Basic Parsing with Context-Free Grammars

Some slides adapted from Karl Stratos and from Chris Manning

Announcements

- HW 2 out
- Midterm on 10/19 (see website). Sample questions will be provided.
- Sign up for poll everywhere if you haven't
- Today: dependency parsing

Soundquality on video

- If you encounter poor quality on video of lectures, send email to support@cvn.columbia.edu with the lecture date and timecode where the issue occurs

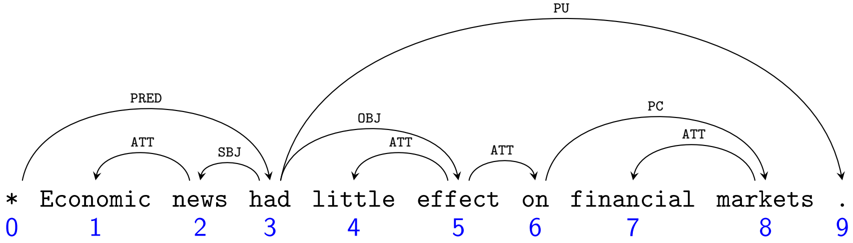
Overview

- Dependency Parsing
- Transition-Based Framework
    - Configuration
    - Transitions
- Transition Systems
    - Arc-Standard
    - Arc-Eager
- Implementation
    - Training
    - Greedy Parser
    - Beam Search Parser

Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

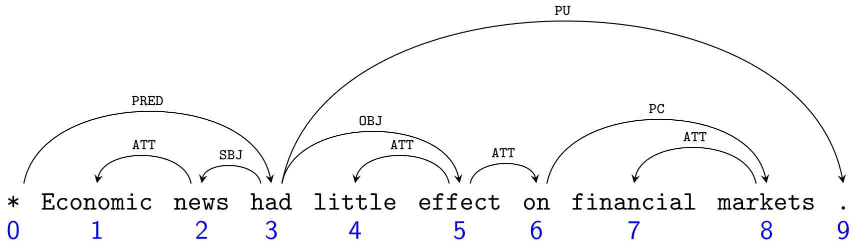
$\Downarrow$



Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

$\Downarrow$



$$A = \{(0, \texttt{PRED}, 3), (3, \texttt{SBJ}, 2), (2, \texttt{ATT}, 1), (3, \texttt{OBJ}, 5),$$
$$(3, \texttt{PU}, 9), (5, \texttt{ATT}, 4), (5, \texttt{ATT}, 6), (6, \texttt{PC}, 8), (8, \texttt{ATT}, 7)\}$$

Valid Dependency Tree

1. (Root): 0 must not have a parent.

$$\begin{tikzcd} 0 & \cdots & i \arrow[ll, bend right, "l"'] \end{tikzcd}$$

2. (Connected): There must be a path from 0 to every $i \in \mathcal{N}$.
3. (Tree): A node must not have more than one parent.

$$\begin{tikzcd} i \arrow[r, bend left, "l"] & j & k \arrow[l, bend right, "l'"'] \end{tikzcd}$$

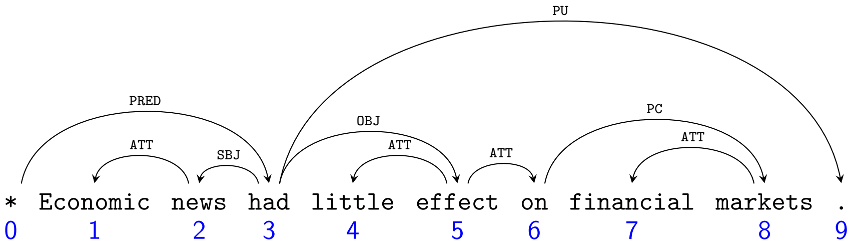
4. (Acyclic): Nodes must not form a cycle.

$$\begin{tikzcd} i_0 \arrow[r, bend left, "l_1"] & \cdots \arrow[r, bend left, "l_{n-2}"] & i_{n-1} \arrow[ll, bend right, "l_{n-1}"'] \end{tikzcd}$$

Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

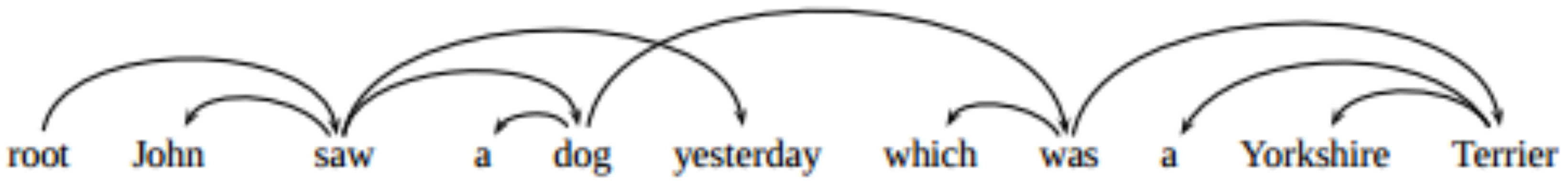
$$\Downarrow$$



$$A = \{(0, \texttt{PRED}, 3), (3, \texttt{SBJ}, 2), (2, \texttt{ATT}, 1), (3, \texttt{OBJ}, 5), (3, \texttt{PU}, 9), (5, \texttt{ATT}, 4), (5, \texttt{ATT}, 6), (6, \texttt{PC}, 8), (8, \texttt{ATT}, 7)\}$$

Projective

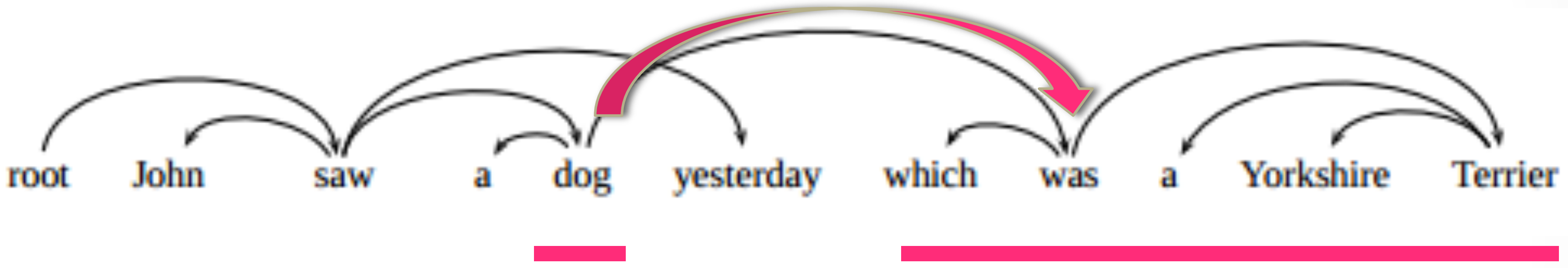
- Can arrows cross -> non-projective



- A valid dependency tree is projective if for every arc (i, l, j) there is a path from I to k for all i<k<j.

Projective

- Can arrows cross -> non-projective

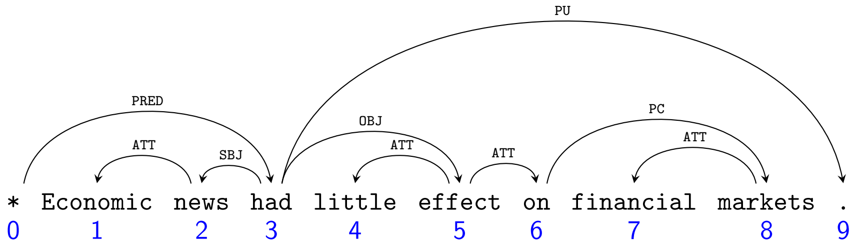


- A valid dependency tree is projective if for every arc (i, l, j) there is a path from I to k for all i<k<j.

Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

$$\Downarrow$$



$$A = \{(0, \texttt{PRED}, 3), (3, \texttt{SBJ}, 2), (2, \texttt{ATT}, 1), (3, \texttt{OBJ}, 5), (3, \texttt{PU}, 9), (5, \texttt{ATT}, 4), (5, \texttt{ATT}, 6), (6, \texttt{PC}, 8), (8, \texttt{ATT}, 7)\}$$

Dependency Parsing = Arc Finding

- Sentence: $x_1 \dots x_m$
- Associated nodes: $\mathcal{N} = \{0, 1, \dots, m\}$
  - Convention: leftmost root 0
- Labels: $L = \{\texttt{PRED}, \texttt{SBJ}, \dots\}$

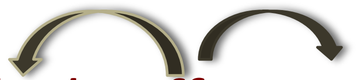
Goal. Find a set of **labeled, directed arcs**

$$A \subseteq \mathcal{N} \times L \times \mathcal{N}$$

that corresponds to a **correct dependency tree** for $x_1 \dots x_m$.

What information useful?

- Lexical affiniities
  - *financial markets*
- Dependency distance
- Intervening material
  - *little* in *had little effect*
  - Not *little gave effect*
- Valency of heads (subcategorization)



  - *little effect on financial markets*

Methods of Dependency Parsing

- Dynamic programming
  Eisner (1996): algorithm with complexity O(n^3) by producing parse items with heads at the end instead of the middle
- Graph algorithms
  Create a Minimum Spanning Tree for a sentence (e..g, McDonald's MSTParser 2005)
- Constraint Satisfaction
  Edges are eliminated that don't satisfy hard constraints (Karlsson 1990
- Transition-based parsing (or deterministic based parsing
  Greedy choice of attachments guided by good machine learning. MaltParser (Nivre 2003)

Slide adapted from Manning

Greedy Transition-based Parsing (Nivre 2003)

- Simple form of greedy discriminative dependency parser
- Bottom-up
- Similar to shift-reduce
- The parser has:
  - A stack , written with top to the right
    - Starts with ROOT
  - A buffer ,written with top to the left
    - Starts with input sentence
  - A set of dependency arcs A
    - Which starts off empty
  - A set of actions

Parser Configuration

Triple $c = (\sigma, \beta, A)$ where

- $\sigma = [\ldots\ i]$: "stack" of $\mathcal{N}$ with i at the top
- $\beta = [i\ \ldots]$: "buffer" of $\mathcal{N}$ with i at the front
- $A \subseteq \mathcal{N} \times L \times \mathcal{N}$: arcs

Notation

- $\mathcal{C}$ denotes the space of all possible configurations.
- $c.\sigma$, $c.\beta$, $c.A$ denote stack, buffer, arcs of $c \in \mathcal{C}$.

Configuration-Based Parsing Scheme

Initial configuration

$$c_0 := ([0], [1 \ldots m], \{\ \})$$

Apply "transitions" until we reach **terminal** c_T (defined later)

$$c_0 \xrightarrow{t_0} c_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{T-1}} c_T$$

and return as a parse

$$c_T.A$$

Shift and Reduce

SHIFT $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$

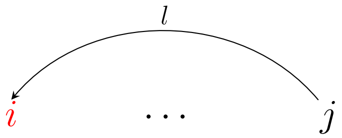
Illegal if β is empty.

REDUCE $(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$

Illegal if i does not have a parent.

Left-Arc

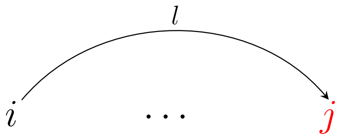
$$\textbf{LEFT}_l \quad (\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j, l, i)\})$$



Illegal if either $i = 0$ or i already has a parent.

Right-Arc

$$\textbf{RIGHT}_l \quad (\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i, l, j)\})$$



Illegal if j already has a parent.

Definition

$2\,|L| + 1$ possible transitions $\mathcal{T}^{\text{std}}$

- **SHIFT**: $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
- **LEFT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j, l, i)\})$$

- **RIGHT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i, l, j)\})$$

Terminal condition: $c.\sigma = [0]$ and $c.\beta = [\,]$

It is time for sleep.
The dogs go to sleep.
They will sleep all night.

