Machine Learning & Data-Mining
ILP

Dr. Andreas Karwath

Institute of Computer Science
University of Freiburg, Germany

Knowledge Discovery in DB (KDD)

"automatic extraction of novel, useful, and valid knowledge from large sets of data"

Different Kinds of:

- Knowledge
  - Rules
  - Decision trees
  - Cluster hierarchies
  - Association rules
  - Statistically unusual subgroups
  - ...
- Data

Relational Analysis

Would it not be nice to have analysis methods and data mining systems capable of directly working with multiple relations as they are available in relational database systems?

Structured data

- Sequences
  - biology, web-logs, alarm sequences, etc.
  - E.g. biology,
    - proteins are sequences of amino acids
    - predict structure of proteins
- Trees
  - XML documents and document classification
  - Parse trees

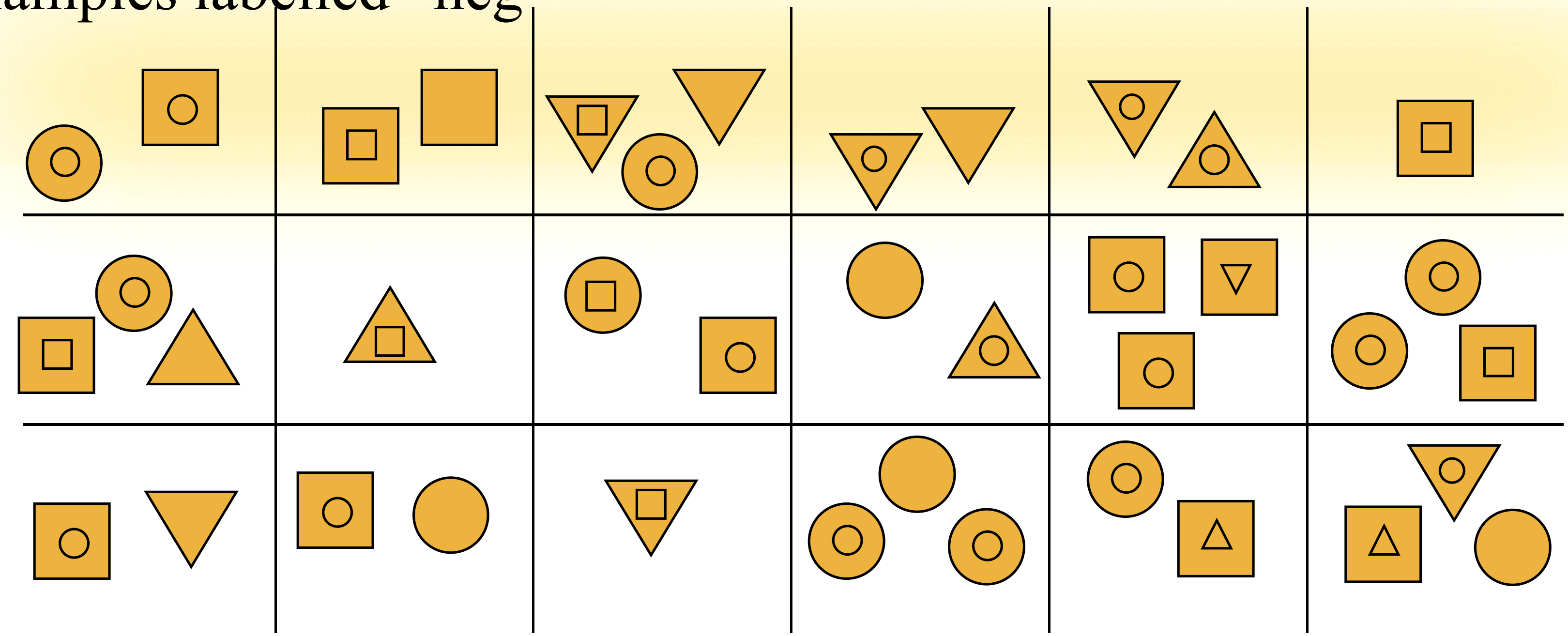
Structured Data

- Graphs
  - Molecules, scenes (bongard problem)
  - Link discovery and analysis
    - Citation analysis, social networks, protein interaction networks, …
    - E.g. Citation analysis
      - nodes are papers and authors.
      - link when paper x cites paper y (or author x cites author y)
    - E.g. social networks
      - US : “homeland security”; analysing terrorist networks

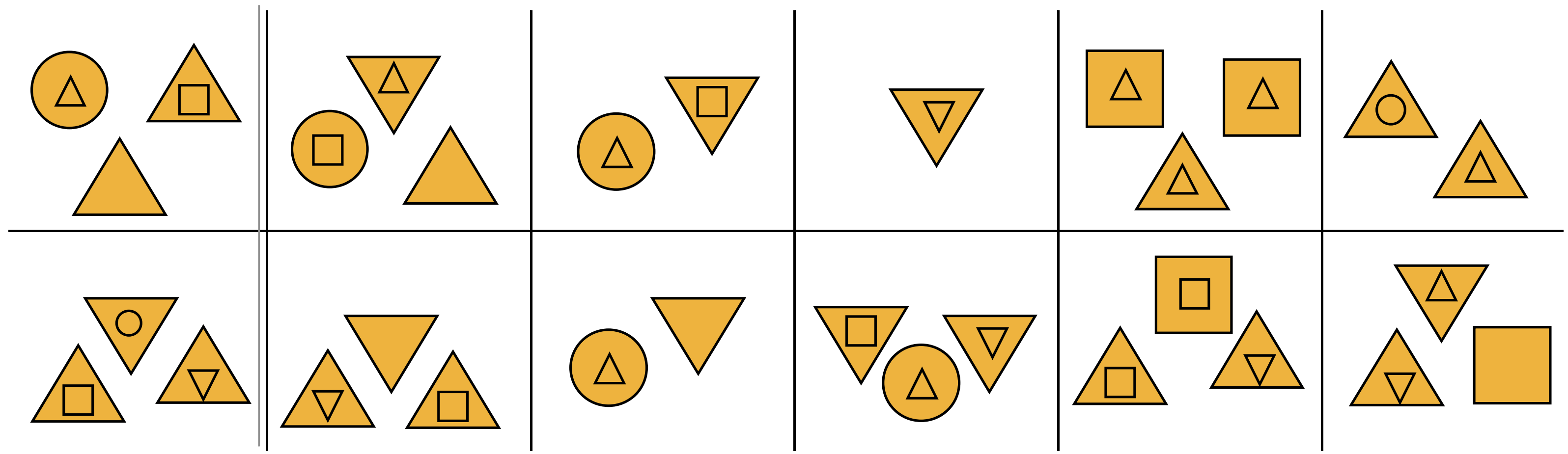
Relational data mining and ILP

- Most databases are relational
- Multiple-table - Multiple-Tuple
- Many data sets cannot elegantly be represented using simple (attribute-value) representation
- Generalizes graph mining setting

Examples labelled “neg”



Examples labelled “pos”

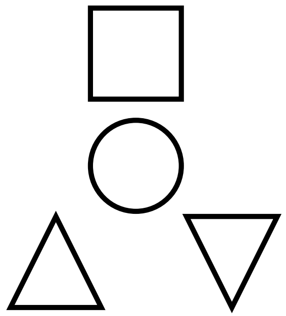


How to deal with such data ?

