INF5830
Classical Approaches to Dependency Parsing

Andrey Kutuzov

With thanks to Lilja Øvrelid, Sandra Kübler and Joakim Nivre
Contents

Introduction

General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
This lecture

- Transition-based dependency parsing:
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
  - Parsing algorithms;
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
  - Parsing algorithms;
  - Machine learning;
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
  - Parsing algorithms;
  - Machine learning;

- Available resources:
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
  - Parsing algorithms;
  - Machine learning;

- Available resources:
  - Treebanks;
This lecture

- Transition-based dependency parsing:
  - Syntactic representations;
  - Parsing algorithms;
  - Machine learning;

- Available resources:
  - Treebanks;
  - Parsers.
Contents

Introduction

General issues

Data-driven dependency parsing

Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features

Advanced transition parsers

Non-Projective Dependency Parsing

Treebanks

Parsers

Summary

References
The task of syntactic parsing

▶ How we can automatically produce dependency structures?
The task of syntactic parsing

- How we can automatically produce dependency structures?
- ...or automatically determine the syntactic structure for a given sentence.
The task of syntactic parsing

- How we can **automatically** produce dependency structures?
- ...or automatically determine the syntactic structure for a given sentence.
- Traditionally (for phrase-structure grammars):
The task of syntactic parsing

- How we can automatically produce dependency structures?
- ...or automatically determine the syntactic structure for a given sentence.
- Traditionally (for phrase-structure grammars):
  - search through all possible trees for a sentence;
The task of syntactic parsing

- How we can **automatically** produce dependency structures?
- ...or automatically determine the syntactic structure for a given sentence.
- Traditionally (for phrase-structure grammars):
  - search through all possible trees for a sentence;
  - bottom-up vs top-down approaches.

- Natural languages are very ambiguous.

- Natural languages are very ambiguous.
- Often there is more than one possible structure for a sentence.

- Natural languages are very ambiguous.
- Often there is more than one possible structure for a sentence.
- A very common problem.

- Natural languages are very ambiguous.
- Often there is more than one possible structure for a sentence.
- A very common problem.
- In fact, a long sentence can have thousands of possible syntactic structures (parsings).
### Why difficult? Ambiguity.

- Natural languages are very ambiguous.
- Often there is more than one possible structure for a sentence.
- A very common problem.
- In fact, a long sentence can have thousands of possible syntactic structures (parsings).

#### PoS-ambiguities

<table>
<thead>
<tr>
<th>VB</th>
<th>VBZ</th>
<th>VBP</th>
<th>VBZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>NNS</td>
<td>NN</td>
<td>NNS</td>
</tr>
<tr>
<td>Fed</td>
<td>raises</td>
<td>interest</td>
<td>rates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.5 %</td>
</tr>
</tbody>
</table>

#### Attachment ambiguities

in effort to control inflation
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
- Humans resolve this ambiguity almost without noticing it.
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
- Humans resolve this ambiguity almost without noticing it.
- For computers, this is a challenge.
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
- Humans resolve this ambiguity almost without noticing it.
- For computers, this is a challenge.
- These parsings are still possible and an automatic parser must somehow discard them.
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
- Humans resolve this ambiguity almost without noticing it.
- For computers, this is a challenge.
- These parsings are still possible and an automatic parser must somehow discard them.
- Multiple ambiguities lead to a combinatorial explosion in the number of possible structures for a sentence.
Ambiguity

- The vast majority of these hypothetical structures are wildly implausible.
- Humans resolve this ambiguity almost without noticing it.
- For computers, this is a challenge.
- These parseings are still possible and an automatic parser must somehow discard them.
- Multiple ambiguities lead to a combinatorial explosion in the number of possible structures for a sentence.
- A natural language parser has to search through all of these alternatives, and find the most plausible structure given the context.
Text parsing

- Goal: parse unrestricted text in natural language.
Text parsing

► Goal: parse unrestricted text in natural language.
  ► Given a text \( T = (x_1, \ldots, x_2) \) in language \( L \), derive the correct analysis for every sentence \( x_i \in T \).
Text parsing

- Goal: parse unrestricted text in natural language.
  - Given a text $T = (x_1, \ldots, x_n)$ in language $L$, derive the correct analysis for every sentence $x_i \in T$.

- Challenges:
  - robustness: at least one analysis;
Text parsing

Goal: parse unrestricted text in natural language.

Given a text \( T = (x_1, \ldots, x_2) \) in language \( L \), derive the correct analysis for every sentence \( x_i \in T \).

Challenges:

- robustness: at least one analysis;
- disambiguation: at most one analysis;
Text parsing

- **Goal**: parse unrestricted text in natural language.
  - Given a text \( T = (x_1, \ldots, x_2) \) in language \( L \), derive the correct analysis for every sentence \( x_i \in T \).

- **Challenges**:
  - **robustness**: at least one analysis;
  - **disambiguation**: at most one analysis;
  - **accuracy**: correct analysis (for every sentence);
Text parsing

- **Goal:** parse unrestricted text in natural language.
  - Given a text $T = (x_1, \ldots, x_n)$ in language $L$, derive the correct analysis for every sentence $x_i \in T$.

- **Challenges:**
  - **robustness:** at least one analysis;
  - **disambiguation:** at most one analysis;
  - **accuracy:** correct analysis (for every sentence);
  - **efficiency:** reasonable time and memory usage.
Text parsing

- **Goal:** parse unrestricted text in natural language.
  - Given a text $T = (x_1, \ldots, x_n)$ in language $L$, derive the correct analysis for every sentence $x_i \in T$.

- **Challenges:**
  - **robustness:** at least one analysis;
  - **disambiguation:** at most one analysis;
  - **accuracy:** correct analysis (for every sentence);
  - **efficiency:** reasonable time and memory usage

- **Two different methodological strategies:**
Text parsing

- **Goal**: parse unrestricted text in natural language.
  - Given a text $T = (x_1, \ldots, x_2)$ in language $L$, derive the correct analysis for every sentence $x_i \in T$.

