(X,W).`

Bottom-up induction: example

- Treat positive examples + ground background facts as **body**
- Choose two examples as **heads** and **anti-unify**

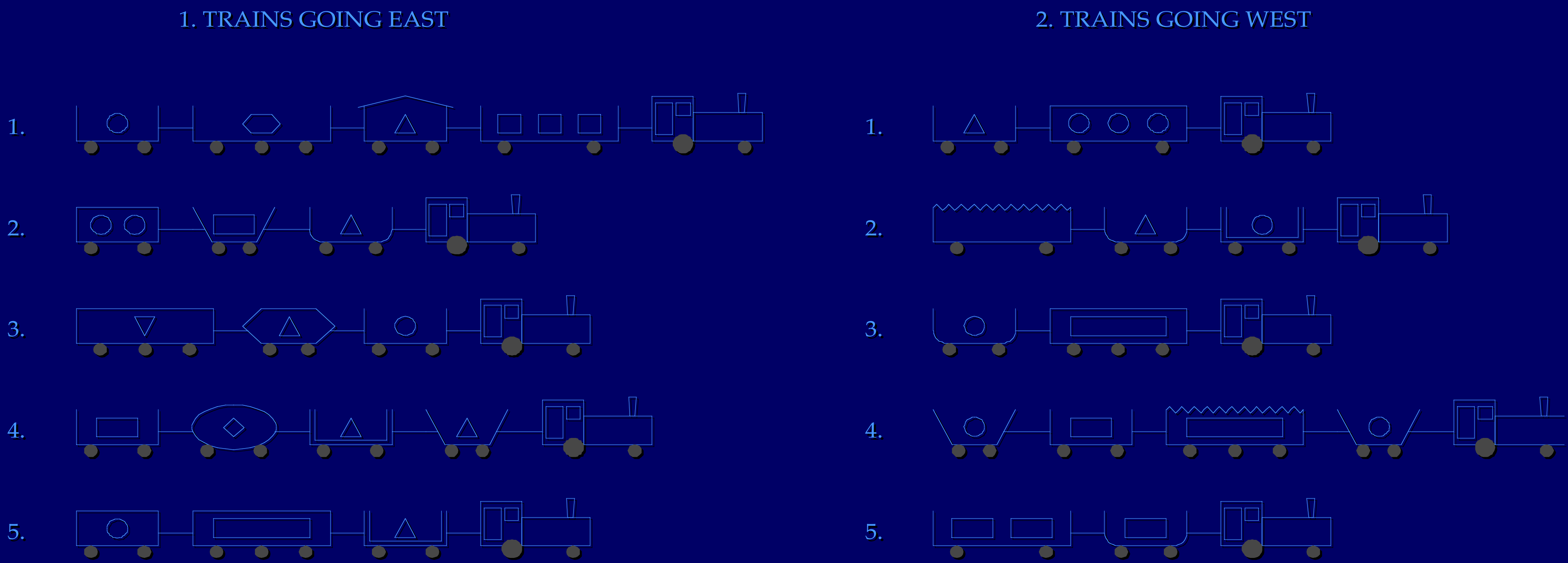
```
q([1,2],[3,4],[1,2,3,4]):-
   q([1,2],[3,4],[1,2,3,4]),q([a],[],[a]),q([],[],[]),q([2],[3,4],[2,3,4])

q([a],[],[a]):-
   q([1,2],[3,4],[1,2,3,4]),q([a],[],[a]),q([],[],[]),q([2],[3,4],[2,3,4])

q([A|B],C,[A|D]):-
   q([1,2],[3,4],[1,2,3,4]),q([A|B],C,[A|D]),q(W,C,X),q([S|B],[3,4],[S,T,U|V]),
   q([R|G],K,[R|L]),q([a],[],[a]),q(Q,[],Q),q([P],K,[P|K]),
   q(N,K,O),q(M,[],M),q([],[],[]),q(G,K,L),
   q([F|G],[3,4],[F,H,I|J]),q([E],C,[E|C]),q(B,C,D),q([2],[3,4],[2,3,4])
```

- Generalise by **removing literals** until negative examples covered

ILP systems

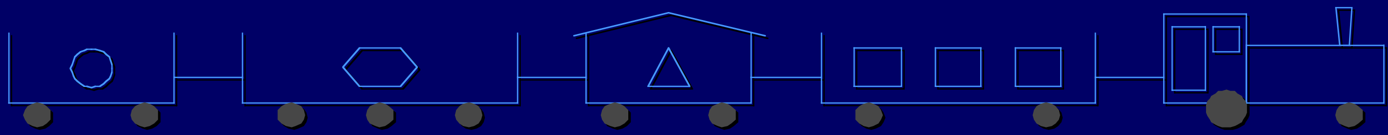
- **MIS** (Shapiro, 1983)
  - top-down, incremental, non-heuristic
- **CIGOL** (Muggleton & Buntine, 1988)
  - bottom-up (inverting resolution), incremental, compression
- **FOIL** (Quinlan, 1990)
  - top-down, non-incremental, information-gain
- **GOLEM** (Muggleton & Feng, 1990)
  - bottom-up, non-incremental, compression
- **LINUS** (Lavrac, Dzeroski & Grobelnik, 1991)
  - transformation to attribute-value learning
- **PROGOL** (Muggleton, 1995)
  - hybrid, non-incremental, compression

