Topics in Natural Language Processing

Shay Cohen

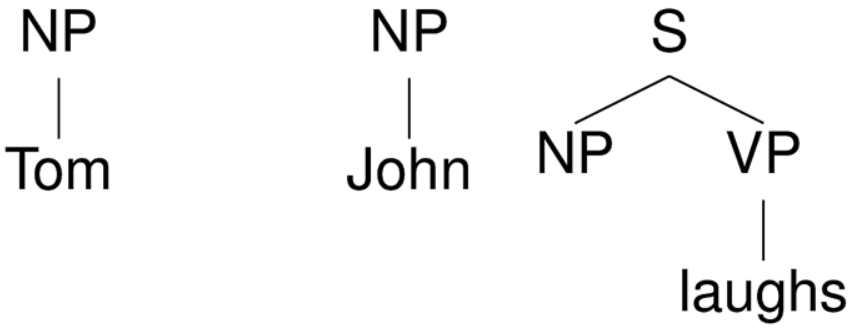
Institute for Language, Cognition and Computation

University of Edinburgh

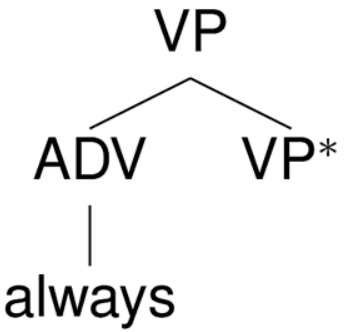
Lecture 5

Tree Adjoining Grammars

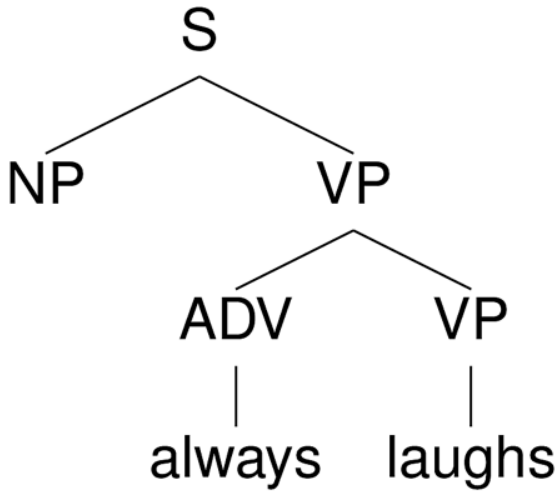
Initial trees



Auxiliary trees



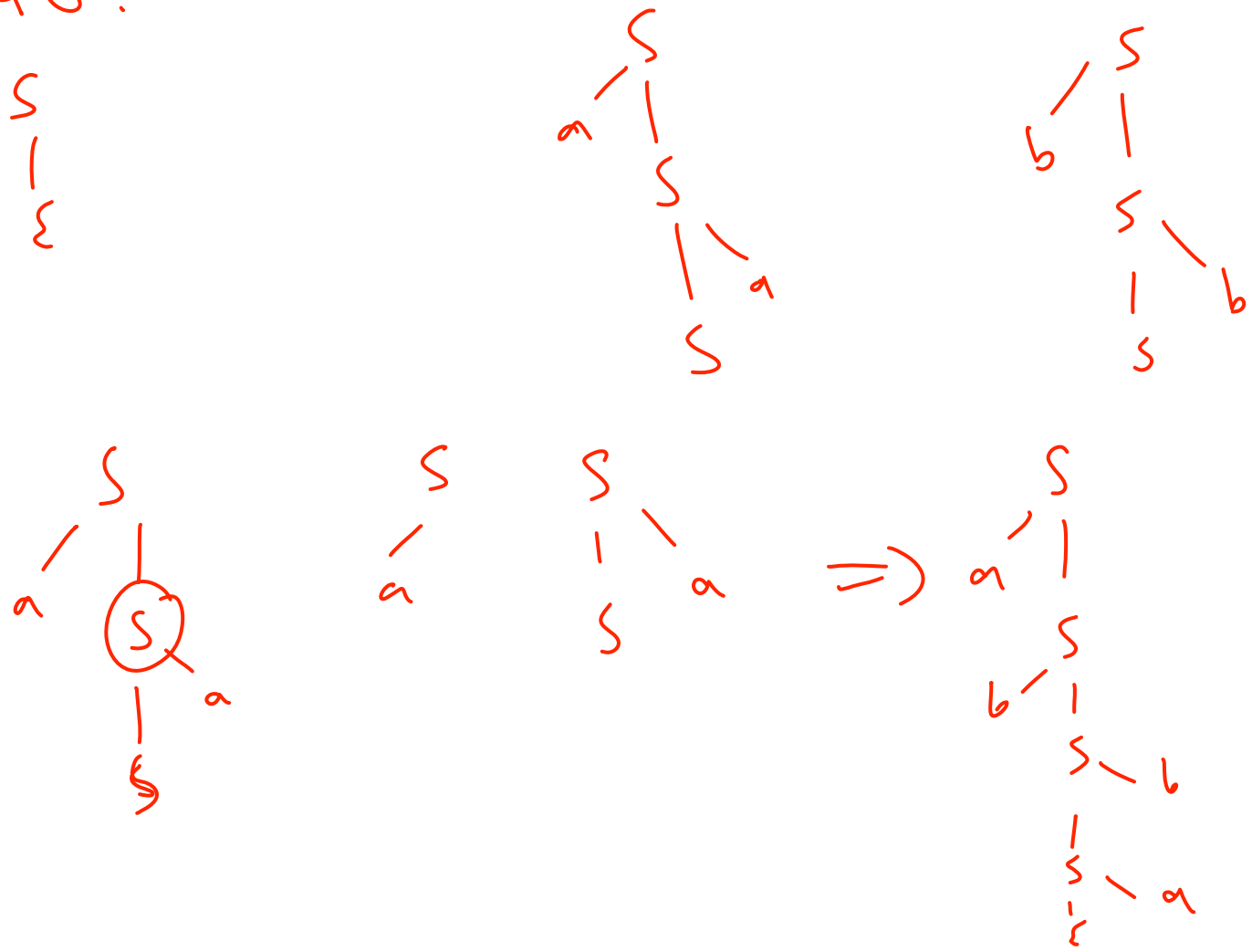
Derivational process: break the tree (S (NP) (VP laughs)) on the VP node and insert the auxiliary tree:



Tree Adjoining Grammars

Quick question: is $\{ww \mid w \in \Sigma^*\}$ a context-free language? No.

TAG:



Tree Adjoining Grammars

Quick question: is $\{ww \mid w \in \Sigma^*\}$ a context-free language?

Is it a tree adjoining language?