- **Challenges**:
  - robustness: at least one analysis;
  - disambiguation: at most one analysis;
  - accuracy: correct analysis (for every sentence);
  - efficiency: reasonable time and memory usage

- **Two different methodological strategies**:
  - grammar-driven;
Text parsing

- **Goal:** parse unrestricted text in natural language.
  - Given a text $T = (x_1, \ldots, x_n)$ in language $L$, derive the correct analysis for every sentence $x_i \in T$.

- **Challenges:**
  - **robustness:** at least one analysis;
  - **disambiguation:** at most one analysis;
  - **accuracy:** correct analysis (for every sentence);
  - **efficiency:** reasonable time and memory usage

- **Two different methodological strategies:**
  - **grammar-driven**;
  - **data-driven**.
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
- Grammar-driven parsing: possible trees defined by the grammar.
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
- Grammar-driven parsing: possible trees defined by the grammar.
- Problems with coverage:
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
- Grammar-driven parsing: possible trees defined by the grammar.
- Problems with coverage:
  - only around 70% of all sentences were assigned an analysis.
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
- Grammar-driven parsing: possible trees defined by the grammar.
- Problems with coverage:
  - only around 70% of all sentences were assigned an analysis.
- Most sentences were assigned very many analyses by a grammar:
Back in the days (90s)

- Parsers assigned linguistically detailed syntactic structures (based on linguistic theories).
- Grammar-driven parsing: possible trees defined by the grammar.
- Problems with coverage:
  - only around 70% of all sentences were assigned an analysis.
- Most sentences were assigned very many analyses by a grammar:
  - no way of choosing between them.
Grammar-driven parsing

- A formal grammar $G$ defines:

  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
  - robustness: analyze any input sentence:
    - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.

- disambiguation:
  - number of analyses assigned by grammar may be very large,
  - probabilistic extensions to pick the most likely parse, e.g. PCFG.

- accuracy: assumed advantage, but requires joint optimization of robustness and disambiguation.
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser

- Robustness (analyze any input sentence):
  - Some input sentences $x_i$ are not in $L(G)$.

- Disambiguation:
  - Number of analyses assigned by grammar may be very large.
  - Probabilistic extensions to pick the most likely parse, e.g., PCFG.

- Accuracy: Assumed advantage, but requires joint optimization of robustness and disambiguation.
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
- robustness (analyze any input sentence):
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser

- **robustness** (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
- **robustness** (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
- robustness (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.
- disambiguation:
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
- **robustness** (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.
- **disambiguation**:
  - number of analyses assigned by grammar may be very large,
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser
- robustness (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.
- disambiguation:
  - number of analyses assigned by grammar may be very large,
  - probabilistic extensions to pick the most likely parse, e.g. PCFG.
Grammar-driven parsing

- A formal grammar $G$ defines:
  - the language $L(G)$ that can be parsed,
  - the class of analyses returned by the parser

- **robustness** (analyze any input sentence):
  - some input sentences $x_i$ are not in $L(G)$;
  - constraint relaxation, partial parsing.

- **disambiguation**:
  - number of analyses assigned by grammar may be very large,
  - probabilistic extensions to pick the most likely parse, e.g. PCFG.

- **accuracy**: assumed advantage, but requires joint optimization of robustness and disambiguation.
Enter data-driven (statistical) parsing

▶ Today data-driven/statistical parsing is available for most languages and syntactic frameworks.
Enter data-driven (statistical) parsing

- Today data-driven/statistical parsing is available for most languages and syntactic frameworks.
- Data-driven approaches: possible trees defined by the treebank (may also additionally involve a grammar).
Enter data-driven (statistical) parsing

- Today data-driven/statistical parsing is available for most languages and syntactic frameworks.
- Data-driven approaches: possible trees defined by the treebank (may also additionally involve a grammar).
- Produce one analysis (hopefully the most likely one) for any sentence.
Enter data-driven (statistical) parsing

- Today data-driven/statistical parsing is available for most languages and syntactic frameworks.
- Data-driven approaches: possible trees defined by the treebank (may also additionally involve a grammar).
- Produce one analysis (hopefully the most likely one) for any sentence.
- And get most of them correct.
Enter data-driven (statistical) parsing

- Today data-driven/statistical parsing is available for most languages and syntactic frameworks.
- Data-driven approaches: possible trees defined by the treebank (may also additionally involve a grammar).
- Produce one analysis (hopefully the most likely one) for any sentence.
- And get most of them correct.
- Very active field of research, state-of-the-art is pushed further at almost every major NLP conference.
Data-driven dependency parsing

Introduction
General issues

Data-driven dependency parsing

Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features

Advanced transition parsers

Non-Projective Dependency Parsing

Treebanks

Parsers

Summary

References
Data-driven parsing

1. **formal model** $M$ defining possible analyses for sentences in $L$;
Data-driven parsing

1. formal model $M$ defining possible analyses for sentences in $L$;
2. A sample of annotated text $S = (x_1, \ldots, x_m)$ from $L$;
Data-driven parsing

1. **formal model** \( M \) defining possible analyses for sentences in \( L \);
2. A sample of annotated text \( S = (x_1, \ldots, x_m) \) from \( L \);
3. An inductive inference scheme \( I \) defining actual analyses for the sentences of a text \( T = (x_1, \ldots, x_n) \) in \( L \), relative to \( M \) and \( S \).
Data-driven parsing

1. formal model $M$ defining possible analyses for sentences in $L$;
2. A sample of annotated text $S = (x_1, \ldots, x_m)$ from $L$;
3. An inductive inference scheme $I$ defining actual analyses for the sentences of a text $T = (x_1, \ldots, x_n)$ in $L$, relative to $M$ and $S$.

- $S$ is the training data: contains representations satisfying $M$;
Data-driven parsing

1. **formal model** $M$ defining possible analyses for sentences in $L$;
2. A sample of annotated text $S = (x_1, \ldots, x_m)$ from $L$;
3. An inductive inference scheme $I$ defining actual analyses for the sentences of a text $T = (x_1, \ldots, x_n)$ in $L$, relative to $M$ and $S$.