- Using traditional flat (i.e. attribute value representations) ?
- Either
  - serious loss of information, or
  - combinatorial explosion.
- An example …

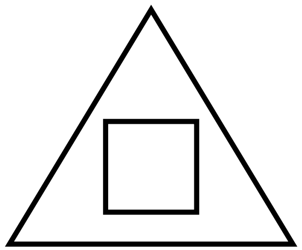
Shape

- squares
- circles
- triangles (up/down)



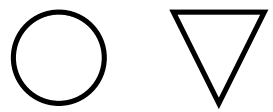
Position

- “in” relation



How to represent in AVL ?

- Assume fixed number of objects



	Object 1	Pointing	Object 2	Pointing	Class
1	circle	N/A	triangle	down	positive

	Object 1	Pointing	Object 2	Pointing	Class
1'	triangle	up	circle	N/A	positive

Problem 1

- equivalent : change objects 1 and 2
  - exponential number of (equivalent) representations of examples and rules
  - if object1 = circle then positive
  - if object2 = circle then positive
  - existing propositional algorithms do not handle this

Problem 2

- relations:
  - leftof(object1,object2) = true
  - leftof(object2,object1) = false
  - many more false facts than true ones

Object 1	Object 2	leftof(1,2)	leftof(2,1)	Class
circle	triangle	TRUE	FALSE	positive

Problem 3

– Variable number of objects

Object 1	Object 2	Object 3	leftof(1,2)	leftof(1,3)	leftof(2,3)	Class
circle	triangle	N/A	TRUE	N/A	N/A	positive
square	triangle	circle	TRUE	FALSE	NO	yes

– Table explodes
– Contains too much N/A values
– Further arguments [De Raedt, ILP98]

Single Table vs Relational DM

The same problems still remain.

But solutions?

More problems:

- Extending the key notations
- Efficiency concerns

Relational Data Mining

Data Mining : ML :: Relational Data Mining : **ILP**

Initially

Binary Classification

Now

Classification, Regression, Clustering, Association Analysis

ILP

Inductive Logic Programming:

- Is a sub-area of Machine Learning, that in turn is part of Artificial Intelligence
- Uses contributions from Logic Programming and Statistics
- Tries to automate the induction processes

Deductive Vs Inductive Reasoning

$T \quad \cup \quad B \quad \rightarrow \quad E$ **(deduce)**

T	B	E
parent(X,Y) :- mother(X,Y). parent(X,Y) :- father(X,Y).	mother(mary,vinni). mother(mary,andre). father(carrey,vinni). father(carry,andre).	parent(mary,vinni). parent(mary,andre). parent(carrey,vinni). parent(carrey,andre).

$E \quad \cup \quad B \quad \rightarrow \quad T$ **(induce)**

E	B	T
parent(mary,vinni). parent(mary,andre). parent(carrey,vinni). parent(carrey,andre).	mother(mary,vinni). mother(mary,andre). father(carrey,vinni). father(carry,andre).	parent(X,Y) :- mother(X,Y). parent(X,Y) :- father(X,Y).

ILP: Objective

Given a dataset:

- Positive examples (*E*+) and optionally negative examples (*E*-)
- Additional knowledge about the problem/application domain (Background Knowledge *B*)
- Set of constraints to make the learning process more efficient (*C*)

Goal of an ILP system is to find a set of hypothesis that:

- Explains (covers) the positive examples - Completeness
- Are consistent with the negative examples - Consistency

$$h \in H : \forall p \in P : \text{covers}(h, p) \ \wedge \ \forall n \in N : \neg \text{covers}(h, n)$$

DB vs. Logic Programming

DB Terminology	LP Terminology
– Relation name *p*	– Predicate symbol *p*
– Attribute of relation *p*	– Argument of predicate *p*
– Tuple $<a_1,\ldots,a_n>$	– Ground fact $p(a_1,\ldots,a_n)$
– Relation *p* a set of tuples	– Predicate *p* defined extensionally by a set of ground facts
– Relation *q* defined as a view	– Predicate *q* defined intentionally by a set of rules (clauses)

Relational Pattern

IF Customer(C1,Age1,Income1,TotSpent1,BigSpender1)
AND MarriedTo(C1,C2)
AND Customer(C2,Age2,Income2,TotSpent2,BigSpender2)
AND Income2 $\geq$ 10000
THEN BigSpender1 = Yes

big_spender(C1,Age1,Income1,TotSpent1) $\leftarrow$
married_to(C1,C2) $\wedge$
customer(C2,Age2,Income2,TotSpent2,BigSpender2) $\wedge$
Income2 $\geq$ 10000

A Generic ILP Algorithm

procedure ILP (Examples)

INITIALIZE (*Theories, Examples*)

repeat

  T = SELECT (*Theories, Examples*)

  $\{T_i\}^n_{i=1}$ = REFINE (T, *Examples*)

  Theories = REDUCE (*Theories* $\cup \bigcup_{i=1}^{n} T_i$, *Examples*)

until STOPPINGCRITERION (*Theories, Examples*)

return (*Theories*)

Procedures for a Generic ILP Algo.

- INITIALIZE: initialize a set of theories

  (e.g. *Theories* = {true} or *Theories* = *Examples*)
- SELECT: select the most promising candidate theory
- REFINE: apply refine operators that guarantee new theories (*specialization, generalization*,…).
- REDUCE: discard unpromising theories
- STOPPINGCRITERION: determine whether the current set of theories is already good enough

  (e.g. when it contains a complete and consistent theory)

SELECT and REDUCE together implement the search strategy.

(e.g. *hill-climbing*: REDUCE = only keep the best theory.)

