Advanced Artificial Intelligence

Part II. Statistical NLP

Probabilistic DCGs and SLPs

Wolfram Burgard, Luc De Raedt, Bernhard Nebel, Lars Schmidt-Thieme

Many slides taken from Kristian Kersting

Overview

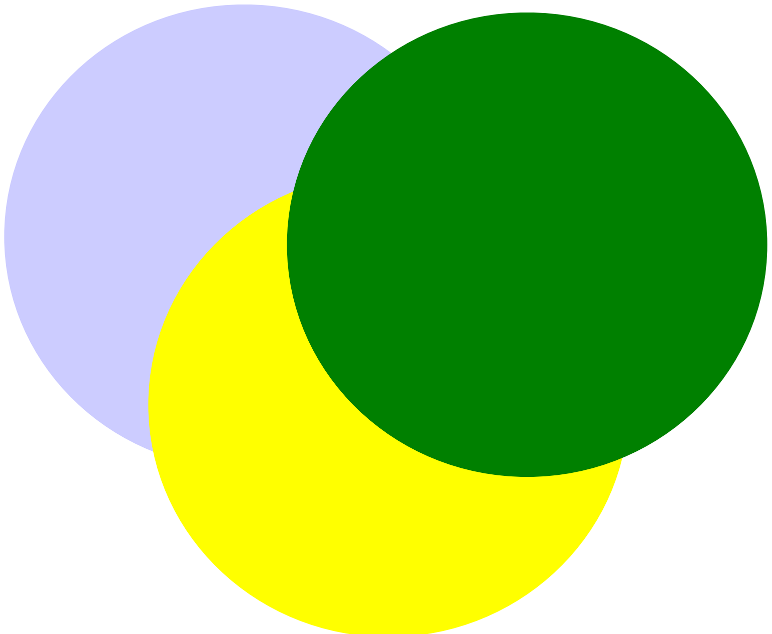
- Expressive power of PCFGs, HMMs, BNs still limited
  - First order logic is more expressive
- Why not combine logic with probabilities ?
  - Probabilistic logic learning
  - Statistical relational learning
- One example in NLP
  - Probabilistic DCGs (I.e. PCFGs + Unification)

Context

One of the key open questions of artificial intelligence concerns

"probabilistic logic learning",

i.e. the integration of
probabilistic reasoning
with
first order logic
representations and
machine learning.



Sometimes called Statistical Relational Learning

So far

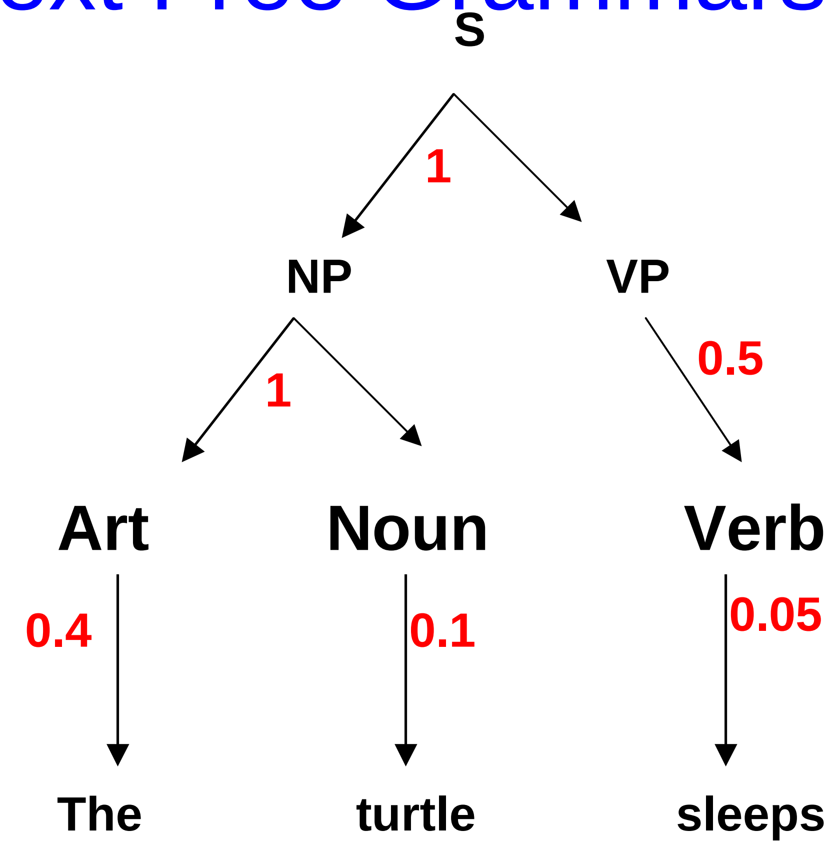
- We have largely been looking at probabilistic representations and ways of learning these from data
  - BNs, HMMs, PCFGs
- Now, we are going to look at their expressive power, and make traditional probabilistic representations more expressive using logic
  - Probabilistic First Order Logics
  - Lift BNs, HMMs, PCFGs to more expressive frameworks
  - Upgrade also the underlying algorithms