- $S$ is the **training data**: contains representations satisfying $M$;
- a **treebank**: manually annotated with correct analysis;
Data-driven parsing

1. formal model $M$ defining possible analyses for sentences in $L$;
2. A sample of annotated text $S = (x_1, \ldots, x_m)$ from $L$;
3. An inductive inference scheme $I$ defining actual analyses for the sentences of a text $T = (x_1, \ldots, x_n)$ in $L$, relative to $M$ and $S$.

- $S$ is the training data: contains representations satisfying $M$;
- a treebank: manually annotated with correct analysis;
- $I$ based on supervised machine learning.
Data-driven parsing

- **Robustness**: depends on $M$ and $I$, but usually designed such that any input string is assigned at least one analysis.
Data-driven parsing

- **Robustness**: depends on $M$ and $I$, but usually designed such that any input string is assigned at least one analysis.
- **Disambiguation**: severe problem, solved by inductive inference scheme.
Data-driven parsing

- Robustness: depends on $M$ and $I$, but usually designed such that any input string is assigned at least one analysis.
- Disambiguation: severe problem, solved by inductive inference scheme.
- Improved accuracy represents main challenge.
Data-driven parsing

- **Robustness**: depends on $M$ and $I$, but usually designed such that any input string is assigned at least one analysis.
- **Disambiguation**: severe problem, solved by inductive inference scheme.
- Improved **accuracy** represents main challenge.
- **Efficiency**: varies.
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
  - single-head,
Data-driven dependency parsing

- \( M \) defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
  - single-head,
  - (projective).
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
  - single-head,
  - (projective).

- $I$ may be defined in different ways:
  - parsing method (deterministic, non-deterministic, etc),
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
  - single-head,
  - (projective).

- It may be defined in different ways:
  - parsing method (deterministic, non-deterministic, etc),
  - machine learning algorithm, feature representations.

- Two main classic approaches: transition-based and graph-based models [McDonald and Nivre 2007],
Data-driven dependency parsing

- $M$ defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected,
  - acyclic,
  - single-head,
  - (projective).

- $I$ may be defined in different ways:
  - parsing method (deterministic, non-deterministic, etc),
  - machine learning algorithm, feature representations.

- Two main classic approaches: transition-based and graph-based models [McDonald and Nivre 2007],

- Today we deal with the transition-based parsing.
Contents

Introduction
General issues
Data-driven dependency parsing

Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features

Advanced transition parsers

Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
Transition-based approaches

▶ Basic idea:
Transition-based approaches

- Basic idea:
  - define a transition system for mapping a sentence to its correct dependency graph.
Transition-based approaches

Basic idea:

- define a transition system for mapping a sentence to its correct dependency graph.
- **Learning:** induce a model for predicting the next state transition, given the transition history.
Transition-based approaches

- Basic idea:
  - define a transition system for mapping a sentence to its correct dependency graph.
  - **Learning**: induce a model for predicting the next state transition, given the transition history.
  - **Parsing**: Construct the optimal transition sequence, given the induced model.
Transition-based approaches

- **Basic idea:**
  - define a transition system for mapping a sentence to its correct dependency graph.
  - **Learning:** induce a model for predicting the next state transition, given the transition history.
  - **Parsing:** Construct the optimal transition sequence, given the induced model.

- **Characteristics:**
  - stack-based,
Transition-based approaches

- Basic idea:
  - define a transition system for mapping a sentence to its correct dependency graph.
  - Learning: induce a model for predicting the next state transition, given the transition history.
  - Parsing: Construct the optimal transition sequence, given the induced model.

- Characteristics:
  - stack-based,
  - local inference,
Transition-based approaches

- Basic idea:
  - define a transition system for mapping a sentence to its correct dependency graph.
  - **Learning**: induce a model for predicting the next state transition, given the transition history.
  - **Parsing**: Construct the optimal transition sequence, given the induced model.

- Characteristics:
  - stack-based,
  - local inference,
  - greedy search,
Transition-based approaches

- **Basic idea:**
  - define a transition system for mapping a sentence to its correct dependency graph.
  - **Learning:** induce a model for predicting the next state transition, given the transition history.
  - **Parsing:** Construct the optimal transition sequence, given the induced model.

- **Characteristics:**
  - stack-based,
  - local inference,
  - greedy search,
  - usually produces projective trees.
So called Shift-Reduce paradigm existed since 1970s...
So called **Shift-Reduce** paradigm existed since 1970s...

but it was **Joachim Nivre** who defined the modern, deterministic, transition-based dependency parsing [Nivre 2003].
So called Shift-Reduce paradigm existed since 1970s...

but it was Joachim Nivre who defined the modern, deterministic, transition-based dependency parsing [Nivre 2003].

Important name to know: current president of the Association for Computational Linguistics.
Transition-based Parsing

- Key notions: configurations and transitions;
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
- a set of transition operators, which when applied to a configuration, produce new configurations.
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
- a set of transition operators, which when applied to a configuration, produce new configurations.

Workflow

- Parsing as a set of transitions between parse configurations: deterministic sequence of elementary parsing actions.
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
- a set of transition operators, which when applied to a configuration, produce new configurations.

Workflow

- Parsing as a set of transitions between parse configurations: deterministic sequence of elementary parsing actions.
- The parse oracle predicts the next parse action (transition), based on the current parse configuration.
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
- a set of transition operators, which when applied to a configuration, produce new configurations.

Workflow

- Parsing as a set of transitions between parse configurations: deterministic sequence of elementary parsing actions.
- The parse oracle predicts the next parse action (transition), based on the current parse configuration.
- The oracle is trained with supervised machine learning.
Transition-based parsing

- Key notions: configurations and transitions;
- configuration is the current state of data structures;
- a set of transition operators, which when applied to a configuration, produce new configurations.