East-West trains



ILP representation (flattened)

- **Example:**
  `eastbound(t1).`



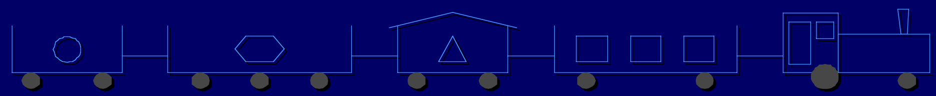
- **Background theory:**

```
car(t1,c1).         car(t1,c2).         car(t1,c3).         car(t1,c4).
rectangle(c1).      rectangle(c2).      rectangle(c3).      rectangle(c4).
short(c1).          long(c2).           short(c3).          long(c4).
none(c1).           none(c2).           peaked(c3).         none(c4).
two_wheels(c1).     three_wheels(c2).   two_wheels(c3).     two_wheels(c4).
load(c1,l1).        load(c2,l2).        load(c3,l3).        load(c4,l4).
circle(l1).         hexagon(l2).        triangle(l3).       rectangle(l4).
one_load(l1).       one_load(l2).       one_load(l3).       three_loads(l4).
```

- **Hypothesis:**

```
eastbound(T):-car(T,C),short(C),not none(C).
```

ILP representation (terms)



- **Example:**

```
eastbound([c(rectangle,short,none,2,l(circle,1)),
           c(rectangle,long,none,3,l(hexagon,1)),
           c(rectangle,short,peaked,2,l(triangle,1)),
           c(rectangle,long,none,2,l(rectangle,3))]).
```

- **Background theory: empty**

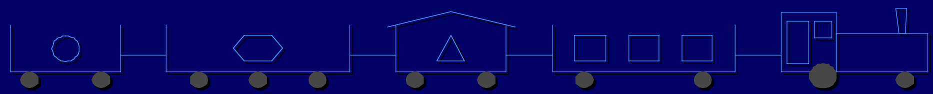
- **Hypothesis:**

```
eastbound(T):-member(C,T),arg(2,C,short),
                          not arg(3,C,none).
```

ILP representation (strongly typed)

- Type signature:

```
data Shape  = Rectangle | Hexagon | …;   data Length = Long | Short;
data Roof   = None | Peaked | …;         data Object = Circle | Hexagon | …;

type Wheels = Int;     type Load = (Object,Number);       type Number = Int
type Car    = (Shape,Length,Roof,Wheels,Load);            type Train = [Car];

eastbound::Train->Bool;
```



- Example:

```
eastbound([(Rectangle,Short,None,2,(Circle,1)),
           (Rectangle,Long,None,3,(Hexagon,1)),
           (Rectangle,Short,Peaked,2,(Triangle,1)),
           (Rectangle,Long,None,2,(Rectangle,3))]) = True
```

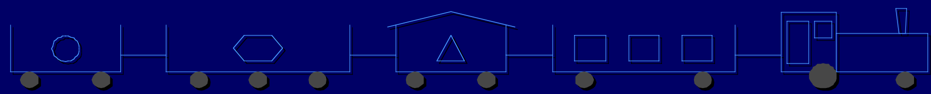
- Hypothesis:

```
eastbound(t) = (exists \c -> member(c,t) &&
                    LengthP(c)==Short && RoofP(c)!=None)
```

ILP representation (strongly typed)

- Type signature:

```
data Shape  = Rectangle | Hexagon | …;   data Length = Long | Short;
data Roof   = None | Peaked | …;          data Object = Circle | Hexagon | …;

type Wheels = Int;      type Load = (Object,Number);        type Number = Int
type Car    = (Shape,Length,Roof,Wheels,Load);              type Train = [Car];

eastbound::Train->Bool;
```



- Example:

```
eastbound([(Rectangle,Short,None,2,(Circle,1)),
           (Rectangle,Long,None,3,(Hexagon,1)),
           (Rectangle,Short,Peaked,2,(Triangle,1)),
           (Rectangle,Long,None,2,(Rectangle,3))]) = True
```

- Hypothesis:

```
eastbound(t) = (exists \c -> member(c,t) &&
                        LengthP(c)==Short && RoofP(c)!=None)
```

ILP representation (database)

LOAD_TABLE

LOAD	CAR	OBJECT	NUMBER
l1	c1	circle	1
l2	c2	hexagon	1
l3	c3	triangle	1
l4	c4	rectangle	3
…	…	…	

TRAIN_TABLE

TRAIN	EASTBOUND
t1	TRUE
t2	TRUE
…	…
t6	FALSE
…	…

CAR_TABLE

CAR	TRAIN	SHAPE	LENGTH	ROOF	WHEELS
c1	t1	rectangle	short	none	2
c2	t1	rectangle	long	none	3
c3	t1	rectangle	short	peaked	2
c4	t1	rectangle	long	none	2
…	…	…			…

```
SELECT DISTINCT TRAIN_TABLE.TRAIN FROM TRAIN_TABLE, CAR_TABLE WHERE
        TRAIN_TABLE.TRAIN = CAR_TABLE.TRAIN AND
        CAR_TABLE.SHAPE = 'rectangle' AND
        CAR_TABLE.ROOF != 'none'
```

