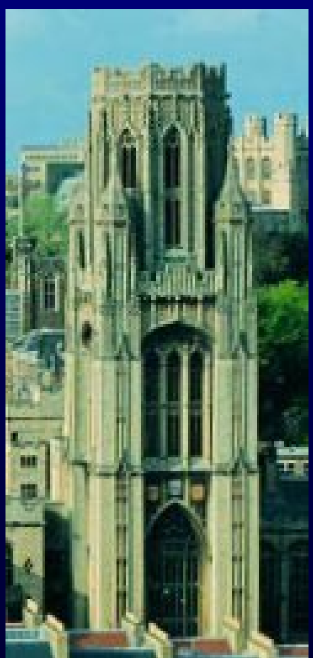
The logic of learning:
logic and knowledge representation in machine learning



Peter A. Flach

Department of Computer Science

University of Bristol

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

Overview of this talk

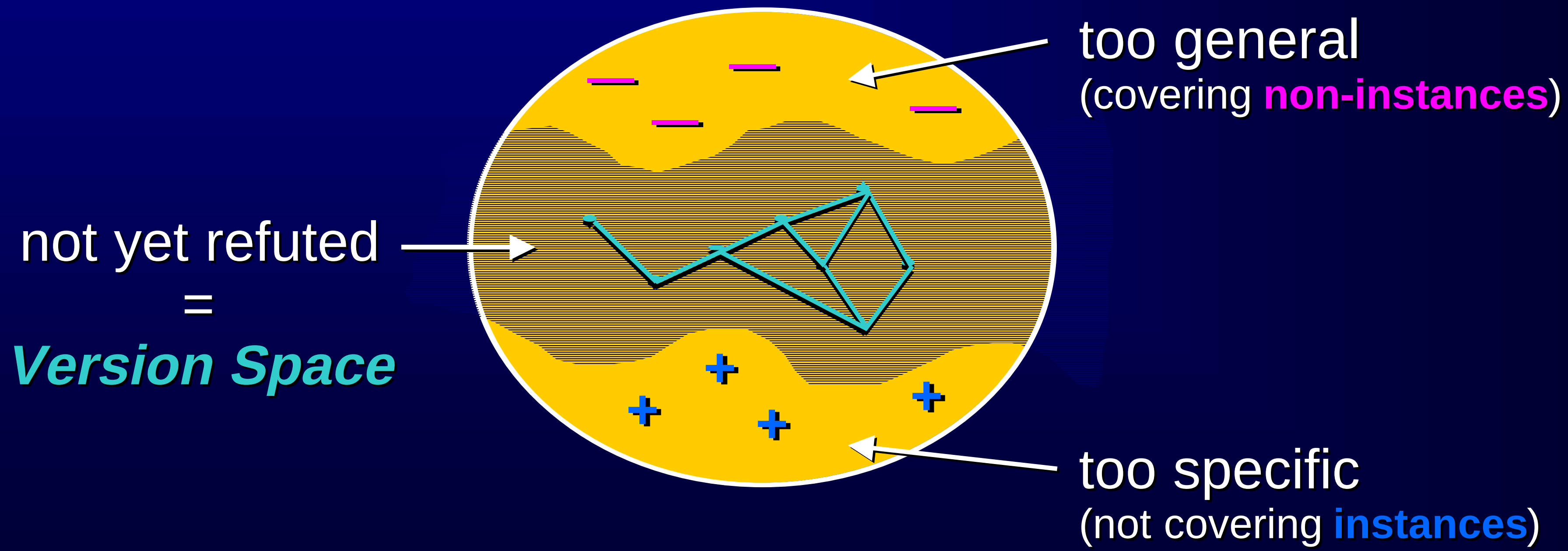
- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook

Overview of this talk

- **A (very) quick overview of ILP**
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook

Inductive concept learning

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



Concept learning in logic

- **Given**:
  - ***positive examples P***: facts to be entailed,
  - ***negative examples N***: facts not to be entailed,
  - ***background knowledge 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***).

ILP methods

- **top-down** (language-driven)
  - **descend** the generality ordering
    - start with **short, general rule**
  - **specialise** by
    - substituting variables
    - adding conditions

- **bottom-up** (data-driven)
  - **climb** the generality ordering
    - start with **long, specific rule**
  - **generalise** by
    - introducing variables
    - removing conditions