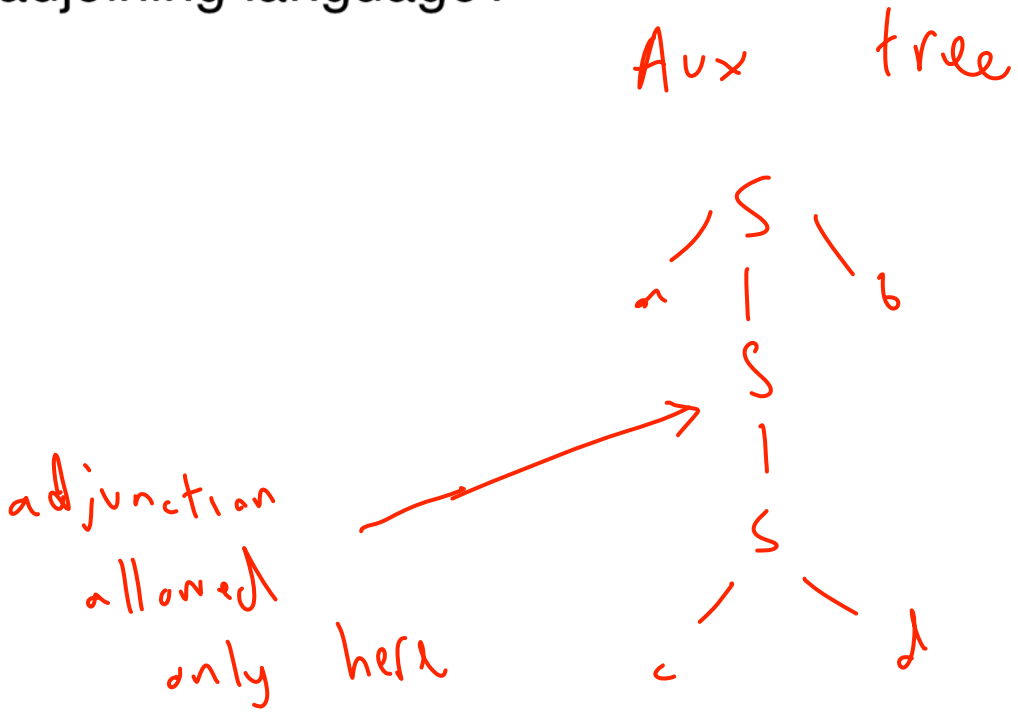
Tree Adjoining Grammars

Another quick question: is $\{a^n b^n c^n d^n \mid n \geq 1\}$ a context-free language? No

Tree Adjoining Grammars

Another quick question: is $\{a^n b^n c^n d^n \mid n \geq 1\}$ a context-free language? NO

Is it a tree adjoining language?