Example for arc-standard

- *They sleep all night*

START

[ROOT] They sleep all night

Example for arc-standard

- *They sleep all night*

START

[ROOT] They sleep all night

SHIFT

[ROOT] They sleep all night

Example for arc-standard

- *They sleep all night*

START

[ROOT] They sleep all night

SHIFT

[ROOT] They sleep all night

SHIFT

[ROOT] They sleep all night

Example for arc-standard

- *They sleep all night*

LEFT ARC

[ROOT] They sleep

$\Downarrow$

[ROOT] sleep

all night

Arcs

NSUBJ (sleep -> They

Example for arc-standard

- *They sleep all night*

Arcs

LEFT ARC

[ROOT] They sleep

NSUBJ (sleep -> They

[ROOT] sleep

all night

SHIFT

[ROOT] sleep all

night

Example for arc-standard

- *They sleep all night*

LEFT ARC

[ROOT] They sleep

[ROOT] sleep | all night

SHIFT

[ROOT] sleep all | night

SHIFT

[ROOT] sleep all night |

Arcs

NSUBJ (sleep -> They

Example for arc-standard

- *They sleep all night*

LEFT ARC

[ROOT] sleep all night

LEFT ARC

[ROOT] sleep night

Arcs

NSUBJ (sleep -> They)

ATT (night -> all)

Example for arc-standard

- *They sleep all night*

Arcs

LEFT ARC

NSUBJ (sleep -> They)

[ROOT] sleep all night

LEFT ARC

ATT (night -> all)

[ROOT] sleep night

RIGHT ARC

OBJ(sleep -> night)

[ROOT] sleep

Example for arc-standard

- *They sleep all night*

LEFT ARC

[ROOT] sleep all night

LEFT ARC

[ROOT] sleep night

RIGHT ARC

[ROOT] sleep

RIGHT ARC

{ROOT}

Arcs

NSUBJ (sleep -> They)

ATT (night -> all)

OBJ(sleep -> night)

PRED (ROOT -> sleep)

FINiSH

Definition

$2\,|L|+1$ possible transitions $\mathcal{T}^{\text{std}}$

- **SHIFT**: $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
- **LEFT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j, l, i)\})$$

- **RIGHT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i, l, j)\})$$

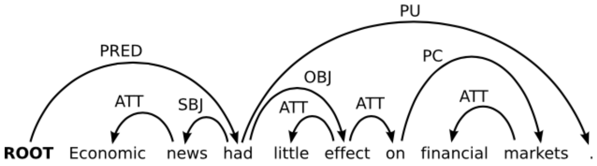
Terminal condition: $c.\sigma = [0]$ and $c.\beta = [\,]$

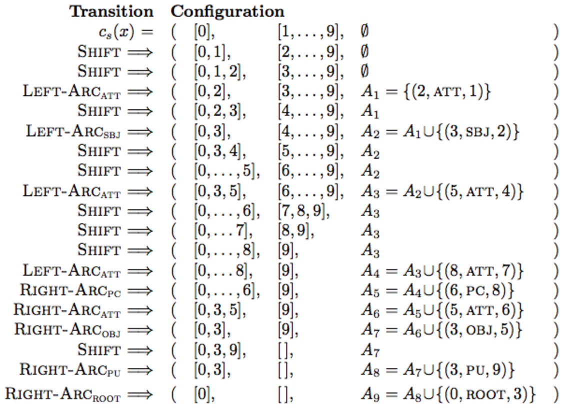
Properties

- Makes **exactly** $2m$ transitions to parse $x_1 \dots x_m$. Why?
- **Bottom-up**: a node must collect all its children before getting a parent. Why?
- **Sound**: if c is terminal, $c.A$ forms a valid projective tree.
- **Complete**: every valid projective tree A can be produced from c_0 by some sequence of transitions $t_0 \dots t_{T-1} \in \mathcal{T}^{\text{std}}$.

