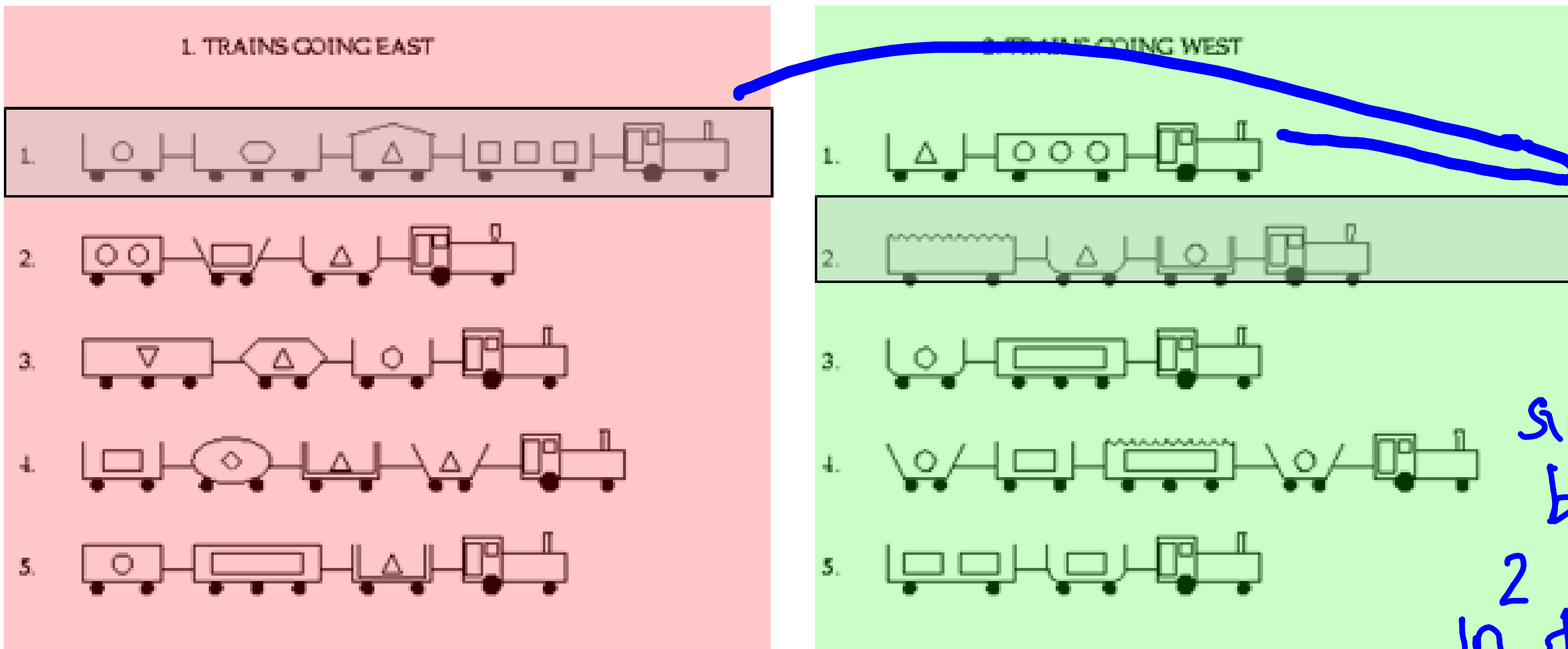
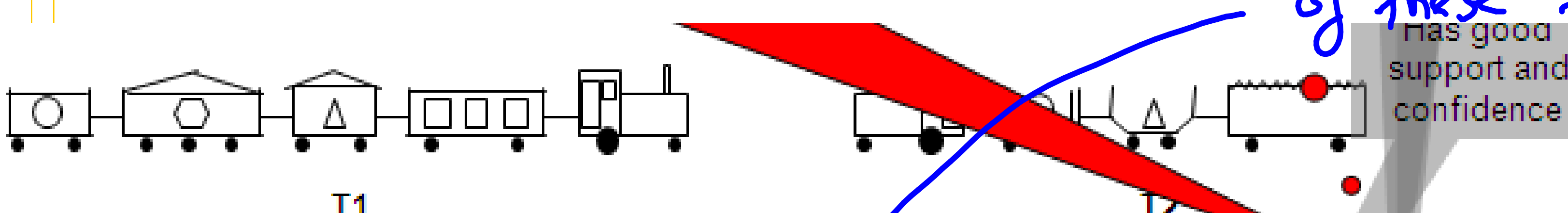
More on Klog