Complexity of ILP problems

- Simplest case: single table with primary key
  - example corresponds to tuple of constants
  - ***attribute-value*** or ***propositional*** learning
- Next: single table without primary key
  - example corresponds to set of tuples of constants
  - ***multiple-instance*** problem
- Complexity resides in many-to-one foreign keys
  - lists, sets, multisets
  - ***non-determinate*** variables

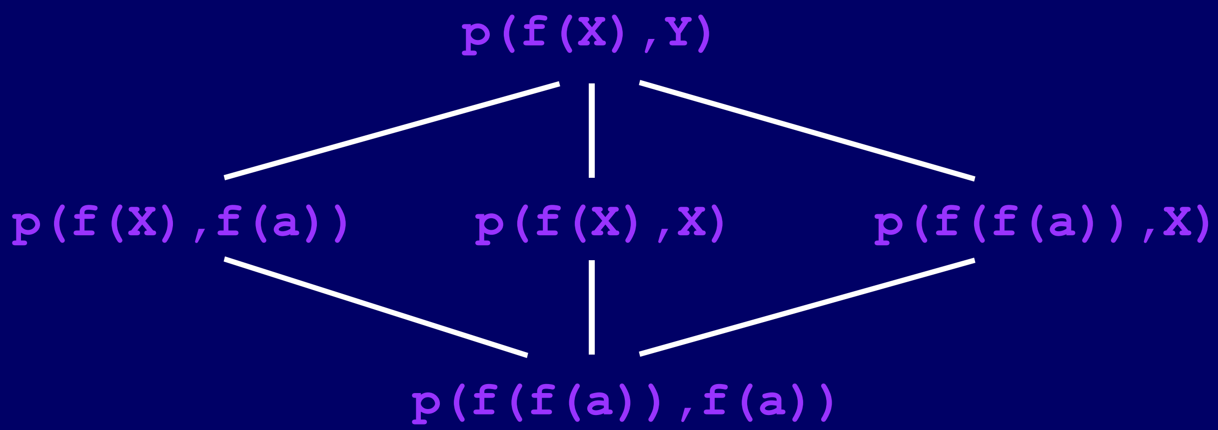
ILP representations: summary

- Term representation collects (almost) all information about individual in one term
  - what about graphs?
- Strongly typed language provides strong bias
  - assumes term representation
- Flattened representation for multiple individuals
  - structural predicates and utility predicates
- NB. assumes ***individual-centred*** classification problem
  - not: logic program synthesis

Generality

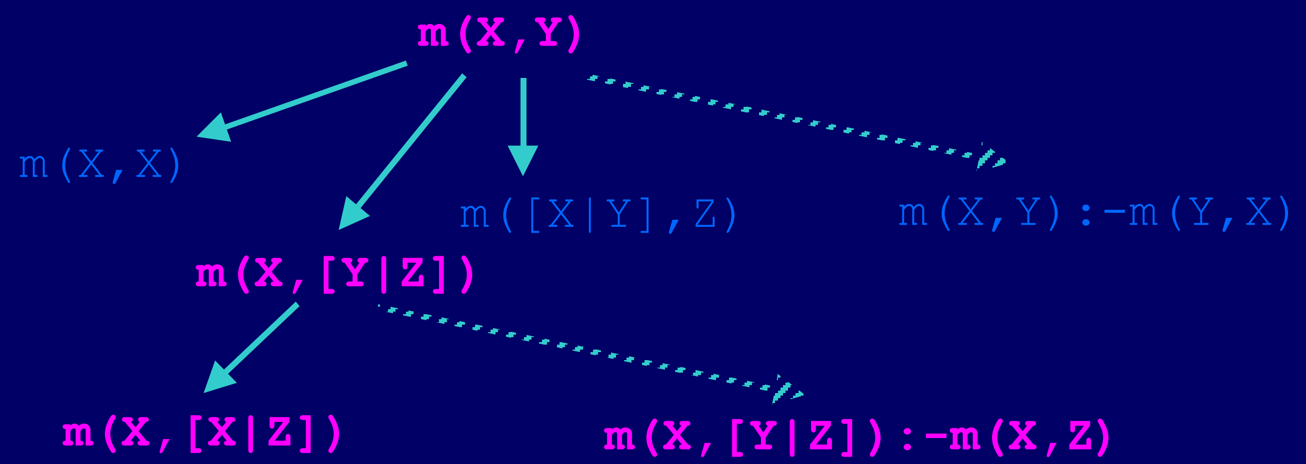
- Generality is primarily an **extensional** notion:
  - one predicate definition is more general than another if its ***extension*** is a proper **superset** of the latter's extension
- This can be used to **structure** and **prune** the hypothesis space
  - if a rule does not cover a positive example, none of its specialisations will
  - if a rule covers a negative example, all of its generalisations will
- We need an ***intensional*** notion of generality, operating on formulae rather than extensions
  - generality of terms, clauses, and theories