$$
\begin{aligned}
t_i &= \mathsf{Oracle}^{\text{std}}(c_i) \\
c_{i+1} &= t_i(c_i)
\end{aligned}
$$

Example Parse (Nivre 2013)





Transition	**Configuration**		
$c_s(x) =$	($[0]$,	$[1, \ldots, 9]$,	$\emptyset$)
$\textsc{Shift} \Longrightarrow$	($[0, 1]$,	$[2, \ldots, 9]$,	$\emptyset$)
$\textsc{Shift} \Longrightarrow$	($[0, 1, 2]$,	$[3, \ldots, 9]$,	$\emptyset$)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 2]$,	$[3, \ldots, 9]$,	$A_1 = \{(2, \textsc{att}, 1)\}$)
$\textsc{Shift} \Longrightarrow$	($[0, 2, 3]$,	$[4, \ldots, 9]$,	A_1)
$\textsc{Left-Arc}_{\textsc{sbj}} \Longrightarrow$	($[0, 3]$,	$[4, \ldots, 9]$,	$A_2 = A_1 \cup \{(3, \textsc{sbj}, 2)\}$)
$\textsc{Shift} \Longrightarrow$	($[0, 3, 4]$,	$[5, \ldots, 9]$,	A_2)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 5]$,	$[6, \ldots, 9]$,	A_2)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 3, 5]$,	$[6, \ldots, 9]$,	$A_3 = A_2 \cup \{(5, \textsc{att}, 4)\}$)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 6]$,	$[7, 8, 9]$,	A_3)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots 7]$,	$[8, 9]$,	A_3)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 8]$,	$[9]$,	A_3)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, \ldots 8]$,	$[9]$,	$A_4 = A_3 \cup \{(8, \textsc{att}, 7)\}$)
$\textsc{Right-Arc}_{\textsc{pc}} \Longrightarrow$	($[0, \ldots, 6]$,	$[9]$,	$A_5 = A_4 \cup \{(6, \textsc{pc}, 8)\}$)
$\textsc{Right-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 3, 5]$,	$[9]$,	$A_6 = A_5 \cup \{(5, \textsc{att}, 6)\}$)
$\textsc{Right-Arc}_{\textsc{obj}} \Longrightarrow$	($[0, 3]$,	$[9]$,	$A_7 = A_6 \cup \{(3, \textsc{obj}, 5)\}$)
$\textsc{Shift} \Longrightarrow$	($[0, 3, 9]$,	$[\,]$,	A_7)
$\textsc{Right-Arc}_{\textsc{pu}} \Longrightarrow$	($[0, 3]$,	$[\,]$,	$A_8 = A_7 \cup \{(3, \textsc{pu}, 9)\}$)
$\textsc{Right-Arc}_{\textsc{root}} \Longrightarrow$	($[0]$,	$[\,]$,	$A_9 = A_8 \cup \{(0, \textsc{root}, 3)\}$)

How do we decide the next arc?

- Each action can be predicted by a discriminative classifier over each legal move (Nivre and Hall 2005)
    - SVM
    - Max of 3 untyped choices; max of |R| X 2 +1 when typed where R is # dependency labels
    - Features: top of stack word, top of stack POS what else?
- No search in greedy search
  - Could do beam search
- It provides VERY fast linear time parsing
- The model's accuracy is slightly below the best parser
- It provides fast, close to state of the art parsing

It is time for sleep.
The dogs go to sleep.
They will sleep all night.

Overview

- Dependency Parsing
- Transition-Based Framework
    - Configuration
    - Transitions
- Transition Systems
    - Arc-Standard
    - Arc-Eager
- Implementation
    - Training
    - Greedy Parser
    - Beam Search Parser

Getting Training Data

- **Treebank**: sentence-tree pairs $(x^{(1)}, A^{(1)}) \ldots (x^{(M)}, A^{(M)})$
  - Assume all projective

- For each $A^{(j)}$, use an oracle to extract

$$(c_0^{(j)}, t_0^{(j)}) \ldots (c_{T-1}^{(j)}, t_{T-1}^{(j)})$$

where $t_{T-1}^{(j)}(c_{T-1}^{(j)}).A = A^{(j)}$.