Prob. Definite Clause Grammars

- Recall :
  - Prob. Regular Grammars
  - Prob. Context-Free Grammars
- What about Prob. Turing Machines ? Or Prob. Grammars ?
  - Combine PCFGs with Unification
  - A more general language exists : stochastic logic programs (SLPs)
    - Prolog + PCFGs

Probabilistic Context Free Grammars

1.0 : S -> NP, VP
1.0 : NP -> Art, Noun
0.6 : Art -> a
0.4 : Art -> the
0.1 : Noun -> turtle
0.1 : Noun -> turtles
…
0.5 : VP -> Verb
0.5 : VP -> Verb, NP
0.05 : Verb -> sleep
0.05 : Verb -> sleeps
….



P(parse tree) = 1x1x.5x.1x.4x.05

We defined

- $P(tree|G) = \prod_i p_i^{c_i}$ where i ranges over all rules i used to derive tree, and c_i is the number of times they were applied
- $P(w_{1m}|G) = \sum_j P(tree_j|G)$ where j ranges over all possible parse trees for w_{1m}.
- Key concept : probabilities over derivations !!!

PCFGs

Observe: all derivation/rewriting steps succeed

i.e. S->T,Q

T->R,U

always gives

S-> R,U,Q

Probabilistic Definite Clause Grammar

1.0 : S -> NP(Num), VP(Num)

1.0 NP(Num) -> Art(Num), Noun(Num)

0.6 Art(sing) -> a

0.2 Art(sing) -> the

0.2 Art(plur) -> the

0.1 Noun(sing) -> turtle

0.1 Noun(plur) -> turtles

…

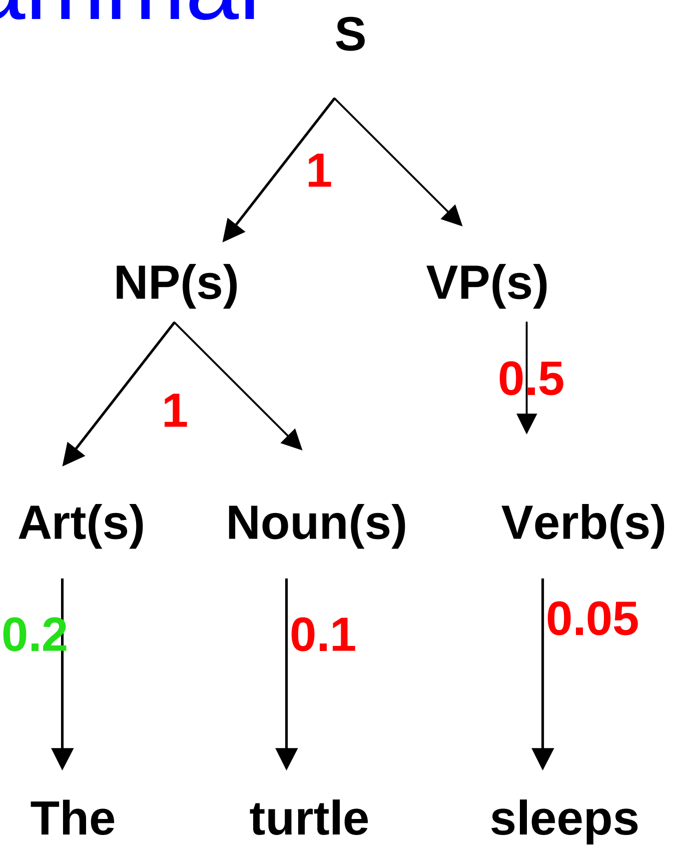
0.5 VP(Num) -> Verb(Num)

0.5 VP(Num) -> Verb(Num), NP(Num)

0.05 Verb(sing) -> sleep

0.05 Verb(plur) -> sleeps

….