Search Algorithms

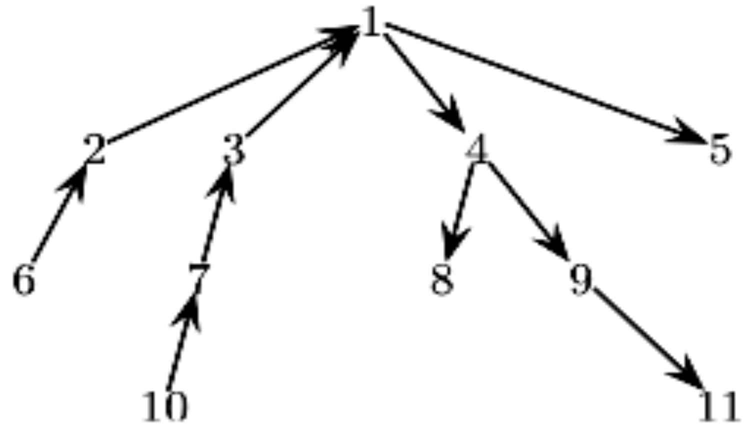
Search Methods

- Systematic Search
  - Depth-first search
  - Breadth-first search
  - Best-first search
- Heuristic Search
  - Hill-climbing
  - Beam-search

Search Direction

- Top-down search: Generic to specific
- Bottom-up search: Specific to general
- Bi-directional search

Example

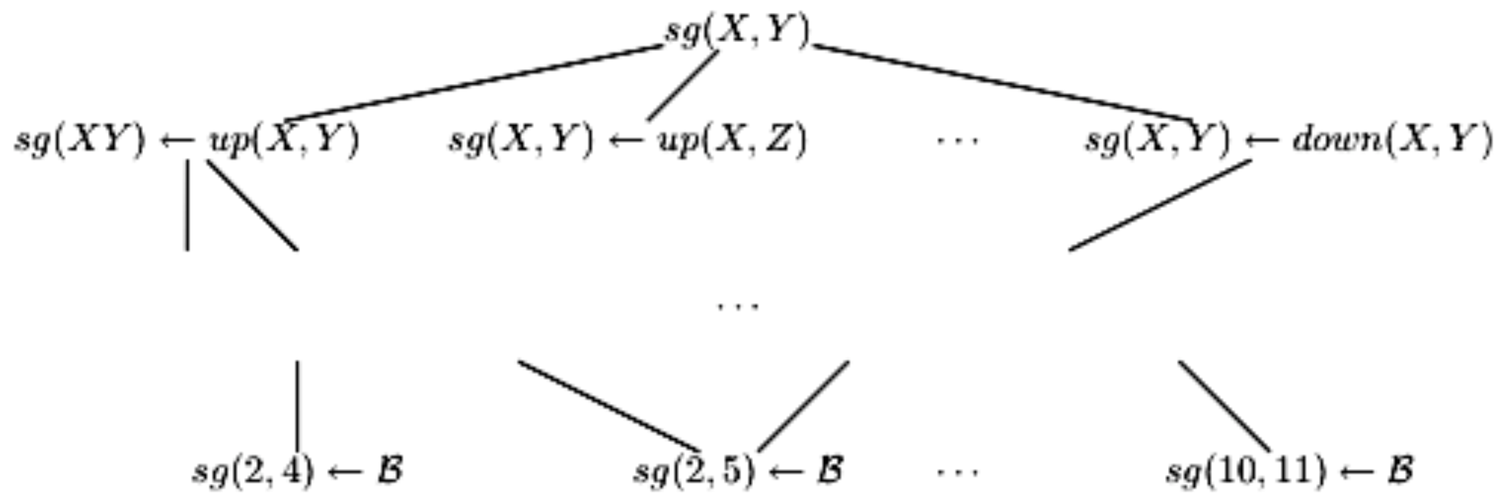


Example: Want to learn $sg(X, Y)$ and are given:

$$\begin{aligned}
\mathcal{B} &= \{up(2,1), up(3,1), up(6,2), up(7,3), up(10,7) \\
&\quad down(1,4), down(1,5), down(4,8), down(4,9), down(9,11)\} \\
\mathcal{E}^+ &= \{sg(2,4), sg(3,5), sg(6,8), sg(7,8), sg(10,11)\} \\
\mathcal{E}^- &= \{sg(2,3), sg(3,8), sg(5,11), sg(4,7)\}
\end{aligned}$$

Search Space

Search space is structured from general to specific:

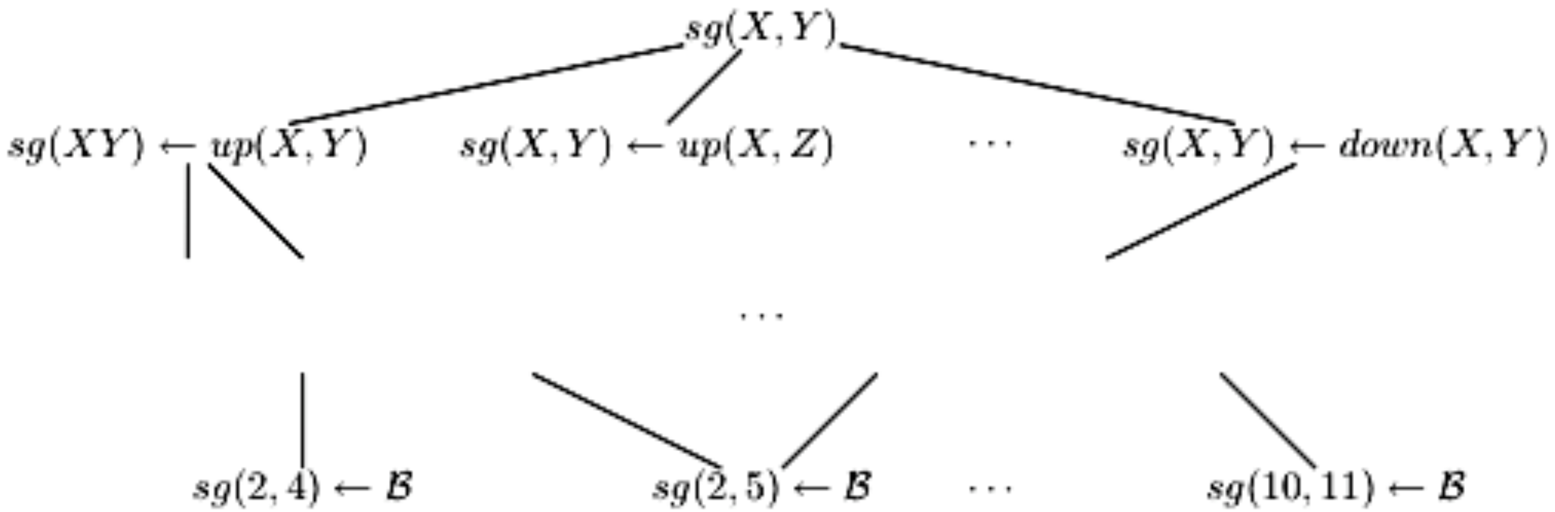


Would like to use entailment to order space, but

$$sg(X,Y) \models sg(X,Y) \leftarrow up(X,Y)?$$

is undecidable

Search Space



So use θ-subsumption instead:

$$c \succeq c' \text{ if } \exists \theta \text{ s.t. } c\theta \subseteq c'$$

Search Space as Lattice

- Search space is a lattice under θ-subsumption
- There exists a *lub* and *glb* for every pair of clauses
- *lub* is 'least general generalization'
- Bottom-up approaches find the *lgg* of the positive examples