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**

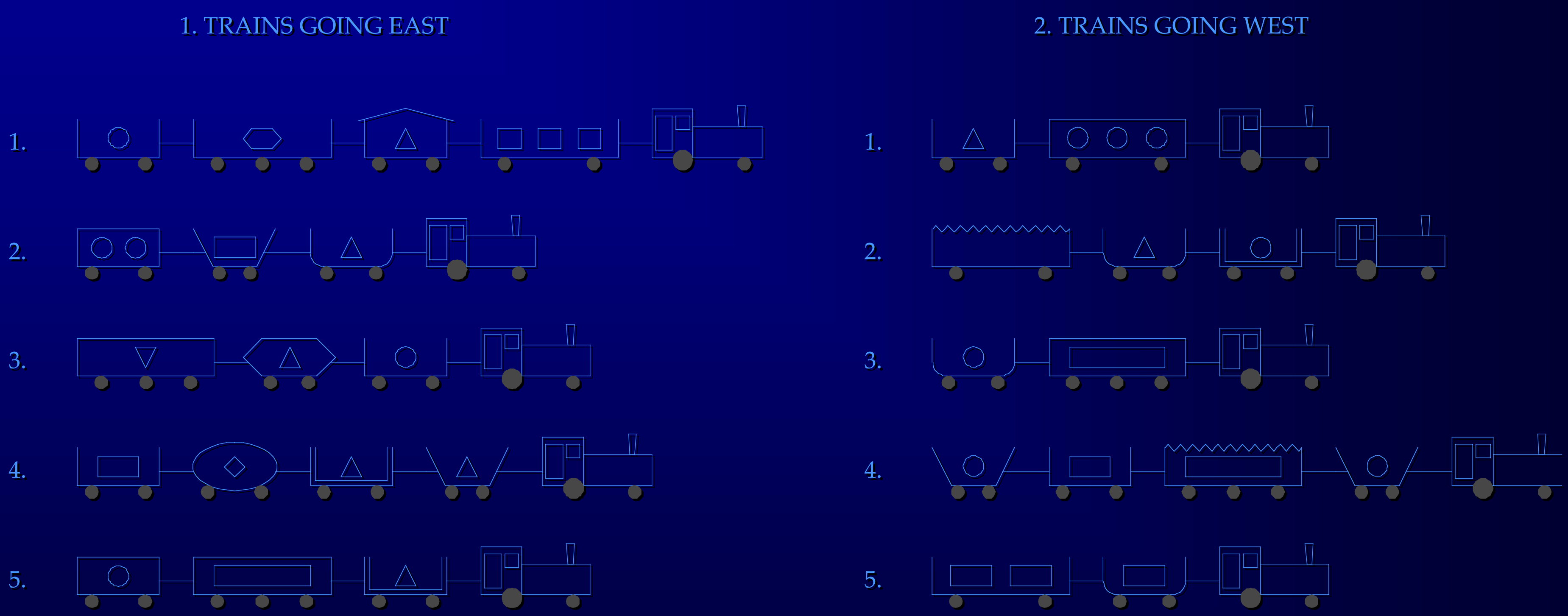
```
q([1,2],[3,4],[1,2,3,4]):-
   q([1,2],[3,4],[1,2,3,4]),q([a],[],[a]),q([],[],[]),q([2],[3,4],[2,3,4])

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

Progol 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 $\geq$ 0.168; or
  - (7) it has an hydroxy oxygen with a partial charge $\geq$ -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 $\leq$ -0.144 and tests positive on Progol's mutagenicity rules.

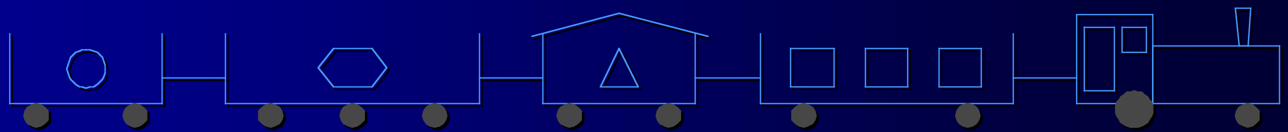
ILP example: East-West trains



Prolog representation (flattened)

- Example:

  `eastbound(t1).`



- Background knowledge:

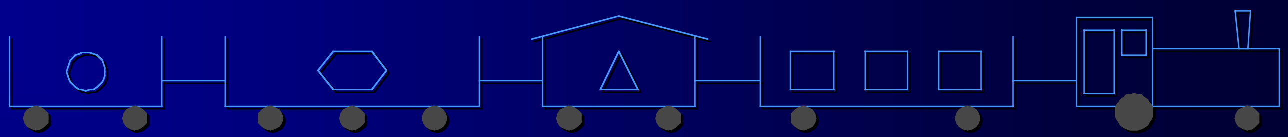
```
car(t1,c1).       car(t1,c2).         car(t1,c3).       car(t1,c4).
rectangle(c1).    rectangle(c2).      rectangle(c3).    rectangle(c4).
short(c1).        long(c2).           short(c3).        long(c4).
open(c1).         open(c2).           peaked(c3).       open(c4).
two_wheels(c1).   three_wheels(c2).   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 open(C).`

Prolog representation (flattened)

- Example:
  `eastbound(t1).`

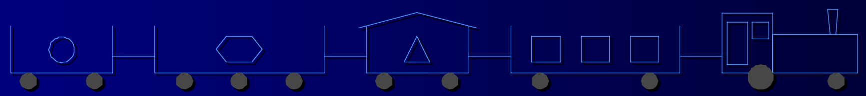


- Background knowledge:

```
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).
open(c1).          open(c2).           peaked(c3).        open(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).
```

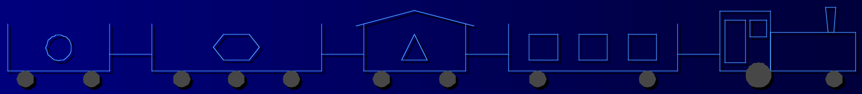
- Hypothesis:
  `eastbound(T):-car(T,C),short(C),not open(C).`

Prolog representation (terms)



- Example:
```
eastbound([c(rectangle,short,open,2,l(circle,1)),
           c(rectangle,long,open,3,l(hexagon,1)),
           c(rectangle,short,peaked,2,l(triangle,1)),
           c(rectangle,long,open,2,l(rectangle,3))]).
```
- Background knowledge: `member/2`, `arg/3`
- Hypothesis:
```
eastbound(T):-member(C,T),arg(2,C,short),
                          not arg(3,C,open).
```

Prolog representation (terms)



- Example:

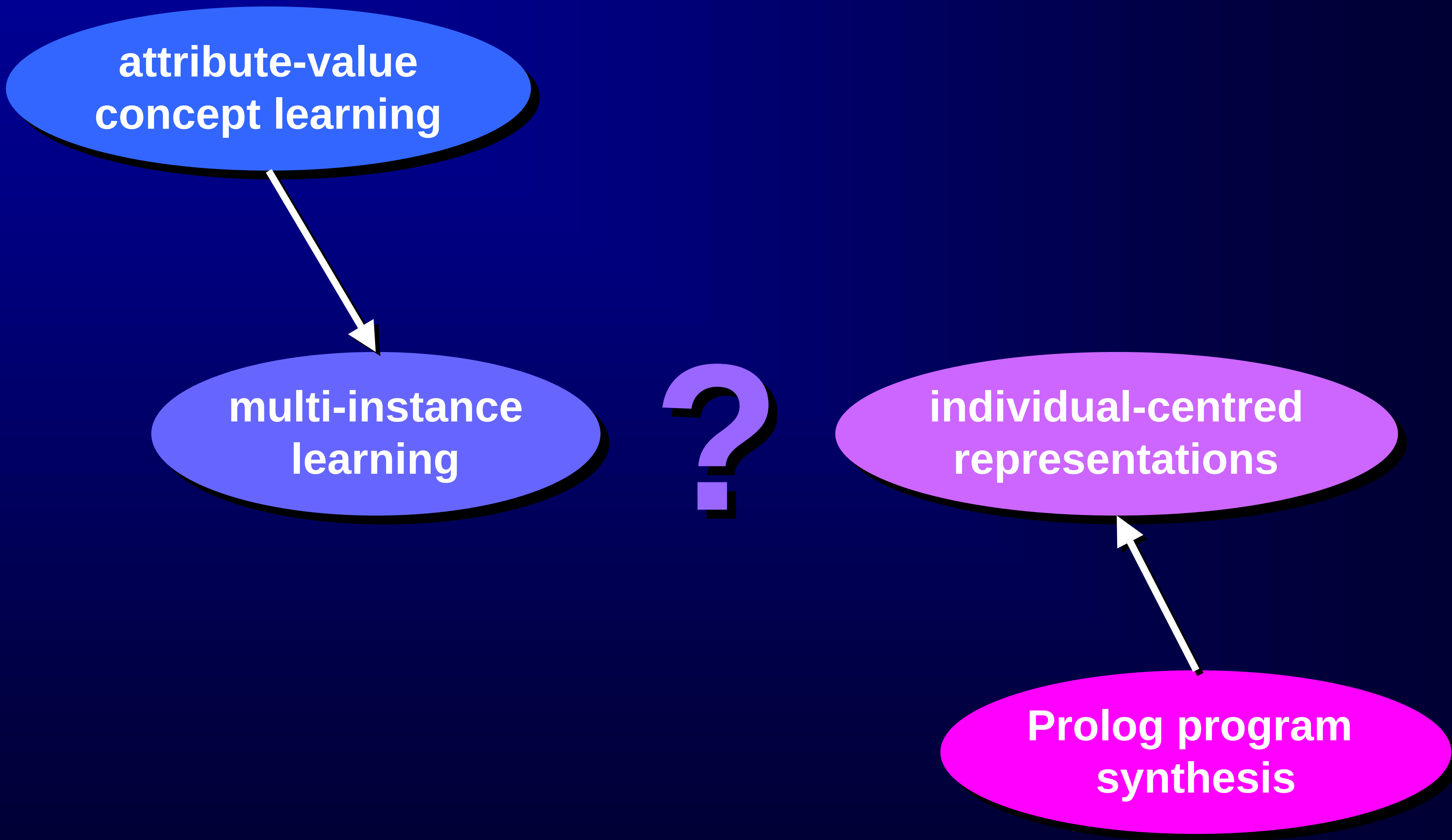
```
eastbound([c(rectangle,short,open,2,l(circle,1)),
           c(rectangle,long,open,3,l(hexagon,1)),
           c(rectangle,short,peaked,2,l(triangle,1)),
           c(rectangle,long,open,2,l(rectangle,3))]).
```