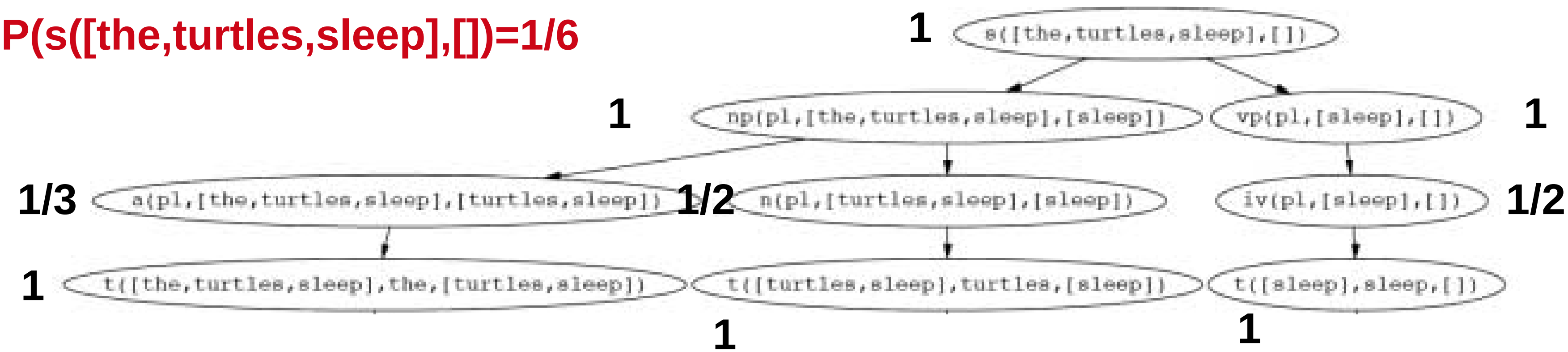


P(derivation tree) = 1x1x.5x.1x .2 x.05

In SLP notation

```
     sentence(A, B) :- noun_phrase(C, A, D), verb_phrase(C, D, B).
1    noun_phrase(A, B, C) :- article(A, B, D), noun(A, D, C).
     verb_phrase(A, B, C) :- intransitive_verb(A, B, C).
     article(singular, A, B) :- terminal(A, a, B).
1/3  article(singular, A, B) :- terminal(A, the, B).
     article(plural, A, B)   :- terminal(A, the, B).
     noun(singular, A, B) :- terminal(A, turtle, B).
1/2  noun(plural, A, B)   :- terminal(A, turtles, B).
     intransitive_verb(singular, A, B) :- terminal(A, sleeps, B).
     intransitive_verb(plural, A, B) :- terminal(A, sleep, B).
1    terminal([A|B],A,B).
```

P(s([the,turtles,sleep],[])=1/6



Stochastic Logic Programs

- Correspondence between CFG - SLP
  - Symbols - Predicates
  - Rules - Clauses
  - Derivations - SLD-derivations/Proofs
- So,
  - a stochastic logic program is an annotated logic program.
  - Each clause has an associated probability label. The sum of the probability labels for clauses defining a particular predicate is equal to 1.

Probabilistic Definite Clause Grammar

1.0 : S -> NP(Num), VP(Num)

1.0 NP(Num) -> Art(Num),
Noun(Num)

0.6 Art(sing) -> a

0.2 Art(sing) -> the

0.2 Art(sing) -> the

0.1 Noun(sing) -> turtle

0.1 No

…

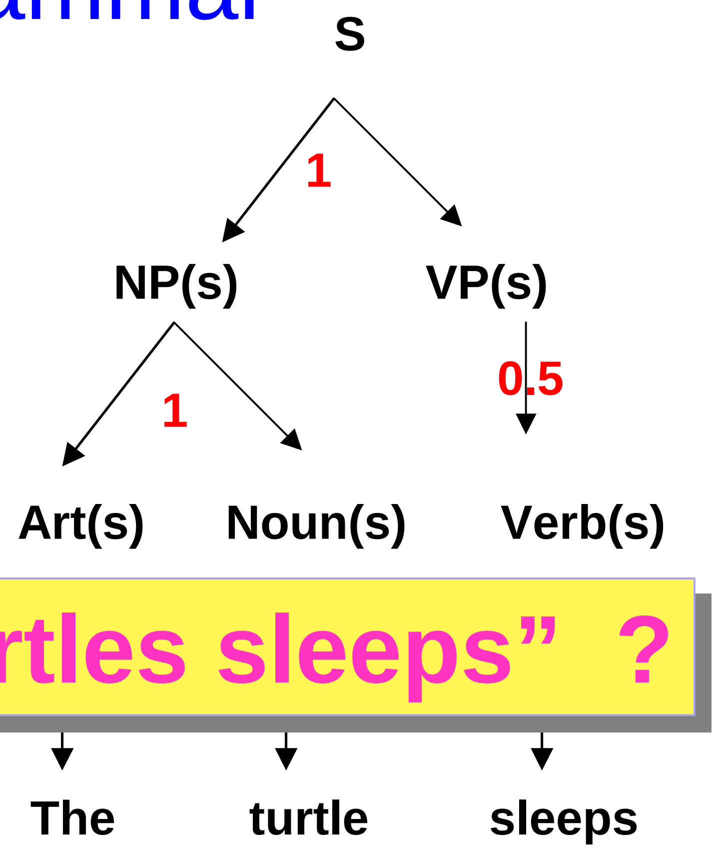
0.5 VP(Num) -> Verb(Num)