Klog tries to compute similarities between 2 trains in the space of these tables



TRAIN	
	DIR
T1	East
T2	West

HAS_CAR		
	CarID	CarName
T1	C1_1	first
T1	C1_2	second
T1	C1_3	third
T1	C1_4	fourth
T2	C2_1	first
T2	C2_2	second
T2	C2_3	third

SHORT
CarID
C1_2
C1_4
C2_1
C2_3

CLOSED
CarID
C1_1
C1_2
C1_3
C1_4
C2_3

LOADS		
	...ape	Qty
	...quare	3
	...iangle	1
C1_3	Hexagon	1
C1_4	Circle	1
C2_1	Circle	1
C2_2	Triangle	1
C2_3	.	.

Eastbound trains have:

a long car C1, and

C1 is closed -roofed, and

C1 has 3 square loads, and

a short car C2, and

C2 is closed, and

C2 has 1 triangular load, and

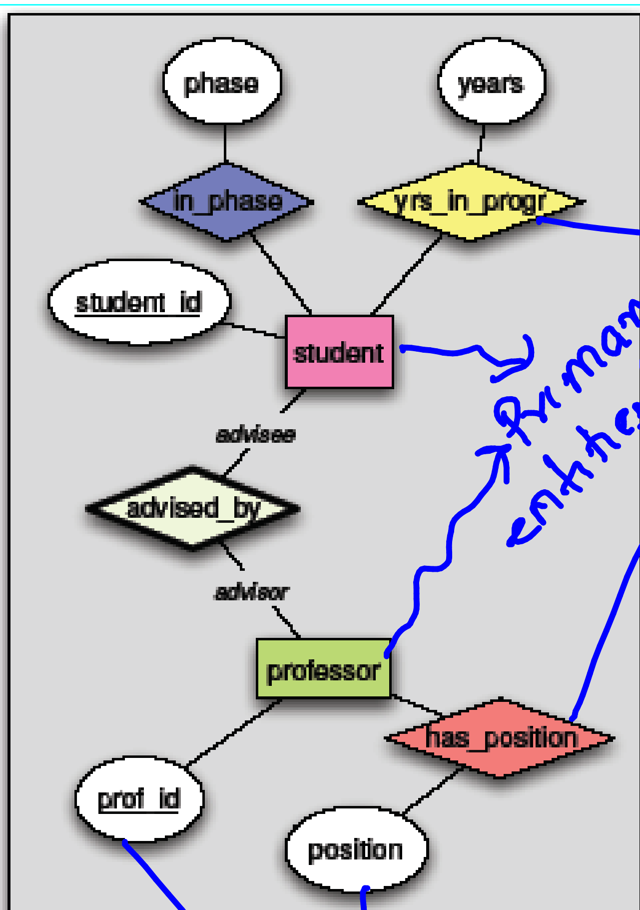
a long car C3,

...

...