Generalization relation

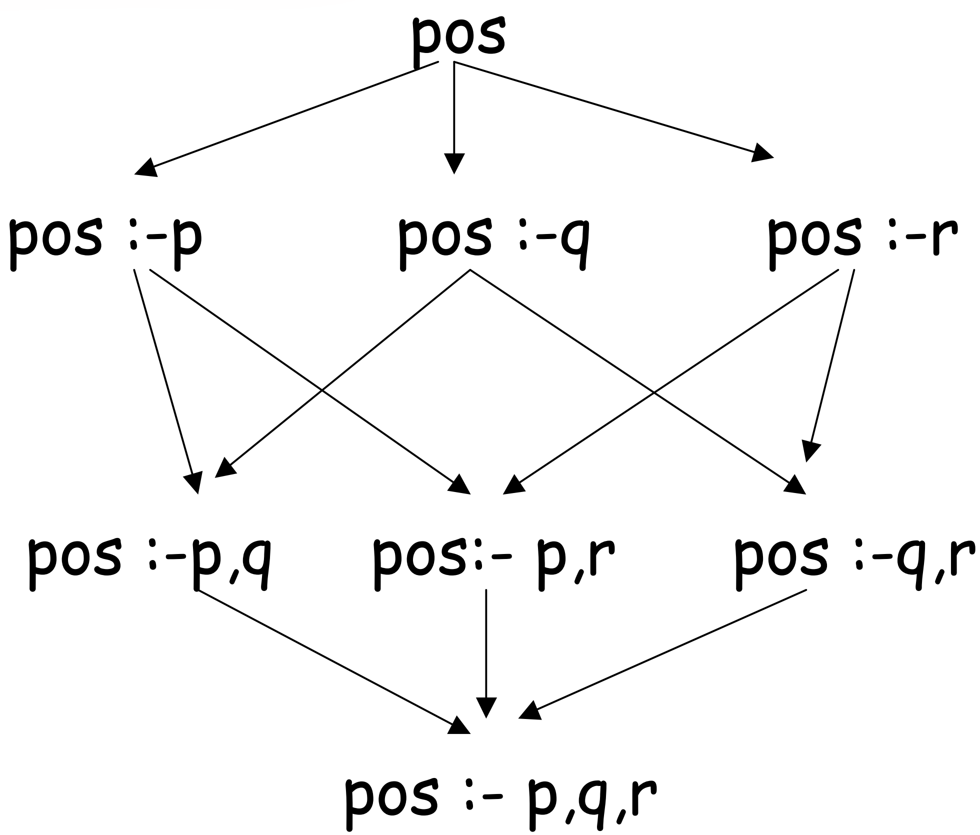
- Typically theta-subsumption is being used
- Let’s introduce specialization operators under theta-subsumption gradually
  - Propositional logic
  - Atoms
  - Queries and clauses

Subsumption in Propositional logic

Clause g subsumes clause s
if and only g |= s
or, equivalently
$g \subseteq s$

pos :- p,q,r |= pos :- p,q,r,s,t
because
$\{pos, \neg p, \neg q, \neg r\} \subseteq \{pos, \neg p, \neg q, \neg r, \neg s, \neg t\}$

Subsumption in propositional logic



Subsumption in propositional logic

- Perfect structure
- Complete lattice
  - any two clauses have unique
    - least upper bound (least general generalization)
    - greatest lower bound
- No syntactic variants
- Easy specialization, generalization

Refinement operators

Specialization operator :

$$\rho_{spec}(h) = \{s \mid s \text{ is a proper minimal specialisation of } h\}$$

Generalization operator :

$$\rho_{gen}(h) = \{g \mid g \text{ is a proper minimal generalization of } h\}$$

Ref. Operators for propositional clauses

Specialization operator :

$$\rho(h) = \{h \wedge l \mid l \text{ is a literal }\}$$

Generalization operator :

$$\rho(h) = \{g \mid \; g \text{ is } h \text{ with a literal } l \text{ deleted}\}$$

Subsumption in logical atoms

- g subsumes s if and only if there is a substitution θ such that $g\theta = s$
- Still nice properties and complete lattice up to variable renaming
  - p(X,a) and p(U,a)
  - greatest lower bound = unification
  - unification p(X,a) and p(b,U) gives p(b,a)
  - least upper bound = anti-unification = lgg
  - lgg p(X,a,b) and p(c,a,d) = p(X,a,Y)

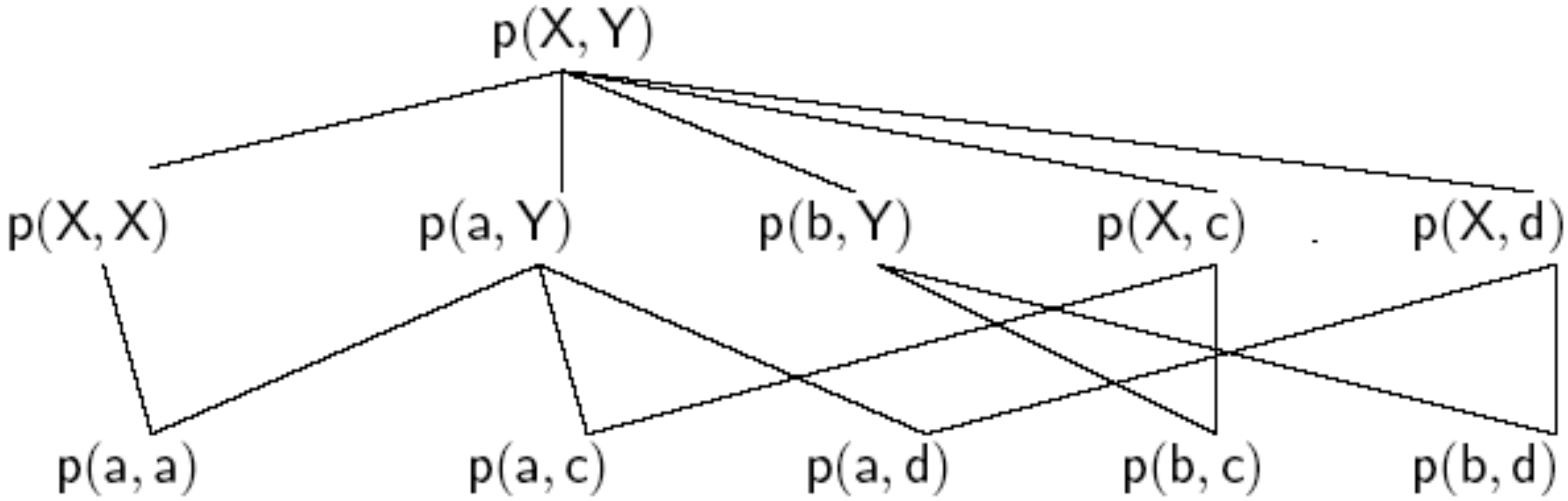


Fig. 5.2. Part of the lattice on atoms.

