A Transition-Based Directed Acyclic Graph Parser for Universal Conceptual Cognitive Annotation

Daniel Hershcovich, Omri Abend and Ari Rappoport

ACL 2017
TUPA — Transition-based UCCA Parser

The first parser to support the combination of three properties:
1. **Non-terminal nodes** — entities and events over the text
TUPA — Transition-based UCCA Parser

The **first parser** to support the combination of three properties:

1. **Non-terminal nodes** — entities and events over the text
2. **Reentrancy** — allow argument sharing
The **first parser** to support the combination of three properties:

1. **Non-terminal nodes** — entities and events over the text
2. **Reentrancy** — allow argument sharing
3. **Discontinuity** — conceptual units are split

— needed for many semantic schemes (e.g. AMR, UCCA).
Introduction
Linguistic Structure Annotation Schemes

- Syntactic dependencies
- Semantic dependencies (Oepen et al., 2016)

Syntactic (UD)

Semantic (DM)

Bilexical dependencies.
Linguistic Structure Annotation Schemes

- Syntactic dependencies
- Semantic dependencies (Oepen et al., 2016)
- Semantic role labeling (PropBank, FrameNet)
- AMR (Banarescu et al., 2013)
- UCCA (Abend and Rappoport, 2013)
- Other semantic representation schemes

Semantic representation schemes attempt to abstract away from syntactic detail that does not affect meaning:

\[
... \text{bathed} = ... \text{took a bath}
\]

\(^1\text{See recent survey (Abend and Rappoport, 2017)}\)
The UCCA Semantic Representation Scheme
Universal Conceptual Cognitive Annotation (UCCA)

Cross-linguistically applicable (Abend and Rappoport, 2013).
Stable in translation (Sulem et al., 2015).

**English**

IBM happened to choose a company with a crucial vulnerability, despite vetting.

**Hebrew**

IBM ש-בדקה למורת פגיעה ממאבד ב-חוברה be-xevra me’od pgi’a lamrot še-badka ota meroš
Universal Conceptual Cognitive Annotation (UCCA)

Rapid and intuitive annotation interface (Abend et al., 2017). Usable by non-experts. [ucca-demo.cs.huji.ac.il]

Facilitates semantics-based human machine translation evaluation (Birch et al., 2016). [ucca.cs.huji.ac.il/mteval]
UCCA generates a directed acyclic graph (DAG). Text tokens are terminals, complex units are non-terminal nodes. Remote edges enable reentrancy for argument sharing. Phrases may be discontinuous (e.g., multi-word expressions).
Transition-based UCCA Parsing
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004).
Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004).
Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.

Initial state:

stack

buffer

You want to take a long bath
Transition-Based Parsing

First used for dependency parsing (Nivre, 2004). Parse text $w_1 \ldots w_n$ to graph $G$ incrementally by applying transitions to the parser state: stack, buffer and constructed graph.

Initial state:

```
stack

buffer

●
```

You want to take a long bath

TUPA transitions:

\{\text{Shift, Reduce, Node}_X, \text{Left-Edge}_X, \text{Right-Edge}_X, \text{Left-Remote}_X, \text{Right-Remote}_X, \text{Swap, Finish}\}

Support non-terminal nodes, reentrancy and discontinuity.
Example

⇒ SHIFT

\begin{center}

stack

\begin{tabular}{|c|c|}
\hline
• & You \\
\hline
\end{tabular}

\begin{tabular}{llllll}
want & to & take & a & long & bath \\
\hline
\end{tabular}

buffer

\begin{tabular}{|c|}
\hline
graph & • \\
\hline
\end{tabular}

\end{center}
Example

⇒ \text{RIGHT-EDGE}_A

stack

\begin{array}{ll}
\text{You} & \text{buffer} \\
\end{array}

\begin{array}{cccc}
\text{want} & \text{to} & \text{take} & \text{a} \\
\text{long} & \text{bath} \\
\end{array}

graph

You

A
Example

\[ \Rightarrow \text{SHIFT} \]

stack

[●] You [want]

buffer

to take a long bath

diagram

graph

You

A
Example

⇒ SWAP

stack

[● want]

buffer

[You to take a long bath]

graph

A

You
Example

⇒ $\text{RIGHT-EDGE}_P$

stack

\[ \begin{array}{c}
\bullet \\
\text{want}
\end{array} \]

buffer

\[ \begin{array}{c}
\text{You} \\
to \\
take \\
a \\
\text{long} \\
bath
\end{array} \]

graph

\[ \begin{array}{c}
A \\
\downarrow \\
P \\
\text{You} \\
\text{want}
\end{array} \]
Example