Goal of Klog: Compute similarity

between examples that "efficiently" & "nicely" captures the effect of enumerating all "description" (in First Order Logic) of the example(s)



primary entities (trains)

name of table (in above example of trains)

values of cells in table

If we had info on which trains run after which trains, we would have had multiple rectangular boxes & edges between them

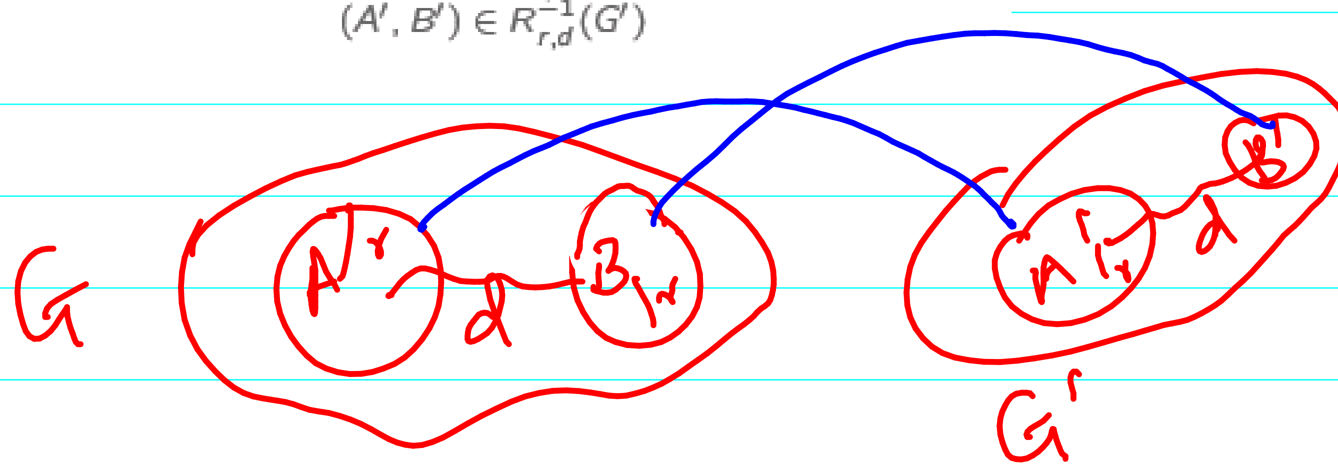
Not specified in tables but are "views" defined through this "join" like query

```
signature on_same_paper(
    student_id::student,
    prof_id::professor
):intensional.

on_same_paper(S,P) :-
     student(S), professor(P),
     publication(Pub, S),
     publication(Pub,P).

signature on_same_course(
   student_id::student,
   prof_id::professor
):intensional.
on_same_course(S,P) :-
     professor(P), student(S),
     ta(Course,S,Term),
     taught_by(Course,P,Term).
```