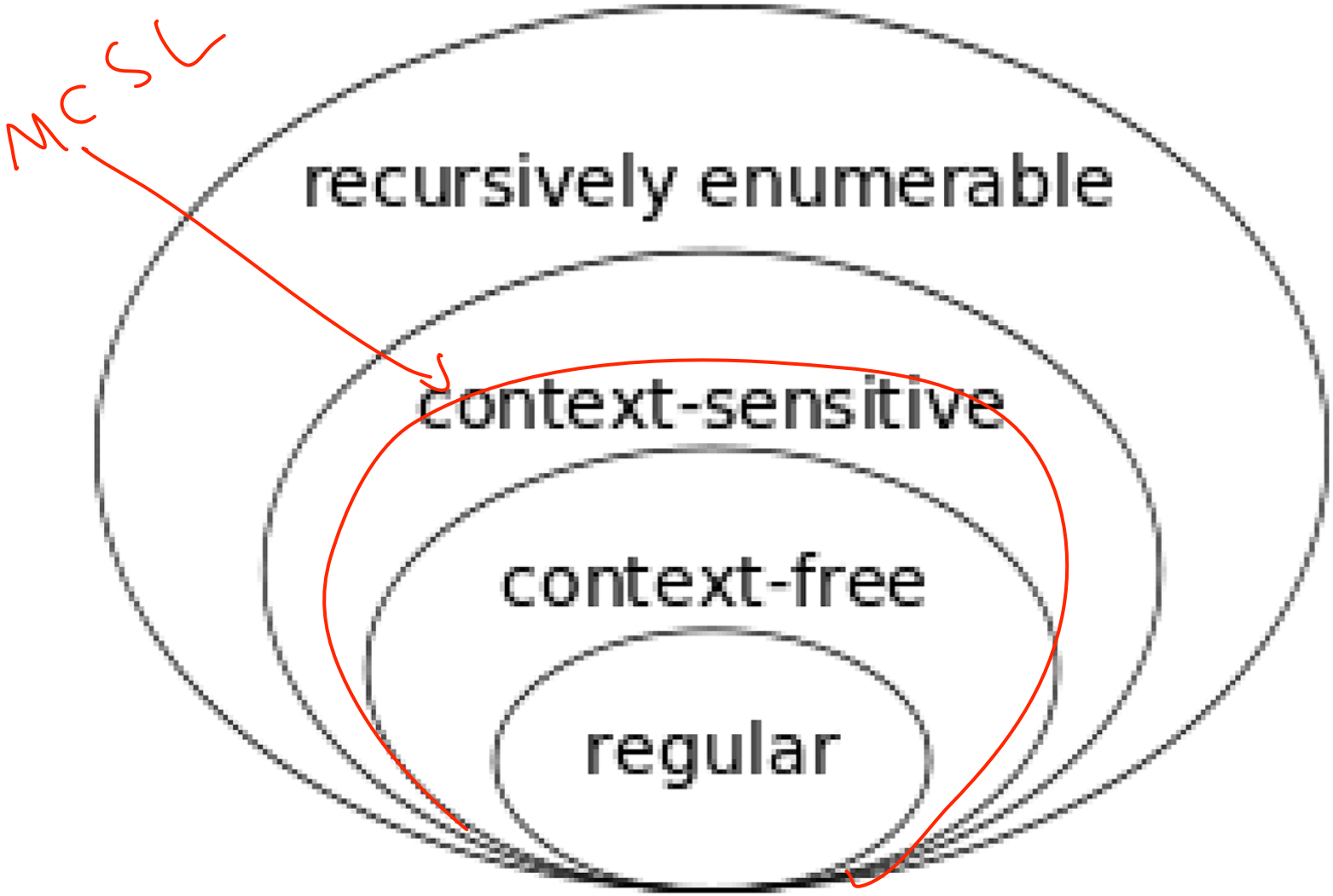
Tree Adjoining Grammars

They add the “minimum needed” in order to capture phenomena such as cross-serial dependencies

They are part of a family of grammar formalisms called “mildly context sensitive”

Other examples which are weakly equivalent: combinatory categorial grammars, head grammars, linear indexed grammars

The Chomsky Hierarchy Plus

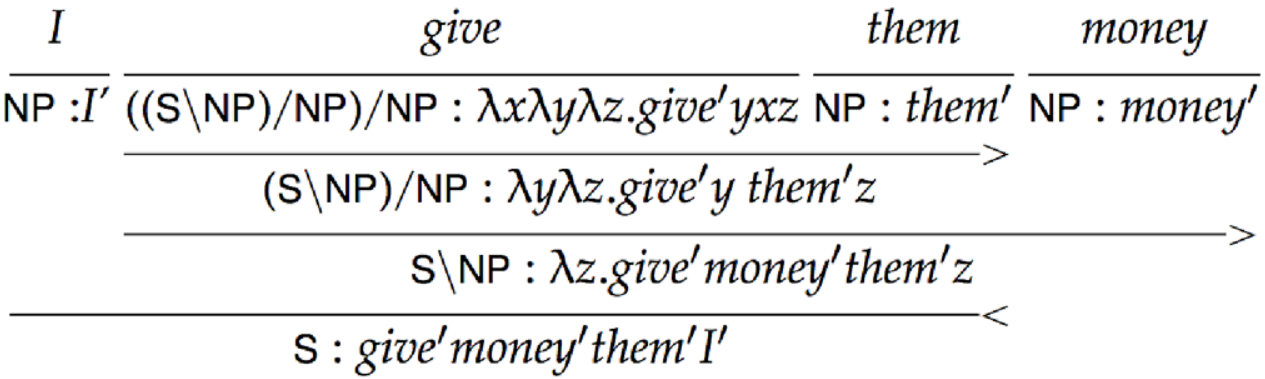


Another Mildly Context-Sensitive Formalism: CCG

Combinatory Categorial Grammars (due to Mark Steedman):

- Give easy access to logical form *semantics*
- Categories are "functions". There are some atomic categories (**NP** for noun phrase, **S** for sentence) and composed ones such as verb: **S** $\backslash$ **NP**: a category that takes NP on the right and gives back an S
- Main operations:
  - Application: $X/Y, Y \rightarrow X$
  - Application: $Y, X\backslash Y \rightarrow X$
  - Composition (forward): $X/Y, Y/Z \rightarrow X/Z$
  - Composition (backward): $X\backslash Y, Y\backslash Z \rightarrow X\backslash Z$
- Steedman (2000) also uses crossed composition, generalised composition, generalised crossed composition and type-raising.

