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

Class 1
Class 2
Class 3
Class 4
Example 2: decision tree

if $X \leq x_1$ then ♦
else if $Y \leq y_1$ then ■
else if $Y \leq y_2$ 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*

\[\text{not yet refuted} = \text{Version Space}\]

- too general (covering *non-instances*)
- too specific (not covering *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, \ldots, D\}
  - covering approach (but stopping criteria relaxed)
  - unordered rules: rule $\text{Class IF Conditions}$ is learned by first determining $\text{Class}$ and then $\text{Conditions}$
  - ordered rules: rule $\text{Class IF Conditions}$ is learned by first determining $\text{Conditions}$ and then $\text{Class}$
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}
for each class $C_i$ do

- $E_i := P_i \cup N_i$ ($P_i$ positive, $N_i$ negative)
- $\text{RuleSet}(C_i) := \text{empty}$

repeat \{find-set-of-rules\}

- find-one-rule $R$ covering some positive examples and no negatives
- add $R$ to $\text{RuleSet}(C_i)$
- delete from $P_i$ all positive examples covered by $R$

until $P_i = \text{empty}$
for each class $C_i$ do

- $E_i := P_i U N_i$, RuleSet($C_i$) := empty

repeat {find-set-of-rules}

- $R := \text{Class} = C_i \text{ IF } \text{Conditions}$, Conditions := true

repeat {learn-one-rule}

- $R' := \text{Class} = C_i \text{ IF } \text{Conditions AND Cond}$
  (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 C₁, ..., Cₙ 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

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
Learn-one-rule as heuristic search

1. Play tennis = yes  IF  true  [9+,5−] (14)

2. Play tennis = yes  IF  Wind=weak  [6+,2−] (8)

3. Play tennis = yes  IF  Wind=strong  [3+,3−] (6)

4. Play tennis = yes  IF  Humidity=normal  [6+,1−] (7)

5. Play tennis = yes  IF  Humidity=high  [3+,4−] (7)

6. Play tennis = yes  IF  Humidity=normal, Wind=weak
7. Play tennis = yes  IF  Humidity=normal, Wind=strong
8. Play tennis = yes  IF  Humidity=normal, Outlook=sunny  [2+,0−] (2)
9. Play tennis = yes  IF  Humidity=normal, Outlook=rain
Heuristics for learn-one-rule

- Evaluating accuracy of a rule:
  \[ A(C_i \text{ IF Conditions}) = p(C_i | \text{Conditions}) \]

- Estimating probability with relative frequency:
  covered positives / covered examples
  
  \[ [6+,1-] (7) = \frac{6}{7}, \quad [2+,0-] (2) = \frac{2}{2} = 1 \]
Probability estimates

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

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

- 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 $m$)

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

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

\[
p(+ | R) = \frac{n^+(R) + m \cdot p_a( +)}{n(R) + m}
\]
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 $C_1, \ldots, C_n$ 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

1. RuleList := empty; $E_{cur} := E$
2. repeat
   1. learn-one-rule R
   2. RuleList := RuleList ++ R
   3. $E_{cur} := E_{cur} - \{\text{all examples covered by } R\}$
3. until performance(R, $E_{cur}$) < ThresholdR
4. RuleList := sort RuleList by performance(R, E)
5. 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:
  - database is *fixed-size vector of values*
  - features are those given in the dataset

- **First-order** representations:
  - database is *flexible-size, structured object*
    - sequence, set, graph
    - hierarchical: e.g. set of sequences
  - features need to be *selected* from potentially infinite set
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.
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:**
  \[ \forall X: \text{bachelor}(X) \leftrightarrow \text{male}(X) \land \text{adult}(X) \land \neg \text{married}(X) \]

- **clausal logic:**
  
  \[
  \text{bachelor}(X);\text{married}(X) :\neg \text{male}(X),\text{adult}(X).
  \]

  \[
  \text{male}(X) :\neg \text{bachelor}(X).
  \]

  \[
  \text{adult}(X) :\neg \text{bachelor}(X).
  \]

  \[
  :\neg \text{bachelor}(X),\text{married}(X).
  \]

  **indefinite clause**

  **definite (Horn) clauses**

  **denial**
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

<table>
<thead>
<tr>
<th>example</th>
<th>action</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>+p(b, [b])</td>
<td>add clause</td>
<td>p(X,Y).</td>
</tr>
<tr>
<td>-p(x, [])</td>
<td>specialise</td>
<td>p(X, [V</td>
</tr>
<tr>
<td>-p(x, [a,b])</td>
<td>specialise</td>
<td>p(X, [X</td>
</tr>
<tr>
<td>+p(b, [a,b])</td>
<td>add clause</td>
<td>p(X, [X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p(X, [V</td>
</tr>
</tbody>
</table>
**Bottom-up induction: example**

- Treat positive examples + ground background facts as **body**

- Choose two examples as **heads** and **anti-unify**

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

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

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

- **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

1. TRAINS GOING EAST

1. 