Workflow

- Parsing as a set of transitions between parse configurations: deterministic sequence of elementary parsing actions.
- The parse oracle predicts the next parse action (transition), based on the current parse configuration.
- The oracle is trained with supervised machine learning.
- Goal: to find correct final configuration (dependency graph), where all words are accounted for.
Transition-based parsing

We search through a space of configurations for a sequence of transitions that leads from a start state to a desired goal state.
Transition-based parsing

We search through a space of configurations for a sequence of transitions that leads from a start state to a desired goal state.

**Algorithm 2 Transition-based parser**

1. **function** DependencyParse(words)
2. \( state \leftarrow ([root], [words], []) \) initial configuration
3. **while** state not final **do**
4. \( t \leftarrow ORACLE(state) \) choose a transition operator
5. \( state \leftarrow APPLY(t, state) \) apply it, creating a new state
6. **return** state
7. **return** dependency tree
Data structures

- $S$ is the parse stack – a list of partially processed tokens which are candidates for dependency arcs $[\ldots, w_i]_S$;

- $I$ is the queue of remaining input tokens $[w_j; \ldots]$;

- $G$ represents the dependency graph under construction $[d_x; \ldots]$.
Data structures

- $S$ is the **parse stack** – a list of partially processed tokens which are candidates for dependency arcs $[\ldots, w_i]_S$;
- $I$ is the **queue of remaining input tokens** $[w_j, \ldots]_Q$;
Data structures

- $S$ is the parse stack – a list of partially processed tokens which are candidates for dependency arcs $[\ldots, w_i]_S$;
- $I$ is the queue of remaining input tokens $[w_j, \ldots]_Q$;
- $G$ represents the dependency graph under construction $[d_x, \ldots]_G$
Data structures

- $S$ is the **parse stack** – a list of partially processed tokens which are candidates for dependency arcs $[\ldots, w_i]_S$;
- $I$ is the **queue of remaining input tokens** $[w_j, \ldots]_Q$;
- $G$ represents the **dependency graph under construction** $[d_x, \ldots]_G$. 

![Diagram of data structures](image)
Transition operators

Parsing actions are built from atomic actions:
Transition operators

Parsing actions are built from atomic actions:

- Adding arcs operations:
Transition operators

Parsing actions are built from atomic actions:

- Adding arcs operations:
  - $w_i \rightarrow w_j$,
Transition operators

Parsing actions are built from atomic actions:

- Adding arcs operations:
  - $w_i \rightarrow w_j$,
  - $w_i \leftarrow w_j$,
Transition operators

Parsing actions are built from atomic actions:

- **Adding arcs** operations:
  - $w_i \rightarrow w_j$,
  - $w_i \leftarrow w_j$,
  - arcs can be labeled
Transition operators

Parsing actions are built from atomic actions:

- **Adding arcs** operations:
  - $w_i \rightarrow w_j$,
  - $w_i \leftarrow w_j$,
  - arcs can be labeled

- **Reduce** operation: mark the word as processed.
Transition operators

Parsing actions are built from atomic actions:

- **Adding arcs** operations:
  - $w_i \rightarrow w_j$,
  - $w_i \leftarrow w_j$,
  - arcs can be labeled

- **Reduce** operation: mark the word as processed.

- **Shift** operation (move the word to the stack queue).
Let's try to parse a sentence!
Transition-based dependency parsing

Let's try to parse a sentence!

‘Book me the morning flight’
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: SHIFT, RightArc, LeftArc.
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td></td>
<td>RightArc</td>
<td>(book!me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td></td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

*Book me the morning flight*

Actions inventory: SHIFT, RightArc, LeftArc.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
</tbody>
</table>
Let’s try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td><em>(book → me)</em></td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘*Book me the morning flight*’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td><em>(book → me)</em></td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!
‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘*Book me the morning flight*’

**Actions inventory:** `SHIFT, RightArc, LeftArc`.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td><em>(book → me)</em></td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: SHIFT, RightArc, LeftArc.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

**Actions inventory:** SHIFT, RightArc, LeftArc.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book $\rightarrow$ me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Let’s try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc**.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(the ← flight)</td>
</tr>
</tbody>
</table>

Transition-based dependency parsing
Let’s try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RightArc</td>
<td>(book → flight)</td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: SHIFT, RightArc, LeftArc.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RightArc</td>
<td>(book → flight)</td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘Book me the morning flight’

Actions inventory: **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>$(book \rightarrow me)$</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>$(morning \leftarrow flight)$</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>$(the \leftarrow flight)$</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RightArc</td>
<td>$(book \rightarrow flight)$</td>
</tr>
</tbody>
</table>
Let's try to parse a sentence!

‘*Book me the morning flight*’

**Actions inventory:** **SHIFT, RightArc, LeftArc.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word list</th>
<th>Action</th>
<th>Relation added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RightArc</td>
<td>(<em>book</em> → <em>me</em>)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(<em>morning</em> ← <em>flight</em>)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LeftArc</td>
<td>(<em>the</em> ← <em>flight</em>)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RightArc</td>
<td>(<em>book</em> → <em>flight</em>)</td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td>RightArc</td>
<td>(<em>root</em> → <em>book</em>)</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>
Training an oracle

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle to tell which actions to take.
Training an oracle

- Data-driven deterministic parsing:
  - Deterministic parsing requires an **oracle** to tell which actions to take.
  - An oracle can be approximated by a **classifier**.
Training an oracle

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle to tell which actions to take.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.
Training an oracle

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle to tell which actions to take.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.

- Learning methods:
Training an oracle

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle to tell which actions to take.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.

