A dependency parse

A peculiarity of the dependency structure in figure 1.1 is that we have inserted an artificial word root before the first word of the sentence. This is a mere technicality, which simplifies both formal definitions and computational implementations. In particular, we can normally assume that every real word of the sentence should have a syntactic head. Thus, instead of saying that the verb had lacks a syntactic head, we can say that it is a dependent of the artificial word root. In chapter 2, we will define dependency structures formally as labeled directed graphs, where nodes correspond to words (including root) and labeled arcs correspond to typed dependency relations.

The information encoded in a dependency structure representation is different from the information captured in a phrase structure representation, which is the most widely used type of syntactic representation in both theoretical and computational linguistics. This can be seen by comparing the dependency structure in figure 1.1 to a typical phrase structure representation for the same sentence, shown in figure 1.2. While the dependency structure represents head-dependent relations between words, classified by functional categories such as subject (SBJ) and object (OBJ), the phrase structure represents the grouping of words into phrases, classified by structural categories such as noun phrase (NP) and verb phrase (VP).
Different kinds of dependencies

Head-argument: *eat sushi*
Arguments may be obligatory, but can only occur once. The head alone cannot necessarily replace the construction.

Head-modifier: *fresh sushi*
Modifiers are optional, and can occur more than once. The head alone can replace the entire construction.

Head-specifier: *the sushi*
Between function words (e.g. prepositions, determiners) and their arguments. Syntactic head ≠ semantic head

Coordination: *sushi and sashimi*
Unclear where the head is.

What is a dependency?

Dependencies are (labeled) asymmetrical binary relations between two lexical items (words).

There is a syntactic relation between a head H and a dependent D in a construction C if:
- the head H determines the syntactic category of the construction C.
- the head H determines the semantic category of the construction C; D gives semantic specification.
- the head H is obligatory. D may be optional.
- the head selects D and determines whether D is obligatory or not.
- The form of D depends on the head H (agreement)
- The linear position of D depends on the head H.

Dependency structures

Dependencies form a graph over the words in a sentence.

This graph is connected (every word is a node) and (typically) acyclic (no loops).

Single-head constraint:
Every node has at most one incoming edge. This implies that the graph is a rooted tree.

From CFGs to dependencies

Assume each CFG rule has one head child (bolded) The other children are dependents of the head.

```
S                → NP VP
VP                → V NP NP
NP                → DT NOUN
NOUN              → ADJ N
```

The headword of a constituent is the terminal that is reached by recursively following the head child.
(here, V is the head word of S, and N is the head word of NP).

If in rule XP → X Y, X is head child and Y dependent, the headword of Y depends on the headword of X.

The maximal projection of a terminal w is the highest nonterminal in the tree that w is headword of.
Here, Y is a maximal projection.
Context-free grammars

CFGs capture only **nested** dependencies

- The dependency graph is a **tree**
- The dependencies **do not cross**

Beyond CFGs: Nonprojective dependencies

Dependencies: **tree with crossing branches**

- (Non-local) **scrambling** (free word order languages)
  *Die Pizza hat Klaus versprochen zu bringen*
- **Extraposition** (The **guy is coming who is wearing a hat**)
- **Topicalization** (*Cheeseburgers, I thought he likes*)

Dependency Treebanks

Dependency treebanks exist for many languages:

- Czech
- Arabic
- Turkish
- Danish
- Portuguese
- Estonian

Phrase-structure treebanks (e.g. the Penn Treebank) can also be translated into dependency trees (although there might be noise in the translation)

The Prague Dependency Treebank

Three levels of annotation:

- **morphological**: [≤2M tokens]
  - Lemma (dictionary form) + detailed analysis
  - (15 categories with many possible values = 4,257 tags)

- **surface-syntactic (“analytical”)**: [1.5M tokens]
  - Labeled dependency tree encoding grammatical functions (subject, object, conjunct, etc.)