Lgg of atoms

– lgg of terms (variables or constants) :

  lgg(t,t) = t

  lgg(f,g) = V (throughout)

  lgg(X,g) = V (throughout)

– lgg of atoms :

  $lgg(p(s_1, \ldots, s_n), p(t_1, \ldots, t_n)) =$

  $\quad p(lgg(s_1,t_1), \ldots, lgg(s_n,t_n))$

  $lgg(p(s_1, \ldots, s_n), q(t_1, \ldots, t_m)) =$ undefined

Operators

- Specialization operator :
  - apply a substitution { X / Y } where X,Y already appear in atom
  - apply a substitution {X / c } where c is a constant

Theta-subsumption (Plotkin 70)

- Most important framework for inductive logic programming. Used by all major ILP systems.
- S and G are single clauses
- Combines propositional subsumption and subsumption on logical atoms
- c1 theta-subsumes c2 if and only if there is a substitution θ such that c1 $\theta \subseteq$ c2
- c1 : father(X,Y) :- parent(X,Y),male(X)
- c2 : father(jef,paul) :- parent(jef,paul), parent(jef,an), male(jef), female(an)
- θ = { X / jef, Y /paul }

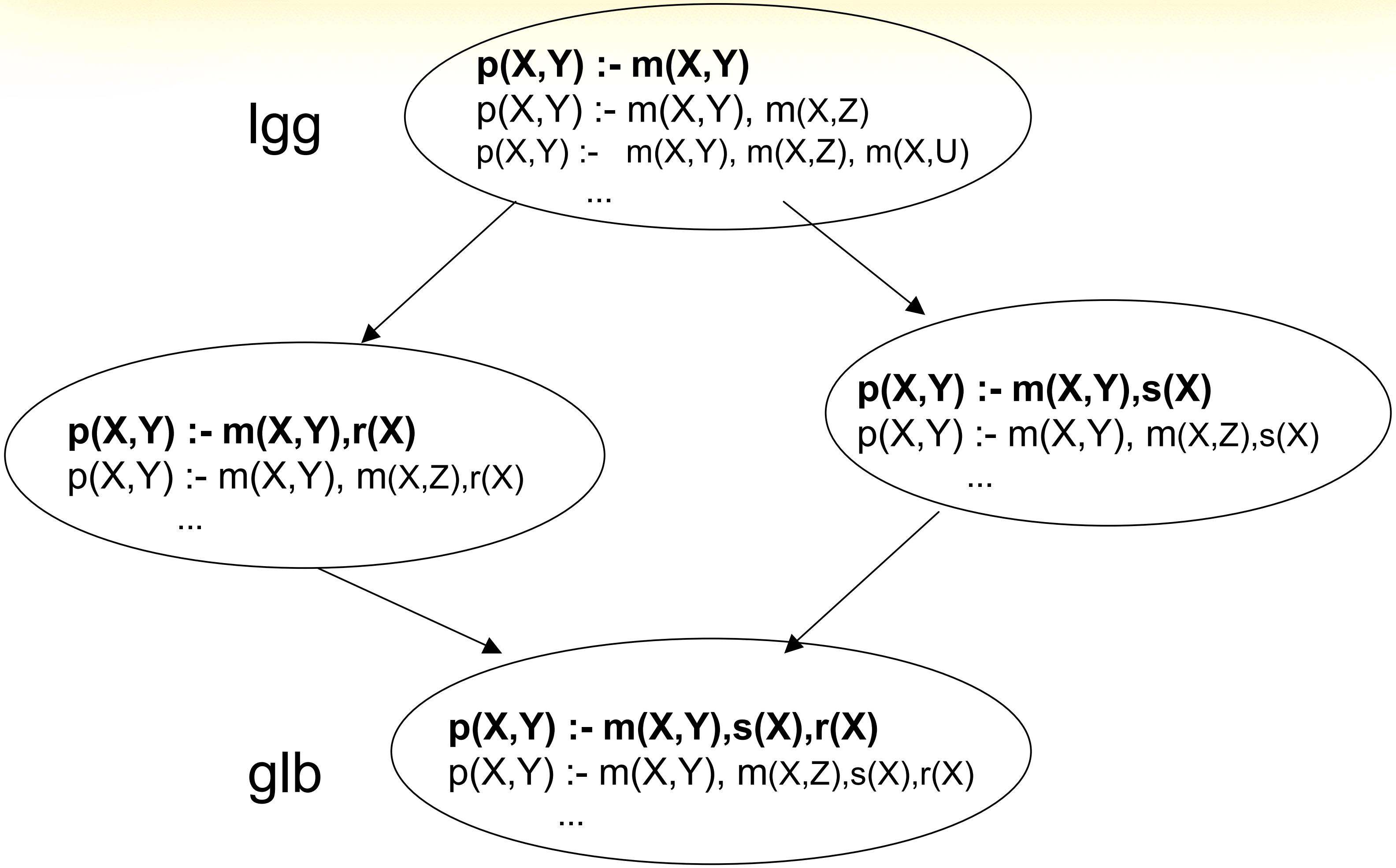
- d1 : p(X,Y) :- q(X,Y), q(Y,X)
- d2 : p(Z,Z) :- q(Z,Z)
- d3 : p(a,a) :- q(a,a)
- theta(1,2) : {X / Z, Y /Z}
- theta(2,3) : {Z/a}
- d1 is a generalization of d3
- Mapping several literals onto one leads (sometimes) to combinatorial problems

Properties

- Soundness :

  if c1 theta-subsumes c2 then c1 |= c2
- Complete (when only variables and constants)
- Decidable (but NP-complete)
- transitive and reflexive but not anti-symmetric
- Lattice at the level of equivalence classes

Structure


lgg
p(X,Y) :- m(X,Y)
p(X,Y) :- m(X,Y), m(X,Z)
p(X,Y) :- m(X,Y), m(X,Z), m(X,U)
...
p(X,Y) :- m(X,Y),r(X)
p(X,Y) :- m(X,Y), m(X,Z),r(X)
...
p(X,Y) :- m(X,Y),s(X)
p(X,Y) :- m(X,Y), m(X,Z),s(X)
...
glb
p(X,Y) :- m(X,Y),s(X),r(X)
p(X,Y) :- m(X,Y), m(X,Z),s(X),r(X)
...