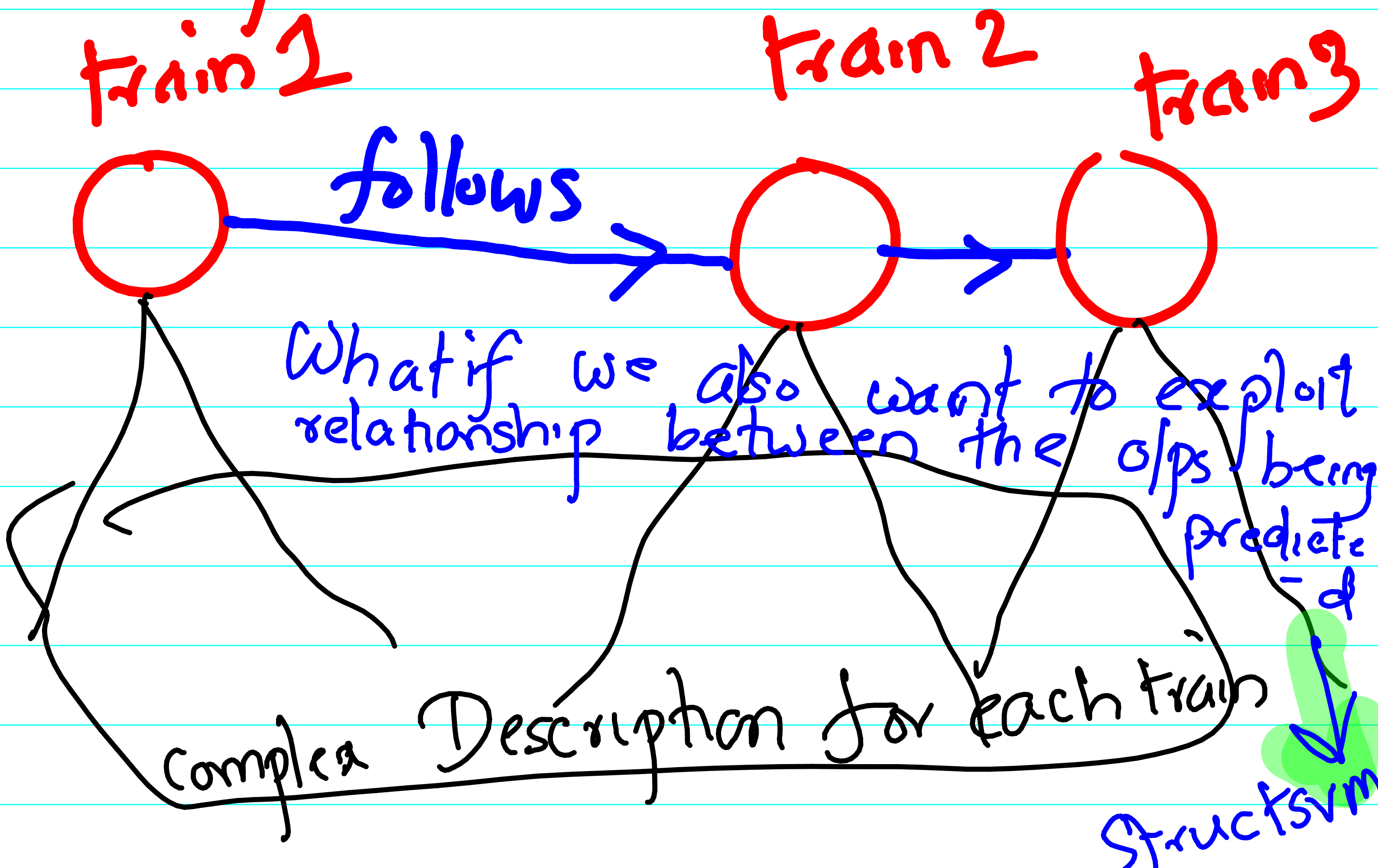
Specified as before in relational tables

- $\kappa_{r,d}$ counts common NP's between two graphs.

$$\kappa_{r,d}(G, G') = \sum_{\substack{(A,B) \in R^{-1}_{r,d}(G) \\ (A',B') \in R^{-1}_{r,d}(G')}} \delta(A, A')\delta(B, B')$$