0.5 VP(Num) -> Verb(Num), NP(Num)

0.05 Verb(sing) -> sleep

0.05 Verb(plur) -> sleeps

….



What about "A turtles sleeps" ?

P(derivation tree) = 1x1x.5x.1x .2 x.05

PDCGs

Observe: some derivations/resolution steps fail

e.g. NP(*Num*)–>Art(*Num*), Noun(*Num*)

and Art(sing)-> a and Noun(plur)->turtles

Interest in successful derivations/proofs/refutations

-> normalization necessary

PDCGs : distributions

- let us consider derivations (i.e., parse trees) d for variabelized non-terminals of the form $p(X_1, ..., X_n)$ with the X_i different variables; corresponds to a non-terminal in a PCFG
- $P_D(der\ d|PDCG) = \prod_i p_i^{c_i}$ where i ranges over all rules used in the derivation $der\ d$ for goal d and c_i is the number of times i was applied; so far this is similar as for PCFGs
- **key difference with PCFGs**:
  - derivations in PCFGs always succeed,
  - proofs in PDCGs can fail;

PCFGs : distributions

- *refutations* are successful parse trees
- we are interested in the conditional probability of a derivation given that we know it is succesful/a refutation
- Therefore, define$P_R(ref\ d|PDCG) = \frac{P_D(ref\ \ d)}{\sum_j P_D(ref_j\ \ d|PDCG)}$, the probability P_R of refutation ref of d; where j ranges over all refutations ref_d of the goal d
- For sentences s define: $P_S(sent\ s|PDCG) = \sum_j P(ref_j\ \ s|PDCG)$ where ref_j ranges over all possible refutations of s starting from the starting symbol $d(X_1, ..., X_n)$

Sampling

- PRGs, PCFGs, PDCGs, and SLPs can also be used for sampling sentences, ground atoms that follow from the program
- Rather straightforward. Consider PDCGs:
  - Probabilistically explore parse-tree
  - At each step, select possible resolvents using the probability labels attached to clauses
  - If derivation succeeds, return corresponding sentence
  - If derivation fails, then restart.

Questions we can ask (and answer) about PDCGs and SLPs

- Compute the probability $P_S(s|PDCG)$ of a sentence s
- Find the most likely refutation (successful parse tree) r for a sentence or non-terminal d, i.e.
  $argmax_{ref} P_R(ref\ d)$
- For a given PDCG and set of sentences S
  for a non-terminal d, compute the parameters λ
  that maximize$(\prod_{s \in S} P_S(s|PDCG(\lambda))$

Answers

- The algorithmic answers to these questions, again extend those of PCFGs and HMMs, in particular,
  - Tabling is used (to record probabilities of partial proofs/parse trees and intermediate results)
  - Failure Adjusted EM (FAM) is used to solve parameter re-estimation problem
    - Only sentences observed
    - Unobserved: Possible refutations and derivations for observed sentences
    - Therefore EM, e.g., Failure Adjusted Maximisation (Cussens) and more recently (Sato) for PRISM
    - Topic of recent research

Answers

- There is a decent implementation of these ideas in the system PRISM by Taisuke Sato for a variant of SLPs/PDCGs
- http://sato-www.cs.titech.ac.jp/prism/

Structure Learning

- From proof trees : De Raedt et al AAAI 05
  - Learn from proof-trees (parse trees) instead of from sentences
  - Proof-trees carry much more information
  - Upgrade idea of tree bank grammars and PCFGs
- Given
  - A set of proof trees
- Find
  - A PDCG that maximizes the likelihood