Properties (2)

- Equivalence classes [c] :
  - parent(X,Y) :- mother(X,Y), mother(X,Z)
  - parent(X,Y) :- mother(X,Y)
- c1 is a reduced clause of c2 iff c1 minimal subset of literals of c2 that is equivalent with c2
  - parent(X,Y) :- mother(X,Y), mother(X,Z)
  - parent(X,Y) :- mother(X,Y) : reduced form
  - this gives an algorithm for reduction
  - reduced class = representative of equivalence class, unique up to variable renaming

Properties (3)

- Equivalence classes induce a lattice L
  - any two equivalence classes have least upper bound (least general generalization - lgg)
  - any two equivalence classes have greatest lower bound
- infinite descending and ascending chains exist, e.g.
  - :- p(X1,X2),p(X2,X1)
  - :- p(X1,X2),p(X2,X1), p(X1,X3),p(X3,X1),p(X2,X3),p(X3,X2)
  - :-{ p(Xi,Xj) for which i=\=j and i and j between 1 and n }
  - ….
  - :- p(X1,X1)

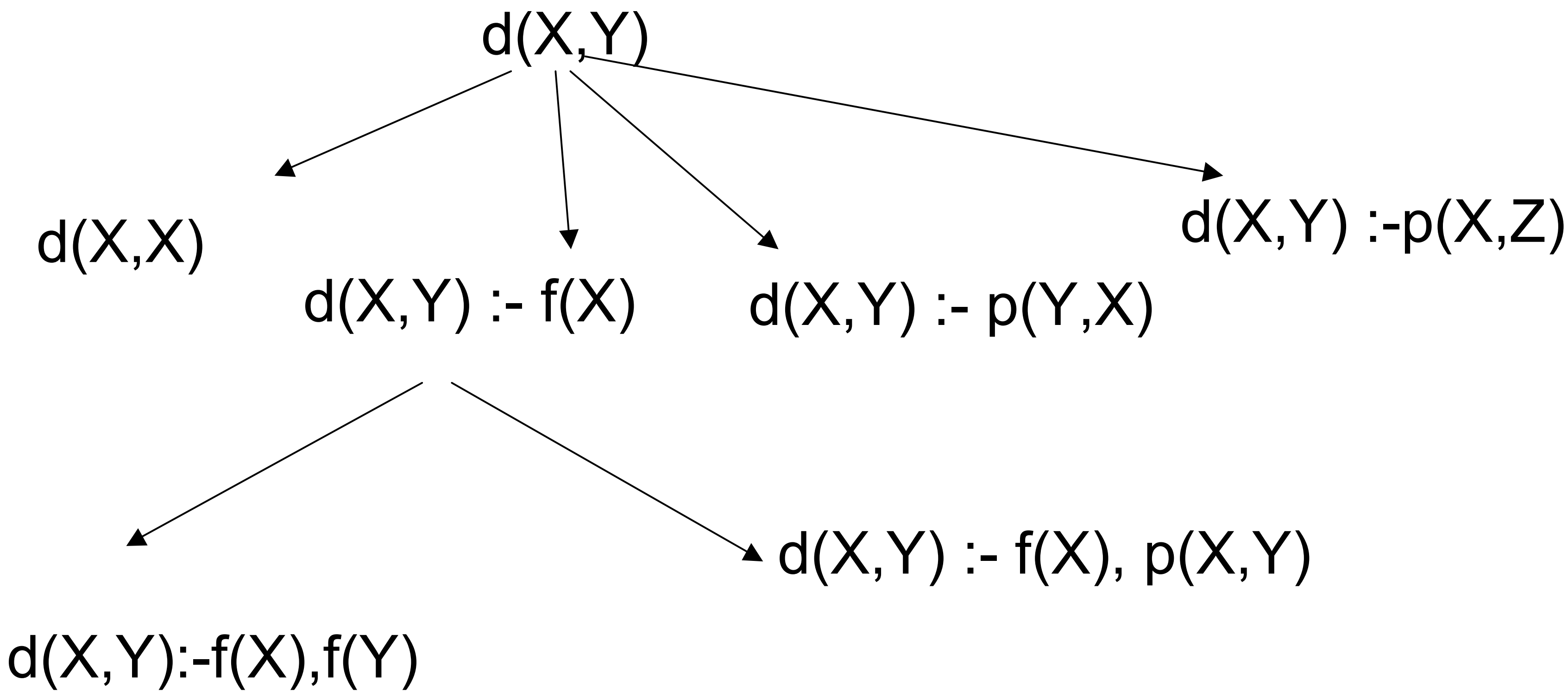
Lgg of clauses

- lgg of literals (= atoms or negated atoms) :
  lgg(atom1,atom2) = see above
  lgg(not atom1, not atom2) = not lgg(atom1, atom2)
  lgg(not atom1, atom2) = undefined
- lgg of clauses :
  lgg({l1, … lm}, {k1, … , kn}) = {lgg(li,kj) | lgg(li,kj) defined}
- f(t,a) :- p(t,a), m(t), f(a)
- f(j,p) :- p(j,p), m(j), m(p)
- lgg = f(X,Y) :- p(X,Y), m(X), m(Z)

Specialization Operators (for most practical purposes)

- Refinement operator (Shapiro) :
  - rho(c) = {c' | c' is a maximally general specialization of c } (theory)
  - rho(c) $\subseteq$ { c U {l} | l is literal } U {cθ | θ is a substitution } (practice)
  - rho(parent(X,Y)) includes :
    - parent(X,X)
    - parent(X,Y) :- male(X)
    - parent(X,Y) :- parent(Y,Z),
    - ….

d: daughter, p: parent, f: female, m : male



Basics of ILP cont'd

- Bottom-up approach of finding clauses leads to long clauses through *lgg.*
- Thus, prefer top-down approach since shorter and more general clauses are learned
- Two ways of doing top-down search
  - FOIL: greedy search using information gain to score
  - PROGOL: branch-and-bound, using *P-N-l* to score, uses saturation to restrict search space
- Usually, refinement operator is to
  - Apply substitution
  - Add literal to body of a clause

FOIL

- Greedy search, score each clause using information gain:
  - Let c_1 be a specialization of c_2
  - Then $WIG(c_1,c_2)$ (weighted information gain) is

$$WIG(c_1, c_2) = p_2^{\oplus\oplus}\left(I\left(c_1\right) - I\left(c_2\right)\right)$$

$$I(c) = -\log_2 \frac{p}{p+n}$$

  - Where $p^{\oplus\oplus}$ is the number of possible bindings that make the clause cover positive examples, p is the number of positive examples covered and n is the number of negative examples covered.
  - Background knowledge (B) is limited to ground facts.