CCG Derivation



(From Hockenmaier and Steedman (2007))

- A CCG consists of a lexicon that attaches each word a category and a semantic attachment (in the form of a λ expression)
- A certain version of CCG is *weakly* equivalent to tree adjoining grammars (Vijay-Shanker and Weir, 1994)

A More Powerful Formalism: LCFRS

Linear Context-Free Rewriting Systems are a more powerful formalism than CCG and TAG, which is still considered mildly context sensitive.

A rewrite rule in an LCFRS can generate several discontinuous strings that can move around in the derivation tree to different places.

A version of LCFRS has been used by Stabler to formalise the minimalist programme of Chomsky, where “movement” of constituents is a central part of the theory

Recipe for Mildly Context-Sensitive Formalisms

A set $\mathcal{L}$ of languages is mildly context-sensitive iff:

- $\mathcal{L}$ contains all context-free languages
- $\mathcal{L}$ can describe cross-serial dependencies: There is an $n \geq 2$ such that $\{w^k \mid w \in \mathcal{T}^*\} \in \mathcal{L}$ for all $k \leq n$
- The languages in $\mathcal{L}$ are polynomially parsable
- The languages in $\mathcal{L}$ have the constant growth property (if we order the words by their length, the length grows in constant steps)

A formalism is mildly context-sensitive iff the set of languages it defines is mildly context-sensitive

Theory of Syntax

Mainstream claim in CL is that mild context-sensitivity in some form is sufficient to capture any natural language, most likely in the form of TAG and CCG.

Just like any other scientific theory, if you want to prove otherwise, you need to falseify this theory by giving an example that shows language is not mildly context-sensitive.

There have been some attempts to construct such counterexamples, but most of them turned out to be either ill-constructed or use wrong linguistic data.

Probabilistic Grammars

We augment the rules with probabilities

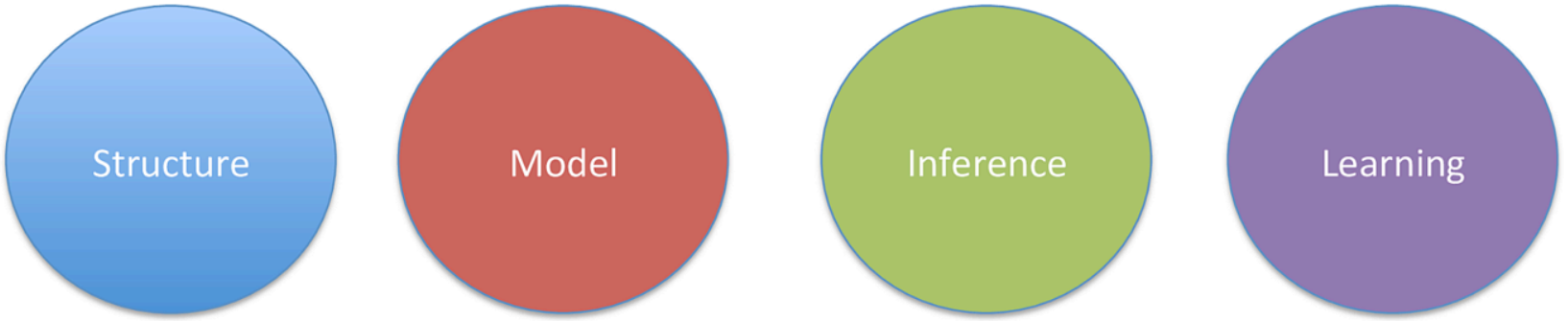
The probability of a derivation is then the product of all rule probabilities:

Often can be thought of as a generative process: we start with the initial symbol and probabilistically choose rules until we reach terminal nodes

There are also weighted versions:

Solving an NLP Problem

When modelling a new problem in NLP, need to address four issues:



Inference

A process in which one has to take a model and an input and predict or calculate some quantity in the output space