Generality of terms



- The set of first-order terms is a **lattice**:
  - t_1 is more general than t_2 iff for some substitution θ: $t_1\theta = t_2$
  - glb ⇒ unification, lub ⇒ anti-unification
  - Specialisation ⇒ applying a substitution
  - Generalisation ⇒ applying an inverse substitution

Generality of clauses



- The set of (equivalence classes of) clauses is a **lattice**:
  - C_1 is more general than C_2 iff for some substitution θ: $C_1\theta \subseteq C_2$
  - glb $\Rightarrow$ θ-MGS, lub $\Rightarrow$ θ-LGG
  - Specialisation $\Rightarrow$ applying a substitution and/or adding a literal
  - Generalisation $\Rightarrow$ applying an inverse substitution and/or removing a literal
  - NB. There are infinite chains!

θ-LGG: examples

```
a([1,2],[3,4],[1,2,3,4]):-a([2],[3,4],[2,3,4])
a([a]  ,[]   ,[a]      ):-a([] ,[]   ,[]     )
a([A|B],C    ,[A|D]    ):-a(B  ,C    ,D      )

m(c,[a,b,c]):-m(c,[b,c]),m(c,[c])
m(a,[a,b  ]):-m(a,[a  ])
m(P,[a,b|Q]):-m(P,[R|Q]),m(P,[P])
```

θ-subsumption vs. implication

- Logical implication is **strictly stronger** than θ-subsumption
  - e.g. `p([V|W]):-p(W)` `p([X,Y|Z]):-p(Z)`
  - this happens when the resolution derivation requires the left-hand clause more than once
- i-LGG of definite clauses is **not unique**
  - i-LGG(`p([A,B|C]):-p(C)`,`p([P,Q,R|S]):-p(S)`) = {`p([X|Y]):-p(Y)`,`p([X,Y|Z]):-p(V)`}
- Logical implication between clauses is undecidable, θ-subsumption is NP-complete

Generality of theories

- Simplification 1: $T_1=B\cup\{C_1\}$ and $T_2=B\cup\{C_2\}$ differ just in one clause
- Simplification 2: approximate B by finite ground model B′
- form clauses C_{1B} and C_{2B} by adding ground facts in B′ to bodies
- θ-RLGG(C_1,C_2,B) = θ-LGG(C_{1B},C_{2B})

θ-RLGG: example

```
a([1,2],[3,4],[1,2,3,4]):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                a([2],[3,4],[2,3,4]).

a([a]  ,[]   ,[a]      ):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                a([2],[3,4],[2,3,4]).

a([A|B],C    ,[A|D]    ):-
      a([1,2],[3,4],[1,2,3,4]), a([A|B],C,[A|D]), a(E,C,F),
      a([G|B],[3,4],[G,H,I|J]),
      a([K|L,M,[K|N]), a([a],[],[a]), a(O,[],O),
      a([P],M,[P|M]),
      a(Q,M,R), a(S,[],S), a([],[],[]), a(L,M,N),
      a([T|L],[3,4],[T,U,V|W]), a(X,C,[X|C]), a(B,C,D),
      a([2],[3,4],[2,3,4]).
```

θ-RLGG: example

```
a([1,2],[3,4],[1,2,3,4]):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                 a([2],[3,4],[2,3,4]).

a([a]  ,[]   ,[a]      ):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                 a([2],[3,4],[2,3,4]).

a([A|B],C    ,[A|D]    ):-
      a([1,2],[3,4],[1,2,3,4]), a([A|B],C,[A|D]), a(E,C,F),
      a([G|B],[3,4],[G,H,I|J]),
      a([K|L,M,[K|N]), a([a],[],[a]), a(O,[],O),
      a([P],M,[P|M]),
      a(Q,M,R), a(S,[],S), a([],[],[]), a(L,M,N),
      a([T|L],[3,4],[T,U,V|W]), a(X,C,[X|C]), a(B,C,D),
      a([2],[3,4],[2,3,4]).
```

θ-RLGG: example

```
a([1,2],[3,4],[1,2,3,4]):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                a([2],[3,4],[2,3,4]).

a([a]   ,[]    ,[a]        ):-
      a([1,2],[3,4],[1,2,3,4]), a([a],[],[a]),
      a([],[],[]),                a([2],[3,4],[2,3,4]).

a([A|B],C     ,[A|D]       ):-
      a([1,2],[3,4],[1,2,3,4]), a([A|B],C,[A|D]), a(E,C,F),
      a([G|B],[3,4],[G,H,I|J]),
      a([K|L,M,[K|N]), a([a],[],[a]), a(O,[],O),
      a([P],M,[P|M]),
      a(Q,M,R), a(S,[],S), a([],[],[]), a(L,M,N),
      a([T|L],[3,4],[T,U,V|W]), a(X,C,[X|C]), a(B,C,D),
      a([2],[3,4],[2,3,4]).
```