- Background knowledge: `member/2`, `arg/3`

- Hypothesis:

```
eastbound(T):-member(C,T),arg(2,C,short),
                          not arg(3,C,open).
```

Machine learning vs. ILP



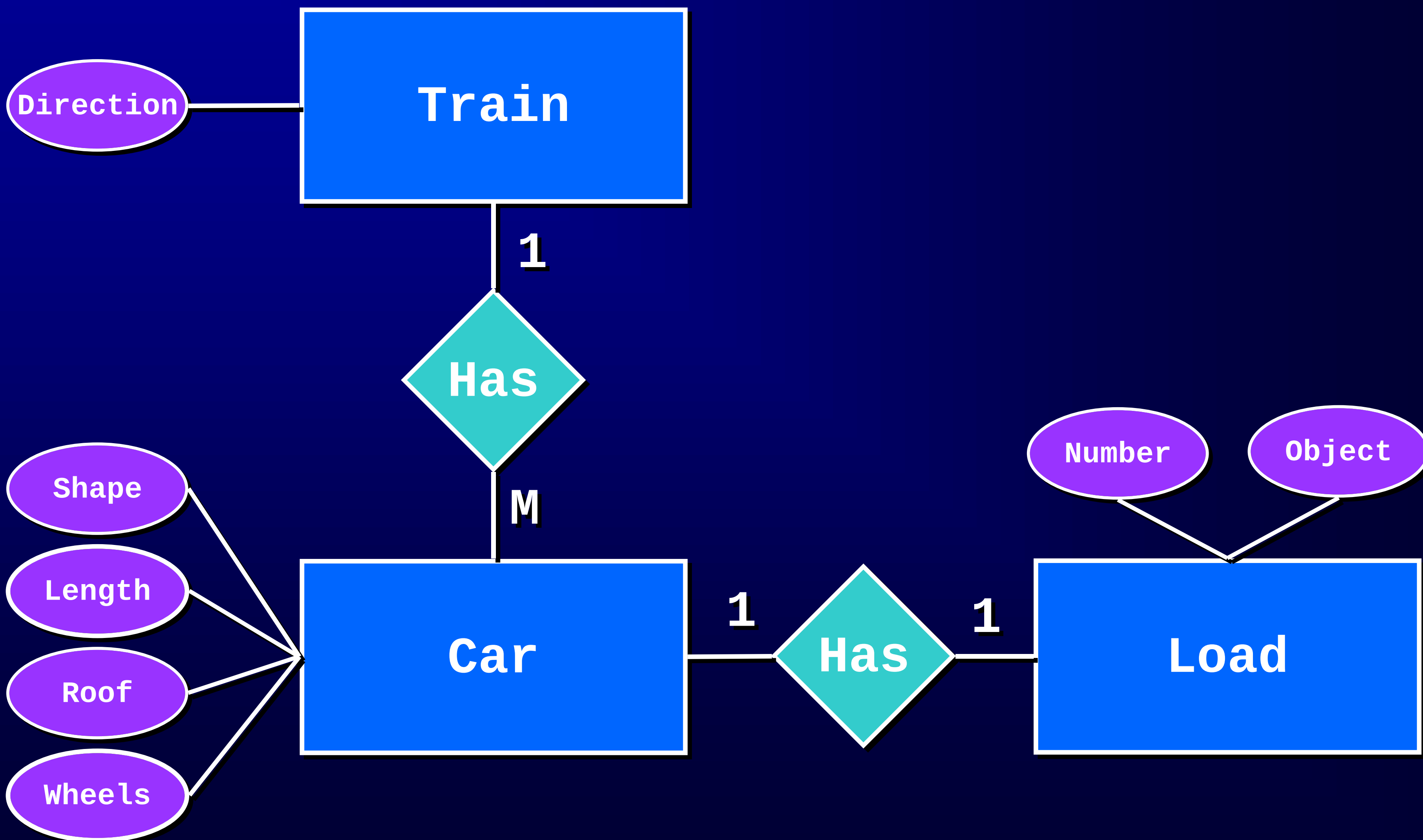
Overview of this talk

- A quick overview of ILP
- **Knowledge representation**
  - **individual-centred representations**
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook

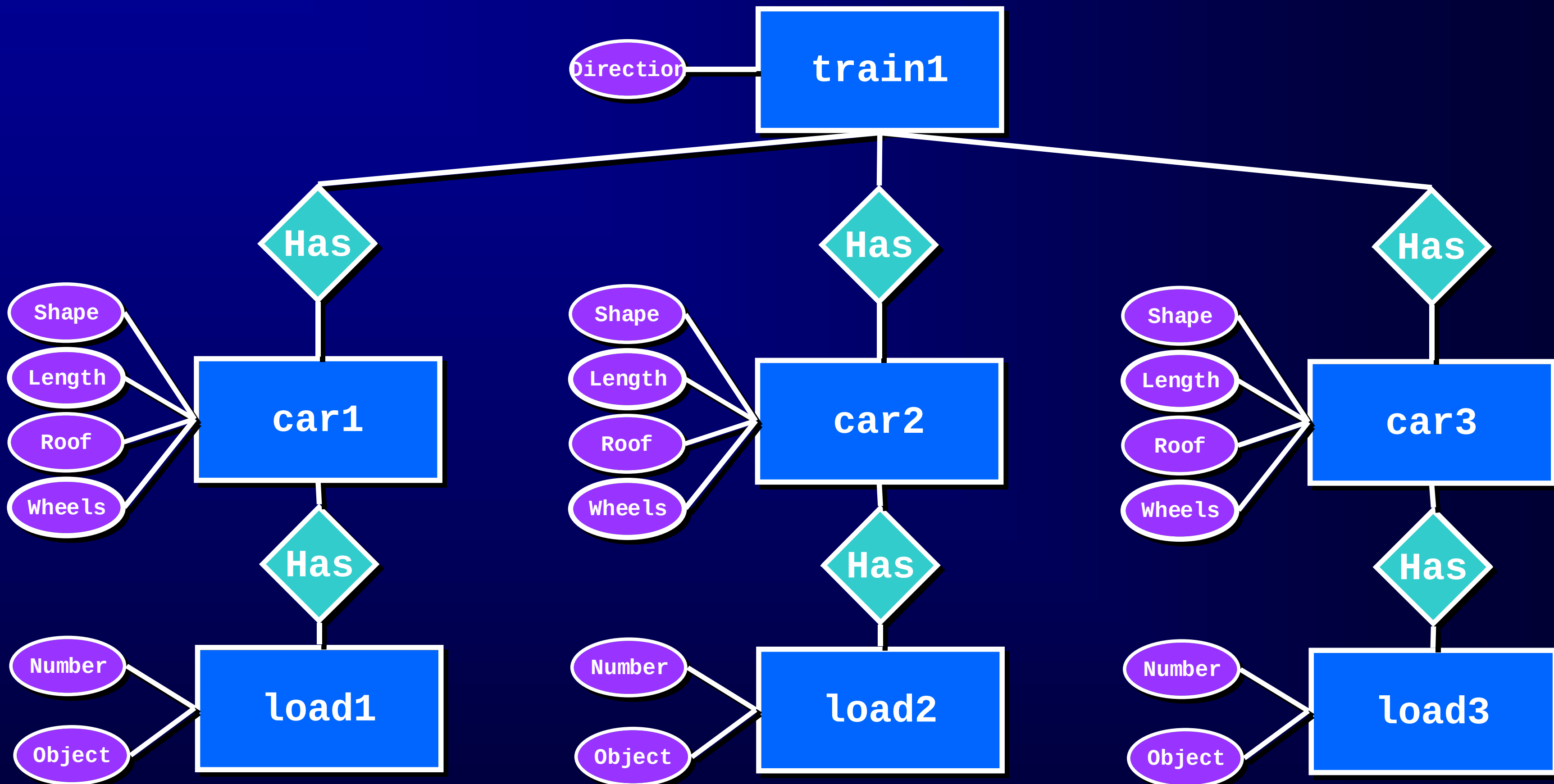
Knowledge Representation

- Entity-Relationship (ER) diagrams
- Relational Database
- Individual-Centred Representations
- Strongly typed language
- XML?

ER diagram for East-West trains



A particular train



Database representation

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	DIRECTION
t1	EAST
t2	EAST
…	…
t6	WEST
…	…

CAR_TABLE

CAR	TRAIN	SHAPE	LENGTH	ROOF	WHEELS
c1	t1	rectangle	short	open	2
c2	t1	rectangle	long	open	3
c3	t1	rectangle	short	peaked	2
c4	t1	rectangle	long	open	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 != 'open'

Individual-centred representations

- ER diagram is a tree (approximately)
  - root denotes individual
  - looking downwards from the root, only one-to-one or one-to-many relations are allowed
  - one-to-one cycles are allowed
- Database can be partitioned into sub-databases each describing a single individual
- Alternative: all information about a single individual packed together in a term
  - tuples, lists, sets, multisets, trees, …

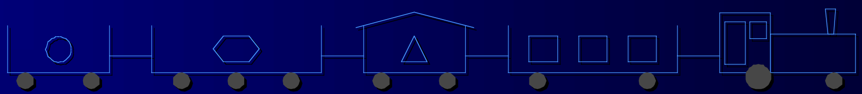
Strongly typed languages

- Type signature specifies 'data model'
  - similar to ER diagram
- Each example described by single statement
- Hypothesis construction guided by types
  - interaction between **structural functions/predicates** referring to subterms and **utility predicates** giving properties of subterms
- Example language: Escher
  - functional logic programming

East-West trains in Escher

- Type signature:

```
data Shape  = Rectangle | Hexagon | …;  data Length = Long | Short;
data Roof   = Open | 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,Open,2,(Circle,1)),
           (Rectangle,Long,Open,3,(Hexagon,1)),
           (Rectangle,Short,Peaked,2,(Triangle,1)),
           (Rectangle,Long,Open,2,(Rectangle,3))]) = True
```

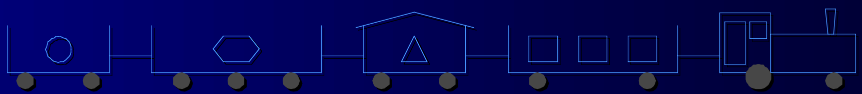
- Hypothesis:

```
eastbound(t) = (exists \c -> member(c,t) &&
                         LengthP(c)==Short && RoofP(c)!=Open)
```

East-West trains in Escher

- Type signature:

```
data Shape  = Rectangle | Hexagon | …;  data Length = Long | Short;
data Roof   = Open | 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;
```