Most commonly:

$$y = \arg\max_x p(y \mid x, \theta)$$

or

$$y = \arg\max_x \text{score}(x, y)$$

Types of Inference Algorithms

Types of Inference Algorithms

- Search algorithms
- Dynamic programming
- Sampling algorithms
- Integer linear programming

Search Algorithms

A traditional AI approach to find a solution

There is a “search space,” each element is a node in the graph

There is a “cost” associated with each node

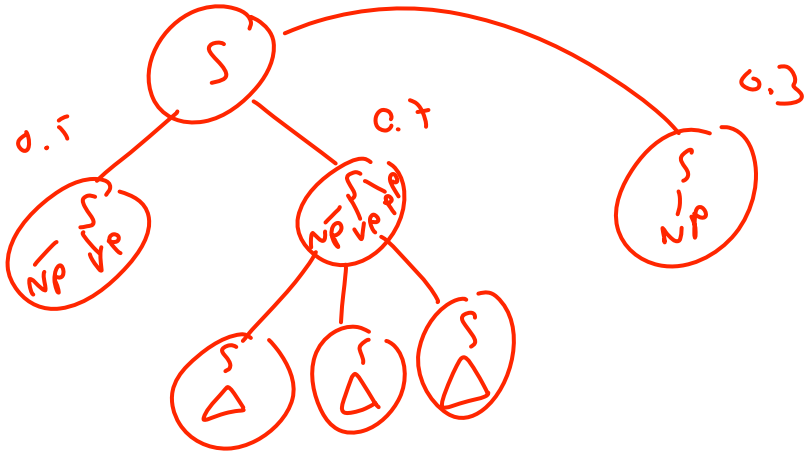
There are edges between these nodes according to simple operators that change one node into another

There is a “goal” node which is the solution we are looking for (with the smallest cost)

There is a strategy to explore the graph and find the goal node

Example Strategy

Best-first search (BFS):



Often coupled with “beam” search

Another Example: A* Search

Exploring the graph by increasing cost. Each step adds to the cost.

Develop paths in the graph using a heuristic of the cost left to reach the goal.

If $h(v)$ is the estimated cost to reach the goal from node v and $f(v)$ is the cost associated with node v, then explore the node with minimum $\min_v f(v) + h(v)$

If $h(v)$ never *overestimates* the cost ("h is admissible") the first time the algorithm finds a goal, that goal is the correct one.

Dynamic Programming Algorithms

- Solve a “bigger” problem by breaking it into “smaller” parts
- The smaller parts are now the bigger problems – work recursively
- Examples: Viterbi algorithm, parsing algorithms such as CKY

The CKY Algorithm and the Inside Algorithm

CKY:

$$\alpha(A, i, j) = \max_{i \le k \le j-1} \max_{A \to B\ C} p(A \to B\ C \mid A)\, \alpha(B, i, k)\, \alpha(C, k+1, j)$$

inside

$$p(x) = \sum_{y} p(y, x)$$

cky. $\max_{y} p(y, x)$

Inside:

$$\alpha(A, i, j) = \sum_{i \le k \le j-1} \sum_{A \to B\ C} p(A \to B\ C \mid A)\, \alpha(B, i, k)\, \alpha(C, k+1, j)$$

Inside and CKY

What is the connection between the inside algorithm and CKY?

CKY:

$$\alpha(A, i, j) = \max_{i \leq k \leq j-1} \max_{A \to B\,C} p(A \to B\,C|A)\alpha(B, i, k)\alpha(C, k+1, j)$$

Inside and CKY

What is the connection between the inside algorithm and CKY?

CKY:

$$\alpha(A,i,j) = \max_{i \leq k \leq j-1} \max_{A \to B\,C} p(A \to B\,C|A)\alpha(B,i,k)\alpha(C,k+1,j)$$

Inside:

$$\alpha(A,i,j) = \sum_{k=i}^{j-1} \sum_{A \to B\,C} p(A \to B\,C|A)\alpha(B,i,k)\alpha(C,k+1,j)$$

The inside algorithm computes the *total probability* of a string – *summing* out all derivations instead of maximising over them

Semirings

What is a semiring?