- Learning methods:
  - **Support vector machines (SVM)**
  - **Memory-based learning (MBL)**
    [Nivre et al. 2004, Nivre and Scholz 2004];
  - **Maximum entropy modeling (MaxEnt)**
    [Cheng et al. 2005].
Generating training data

Learning problem:
- Approximate a function from parser configurations, represented by feature vectors to parser actions, given a training set of gold standard derivations.
Generating training data

- Learning problem:
  - Approximate a function from parser configurations, represented by feature vectors to parser actions, given a training set of gold standard derivations.
- Derive appropriate training instances consisting of configuration-transition pairs from a treebank...
Generating training data

- Learning problem:
  - Approximate a function from parser configurations, represented by feature vectors to parser actions, given a training set of gold standard derivations.
- Derive appropriate training instances consisting of configuration-transition pairs from a treebank...
- ..by simulating the operation of a parser in the context of a reference dependency tree.
Generating training data

- Learning problem:
  - Approximate a function from parser configurations, represented by feature vectors to parser actions, given a training set of gold standard derivations.

- Derive appropriate training instances consisting of configuration-transition pairs from a treebank...

- ..by simulating the operation of a parser in the context of a reference dependency tree.

- In this way, we create the training data for the oracle: SVM, logistic regression or whatever.
Generating training data

- **Learning problem:**
  - Approximate a function from *parser configurations*, represented by feature vectors to *parser actions*, given a training set of gold standard derivations.

- **Derive appropriate training instances** consisting of configuration-transition pairs from a treebank...

- ..by simulating the operation of a parser in the context of a reference dependency tree.

- In this way, we create the training data for the oracle: SVM, logistic regression or whatever.

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Predicted Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[the, flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>2</td>
<td>[root, book, the]</td>
<td>[flight, through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>3</td>
<td>[root, book, the, flight]</td>
<td>[through, houston]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>4</td>
<td>[root, book, flight]</td>
<td>[through, houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>5</td>
<td>[root, book, flight, through]</td>
<td>[houston]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>6</td>
<td>[root, book, flight, through, houston]</td>
<td>[]</td>
<td>LEFTARC</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, flight, through, houston]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>9</td>
<td>[root, book]</td>
<td>[]</td>
<td>RIGHTARC</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
</tr>
</tbody>
</table>
For supervised classification we need features. What can they be?
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
  - Attributes:
    - Lexical: word forms and lemmas;
    - Part-of-speech and morpho-syntactic features (can be given or also inferred);
    - Already inferred dependency types (dynamic features);
    - Distance (between target tokens) and their combinations.
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

▶ Typical features:
  ▶ Tokens:
    ▶ Target tokens themselves;
    ▶ Their linear context (neighbors in S and Q);
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in $S$ and $Q$);
    - Their structural context (parents, children, siblings in $G$: dynamic features)
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in S and Q);
    - Their structural context (parents, children, siblings in G: dynamic features)
  - Attributes:
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in $S$ and $Q$);
    - Their structural context (parents, children, siblings in $G$: dynamic features)
  - Attributes:
    - Lexical: word forms and lemmas;
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in $S$ and $Q$);
    - Their structural context (parents, children, siblings in $G$: dynamic features)
  - Attributes:
    - Lexical: word forms and lemmas;
    - Part-of-speech and morpho-syntactic features (can be given or also inferred);
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

▶ Typical features:
  ▶ Tokens:
    ▶ Target tokens themselves;
    ▶ Their linear context (neighbors in $S$ and $Q$);
    ▶ Their structural context (parents, children, siblings in $G$: dynamic features)
  ▶ Attributes:
    ▶ Lexical: word forms and lemmas;
    ▶ Part-of-speech and morpho-syntactic features (can be given or also inferred);
    ▶ Already inferred dependency types (dynamic features);
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in S and Q);
    - Their structural context (parents, children, siblings in G: dynamic features)
  - Attributes:
    - **Lexical**: word forms and lemmas;
    - **Part-of-speech** and morpho-syntactic features (can be given or also inferred);
    - Already inferred dependency types (dynamic features);
    - **Distance** (between target tokens)
Feature Models

Principally, any property of any element in the stack, the input queue, or in the current set of inferred dependencies.

- Typical features:
  - Tokens:
    - Target tokens themselves;
    - Their linear context (neighbors in S and Q);
    - Their structural context (parents, children, siblings in G: dynamic features)
  - Attributes:
    - **Lexical**: word forms and lemmas;
    - **Part-of-speech** and morpho-syntactic features (can be given or also inferred);
    - Already inferred **dependency types** (dynamic features);
    - **Distance** (between target tokens)
  - ...and their combinations.
Different features for different languages (example for Russian).
Different features for different languages (example for Russian).
Feature Models

- Parse configurations are represented by a set of features, which focus on attributes of the top of the stack, the next input token and neighboring tokens in the stack, input queue and dependency graph.
Parse configurations are represented by a set of features, which focus on attributes of the top of the stack, the next input token and neighboring tokens in the stack, input queue and dependency graph.

<table>
<thead>
<tr>
<th>Feature Model</th>
<th>form</th>
<th>pos</th>
<th>dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>S:top</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>l:next</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>G:head of top</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G:leftmost dependent of top</td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>
Contents

Introduction
General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

- Four parsing actions:
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

▶ Four parsing actions:

1. Shift

$$[\ldots]s \rightarrow [w_i, \ldots]Q$$

$$[\ldots, w_i]s \rightarrow [\ldots]Q$$
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

Four parsing actions:

1. Shift

$[\ldots]s \quad [wi, \ldots]q$

$[\ldots, wi]s \quad [\ldots]q$

2. Reduce

$[\ldots, wi]s \quad [\ldots]q$  $\exists w_k : w_k \rightarrow w_i$

$[\ldots]s \quad [\ldots]q$
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

▶ Four parsing actions:

1. Shift

\[
\begin{array}{c}
[\ldots]s \quad [w_i, \ldots]Q \\
[\ldots, w_i]s \quad [\ldots]Q
\end{array}
\]

2. Reduce

\[
\begin{array}{c}
[\ldots, w_i]s \quad [\ldots]Q \\
[\ldots]s \quad [\ldots]Q
\end{array} \quad \exists w_k : w_k \rightarrow w_i
\]