2. 

3. 

4. 

5. 

2. TRAINS GOING WEST

1. 

2. 

3. 

4. 

5.
Example:

\[
\text{eastbound}(t_1).
\]

Background theory:

\[
\begin{align*}
\text{car}(t_1, c_1). & & \text{car}(t_1, c_2). & & \text{car}(t_1, c_3). & & \text{car}(t_1, c_4). \\
\text{rectangle}(c_1). & & \text{rectangle}(c_2). & & \text{rectangle}(c_3). & & \text{rectangle}(c_4). \\
\text{short}(c_1). & & \text{long}(c_2). & & \text{short}(c_3). & & \text{long}(c_4). \\
\text{none}(c_1). & & \text{none}(c_2). & & \text{peaked}(c_3). & & \text{none}(c_4). \\
\text{two}_\text{wheels}(c_1). & & \text{three}_\text{wheels}(c_2). & & \text{two}_\text{wheels}(c_3). & & \text{two}_\text{wheels}(c_4). \\
\text{load}(c_1, l_1). & & \text{load}(c_2, l_2). & & \text{load}(c_3, l_3). & & \text{load}(c_4, l_4). \\
\text{circle}(l_1). & & \text{hexagon}(l_2). & & \text{triangle}(l_3). & & \text{rectangle}(l_4). \\
\text{one}_\text{load}(l_1). & & \text{one}_\text{load}(l_2). & & \text{one}_\text{load}(l_3). & & \text{three}_\text{loads}(l_4).
\end{align*}
\]

Hypothesis:

\[
\text{eastbound}(T): \neg \text{car}(T, C), \text{short}(C), \neg \text{none}(C).
\]
Example:
\[
\text{eastbound([c(rectangle,short,none,2,1(circle,1)), c(rectangle,long,none,3,1(hexagon,1)), c(rectangle,short,peaked,2,1(triangle,1)), c(rectangle,long,none,2,1(rectangle,3))])}.
\]

Background theory: empty

Hypothesis:
\[
\text{eastbound}(T) :- \text{member}(C,T), \text{arg}(2,C,\text{short}), \neg \text{arg}(3,C,\text{none}).
\]
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,Peked,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

<table>
<thead>
<tr>
<th>LOAD</th>
<th>CAR</th>
<th>OBJECT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>c1</td>
<td>circle</td>
<td>1</td>
</tr>
<tr>
<td>l2</td>
<td>c2</td>
<td>hexagon</td>
<td>1</td>
</tr>
<tr>
<td>l3</td>
<td>c3</td>
<td>triangle</td>
<td>1</td>
</tr>
<tr>
<td>l4</td>
<td>c4</td>
<td>rectangle</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### CAR_TABLE

<table>
<thead>
<tr>
<th>CAR</th>
<th>TRAIN</th>
<th>SHAPE</th>
<th>LENGTH</th>
<th>ROOF</th>
<th>WHEELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>c2</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>3</td>
</tr>
<tr>
<td>c3</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>peaked</td>
<td>2</td>
</tr>
<tr>
<td>c4</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### TRAIN_TABLE

<table>
<thead>
<tr>
<th>TRAIN</th>
<th>EASTBOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>TRUE</td>
</tr>
<tr>
<td>t2</td>
<td>TRUE</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t6</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

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
The set of first-order terms is a lattice:

- $t_1$ is more general than $t_2$ iff for some substitution $\theta$: $t_1\theta = t_2$

- $\text{lub} \Rightarrow \text{unification}$, $\text{glb} \Rightarrow \text{anti-unification}$

- Specialisation $\Rightarrow$ applying a substitution

- Generalisation $\Rightarrow$ applying an inverse substitution
The set of (equivalence classes of) clauses is a **lattice**:

- $C_1$ is more general than $C_2$ iff for some substitution $\theta$: $C_1 \theta \subseteq C_2$
- $\text{lub} \Rightarrow \theta\text{-MGS,} \ \text{glb} \Rightarrow \theta\text{-LGG}$
- 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!
$\theta$-LGG: examples

\begin{align*}
a([1,2],[3,4],[1,2,3,4]) & : -a([2],[3,4],[2,3,4]) \\
a([a],[],[a]) & : -a([],[],[])
\end{align*}

\begin{align*}
a([A|B],C,[A|D]) & : -a(B,C,D)
\end{align*}

\begin{align*}
m(c,[a,b,c]) & : -m(c,[b,c]),m(c,[c]) \\
m(a,[a,b]) & : -m(a,[a])
\end{align*}

\begin{align*}
m(P,[a,b|Q]) & : -m(P,[R|Q]),m(P,[P])
\end{align*}
Logical implication is **strictly stronger** than \( \theta \)-subsumption

- e.g. \( p([V|W]) : \neg p(W) \neq p([X,Y|Z]) : \neg p(Z) \)
- this happens when the resolution derivation requires the left-hand clause more than once

i-LGG of definite clauses is **not unique**

- \( \text{i-LGG}(p([A,B|C]) : \neg p(C), p([P,Q,R|S]) : \neg p(S)) = \{ p([X|Y]) : \neg p(Y), p([X,Y|Z]) : \neg p(V) \} \)

Logical implication between clauses is undecidable, \( \theta \)-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

- $\theta$-RLGG($C_1, C_2, B$) = $\theta$-LGG($C_{1B}, C_{2B}$)
\( \theta\text{-RLGG: example} \)

\[
\begin{align*}
\text{a([1,2],[3,4],[1,2,3,4])} :&= \\
&\text{a([1,2],[3,4],[1,2,3,4]), a([a],[a]),} \\
&\text{a([],[],[]), a([2],[3,4],[2,3,4]).}
\end{align*}
\]

\[
\begin{align*}
\text{a([a],[a])} :&= \\
&\text{a([1,2],[3,4],[1,2,3,4]), a([a],[a]),} \\
&\text{a([],[],[]), a([2],[3,4],[2,3,4]).}
\end{align*}
\]

\[
\begin{align*}
\text{a([A|B],C,[A|D])} :&= \\
&\text{a([1,2],[3,4],[1,2,3,4]), a([A|B],C,[A|D]), a(E,C,F),} \\
&\text{a([G|B],[3,4],[G,H,I|J]),} \\
&\text{a([K|L,M,[K|N]), a([a],[a],O],O),} \\
&\text{a([P],M,[P|M]),} \\
&\text{a(Q,M,R), a(S,[,S), a([],[],[]), a(L,M,N),} \\
&\text{a([T|L],[3,4],[T,U,V|W]), a(X,C,[X|C]), a(B,C,D),} \\
&\text{a([2],[3,4],[2,3,4]).}
\end{align*}
\]
\[\theta\text{-RLGG: example}\]

\[
\begin{align*}
\text{a}([1,2],[3,4],[1,2,3,4]) & : - \\
\text{a}([1,2],[3,4],[1,2,3,4]), & \text{a}([a],[],[]), \\
\text{a}([],[],[]) & , \text{a}([2],[3,4],[2,3,4]).
\end{align*}
\]

\[
\begin{align*}
\text{a}([a],[],[],[]) & : - \\
\text{a}([1,2],[3,4],[1,2,3,4]), & \text{a}([a],[],[]), \\
\text{a}([],[],[]) & , \text{a}([2],[3,4],[2,3,4]).
\end{align*}
\]

\[
\begin{align*}
\text{a}([A|B],C,[A|D]) & : - \\
\text{a}([1,2],[3,4],[1,2,3,4]), & \text{a}([A|B],C,[A|D]), \text{a}(E,C,F), \\
\text{a}([G|B],[3,4],[G,H,I|J]) & , \text{a}([],[],[]), \text{a}(O,[],O), \\
\text{a}([P],M,[P|M]) & , \text{a}(Q,M,R), \text{a}(S,[],S), \text{a}([],[],[]), \text{a}(L,M,N), \\
\text{a}([T|L],[3,4],[T,U,V|W]) & , \text{a}(X,C,[X|C]), \text{a}(B,C,D), \\
\text{a}([2],[3,4],[2,3,4]).
\end{align*}
\]
\( \theta \text{-RLGG: example} \)

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

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

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

The following rule has two features ‘**has a short car**’ and ‘**has a closed car**’:

```
estbound(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**’:

```
estbound(T) :- hasCar(T,C), clength(C, short),
               not croof(C, none).
```
Propositionalising rules

- Equivalently:

  eastbound(T):=\text{hasShortCar}(T),\text{hasClosedCar}(T).

  \text{hasShortCar}(T):=\text{hasCar}(T,C),\text{clength}(C,\text{short}).

  \text{hasClosedCar}(T):=\text{hasCar}(T,C),\text{not croof}(C,\text{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
Type definitions:

\[
data \text{ Outlook} = \text{ Sunny} \mid \text{ Overcast} \mid \text{ Rain};
\]
\[
data \text{ Temperature} = \text{ Hot} \mid \text{ Mild} \mid \text{ Cool};
\]
\[
data \text{ Humidity} = \text{ High} \mid \text{ Normal} \mid \text{ Low};
\]
\[
data \text{ Wind} = \text{ Strong} \mid \text{ Medium} \mid \text{ Weak};
\]
\[
type \text{ Weather} = (\text{ Outlook}, \text{ Temperature}, \text{ Humidity}, \text{ Wind})
\]
\[
\text{playTennis} :: \text{ Weather} \to \text{ Bool};
\]

Examples:

\[
\text{playTennis(Overcast, Hot, High, Weak)} = \text{ True};
\]
\[
\text{playTennis(Sunny, Hot, High, Weak)} = \text{ False};
\]
Attribute-value learning in Escher

Hypothesis:

\[
\text{playTennis}(w) = \\
\begin{cases} 
\text{False} & \text{if (outlookP}(w) == \text{Sunny} \&\& \text{humidityP}(w) == \text{High}) \\
\text{False} & \text{if (outlookP}(w) == \text{Rain} \&\& \text{windP}(w) == \text{Strong}) \\
\text{True} & \text{else}
\end{cases}
\]

---

Hypothesis:

\[
\text{playTennis}(w) = \\
\begin{cases} 
\text{False} & \text{if (outlookP}(w) == \text{Sunny} \&\& \text{humidityP}(w) == \text{High}) \\
\text{False} & \text{if (outlookP}(w) == \text{Rain} \&\& \text{windP}(w) == \text{Strong}) \\
\text{True} & \text{else}
\end{cases}
\]

---

\[
\text{outlookP}::\text{Weather} \rightarrow \text{Outlook}; \\
\text{outlookP}(o,t,h,w) = o;
\]
Hypothesis:

\[
\text{playTennis}(w) =
\begin{cases}
  \text{False} & \text{if } (\text{outlookP}(w) == \text{Sunny} \land \text{humidityP}(w) == \text{High}) \\
  \text{False} & \text{else if } (\text{outlookP}(w) == \text{Rain} \land \text{windP}(w) == \text{Strong}) \\
  \text{True} & \text{else}
\end{cases}
\]

\[
\text{outlookP} :: \text{Weather} \to \text{Outlook};
\]

\[
\text{outlookP}(o,t,h,w) = o;
\]
Attribute-value learning in Escher

**Hypothesis:**

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

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

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

- **Examples:**
  
  \[
  \text{class}\left(\{(\text{In}(\text{Circle},\text{Triangle}),1)\}\right) = \text{Positive};
  \]
  
  \[
  \text{class}\left(\{(\text{Triangle},1), (\text{In}(\text{Circle},\text{Triangle}),1)\}\right) = \text{Positive};
  \]
  
  \[
  \text{class}\left(\{(\text{In}(\text{Triangle},\text{Circle}),1), (\text{Triangle},1)\}\right) = \text{Negative};
  \]

- **Hypothesis:**

  \[
  \text{class}(d) =
  \begin{align*}
  &\text{if } (\exists p \rightarrow p \text{ 'in' } d \land (\exists s t \rightarrow \\
  &\hspace{1cm} \text{shapeP}(p) = \text{In}(s,t) \land s = \text{Circle})) \\
  &\text{then Positive else Negative};
  \end{align*}
  \]
Examples:

\[
\text{class}([\{(\text{In}(\text{Circle}, \text{Triangle}), 1)\}]) = \text{Positive};
\]

\[
\text{class}([\{(\text{Triangle}, 1), (\text{In}(\text{Circle}, \text{Triangle}), 1)\}]) = \text{Positive};
\]

\[
\text{class}([\{(\text{In}(\text{Triangle}, \text{Circle}), 1), (\text{Triangle}, 1)\}]) = \text{Negative};
\]

Hypothesis:

\[
\text{class}(d) =
\begin{align*}
&\text{if (exists } p \rightarrow p \ 'in' \ d \ &\& (\text{exists } s \ t \rightarrow \\
&\quad \\text{shapeP}(p) == \text{In}(s,t) \ &\& s == \text{Circle})) \\
&\quad \text{then Positive else Negative;}
\end{align*}
\]
Type definitions:

```haskell
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:

```plaintext
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;
```

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


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/