- **semantic (“tectogrammatical”)**: [0.8M tokens]
  - Labeled dependency tree for predicate-argument structure, information structure, coreference (not all words included)
  - (39 labels: agent, patient, origin, effect, manner, etc....)
Examples: analytical level

Turkish is an agglutinative language with free word order.
Rich morphological annotations
 Dependencies (next slide) are at the morpheme level

- iyileştirilen
- (literally) while it is being caused to become good
- while it is being improved
- iyí+Adj *DB+Verb+Become*DB+Verb+Caus
  *DB+Verb+Pass+Pos+Pres*DB+Adverb+While

Very small -- about 5000 sentences

METU-Sabanci Turkish Treebank

Universal Dependencies

37 syntactic relations, intended to be applicable to all languages ("universal"), with slight modifications for each specific language, if necessary.
http://universaldependencies.org
Universal Dependency Relations

**Nominal core arguments**: nsubj (nominal subject), obj (direct object), iobj (indirect object)
**Clausal core arguments**: csubj (clausal subject), ccomp (clausal object ["complement"])
**Non-core dependents**: advcl (adverbial clause modifier), aux (auxiliary verb),
**Nominal dependents**: nmod (nominal modifier), amod (adjectival modifier),
**Coordination**: cc (coordinating conjunction), conj (conjunct)

and many more...

Parsing algorithms for DG

‘Transition-based’ parsers:
learn a sequence of actions to parse sentences
**Models:**
State = stack of partially processed items
+ queue/buffer of remaining tokens
+ set of dependency arcs that have been found already
Transitions (actions) = add dependency arcs; stack/queue operations

‘Graph-based’ parsers:
learn a model over dependency graphs
**Models:**
a function (typically sum) of local attachment scores
For dependency trees, you can use a minimum spanning tree algorithm

Transition-based parsing

Transition-based shift-reduce parsing processes the sentence $S = w_0 w_1 ... w_n$ from left to right.
Unlike CKY, it constructs a **single tree**.
N.B: this only works for projective dependency trees

**Notation:**
$w_0$ is a special ROOT token.
$V_S = \{w_0, w_1, ..., w_n\}$ is the vocabulary of the sentence
$R$ is a set of dependency relations

The parser uses three data structures:
- $\sigma$: a **stack** of partially processed words $w_i \in V_S$
- $\beta$: a **buffer** of remaining input words $w_i \in V_S$
- $A$: a **set of dependency arcs** $(w_i, r, w_j) \in V_S \times R \times V_S$
Parser configurations \((\sigma, \beta, A)\)

The **stack** \(\sigma\) is a list of partially processed words
We push and pop words onto/off of \(\sigma\).
\(\sigma|w : w\) is on top of the stack.
Words on the stack are not (yet) attached to any other words.
Once we attach \(w\), \(w\) can’t be put back onto the stack again.

The **buffer** \(\beta\) is the remaining input words
We read words from \(\beta\) (left-to-right) and push them onto \(\sigma\)
\(w|\beta : w\) is on top of the buffer.

The **set of arcs** \(A\) defines the current tree.
We can add new arcs to \(A\) by attaching the word on top of the stack to the word on top of the buffer, or vice versa.

Parser actions

\((\sigma, \beta, A)\): Parser configuration with stack \(\sigma\), buffer \(\beta\), set of arcs \(A\)
\((w, r, w')\): Dependency with head \(w\), relation \(r\) and dependent \(w'\)

**SHIFT:** Push the next input word \(w_i\) from the buffer \(\beta\) onto the stack \(\sigma\)
\((\sigma, w_i|\beta, A) \Rightarrow (\sigma|w_i, \beta, A)\)

**LEFT-ARC:** \(\ldots w_i \ldots w_j \ldots\) (dependent precedes the head)
Attach dependent \(w_i\) (top of stack \(\sigma\)) to head \(w_j\) (top of buffer \(\beta\))
with relation \(r\) from \(w_j\) to \(w_i\). Pop \(w_i\) off the stack.
\((\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})\)

**RIGHT-ARC:** \(\ldots w_i \ldots w_j \ldots\) (dependent follows the head)
Attach dependent \(w_j\) (top of buffer \(\beta\)) to head \(w_i\) (top of stack \(\sigma\))
with relation \(r\) from \(w_i\) to \(w_j\). Move \(w_i\) back to the buffer.
\((\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})\)

An example sentence & parse
Economic news had little effect on financial markets.