Traditional view of rule learning

- **Hypothesis construction**: find a set of *n* rules
  - usually simplified by *n* separate rule constructions
    - exception: HYPER
- **Rule construction**: find a pair (Head, Body)
  - e.g. select class and construct body
    - exceptions: CN2, APRIORI
- **Body construction**: find a set of *m* literals
  - usually simplified by adding one literal at a time
    - problem (ILP): literals introducing new variables

The role of feature construction

- **Hypothesis construction**: find a set of *n* rules
- **Rule construction**: find a pair (Head, Body)
- **Body construction**: find a set of *m* features
- **Feature construction**: find a set of *k* literals
  - e.g. interesting subgroup, frequent itemset
  - discovery task rather than classification task

First-order features

- Features concern interactions of local variables

- The following rule has two features '**has a short car**' and '**has a closed car**':

```
eastbound(T):-hasCar(T,C1),clength(C1,short),
  hasCar(T,C2),not croof(C2,none).
```

- The following rule has one feature '**has a short closed car**':

```
eastbound(T):-hasCar(T,C),clength(C,short),
  not croof(C,none).
```

Propositionalising rules

- Equivalently:

```
eastbound(T):-hasShortCar(T),hasClosedCar(T).
hasShortCar(T):-hasCar(T,C),clength(C,short).
hasClosedCar(T):-hasCar(T,C),not croof(C,none).
```

- Given a way to construct (or choose) first-order features, body construction in ILP is ***propositional***
  - learn non-determinate clauses with LINUS by saturating background knowledge

Declarative bias for first-order features

- Flattened representation, but derived from strongly-typed term representation
  - one free global variable
  - each (binary) structural predicate introduces a new existential local variable and uses either global variable or local variable introduced by other structural predicate
  - utility predicates only use variables
  - all variables are used
- NB. features can be non-boolean

Example: mutagenesis

- 42 regression-unfriendly molecules
- **57 first-order features with one utility literal**
- LINUS using CN2: 83%

```
mutagenic(M,false):-not (has_atom(M,A),atom_type(A,21)),
  logP(M,L),L>1.99,L<5.64.
mutagenic(M,false):-not
  (has_atom(M,A),atom_type(A,195)),
  lumo(M,Lu),Lu>-1.74,Lu<-0.83,
  logP(M,L),L>1.81.
mutagenic(M,false):-lumo(M,Lu),Lu>-0.77.

mutagenic(M,true):-has_atom(M,A),atom_type(A,21),
  lumo(M,Lu),Lu<-1.21.
mutagenic(M,true):-logP(M,L),L>5.64,L<6.36.
mutagenic(M,true):-lumo(M,Lu),Lu>-0.95,
  logP(M,L),L<2.21.
```

Feature construction: summary

- All the expressiveness of ILP is in the features
  - body construction is essentially propositional
  - every ILP system does constructive induction
- Feature construction is a discovery task
  - use of discovery systems such as Warmr, Tertius or Midos
  - alternative: use a relevancy filter

Overview

- Introduction
- Learning rules with CN2
- Learning Prolog rules with ILP
- **Rule learning with other declarative languages**

Attribute-value learning in Escher

- Type definitions:

```
data Outlook = Sunny | Overcast | Rain;
data Temperature = Hot | Mild | Cool;
data Humidity = High | Normal | Low;
data Wind = Strong | Medium | Weak;
type Weather = (Outlook,Temperature,Humidity,Wind)
playTennis::Weather->Bool;
```

- Examples:

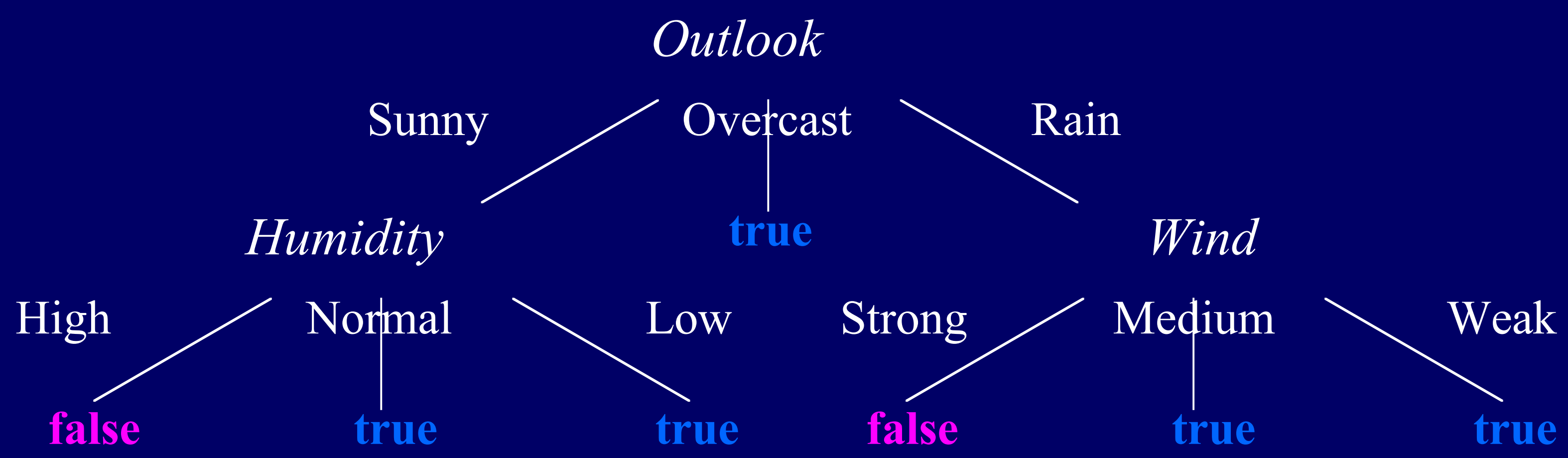
```
playTennis(Overcast,Hot,High,Weak) = True;
playTennis(Sunny,Hot,High,Weak) = False;
```