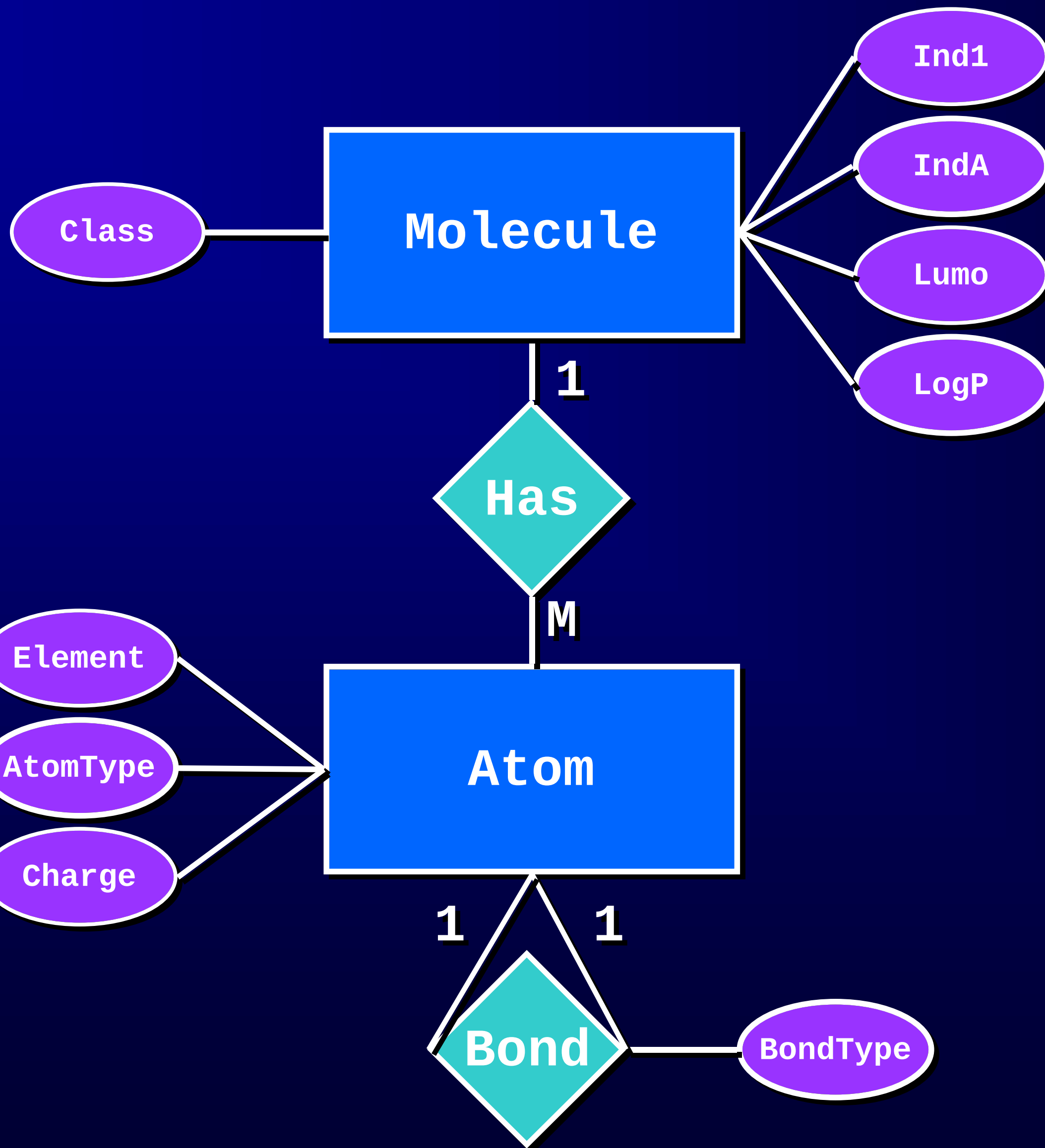
- Example:

```
eastbound([(Rectangle,Short,Open,2,(Circle,1)),
           (Rectangle,Long,Open,3,(Hexagon,1)),
           (Rectangle,Short,Peaked,2,(Triangle,1)),
           (Rectangle,Long,Open,2,(Rectangle,3))]) = True
```

- Hypothesis:

```
eastbound(t) = (exists \c -> member(c,t) &&
                       LengthP(c)==Short && RoofP(c)!=Open)
```

Mutagenesis



Mutagenesis in Escher

- Type signature:

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

type Ind1 = Bool;
type IndA = Bool;
type Lumo = Float;
type LogP = Float;
type AtomID = Int;
type AtomType = Int;
type Charge = Float;
type BondType = Int;
type Atom = (AtomID,Element,AtomType,Charge);
type Bond = ({AtomID},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)))) ||
  …;
```

Complexity of classification problems

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

Understanding ILP

- Back to Prolog: what do we learn from all this?
  - structural predicates introduce local variables, utility predicates consume them
  - interactions between local variables should not be broken up ===> features
  - enhancement of existing transformation methods (e.g. LINUS) through feature construction

The key steps in rule learning

- **Hypothesis construction**: find a set of *n* rules
  - usually simplified by *n* separate rule constructions
- **Rule construction**: find a pair (Head, Body)
  - e.g. select class and construct body
- **Body construction**: find a set of *m* literals
  - usually simplified by adding one literal at a time

The key steps in rule learning

- **Hypothesis construction**: find a set of *n* rules
  - usually simplified by *n* separate rule constructions
- **Rule construction**: find a pair (Head, Body)
  - e.g. select class and construct body
- **Body construction**: find a set of *m* features
  - usually simplified by adding one feature at a time
- **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 one feature '**has a short closed car**':

```
eastbound(T):-car(T,C),short(C),not open(C).
```

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

```
eastbound(T):-
  car(T,C1),short(C1),
  car(T,C2),not open(C2).
```

Propositionalising rules

- Equivalently:

```
eastbound(T):-hasShortCar(T),hasClosedCar(T).

hasShortCar(T):-car(T,C1),short(C1).