PROGOL

- Branch and bound top-down search
- Uses *P-N-l* as scoring function:
  - *P* is number of positive examples covered
  - *N* is number of negative examples covered
  - *l* is the number of literals in the clause
- Preprocessing step: build a bottom clause using a positive example and *B* to restrict search space.
- Uses mode declarations to restrict language
- *B* not limited to ground facts
- While doing branch and bound top-down search:
  - Only use literals appearing in bottom clause to refine clauses.
  - Learned literal is a generalization of this bottom clause.
- Can set depth bound on variable chaining and theorem proving

Example of Bottom Clause

$$E^+ = \{p(a), p(b)\}$$
$$E^- = \{p(c), p(d)\}$$
$$B = \{r(a,b), r(a,c), r(c,d),$$
$$q(X,Y) \leftarrow r(Y,X)\}$$

modes: $p(\mathrm{var})$
$q(\mathrm{var}, \mathrm{var})$
$r(\mathrm{var}, const)$
$r(\mathrm{var}, \mathrm{var})$

- Select a seed from positive examples, *p(a)*, randomly or by order (first uncovered positive example)
- Gather all relevant literals (by forward chaining add anything from *B* that is allowable)

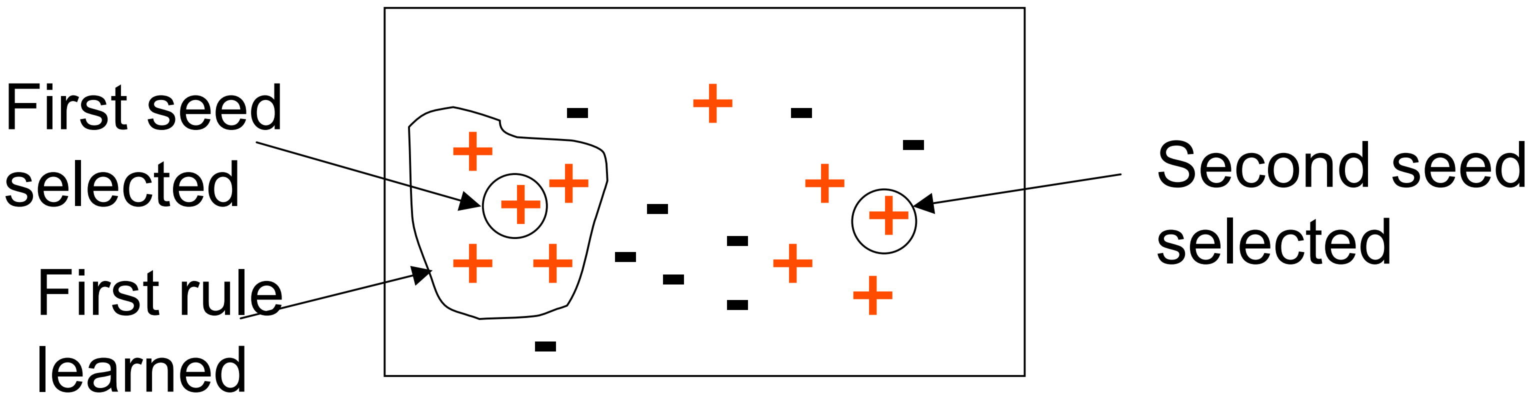
$$p(a) \leftarrow r(a,b), r(a,c), r(c,d), q(b,a),$$
$$q(c,a), q(d,c), r(a,b), r(a,c), r(c,d)$$

- Introduce variables as required by modes

$$p(A) \leftarrow r(A,b), r(A,c), r(C,d), q(B,A),$$
$$q(C,A), q(D,C), r(A,B), r(A,C), r(C,D)$$

Iterate to Learn Multiple Rules

- Select seed from positive examples to build bottom clause.
- Get some rule “If A $\wedge$ B then P”. Now throw away all positive examples that were covered by this rule
- Repeat until there are no more positive examples.



Repetition

- Why ILP is not just Decision Trees.
  - Language is First-Order Logic
    - Natural representation for multi-relational settings
    - Thus, a natural representation for *full* databases
  - Not restricted to the classification task.
  - So then, what is ILP?

What is ILP?
(An obscene generalization)

- A way to search the space of First-Order clauses.
  - With restrictions of course
  - θ-subsumption and search space ordering
  - Refinement operators:
    - Applying substitutions
    - Adding literals
    - Chaining variables

More From Before

- Evaluation of hypothesis requires finding substitutions for each example.
  - This requires a call to PROLOG for *each* example
  - For PROGOL only one substitution required
  - For FOIL *all* substitutions are required (recall the $p^{\oplus\oplus}$ in scoring function)

Scaling in Data Mining

- Scaling to large datasets
  - Increasing number of training examples
- Scaling to size of the examples
  - Increasing number of ground facts in background knowledge

[Tang, Mooney, Melville, UT Austin, MRDM (SIGKDD) 2003.]

Efficiency Issues

- Representational Aspects
- Search
- Evaluation
- Sharing computations
- Memory-wise scalability

Representational Aspects

- Example:
  - Student(string sname, string major, string minor)
  - Course(string cname, string prof, string cred)
  - Enrolled(string sname, string cname)
- In a natural join of these tables there is a one-to-one correspondance between join result and the Enrolled table
- Data mining tasks on the Enrolled table are really propositional
- MRDM is overkill

Representational Aspects

- Three settings for data mining:
  - Find patterns within individuals represented as tuples (single table, propositional)
    - eg. Which minor is chosen with what major
  - Find patterns within individuals represented as sets of tuples (each individual ‘induces’ a sub-database)
    - Multiple tables, restricted to some individual
    - eg. Student X taking course A, usually takes course B
  - Find patters within whole database
    - Mutliple tables
    - eg. Course taken by student A are also taken by student B

Search

- Space restriction
  - Bottom clauses as seen above
- Syntactical biases and typed logic
  - Modes as seen above.
  - Can add types to variables to further restrict language
- Search biases and pruning rules
  - PROGOL's bound (relies on anti-monotonicity of coverage)
- Stochastic search

Evaluation

- Evaluating a clause: get some measure of coverage
  - Match each example to the clause:
    - Run multiple logical queries.
  - Query optimization methods from DB community
    - Rel. Algebra operator reordering
    - BUT: queries for DB are set oriented (bottom-up), queries in PROLOG find a single solution (top-down).