3. Left-Arc

\[
\begin{array}{c}
[\ldots, w_i]s \quad [w_j, \ldots]Q \\
[\ldots]s \quad [w_j, \ldots]Q
\end{array} \quad \neg \exists w_k : w_k \rightarrow w_i \quad w_i \leftarrow w_j
\]

Characteristics:

▶ Arc-eager processing of right-dependents;
▶ Single pass over the input still gives linear time complexity $O(n)$.
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

▶ Four parsing actions:

1. Shift

\[
\begin{array}{c}
\text{[\ldots]s} & \text{[w_i, \ldots]Q} \\
\text{[\ldots, w_i]s} & \text{[\ldots]Q}
\end{array}
\]

2. Reduce

\[
\begin{array}{c}
\text{[\ldots, w_i]s} & \text{[\ldots]Q} \\
\text{[\ldots]s} & \text{[\ldots]Q}
\end{array}
\]

\[
\exists w_k : w_k \rightarrow w_i
\]

3. Left-Arc

\[
\begin{array}{c}
\text{[\ldots, w_i]s} & \text{[w_j, \ldots]Q} \\
\text{[\ldots]s} & \text{[w_j, \ldots]Q}
\end{array}
\]

\[
\neg \exists w_k : w_k \rightarrow w_i
\]

\[
w_i \leftarrow w_j
\]

4. Right-Arc

\[
\begin{array}{c}
\text{[\ldots, w_i]s} & \text{[w_j, \ldots]Q} \\
\text{[\ldots, w_i, w_j]s} & \text{[\ldots]Q}
\end{array}
\]

\[
\neg \exists w_k : w_k \rightarrow w_j
\]

\[
w_i \rightarrow w_j
\]
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

Four parsing actions:

1. Shift

\[
\begin{array}{c}
[\ldots]s & [w_i, \ldots]Q \\
[\ldots, w_i]s & [\ldots]Q
\end{array}
\]

2. Reduce

\[
\begin{array}{c}
[\ldots, w_i]s & [\ldots]Q \\
[\ldots]s & [\ldots]Q
\end{array}
\]

\[
\exists w_k : w_k \rightarrow w_i
\]

3. Left-Arc_r

\[
\begin{array}{c}
[\ldots, w_i]s & [w_j, \ldots]Q \\
[\ldots]s & [w_j, \ldots]Q
\end{array}
\]

\[
\neg \exists w_k : w_k \rightarrow w_i
\]

\[
w_i \leftarrow w_j
\]

4. Right-Arc_r

\[
\begin{array}{c}
[\ldots, w_i]s & [w_j, \ldots]Q \\
[\ldots, w_i, w_j]s & [\ldots]Q
\end{array}
\]

\[
\neg \exists w_k : w_k \rightarrow w_j
\]

\[
w_i \rightarrow w_j
\]

Characteristics:

Arc-eager processing of right-dependents;
Nivre’s Arc-Eager algorithm

It allows words to be attached to their heads as soon as possible, even before we see their dependents.

- Four parsing actions:
  1. Shift
     \[
     \begin{array}{c}
     \cdots \to S \hspace{1cm} [\ldots] Q \hspace{1cm} [\ldots] Q \\
     [\ldots, w_i] S \hspace{1cm} \cdots \to Q
     \end{array}
     \]
  2. Reduce
     \[
     \begin{array}{c}
     \cdots, w_i \to S \hspace{1cm} [\ldots] Q \hspace{1cm} \exists w_k : w_k \to w_i \\
     [\ldots] S \hspace{1cm} [\ldots] Q
     \end{array}
     \]
  3. Left-Arc
     \[
     \begin{array}{c}
     \cdots, w_i \to S \hspace{1cm} [w_j, \ldots] Q \hspace{1cm} \neg \exists w_k : w_k \to w_i \\
     [\ldots] S \hspace{1cm} [w_j, \ldots] Q \hspace{1cm} w_i \leftarrow w_j
     \end{array}
     \]
  4. Right-Arc
     \[
     \begin{array}{c}
     \cdots, w_i \to S \hspace{1cm} [w_j, \ldots] Q \hspace{1cm} \neg \exists w_k : w_k \to w_i \\
     [\ldots, w_i, w_j] S \hspace{1cm} [\ldots] Q \hspace{1cm} w_i \to w_j
     \end{array}
     \]

- Characteristics:
  - Arc-eager processing of right-dependents;
  - Single pass over the input still gives linear time complexity.
Example

[root]_s [Economic news had little effect on financial markets.]_q
Example

Economic

[ root Economic ]\textsubscript{S} [ news had little effect on financial markets . ]\textsubscript{Q}

Shift
Example

$\left[ \text{root}\right]_S \quad \text{Economic} \quad [\text{news had little effect on financial markets .}]_Q$

Left-Arc$_{nmod}$
Example

[\text{root Economic news}]_S [had little effect on financial markets .]_Q

Shift

nmod
Example

Economic news [had little effect on financial markets .]_Q

Left-Arc_{sbj}
Example

[root Economic news had]$_S$ [little effect on financial markets .]$_Q$

Right-Arc$_{pred}$
Example

[root Economic news had little]$_S$ [effect on financial markets .]$_Q$

Shift
Example

```
[root Economic news had]_S little [effect on financial markets .]_Q
```

Left-Arc_{nmod}
Example

Economic news had little effect on financial markets.

Right-Arc_{obj}
Example

Economic news had little effect on financial markets.

Right-Arc_{nmod}
Example

```
[ root: Economic news had little effect on financial markets. ]
```

Shift
Example

Economic news had little effect on financial markets.

Left-Arc_{nmod}
Example

```
[root Economic news had little effect on financial markets]_s  [. ]Q
```

Right-Arc_{pc}
Economic news had little effect on financial markets.

Reduce
Example

Economic news had little effect on financial markets.

Reduce
Example

Economic news had little effect on financial markets.

Reduce
Economic news had little effect on financial markets.

Reduce
Example

Economic news had little effect on financial markets.