Transition Configuration

| (root), [Economic,...], $\emptyset$ |

SH $\Rightarrow$ (root, Economic), [news,...], $\emptyset$

LA_{ATT} $\Rightarrow$ (root), [news,...], $A_1 = \{(news, ATT, Economic)\}$

Economic news had little effect on financial markets.

Transition Configuration

| (root), [Economic,...], $\emptyset$ |

SH $\Rightarrow$ (root, Economic), [news,...], $\emptyset$

LA_{ATT} $\Rightarrow$ (root), [news,...], $A_1 = \{(news, ATT, Economic)\}$
Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH (\rightarrow) [(root), (root, Economic), [Economic, ...]], (\emptyset)</td>
</tr>
<tr>
<td>LA(_ATT) (\rightarrow) [(root), [Economic, ...]], (\emptyset)</td>
</tr>
<tr>
<td>SH (\rightarrow) [(root, news), [news, ...]], (A_1 = {([\text{news}, ATT, Economic])})</td>
</tr>
<tr>
<td>LA(_SS) (\rightarrow) [(root), [had, ...]], (A_2 = A_1 \cup ([\text{had}, SBJ, news])})</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH (\rightarrow) [(root), (root, Economic), [Economic, ...]], (\emptyset)</td>
</tr>
<tr>
<td>LA(_ATT) (\rightarrow) [(root), [Economic, ...]], (A_1 = {([\text{news}, ATT, Economic])})</td>
</tr>
<tr>
<td>SH (\rightarrow) [(root, news), [had, ...]], (A_1})</td>
</tr>
<tr>
<td>LA(_SS) (\rightarrow) [(root), [had, ...]], (A_2 = A_1 \cup ([\text{had}, SBJ, news])})</td>
</tr>
<tr>
<td>SH (\rightarrow) [(root, had), [little, ...]], (A_2})</td>
</tr>
<tr>
<td>LA(_ATT) (\rightarrow) [(root, had), [effect, ...]], (A_1 = A_2 \cup ([\text{effect}, ATT, little})})</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

### Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇔ (root, Economic), (news, …), θ</td>
<td>(root, Economic, …), θ</td>
</tr>
<tr>
<td>LATT ⇒ (root, Economic), (news, …), θ</td>
<td>(root, Economic, …), θ</td>
</tr>
<tr>
<td>SH ⇒ (root, news), (had, …), θ</td>
<td>( A_1 = { \text{news, ATT, Economic} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, news), (had, …), θ</td>
<td>( A_1 = { \text{news, ATT, Economic} } )</td>
</tr>
<tr>
<td>SH ⇒ (root, had), (little, …), θ</td>
<td>( A_2 = A_1 \cup { \text{had, SBJ, news} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, had), (effect, …), θ</td>
<td>( A_2 = A_1 \cup { \text{had, SBJ, news} } )</td>
</tr>
<tr>
<td>SH ⇒ (root, had, little), (effect, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, had, little), (effect, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>SH ⇒ (root, had, effect), (on, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, had, effect), (on, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>SH ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>SH ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
<tr>
<td>LATT ⇒ (root, … on), (financial, markets, …), θ</td>
<td>( A_3 = A_2 \cup { \text{effect, ATT, little} } )</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH (\Rightarrow) ([root], [Economic, ...], \emptyset)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root], [news, ...], A_1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, news], [had, ...], A_1)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root], [had, ...], A_2 = A_1 \cup {had, SBJ, news})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, had], [little, ...], A_2)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root, had, effect], [effect, ...], A_2)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, ... on], [financial, markets, ...], A_3)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root, ... on], [markets, ...], A_4 = A_3 \cup {markets, ATT, financial})</td>
</tr>
<tr>
<td>RA_{EC} (\Rightarrow) ([root, had, effect], [on, ...], A_5 = A_4 \cup {on, PC, markets})</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH (\Rightarrow) ([root], [Economic, ...], \emptyset)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root], [news, ...], A_1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, news], [had, ...], A_1)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root], [had, ...], A_2 = A_1 \cup {had, SBJ, news})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, had], [little, ...], A_2)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, had, little], [effect, ...], A_2)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root, had, effect], [effect, ...], A_3 = A_2 \cup {effect, ATT, little})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, had, effect], [on, ...], A_3)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, ... on], [financial, markets, ...], A_3)</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, ... financial], [markets, ...], A_5)</td>
</tr>
<tr>
<td>LA_{ATT} (\Rightarrow) ([root, ... on], [markets, ...], A_4 = A_3 \cup {markets, ATT, financial})</td>
</tr>
<tr>
<td>RA_{EC} (\Rightarrow) ([root, had, effect], [on, ...], A_5 = A_4 \cup {on, PC, markets})</td>
</tr>
<tr>
<td>RA_{ATT} (\Rightarrow) ([root, had], [effect, ...], A_6 = A_5 \cup {effect, ATT, on})</td>
</tr>
<tr>
<td>RA_{SBF} (\Rightarrow) ([root], [had, ...], A_7 = A_6 \cup {had, OBJ, effect})</td>
</tr>
<tr>
<td>SH (\Rightarrow) ([root, had], [], A_7)</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