hasClosedCar(T):-car(T,C2),not open(C2).
```

- Given a way to construct and select first-order features, body construction in ILP is ***semi-propositional***
  - head and all literals in body have the same global variable(s)
  - corresponds to single table, one row per example

Prolog feature bias

- 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
  - if all structural predicates are one-to-one

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 of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- **Learning as inference**
  - **inductive consequence relations**
- Conclusions and outlook

Inductive consequence relations

- I write ***E* |< *H*** for '*H* is a possible inductive hypothesis given evidence *E*'
  - like deduction: from input to output
  - unlike deduction: possibly unsound
- What are sensible properties of |< ?
- What are possible material definitions of |< ?

General induction postulates

(I1) If α |< and $\alpha\wedge\beta\rightarrow\gamma$, then $\alpha\wedge\gamma$ |< β.

(I2) If α |< and $\alpha\wedge\beta\rightarrow\gamma$, then $\alpha\wedge\neg\gamma$ |≮ β.

(I2′) If $\beta\rightarrow\neg$ then α |≮ β

(I3) If α |< β and $\wedge\beta\rightarrow\gamma$, then α |< $\beta\wedge\gamma$

(I4) If α |< β, then

(I5) If α |< β, then |< β

(I6) If α |< β and $\beta\leftrightarrow\gamma$, then α |< γ.

(I7) If α |< γ and $\alpha\leftrightarrow\beta$, then β |< γ.

Explanatory induction

- E|< H is interpreted as ‘evidence E is ***explained*** by hypothesis H’
  - induction as reverse deduction
- Close link with abduction
  - Peirce: ‘if A were true, C would be a matter of course’
- Depends on notion of explanation

Explanatory induction postulates

(E1) If α |< β, [illegible] and γ |< γ, then α |< γ.

(E2) If γ |< γ and [illegible] |≮ γ, then α |< α.

(E3) If α |< $\beta \wedge \gamma$, th[illegible]n $\beta \rightarrow \alpha$ |< γ.

(E4) If α |< γ and [illegible] < γ, then $\alpha \wedge \beta$ |< γ.

(E5) If α |< γ and [illegible]$\rightarrow \beta$, then β |< γ.

Explanatory semantics

- Let $|\!\sim$ be an ***explanation mechanism***, and define the ***explanatory power*** of a formula α as $C_{\sim} = \{ \gamma \mid \alpha \mid\!\sim \gamma \}$
- The ***explanatory consequence relation*** $|\!<$ based on $|\!\sim$ is defined as

  $\alpha \mid\!< \beta$ iff $C_{\sim}(\alpha) \subseteq C_{\sim}(\beta) \subset \boldsymbol{L}$