R

$\oplus$ $\quad a, b \in R \quad a \oplus b$

$\otimes$ $\quad a, b \in R \quad a \otimes b$

$\bar{1}$

$\bar{0}$

Semirings

What is a semiring?

- A set R
- Two operations: $\oplus$ and $\otimes$
- Identity element $\overline{1}$ for $\otimes$
- Identity element $\overline{0}$ for $\oplus$
- (... and a few more important properties)

CKY and Semirings

CKY:

$$\alpha(A, i, j) = \max_{i \leq k \leq j-1} \max_{A \rightarrow B\,C} p(A \rightarrow B\,C | A) \alpha(B, i, k) \alpha(C, k+1, j)$$

What is the semiring?

$\oplus$ $a \oplus b = \max\{a, b\}$

$\otimes$ $a \otimes b = a \cdot b$

$\overline{1}$ 1

$\overline{0}$ 0

Inside Algorithm and Semirings

CKY:

$$\alpha(A,i,j) = \sum_{i \le k \le j-1} \sum_{A \to B\,C} p(A \to B\,C|A)\alpha(B,i,k)\alpha(C,k+1,j)$$

What is the semiring?

$\oplus$ $a \oplus b = a + b$

$\otimes$ $a \otimes b = a \cdot b$

$\overline{1}$ 1

$\overline{0}$ 0

Log-Domain Trick and Semirings

CKY:

$$\alpha(A, i, j) = \overline{\max_{i \leq k \leq j-1}} \max_{A \to B\,C} \log p(A \to B\,C | A) + \alpha(B, i, k) + \alpha(C, k+1, j)$$

What is the semiring? $R = (-\infty, 0]$

$\oplus$ $a \oplus b = \max\{a, b\}$

$\otimes$ $a \otimes b = a \cdot b$

$\overline{1}$ $\overline{1} = \log 1 = 0$

$\overline{0}$ $\overline{0} = \log 0 = -\infty$

Parsing as Weighted Logic Programming

$$\text{constit}(a, i, j) \oplus = \text{constit}(b, i, k) \otimes \text{constit}(c, k+1, j) \otimes \text{rule}(a \rightarrow b\, c)$$

$$\text{constit}(a, i, i) \oplus = \text{rule}(a \rightarrow w)$$

Goal: $\text{constit}(S, 0, n)$

Weighted Logic Programmes

A succinct useful representation for dynamic programming algorithms

It represents inference algorithms in a generic way

Does not commit to a specific “execution model” – but dynamic programming is often used

Example of a Weighted Logic Programme

We are given a sequence $w_1, \ldots, w_n$ of some symbols.

$$\text{prob}(b, i) \oplus = \text{prob}(a, i-1) \otimes \text{transition}(a \to b) \otimes \text{emission}(b, w_i)$$

$$\text{prob}(a, 1) \oplus = \text{start\_state}(b) \otimes \text{emission}(a, w_1)$$

Example of a Weighted Logic Programme

We are given a sequence $w_1, \ldots, w_n$ of some symbols.

$$\text{prob}(b, i) \oplus = \text{prob}(a, i-1) \otimes \text{transition}(a \to b) \otimes \text{emission}(b, w_i)$$

$$\text{prob}(a, 1) \oplus = \text{start\_state}(b) \otimes \text{emission}(a, w_1)$$

Hidden Markov models and the forward algorithm:

- emission are the emission probabilities
- transition are the transition probabilities
- start_state are the initial probabilities
- $\text{prob}(b, i)$ gives the probability $p(w_1, \ldots, w_i, S_i = b)$

Solving Weighted Logic Programmes

- Memoisation and dynamic programming
- Agenda algorithms. Roughly:
  - Keep a queue ("agenda") of unprocessed items and a chart of processed items
  - Dequeue an item x and create or update all items by using information from the chart together with x. Put all the new items in the queue
  - Depending on the order by which we dequeue from the agenda, we might process items several times
  - Examples of agenda priority values: the value of an item, the value of an item with A* heuristic