Right-Arc$_p$
Beam search

- Transition-based parsing is fast...
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
- Beam search fixes this by constantly searching for alternatives.
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
- Beam search fixes this by constantly searching for alternatives.
- Some heuristic filters ensure the desired beam width.
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
- Beam search fixes this by constantly searching for alternatives.
- Some heuristic filters ensure the desired beam width.
- All applicable operators are applied to the configuration.
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
- **Beam search** fixes this by constantly searching for alternatives.
- Some heuristic filters ensure the desired beam width.
- All applicable operators are applied to the configuration.
- The resulting configurations are scored.

See more in [Zhang and Clark 2008](#).
Beam search

- Transition-based parsing is fast...
- ...because it’s greedy: decisions are made once and forever.
- Beam search fixes this by constantly searching for alternatives.
- Some heuristic filters ensure the desired beam width.
- All applicable operators are applied to the configuration.
- The resulting configurations are scored.
- See more in [Zhang and Clark 2008].
Contents
Introduction
General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
Non-Projective Dependency Parsing

- Many parsing algorithms are restricted to projective dependency graphs.

<table>
<thead>
<tr>
<th>Language</th>
<th>%NPD</th>
<th>%NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Non-Projective Dependency Parsing

- Many parsing algorithms are restricted to projective dependency graphs.
- Is this a problem?
Non-Projective Dependency Parsing

- Many parsing algorithms are restricted to projective dependency graphs.
- Is this a problem?
- Statistics from CoNLL-X Shared Task [Buchholz and Marsi 2006]:

<table>
<thead>
<tr>
<th>Language</th>
<th>% NPD</th>
<th>% NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Non-Projective Dependency Parsing

- Many parsing algorithms are restricted to projective dependency graphs.
- Is this a problem?
- Statistics from CoNLL-X Shared Task [Buchholz and Marsi 2006]:

  - NPD = Non-projective dependencies
  - NPS = Non-projective sentences

<table>
<thead>
<tr>
<th>Language</th>
<th>%NPD</th>
<th>%NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Two Main Approaches

- Algorithms for non-projective dependency parsing:
Two Main Approaches

- Algorithms for non-projective dependency parsing:
  - McDonald’s spanning tree algorithm [McDonald et al. 2005];
Two Main Approaches

- Algorithms for non-projective dependency parsing:
  - McDonald’s spanning tree algorithm [McDonald et al. 2005];
  - Covington’s algorithm [Nivre 2006]
Two Main Approaches

- Algorithms for non-projective dependency parsing:
  - McDonald’s spanning tree algorithm [McDonald et al. 2005];
  - Covington’s algorithm [Nivre 2006]

- Post-processing of projective dependency graphs:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
Non-Projective Parsing Algorithms

- Complexity considerations:
  - Projective (Proj)
  - Non-projective (NonP)

<table>
<thead>
<tr>
<th>Problem/Algorithm</th>
<th>Proj</th>
<th>NonP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic parsing</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[Nivre 2003, Covington 2001]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First order spanning tree</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[McDonald et al. 2005]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contents
Introduction
General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
Treebanks

- Even in the days before data-driven dependency parsing, data was needed to estimate probabilities in formal grammars.
Treebanks

▷ Even in the days before data-driven dependency parsing, data was needed to estimate probabilities in formal grammars.
▷ Collection of sentences manually annotated with the correct parse ⇒ a treebank.
Treebanks

- Even in the days before data-driven dependency parsing, data was needed to estimate probabilities in formal grammars.
- Collection of sentences manually annotated with the correct parse $\Rightarrow$ a treebank.
- Dependency treebanks:
Treebanks

- Even in the days before data-driven dependency parsing, data was needed to estimate probabilities in formal grammars.
- Collection of sentences manually annotated with the correct parse ⇒ a treebank.
- Dependency treebanks:
  - Converted from PSG treebanks;
Treebanks

▶ Even in the days before data-driven dependency parsing, data was needed to estimate probabilities in formal grammars.

▶ Collection of sentences manually annotated with the correct parse ⇒ a treebank.

▶ Dependency treebanks:
  ▶ Converted from PSG treebanks;
  ▶ Genuine dependency treebanks.

▶ Penn Treebank: treebanks from Brown, Switchboard, ATIS og Wall Street Journal corpora [Marcus et al. 1993].
Even in the days before data-driven dependency parsing, data was needed to estimate **probabilities** in formal grammars.

Collection of sentences manually annotated with the correct parse $\Rightarrow$ a **treebank**.

Dependency treebanks:
- Converted from PSG treebanks;
- Genuine dependency treebanks.

Penn Treebank: treebanks from Brown, Switchboard, ATIS og **Wall Street Journal** corpora [Marcus et al. 1993].
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- http://universaldependencies.org/
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- http://universaldependencies.org/
- UD v2 features 70 treebanks in 50 languages (including Norwegian Bokmål and Nynorsk, 300K words each).
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- [http://universaldependencies.org/](http://universaldependencies.org/)
- UD v2 features 70 treebanks in 50 languages (including Norwegian Bokmål and Nynorsk, 300K words each).
- All freely downloadable.

Detailed description in [Nivre et al. 2016](#).