Initial Rule Set DCG

S -> NP(s), VP(s)

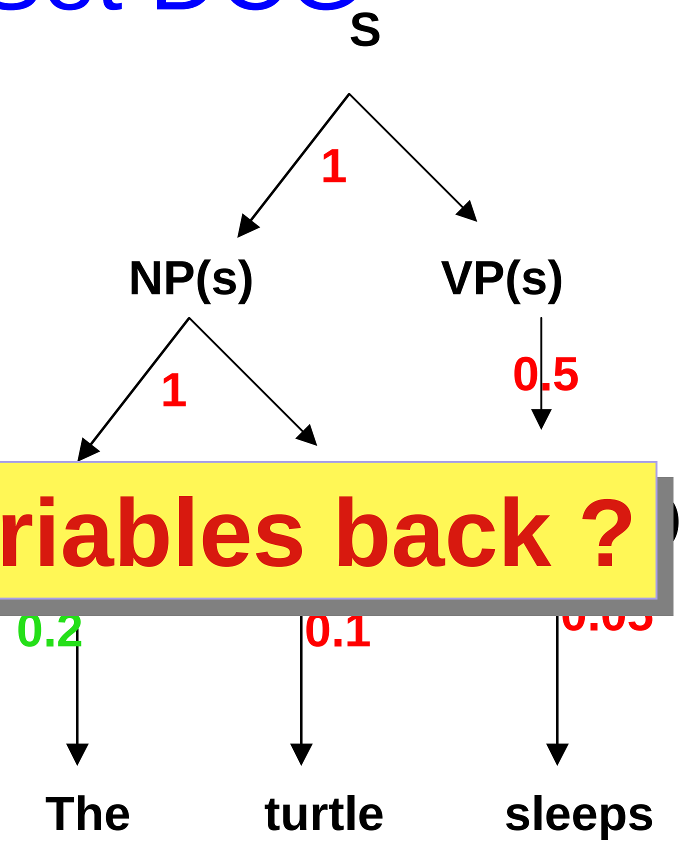
NP(s) -> Art(s), Noun(s)

VP(

Art(s) -> the

Noun(s) -> turtle

Verb(s) -> sleeps



How to get the variables back ?

P(derivation tree) = 1x1x.5x.1x.4x.05

Learning PDCGs from Proof Trees

- Based on Tree-Bank Grammar idea, e.g. Penn Tree Bank

- Key algorithm
  - Let S be the set of all (instantiated) rules that occur in an example proof tree
  - Initialize parameters
  - repeat as long as the score of S improves
    - Generalize S
    - Estimate the parameters of S using Cussens' FAM
      - (which can be simplified - proofs are now observed)
  - Output S

Generalizing Rules in SLPs

- Generalization in logic
  - Take two rules for same predicate and replace them by the lgg under θ -subsumption (Plotkin)
  - Example

    department(cs,nebel) ->
      prof(nebel), in(cs), course(ai), lect(nebel,ai).

    department(cs,burgard) ->
      prof(burgard), in(cs),course(ai), lect(burgard,ai)
  - Induce

    department(cs,P) ->
      prof(P), in(cs),course(ai), lect(P,ai)

Strong logical constraints

- Replacing the rules r1 and r2 by the lgg should preserve the proofs/parse trees !
- So, two rules r1 and r2 should only be generalized when
  - There is a one to one mapping (with corresponding substitutions) between literals in r1, r2 and lgg(r1,r2)
- Exclude

  father(j,a) -> m(j),f(a),parent(j,a)