- (E1–5) are sound and complete if $|\!\sim \; = \; |\!=$

Confirmatory induction

- E |< H is interpreted as 'evidence E ***confirms*** hypothesis H'

- A kind of closed-world reasoning
  - 'assume that everything you haven't seen behaves like something you have seen'
  - closely related to non-monotonic reasoning

Confirmatory induction postulates

(C1) If α |< β and [illegible]$\rightarrow\gamma$, then α |< γ.

(C2) If α |< α and α |≮ $\neg\beta$, then β |< β.

(C3) If α |< β and α |< γ, then α |< $\beta\wedge\gamma$.

(C4) If α |< γ and β |< γ, then $\alpha\vee\beta$ |< γ.

(C5) If α |< β and α |< γ, then $\alpha\wedge\gamma$ |< β.

Confirmatory semantics

- Let **Reg** be a function constr ctin se of *regular models* from observ tior

- The *confirmatory consequ nce ati n* |< based on **Reg** is defined a

  α |< β iff $\emptyset \subset$ **Reg**(α) $\subseteq$ β

- (C1–5) are sound and comp e i **g**(are the most preferred mode of

Overview of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- **Conclusions and outlook**

First-order representations in…

- …probabilistic models
  - Koller's probabilistic relational models
  - first-order Bayesian classification with 1BC
  - towards first-order Bayesian networks
- …support vector machines
  - kernels on sequences
  - a kernel on Escher terms
- …neural networks
  - recurrent NN for Escher terms

The naive Bayes classifier

- Bayesian classifier:

$$\arg\max_c P(c \mid d)$$

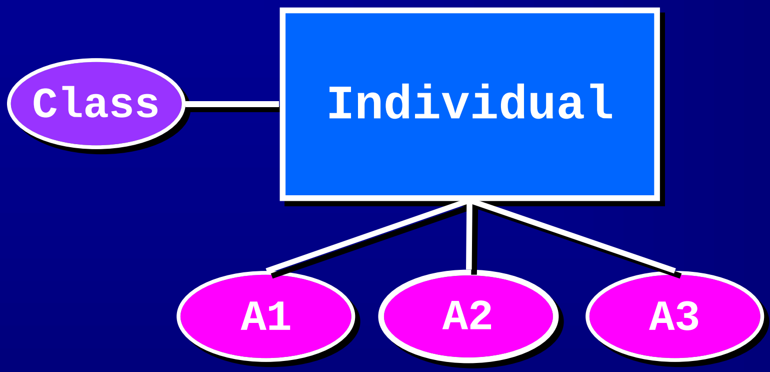
$$= \arg\max_c \frac{P(d \mid c)P(c)}{P(d)}$$

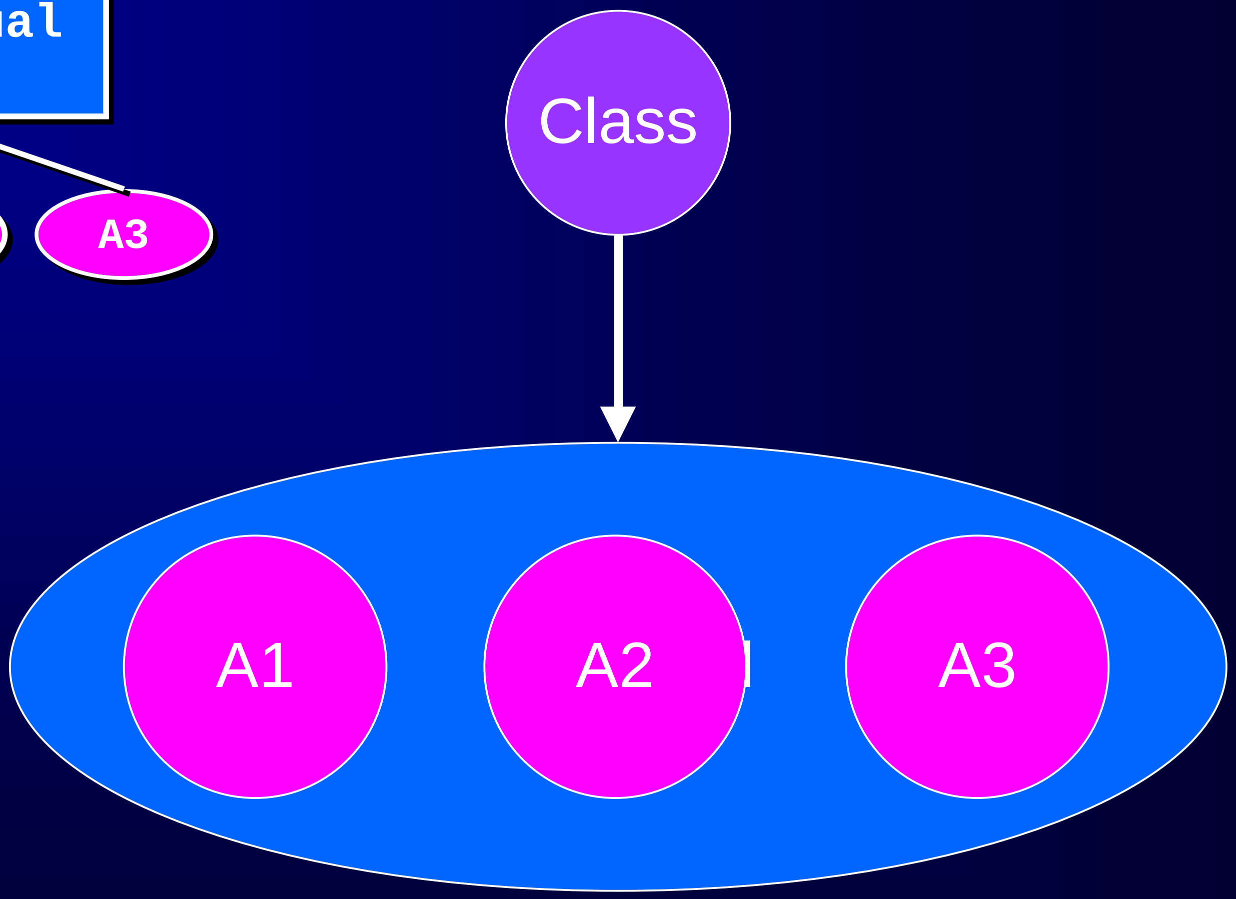
$$= \arg\max_c P(d \mid c)P(c)$$

- Naive Bayes assumption (propositional case):

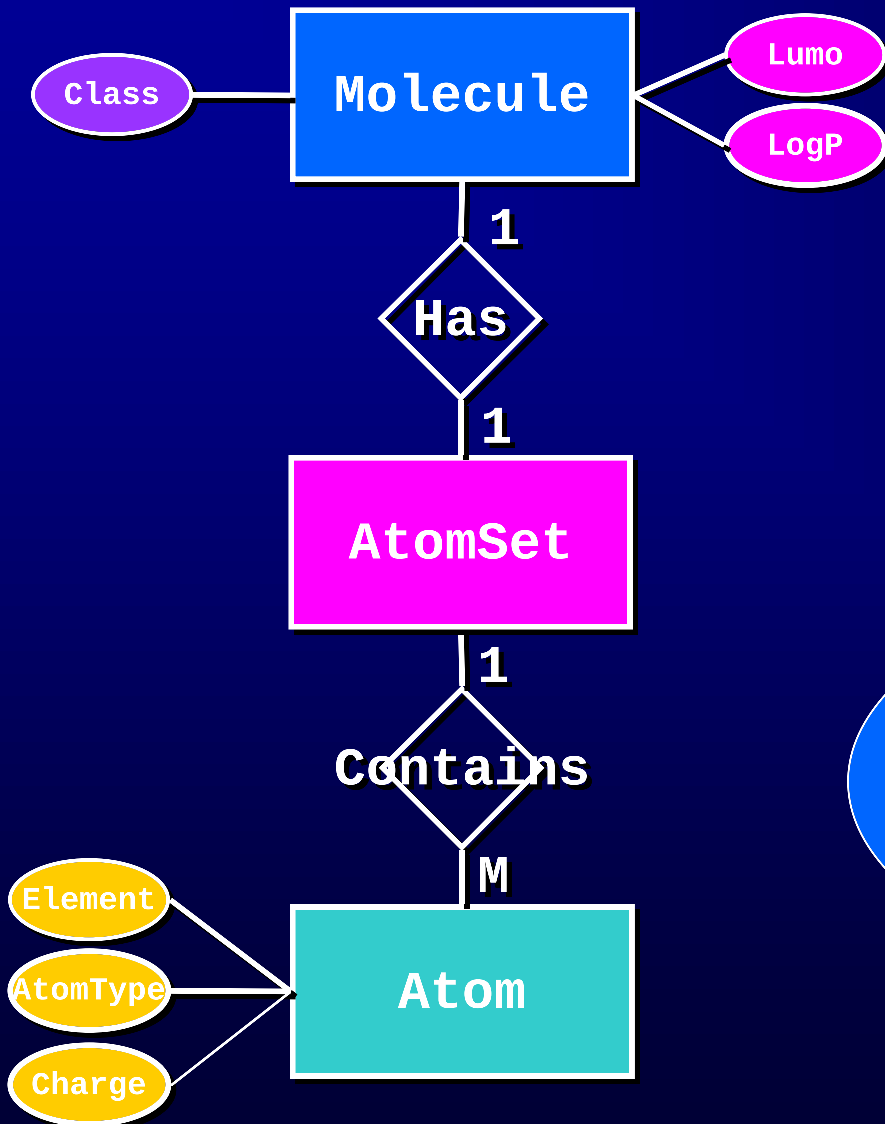
$$= \arg\max_c P(c)\prod_i P(A_i = a_i \mid c)$$

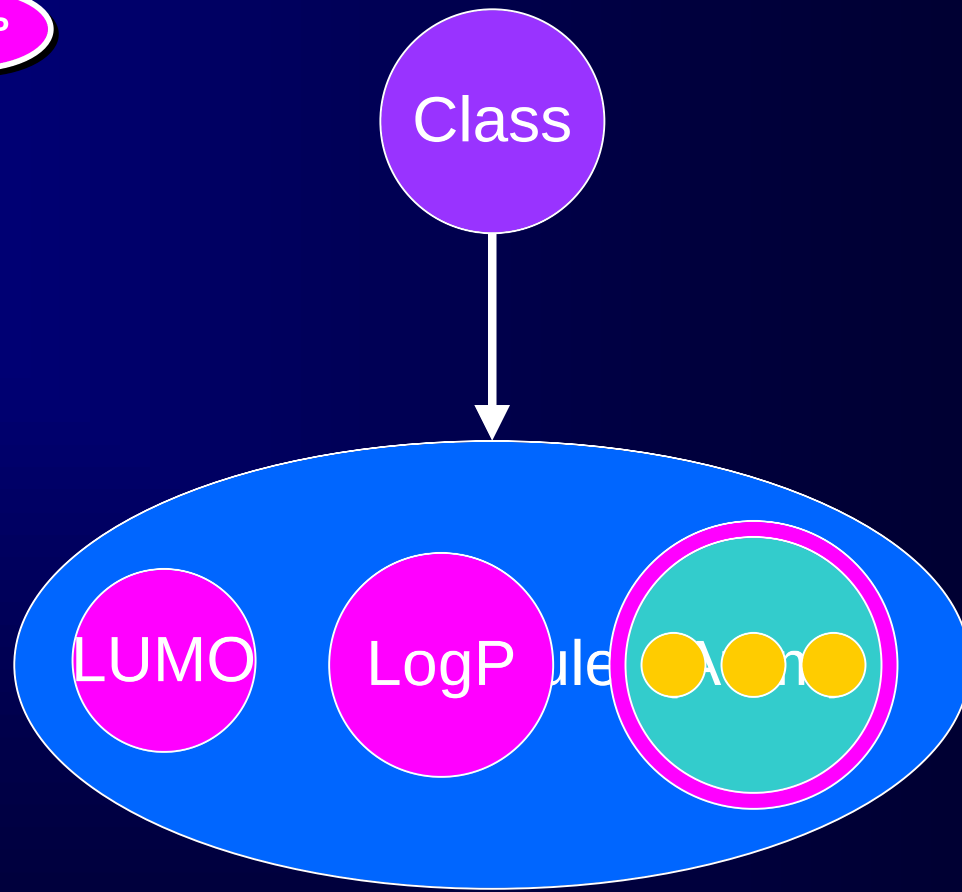
Naive Bayes net





Towards first-order Bayes nets





Support vector machines

- Wide margin classifier
  - support vectors are the datapoints closest to the separating hyperplane
- Kernel: (implicit) transformation to feature space
  - to deal with problems that are not linearly separable in input space
  - feature space is often high-dimensional

Primal and dual form

- Linear classifiers construct a hyperplane separating the input points
  - decision rule $$h(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b)$$