Attribute-value learning in Escher

```
outlookP::Weather->Outlook;
outlookP(o,t,h,w) = o;
```

- Hypothesis:

```
playTennis(w) =
  if (outlookP(w)==Sunny && humidityP(w)==High) then False
  else if (outlookP(w)==Rain && windP(w)==Strong) then False
  else True;
```

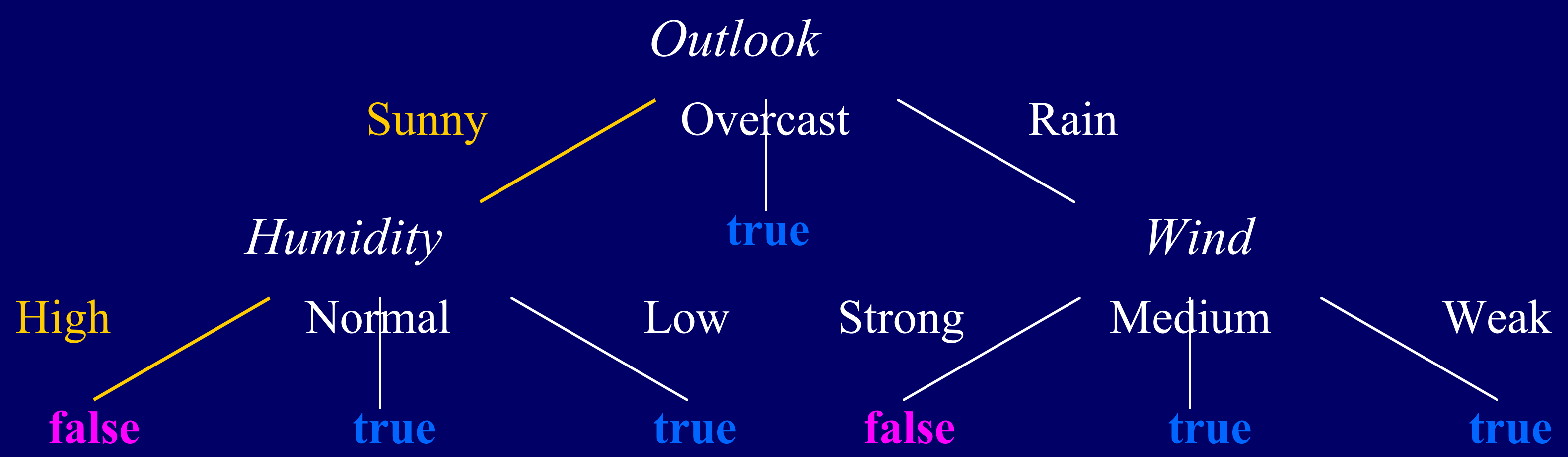


Attribute-value learning in Escher

```
outlookP::Weather->Outlook;
outlookP(o,t,h,w) = o;
```

- Hypothesis:

```
playTennis(w) =
  if (outlookP(w)==Sunny && humidityP(w)==High) then False
  else if (outlookP(w)==Rain && windP(w)==Strong) then False
  else True;
```