Structured output prediction



What if we also want to exploit relationship between the o/ps being predicted

So far: K(train 1, train 2) was computed using complex methods

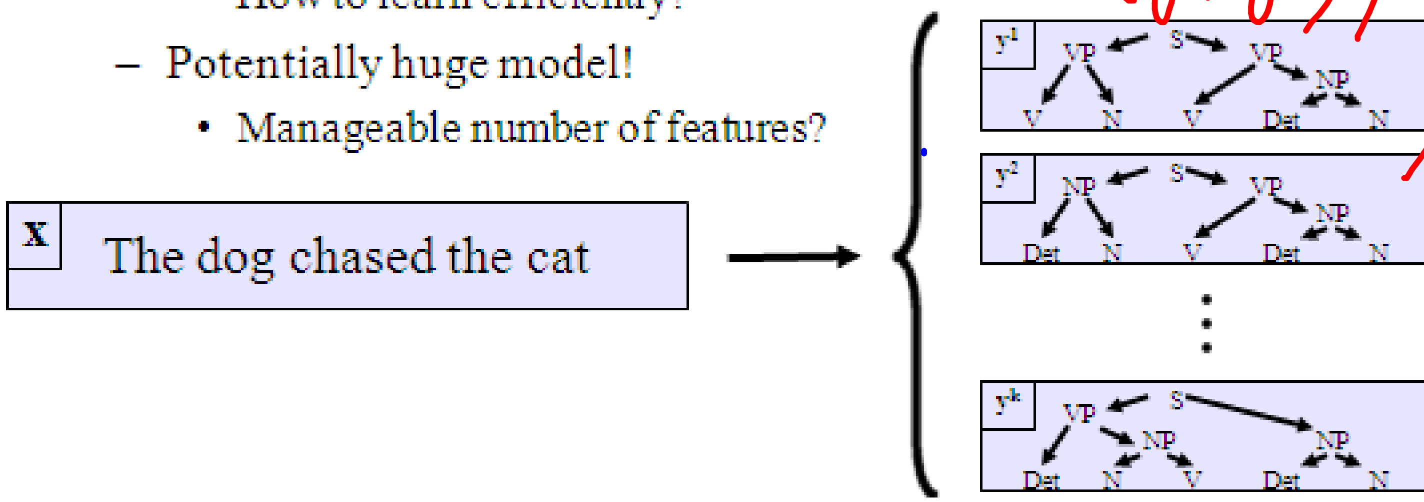
But

$$f(\text{train}) = \sum_{\text{train}(i) \in D} \alpha_i y_i K(\text{train}, \text{train}(i))$$

$y_i = 1$ iff east bound
-1 o/w

We would like to exploit that! $\text{sim}(y^1, y^2) > \text{sim}(y^2, y^k)$

- **Problems:**
  - Exponentially many classes!
    - How to predict efficiently?
    - How to learn efficiently?
  - Potentially huge model!
    - Manageable number of features?



Soln 1:

$$w_y^T \phi(x) \rightarrow w^T \phi(x, y)$$

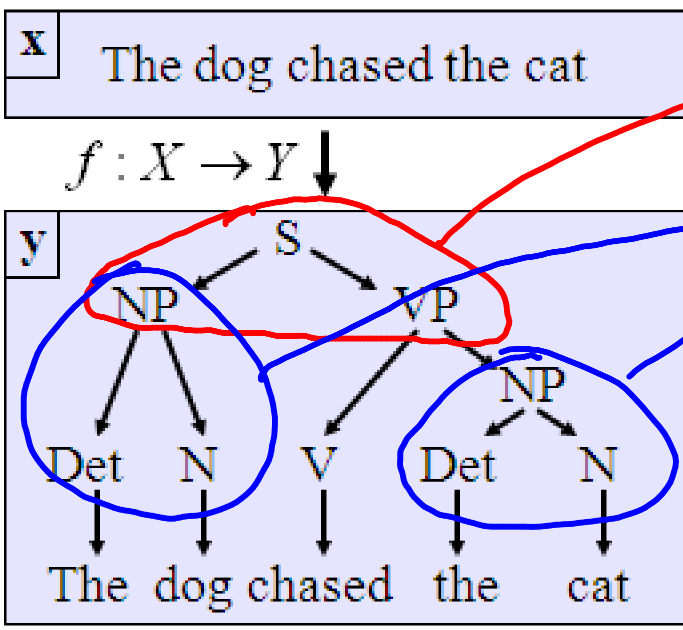
Standard tradeoff

i.e. we need to look at substructures within y the way we looked at substructures within parse trees for x earlier

Joint Feature Map for Trees

- **Weighted Context Free Grammar**
  - Each rule r_i (e.g. $S \rightarrow NP\ VP$) has a weight w_i
  - Score of a tree is the sum of its weights
  - Find highest scoring tree $h(\vec{x}) = argmax_{y \in Y} \left[\vec{w}^T \Phi(x, y) \right]$

CKY Parser



$$\Phi(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ \vdots \\ 0 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{matrix} S \rightarrow NP\ VP \\ S \rightarrow NP \\ NP \rightarrow Det\ N \\ VP \rightarrow V\ NP \\ \\ Det \rightarrow dog \\ Det \rightarrow the \\ N \rightarrow dog \\ V \rightarrow chased \\ N \rightarrow cat \end{matrix}$$