  - hypothesis $$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
  - equivalently $$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_i \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b \right)$$

where α_i represent hypothesis in dual co-ordinates

Kernels

- Learning in feature space:

$$h(\mathbf{x}) = \mathrm{sgn}\left(\sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b\right)$$

- A kernel calculates the inner product directly in input space:

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$$

  - This measures the similarity between **x** and **z** in terms of features ϕ

A kernel for Escher terms

- Let x and z be terms of type T. We define $K_T(x,z)$ recursively as follows:
  - If $T = T_1 \times \ldots \times T_n$ is a tuple type, $x = (x_1,\ldots,x_n)$ and $z = (z_1,\ldots,z_n)$, then $K_T(x,z) = K_{T1}(x_1,z_1) + \ldots + K_{Tn}(x_n,z_n)$.
  - If $T = \{T'\}$ is a set type, $x = \{x_1,\ldots,x_n\}$ and $z = \{z_1,\ldots z_m\}$, then $K_T(x,z) = K_{T'}(x_1,z_1) + \ldots + K_{T'}(x_1,z_m) + K_{T'}(x_2,z_1) + \ldots + K_{T'}(x_2,z_m) + \ldots + K_{T'}(x_n,z_m)$.
  - If $x = f(x_1,\ldots,x_n)$ and $z = f(z_1,\ldots,z_n)$ where f is a data constructor of type T_1 -> ... -> T_n -> T, then $K_T(x,z) = 1 + K_{T1}(x_1,z_1) + \ldots + K_{Tn}(x_n,z_n)$; if x and z have different data constructors then $K_T(x,z) = 0$.

Recurrent neural networks

- Consist of a recurrent or folding part that is unfolded to encode a given input tree, followed by a traditional feed-forward network
- Folding part trained by *backpropagation through structure*
- Generalises naturally to terms

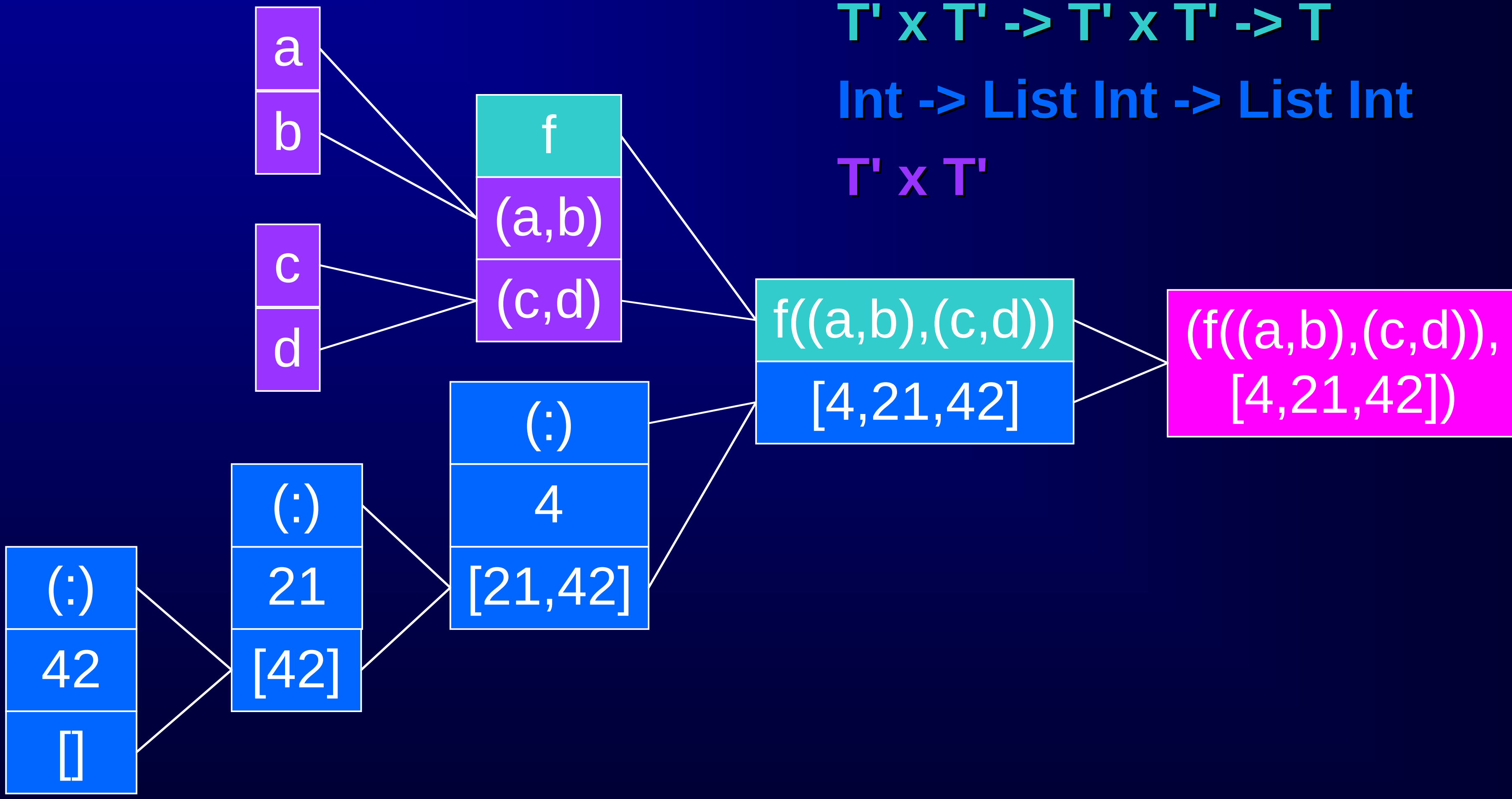
Recurrent NN for Escher terms

T x List Int

T' x T' -> T' x T' -> T

Int -> List Int -> List Int

T' x T'



Concluding remarks

- Data models and knowledge representation are integral parts of any approach to learning, modelling and reasoning
- Individual-centred representation are natural in classification and provide better understanding of the relation with propositional approaches
- There is still much to explore in upgrading existing propositional approaches with richer knowledge representation

Acknowledgements

- Joint work with
  - Nicolas Lachiche
  - John Lloyd
  - Christophe Giraud-Carrier
  - Nada Lavrac
  - Thomas Gaertner
  - Elias Gyftodimos