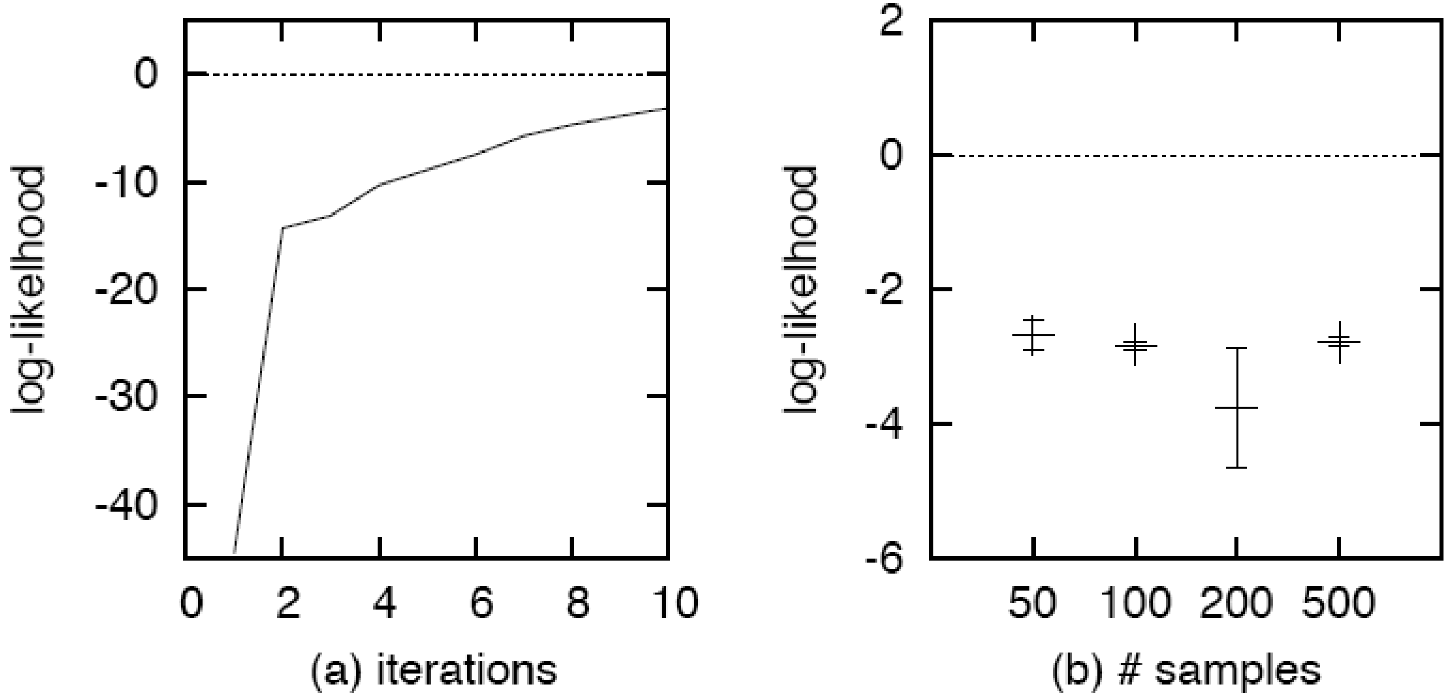
  father(j,t) -> m(j),m(t), parent(j,t)
- Gives

  father(j,P) -> m(j),m(X),parent(j,P)

Experiment

$$
\begin{aligned}
&1 : s(A, B) \leftarrow np(Number, A, C), vp(Number, C, B).\\
&1/2 : np(Number, A, B) \leftarrow det(A, C), n(Number, C, B).\\
&1/2 : np(Number, A, B) \leftarrow pronom(Number, A, B).\\
&1/2 : vp(Number, A, B) \leftarrow v(Number, A, B).\\
&1/2 : vp(Number, A, B) \leftarrow v(Number, A, C), np(D, C, B).\\
&1 : det(A, B) \leftarrow term(A, the, B).\\
&1/4 : n(s, A, B) \leftarrow term(A, man, B).\\
&1/4 : n(s, A, B) \leftarrow term(A, apple, B).\\
&1/4 : n(pl, A, B) \leftarrow term(A, men, B).\\
&1/4 : n(pl, A, B) \leftarrow term(A, apples, B).\\
&1/4 : v(s, A, B) \leftarrow term(A, eats, B).\\
&1/4 : v(s, A, B) \leftarrow term(A, sings, B).\\
&1/4 : v(pl, A, B) \leftarrow term(A, eat, B).\\
&1/4 : v(pl, A, B) \leftarrow term(A, sing, B).\\
&1 : pronom(pl, A, B) \leftarrow term(A, you, B).\\
&1 : term([A|B], A, B) \leftarrow
\end{aligned}
$$

Experiment



In all experiments : correct structure induced !

Conclusions

- SLPs and PDCGs extend PCFGs as a representation
- Proof-trees for PDCGs and PCFGs correspond to parse-trees in PCFGs
- Also : a lot of related work in Statistical Relational Learning and Probabilistic Inductive Logic Programming
  - Combining Graphical Models and Prob. Grammars with ideas (such as unification and relations) from first order logic
  - Many approaches -
    - Bayesian Nets (and Bayesian Logic Programms)
    - Markov Networks (and Markov Logic Networks)
    - …
  - Current research topic
    - EU project APRIL II