PoS tags in the treebanks are also standardized ([Google's Universal PoS tags,](#) [Petrov et al. 2012](#)).
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- http://universaldependencies.org/
- UD v2 features 70 treebanks in 50 languages (including Norwegian Bokmål and Nynorsk, 300K words each).
- All freely downloadable.
- Detailed description in [Nivre et al. 2016].
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- [http://universaldependencies.org/](http://universaldependencies.org/)
- UD v2 features 70 treebanks in 50 languages (including Norwegian Bokmål and Nynorsk, 300K words each).
- All freely downloadable.
- Detailed description in [Nivre et al. 2016].
- ‘to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages, while allowing language-specific extensions when necessary.’
Universal Dependencies treebanks

- May be, the only treebanks you need these days.
- http://universaldependencies.org/
- UD v2 features 70 treebanks in 50 languages (including Norwegian Bokmål and Nynorsk, 300K words each).
- All freely downloadable.
- Detailed description in [Nivre et al. 2016].
- ‘to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages, while allowing language-specific extensions when necessary.’
- PoS tags in the treebanks are also standardized (Google’s Universal PoS tags, [Petrov et al. 2012])
Universal Dependencies treebanks
Other dependency Treebanks

- Ontonotes [Weischedel et al. 2011]
- Arabic: Prague Arabic Dependency Treebank
- Czech: Prague Dependency Treebank
- Danish: Danish Dependency Treebank
- Portuguese: Bosque: Floresta sintá(c)tica
- Slovene: Slovene Dependency Treebank
- Turkish: METU-Sabanci Turkish Treebank
- Prague Dependency Treebank (czech) [Hajič 1998]
- Negra/Tuba-DZ (German)
- Penn (Chinese)
- Norwegian Dependency Treebank
- the CoNLL treebanks...
Contents

Introduction
General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks

Parsers
Summary
References
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s **MSTParser** (graph-based; more on that in the next week);
- **Mate**: https://code.google.com/archive/p/mate-tools/wikis/ParserAndModels.wiki;
- **TurboParser**: http://www.cs.cmu.edu/~ark/TurboParser/;
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
- *TurboParser*:
  http://www.cs.cmu.edu/~ark/TurboParser/;
- *Spacy*: http://spacy.io/;
- ...and many more.

We will mostly work with *MaltParser*.
Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
- *TurboParser*: http://www.cs.cmu.edu/~ark/TurboParser/;
- *Spacy*: http://spacy.io/;
- *Stanford CoreNLP*: https://stanfordnlp.github.io/CoreNLP/depparse.html;
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
- *TurboParser*:
  http://www.cs.cmu.edu/~ark/TurboParser/;
- *Spacy*: http://spacy.io/;
- *Stanford CoreNLP*: https://stanfordnlp.github.io/CoreNLP/depparse.html;
- ...and many more.
Parsers

Trainable and freely available parsers:

- Ryan McDonald’s *MSTParser* (graph-based; more on that in the next week);
- *TurboParser*: http://www.cs.cmu.edu/~ark/TurboParser/;
- *Spacy*: http://spacy.io/;
- *Stanford CoreNLP*: https://stanfordnlp.github.io/CoreNLP/depparse.html;
- ...and many more.
- We will mostly work with *MaltParser*. 
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
- Executable versions for Linux, Windows and MacOS, open source.
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
- Executable versions for Linux, Windows and MacOS, open source.
- Written in *Java*.
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
- Executable versions for Linux, Windows and MacOS, open source.
- Written in *Java*.
- Employs a rich feature history in order to guide parsing.
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
- Executable versions for Linux, Windows and MacOS, open source.
- Written in Java.
- Employs a rich feature history in order to guide parsing.
- May easily be extended to take into account new features of the parse history.
MaltParser

- *MaltParser* is a language-independent system for transition-based dependency parsing [Nivre et al. 2006a].
- Freely available at http://www.maltparser.org/
- Executable versions for Linux, Windows and MacOS, open source.
- Written in *Java*.
- Employs a rich feature history in order to guide parsing.
- May easily be extended to take into account new features of the parse history.
- Includes *MaltEval* tool to evaluate parsing results, along with convenient visualizations.
We will work more with *MaltParser* and *MaltEval* at the next group session.
Contents
Introduction
General issues
Data-driven dependency parsing
Transition-based dependency parsing
  General algorithm
  Data structures
  Transition operators
  Training an oracle
  Features
Advanced transition parsers
Non-Projective Dependency Parsing
Treebanks
Parsers
Summary
References
Part I: Data-driven dependency parsing

- Dependency grammar (today);
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (*Project A*) released October 3;
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (Project A) released October 3;
- Project A (written report due October 27):
Part I: Data-driven dependency parsing

▶ Dependency grammar (today);
▶ Classical data-driven dependency parsing (today);
▶ Modern approaches to dependency parsing (next Tuesday, October 3);
▶ Obligatory assignment 3 (Project A) released October 3;
▶ Project A (written report due October 27):
  ▶ training a parser on one language and evaluating on another;
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (Project A) released October 3;
- Project A (written report due October 27):
  - training a parser on one language and evaluating on another;
  - Universal Dependencies v2 Treebanks: freely available syntactically annotated corpora;
  - MaltParser: freely available software for data-driven dependency parsing;
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (Project A) released October 3;
- Project A (written report due October 27):
  - training a parser on one language and evaluating on another;
  - Universal Dependencies v2 Treebanks: freely available syntactically annotated corpora;
  - MaltParser: freely available software for data-driven dependency parsing;
- Group sessions:
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (Project A) released October 3;
- Project A (written report due October 27):
  - training a parser on one language and evaluating on another;
  - Universal Dependencies v2 Treebanks: freely available syntactically annotated corpora;
  - MaltParser: freely available software for data-driven dependency parsing;
- Group sessions:
  - Getting your hands dirty with MaltParser and experimental methodology (next Monday, October 2);
Part I: Data-driven dependency parsing

- Dependency grammar (today);
- Classical data-driven dependency parsing (today);
- Modern approaches to dependency parsing (next Tuesday, October 3);
- Obligatory assignment 3 (*Project A*) released October 3;
- *Project A* (*written report due October 27*):
  - training a parser on one language and evaluating on another;
  - *Universal Dependencies v2* Treebanks: freely available syntactically annotated corpora;
  - *MaltParser*: freely available software for data-driven dependency parsing;
- Group sessions:
  - Getting your hands dirty with *MaltParser* and experimental methodology (next Monday, October 2);
  - Discussing and working on *Project A* (October 16).
Monday October 2:

- in-class exercises, MaltParser evaluation and experimental methodology
- Please read the items on the mandatory reading list AND [Nivre et al. 2016]

Tuesday October 3:

- Modern approaches to dependency parsing
References


References II


References IV


References


References VI


References VII


References VIII