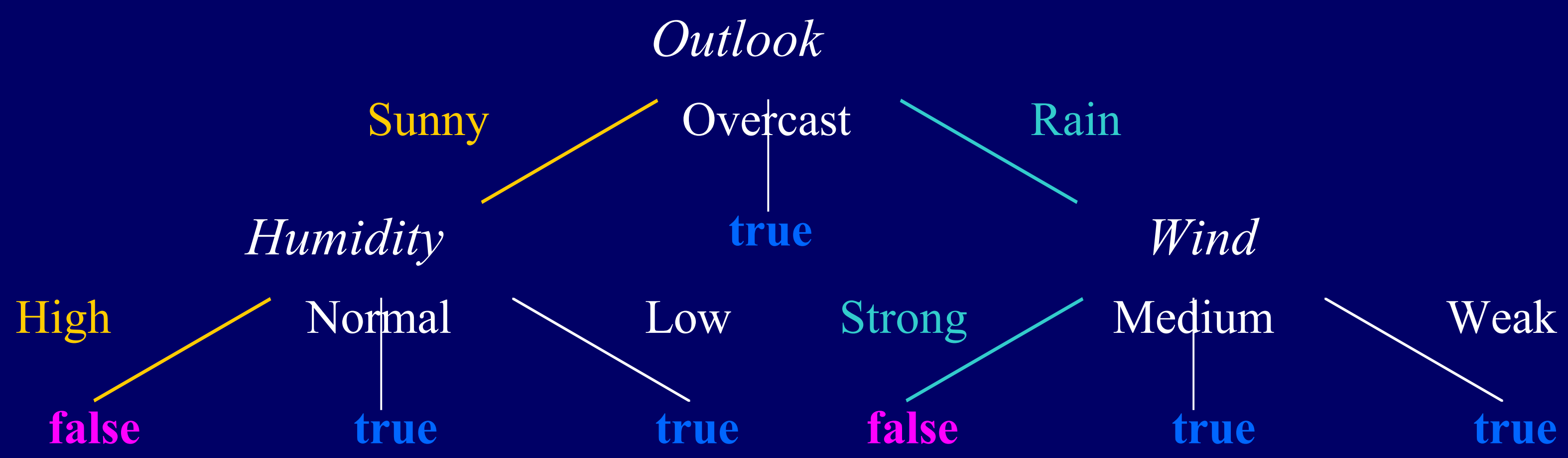


Attribute-value learning in Escher

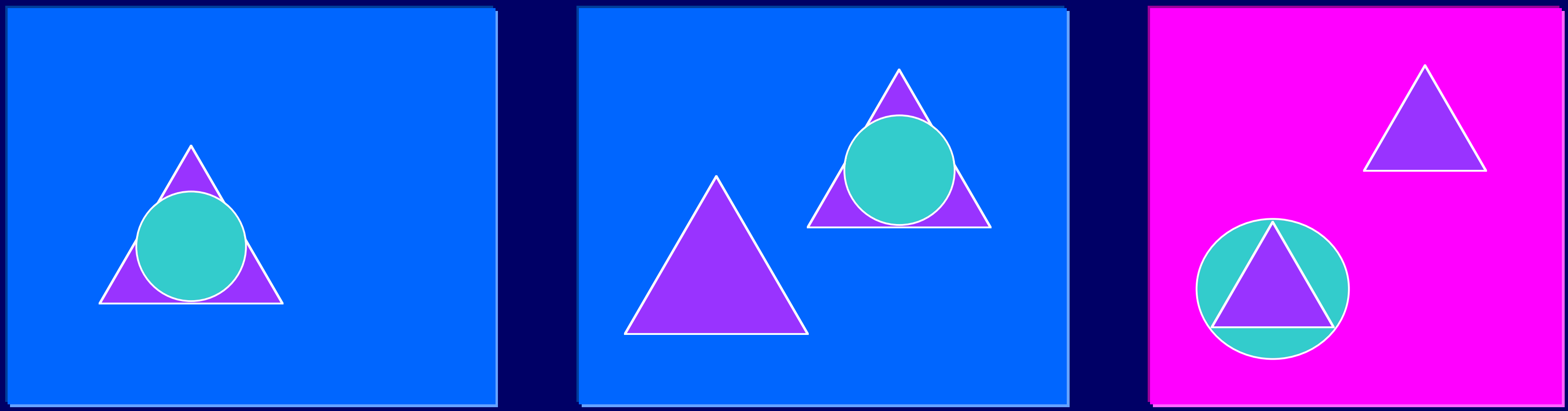
```
outlookP::Weather->Outlook;
outlookP(o,t,h,w) = o;
```

- Hypothesis:

```
playTennis(w) =
  if (outlookP(w)==Sunny && humidityP(w)==High) then False
  else if (outlookP(w)==Rain && windP(w)==Strong) then False
  else True;
```



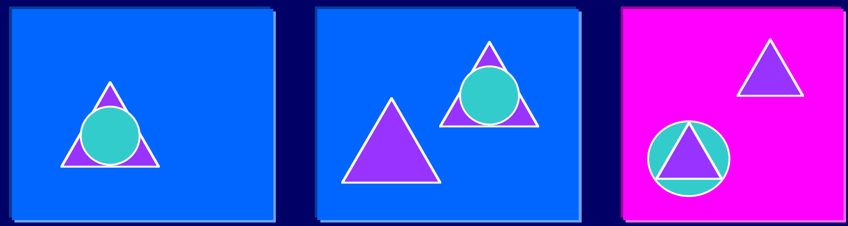
Multi-instance learning in Escher



- Type definitions:

```
data Shape = Circle | Triangle | In(Shape,Shape);
data Class = Positive | Negative;
type Diagram = {(Shape,Int)};
class::Diagram->Class;
```

Multi-instance learning in Escher



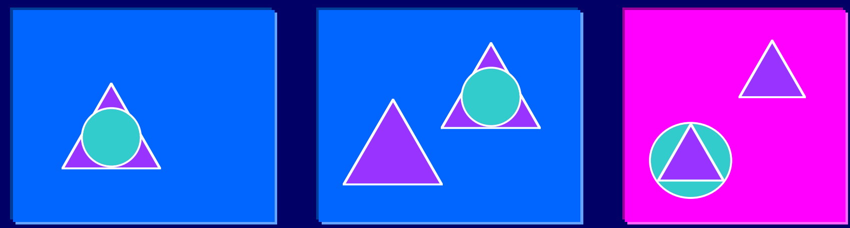
- Examples:

```
class({(In(Circle,Triangle),1)}) = Positive;
class({(Triangle,1),(In(Circle,Triangle),1)}) = Positive;
class({(In(Triangle,Circle),1),(Triangle,1)}) = Negative;
```