$\phi(x, y)$ is a fixed length vector of size $|G|$ i.e. # of prods

This list of productions is manually created and available

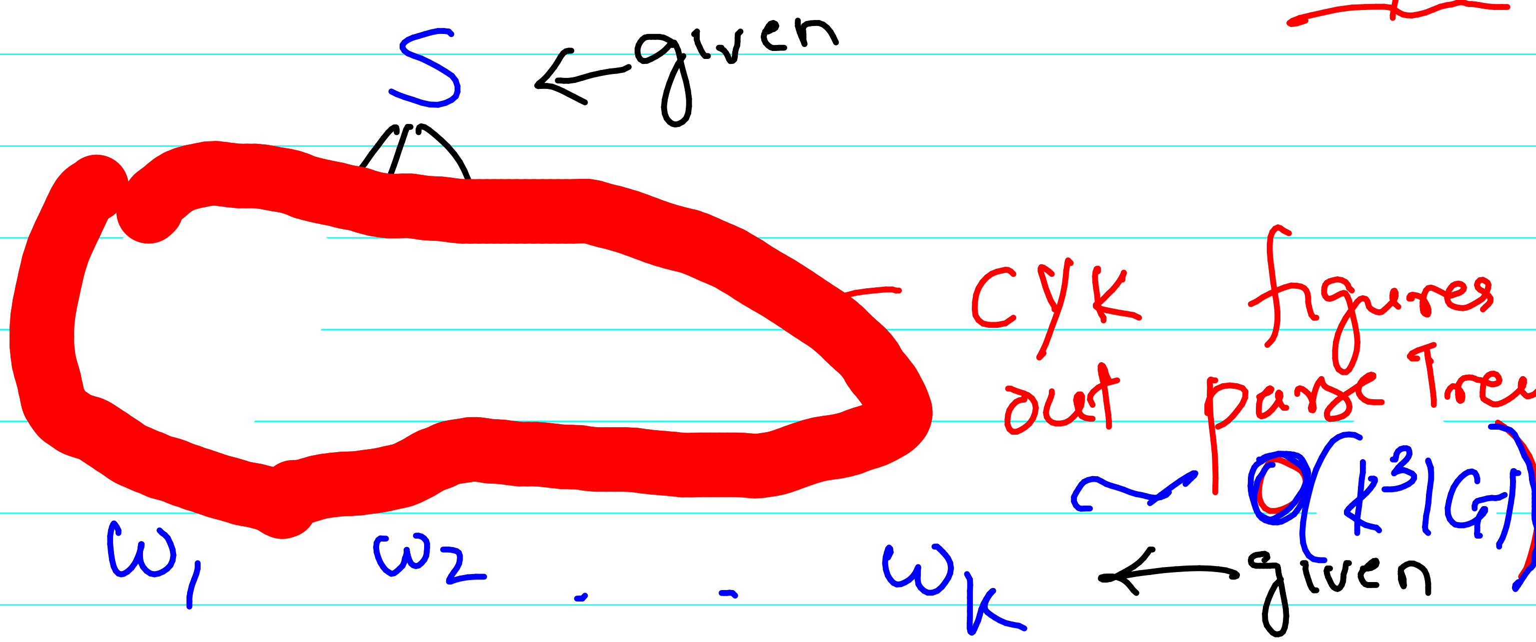
Q1: At the end of training, we have a set of productions along with weight for each. Given a sentence, how to find the "parse tree with highest weight"?

Ans: Modify CYK algo (used for parsing using context free grammar) to handle weights

[Bottom-up parsing] [Based on dynamic programming]

http://en.wikipedia.org/wiki/CYK_algorithm

H/W



Q2: How to allow for slackness in the loss function that accounts for and accommodates "almost similar" parse trees (similar to the correct parse tree)