- We can now use this to train a **classifier**

$$(x^{(j)}, c_i^{(j)}) \mapsto t_i^{(j)}$$

Oracle$^{\text{std}}$

Input: gold arcs A^{gold}, non-terminal configuration $c = (\sigma, \beta, A)$
Output: transition $t \in \mathcal{T}^{\text{std}}$ to apply on c

1. Return **SHIFT** if $|\sigma| = 1$.
2. Otherwise $\sigma = [\ldots\ i\ j]$ for some $i < j$:
   2.1 Return **LEFT**$_l$ if $(j, l, i) \in A^{\text{gold}}$.
   2.2 Return **RIGHT**$_l$ if $(i, l, j) \in A^{\text{gold}}$ and for all $l' \in L, j' \in \mathcal{N}$,

$$(j, l', j') \in A^{\text{gold}} \qquad \Rightarrow \qquad (j, l', j') \in A$$

   2.3 Return **SHIFT** otherwise.

Linear Classifier

- Parameters: $w_t \in \mathbb{R}^d$ for each $t \in \mathcal{T}$

- Each $c \in \mathcal{C}$ for sentence x is "featurized" as $\phi^x(c) \in \mathbb{R}^d$.
  - Classical approach: **binary features** providing useful signals
  - Assumes we have access to POS tags of $x_1 \ldots x_m$.

$$\phi^x_{20134}(c) := \begin{cases} 1 & \text{if } x_{c.\sigma[0]}.\text{POS} = \texttt{NN} \text{ and } x_{c.\beta[0]}.\text{POS} = \texttt{VBD} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi^x_{1988}(c) := \begin{cases} 1 & \text{if } x_{c.\sigma[0]}.\text{POS} = \texttt{VBD} \text{ with leftmost arc } \texttt{SUBJ} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi^x_{42}(c) := \begin{cases} 1 & \text{if } x_{c.\beta[1]} = \texttt{cat} \\ 0 & \text{otherwise} \end{cases}$$

Linear Classifier (Continued)

- **Score** of $t \in \mathcal{T}$ at $c \in \mathcal{C}$ for x:

$$\begin{aligned} \mathsf{score}_x(t|c) &:= w_t \cdot \phi^x(c) \\ &= \sum_{i=1:\ \phi_i^x(c)=1}^{d} [w_t]_i \end{aligned}$$

- From here on, we assume $\{w_t\}_{t \in \mathcal{T}}$ trained from data.

Important Aside

Each c_i is computed from **past decisions** $t_0 \ldots t_{i-1}$.

$$c_i = t_{i-1}(t_{i-2}(\cdots t_0(c_0)))$$

So the score function on c_i is really a **function of** $t_0 \ldots t_{i-1}$.

$$\text{score}_x(t|c) = \text{score}_x(t|t_1 \ldots t_{i-1})$$

Will use c_i and $t_0 \ldots t_{i-1}$ interchangeably.

Overview

- Dependency Parsing
- Transition-Based Framework
  - Configuration
  - Transitions
- Transition Systems
  - Arc-Standard
  - Arc-Eager
- Implementation
  - Training
  - Greedy Parser
  - Beam Search Parser

Greedy

At each configuration c_i, pick

$$t_i \leftarrow \arg\max_{t \in \texttt{LEGAL}(c_i)} \textsf{score}_x(t | t_0 \ldots t_{i-1})$$

Parsing Algorithm

Input: $\{w_t\}_{t \in \mathcal{T}}$, sentence x of length m
Output: arcs representing a dependency tree for x

```
1. $c \leftarrow c_0$
2. While $c.\beta \neq [\,]$,
    2.1 Select
        $$\hat{t} \leftarrow \arg\max_{t \in \texttt{LEGAL}(c)} \textsf{score}_x(t|c)$$
    2.2 Make a transition: $c \leftarrow \hat{t}(c)$.
3. Return $c.A$.
```