Evaluation

- More options
  - *k*-locality, given some bindings, literals in a clause become independent:
    - eg. *?- p(X,Y),q(Y,Z),r(Y,U).*
    - Given a binding for *Y*, proofs of *q* and *r* are independent
    - So, find only one solution for *q*, if no solution found for *r* no need to backtrack.
  - Relax θ-subsumption using stochastic estimates
    - Sample space of substitutions and decide on subsumption based on this sample

Sharing Computations

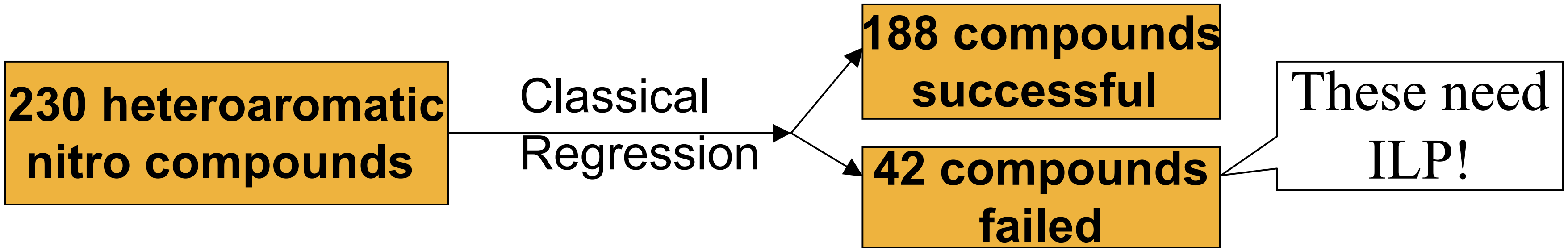
- Materialization of features
- Propositionalization
- Pre-compute some statistics
  - Joint distribution over attributes of a table
  - Query selectivity
- Store proofs, reuse when evaluating new clauses

Memory-wise scalability

- All full ILP systems work on memory databases
  - Exception: TILDE: learns multi-relational decision trees
    - The trick: make example loop the outer loop
- Current solution:
  - Encode data compactly

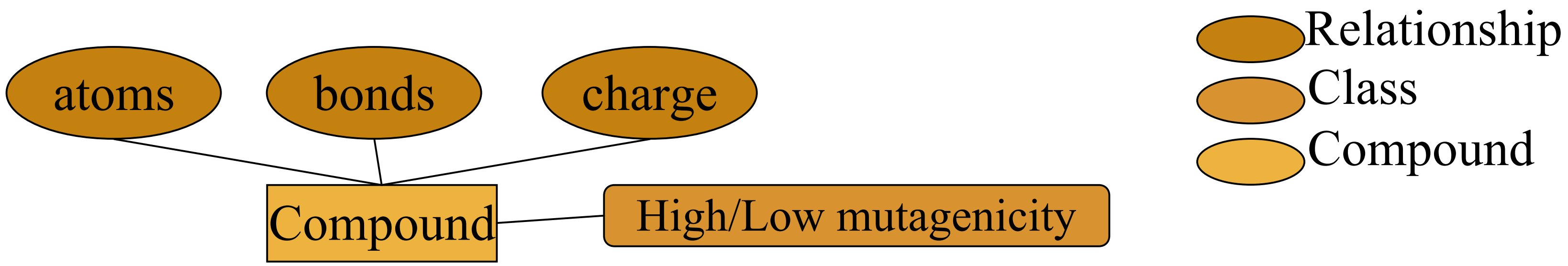
Example Application: Predicting the Mutagenicity of Chemical Compounds(1)

- SAR(Structure/Activity) Relationship of chemical compounds
- Traditional Way - Attribute-based
  - using global attributes of a molecule
    - EX) LUMO(Energy of the Lowest Molecular Orbital)
  - can't use the patterns in the molecule structure



Example Application: Predicting the Mutagenicity of Chemical Compounds(2)

– Using Progol - A ILP System



- Background Knowledge
  - Classified Results of 230 Compounds
  - 18300 Prolog facts
  - LUMO(Energy of Lowest Unoccupied Molecular Orbital)
    - only for the 188 compounds amenable to regression

Example Application: Predicting the Mutagenicity of Chemical Compounds(3)

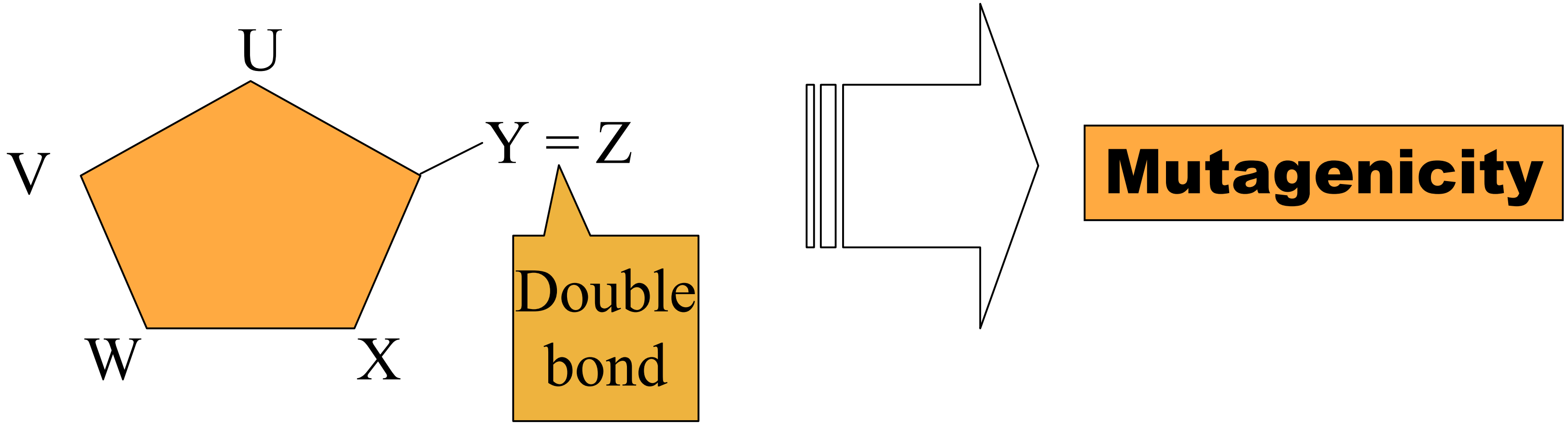
- Result from Progol
  - For 188 Compounds
    - EX) Mutagenic if it has a RUMO value < -1.937
    - 89% Accuracy(matching accuracy of regression)
      - but easy to comrehend and automatically generated
  - The Remaining 42 Compounds
    - 1 Rule with accuracy 88%
      - regression : 62%

Example Application: Predicting the Mutagenicity of Chemical Compounds(4)

- Good Prediction Accuracy
- New Chemical Insight

Discovery of New Structural Feature!

Five-membered aromatic ring with a nitrogen atom



Conclusion

– ILP

Difficult, industrially or scientifically relevant

NOT satisfactorily solved

Relational Background > Attribute-value form

Major Advantage

Background Knowledge in the form of General P rogram like Prolog

Major Obstacle

Inefficiency
- handling numerical data