### Transition-based parsing: assumptions

This algorithm works for projective dependency trees.

**Dependency tree:**

Each word has a single parent

(Each word is a dependent of [is attached to] one other word)

**Projective dependencies:**

There are no crossing dependencies.

For any \(i, j, k\) with \(i < k < j\) if there is a dependency between \(w_i\) and \(w_j\), the parent of \(w_k\) is a word \(w_i\) between (possibly including) \(i\) and \(j\); \(i \leq l \leq j\), while any child \(w_m\) of \(w_k\) has to occur between (excluding) \(i\) and \(j\); \(i < m < j\)

---

**Configuration:**

- **SH** (Shift): Push the next word to the buffer.
- **LA** (Left-Arc): Form a new dependency by adjoining the current word to the top of the stack.
- **RA** (Right-Arc): Form a new dependency by adjoining the current word to the top of the stack.
- **ATT** (Add-ATT): Adjoin the current word to the top of the stack.
- **PRED** (Add-PRED): Add a new dependency to the stack.

**Example configurations:**

- **Top configuration:** \([\text{Economic}, \ldots], \emptyset\)
- **Bottom configuration:** \([\text{Economic}, \ldots], \emptyset\)

---

**Example of parsing:**

1. **Initial configuration:** \([\text{Economic}, \ldots], \emptyset\)
2. **Transition sequence:** SH, LA, RA, ATT
3. **Final configuration:** \([\text{Economic}, \ldots], \emptyset\)
Transition-based parsing

We process the sentence \( S = w_0 w_1 ... w_n \) from left to right (“incremental parsing”)

In the parser configuration \((\sigma | w_i, w_j | \beta, A)\):
- \( w_i \) is on top of the stack. \( w_i \) may have some children
- \( w_j \) is on top of the buffer. \( w_j \) may have some children
- \( w_i \) precedes \( w_j \) (\( i < j \))

We have to either attach \( w_i \) to \( w_j \), attach \( w_j \) to \( w_i \), or decide that there is no dependency between \( w_i \) and \( w_j \).
If we reach \((\sigma | w_i, w_j | \beta, A)\), all words \( w_k \) with \( i < k < j \) have already been attached to a parent \( w_m \) with \( i \leq m \leq j \).

Transition-based parsing in practice

Which action should the parser take under the current configuration?

We also need a parsing model that assigns a score to each possible action given a current configuration.
- Possible actions:
  - SHIFT, and for any relation \( r \): LEFT-ARC\(_r\), or RIGHT-ARC\(_r\)
- Possible features of the current configuration:
  - The top \( \{1,2,3\} \) words on the buffer and on the stack, their POS tags, etc.

We can learn this model from a dependency treebank.