- Hypothesis:

```
class(d) =
    if (exists \p -> p 'in' d && (exists \s t ->
          shapeP(p) == In(s,t) && s == Circle))
    then Positive else Negative;
```

Multi-instance learning in Escher



- Examples:

```
class({(In(Circle,Triangle),1)}) = Positive;
class({(Triangle,1),(In(Circle,Triangle),1)}) = Positive;
class({(In(Triangle,Circle),1),(Triangle,1)}) = Negative;
```

- Hypothesis:

```
class(d) =
   if (exists \p -> p 'in' d && (exists \s t ->
         shapeP(p) == In(s,t) && s == Circle))
   then Positive else Negative;
```

Mutagenesis in Escher

- Type definitions:

```
data Element = Br | C | Cl | F | H | I | N | O | S;

type Ind1 = Bool;
type IndA = Bool;
type Lumo = Float;
type LogP = Float;
type Label = Int;
type AtomType = Int;
type Charge = Float;
type BondType = Int;
type Atom = (Label,Element,AtomType,Charge);
type Bond = ({Label},BondType);
type Molecule = (Ind1,IndA,Lumo,LogP,{Atom},{Bond});

mutagenic::Molecule->Bool;
```

Mutagenesis in Escher

- Examples:

```
mutagenic(True,False,-1.246,4.23,
    {(1,C,22,-0.117),
     (2,C,22,-0.117),
      …,
     (26,O,40,-0.388)},
    {({1,2},7),
      …,
     ({24,26},2)})
    = True;
```

atoms

bonds

- NB. Naming of sub-terms cannot be avoided here, because molecules are graphs rather than trees

Mutagenesis in Escher

- Hypothesis:

```
mutagenic(m) =
  ind1P(m) == True || lumoP(m) <= -2.072 ||
  (exists \a -> a 'in' atomSetP(m) && elementP(a)==C &&
           atomTypeP(a)==26 && chargeP(a)==0.115) ||
  (exists \b1 b2 -> b1 'in' bondSetP(m) && b2 'in' bondSetP(m) &&
           bondTypeP(b1)==1 && bondTypeP(b2)==2 &&
           not disjoint(labelSetP(b1),labelSetP(b2)) ||
  (exists \a -> a 'in' atomSetP(m) &&
           elementP(a)==C && atomTypeP(a)==29 &&
           (exists \b1 b2 ->
                   b1 'in' bondSetP(m) && b2 'in' bondSetP(m) &&
                   bondTypeP(b1)==7 && bondTypeP(b2)==1 &&
                   labelP(a) 'in' labelSetP(b1) &&
                   not disjoint(labelSetP(b1),labelSetP(b2)))) ||
  …;
```

Further reading on ILP

A.F. Bowers, C. Giraud-Carrier, and J.W. Lloyd. Classification of individuals with complex structure. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 81--88. Morgan Kaufmann, 2000.

P.A. Flach, C. Giraud-Carrier, and J.W. Lloyd. Strongly Typed Inductive Concept Learning. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 185--194. Springer-Verlag, 1998.

P.A. Flach. Knowledge representation for inductive learning. In Anthony Hunter and Simon Parsons, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'99)*, volume 1638 of *Lecture Notes in Artificial Intelligence*, pages 160--167. Springer-Verlag, July 1999.

P.A. Flach and N. Lavrac. The role of feature construction in inductive rule learning. In L. De Raedt and S. Kramer, editors, *Proceedings of the ICML2000 workshop on Attribute-Value Learning and Relational Learning: Bridging the Gap*, Stanford University, 2000.

N. Lavrac, S. Dzeroski, and M. Grobelnik. Learning Nonrecursive Definitions of Relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265--281. Springer-Verlag, 1991.

S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339--351. Morgan Kaufmann, 1988.

S. Muggleton and C. Feng. Efficient Induction in Logic Programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281--298. Academic Press, 1992.

S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245--286, 1995.

J.R. Quinlan. Learning logical definitions from Relations. *Machine Learning*, 5:239--266, 1990.

E.Y. Shapiro. An algorithm that infers theories from facts. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 446--452. Morgan Kaufmann, 1981.

See also the ILPnet2 on-line library at http://www.cs.bris.ac.uk/~ILPnet2/Library/

Acknowledgements

- Nada Lavrac (Ljubljana) for the CN2 slides
- John Lloyd, Christophe Giraud-Carrier, Nicolas Lachiche, and other (former) members of the Bristol Machine Learning group for joint research
- This tutorial was financially supported by ILPnet2, the European Network of Excellence on Inductive Logic Programming
  - http://www.cs.bris.ac.uk/~ILPnet2/