⇒ REDUCE

stack

buffer

You to take a long bath

graph

You want

A

P
Example

⇒ **SHIFT**

stack

| ● | You |

buffer

| to | take | a | long | bath |

graph

You

You want
Example

⇒ SHIFT

stack

buffer

You to

take a long bath

graph

You want
Example

⇒ $NODE_F$

stack

You $\rightarrow$ to

buffer

You $\rightarrow$ take a long bath

graph

You $\rightarrow$ want $\rightarrow$ to

You $\rightarrow$ P $\rightarrow$ A $\rightarrow$ to

F
Example

⇒ REDUCE

Youtakealongbath

Youwanttono
Example

⇒ \textbf{SHIFT}

stack

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbullet & You \\
\hline
\end{tabular}
\end{center}

buffer

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
& take & a & long bath \\
\hline
\end{tabular}
\end{center}

graph

You \quad P \quad A

want

to

F
Example

⇒ LEFT-REMOTE$_A$

stack

buffer

<table>
<thead>
<tr>
<th>take</th>
<th>a</th>
<th>long</th>
<th>bath</th>
</tr>
</thead>
</table>

You

You want to take a long bath.

graph

A

P

You

want

to

A

F
Example

⇒ \textbf{SHIFT}

\begin{itemize}
  \item \textbf{You} take
  \item \textbf{a long bath}
\end{itemize}

\begin{itemize}
  \item \textbf{You want}
  \item \textbf{to}
\end{itemize}
Example

⇒ $\text{Node}_C$

stack

-You-
take

buffer

- a long bath

graph

You

want

to

take

A

P

F

C
Example

⇒ REDUCE

stack

You

buffer

You

A

want

to

take

P

A

F

C

a

long

bath
Example

⇒ **SHIFT**

stack

buffer

You buffer a long bath

You want P to take C

You want A to take F

You A P A F C
Example

⇒ $\text{RIGHT-EDGE} P$

stack

buffer

<table>
<thead>
<tr>
<th></th>
<th>You</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

You

want

take

a long bath

A

P

A

P

C

F

to

graph
Example

⇒ SHIFT

stack

buffer

You

a

long

bath

graph

You

want

to

take

A

P

A

F

P

C
Example

⇒ \text{RIGHT-EDGE}_F
Example

⇒ REDUCE

stack

- You
- buffer

long bath

graph

A
P

You
want

You
want
to

A
P

A
P

F
C
F

take
a
Example

$\Rightarrow$ **SHIFT**

- **Stack**
  - You
  - long

- **Buffer**
  - bath

**Graph**

- You
- want
- to
- take
- a

**Notes**

- A: You
- P: long
- C: bath
Example

$\Rightarrow$ SWAP

\begin{align*}
\text{stack} & \quad \text{buffer} \\
\begin{array}{c|c|c}
\bullet & \text{You} & \bullet & \text{long} \\
\end{array}
\end{align*}

\begin{tikzpicture}[thick,->,>=stealth,shorten >=1pt,auto,main node/.style={circle,fill=black,draw,font={\footnotesize}}]
  \node[main node] (1) {You};
  \node[main node] (2) [right of=1, above] {want};
  \node[main node] (3) [below of=2] {to};
  \node[main node] (4) [right of=3] {take};
  \node[main node] (5) [right of=2] {a};
  \node[main node] (6) [right of=1] {P};
  \node[main node] (7) [right of=6] {F};
  \node[main node] (8) [above of=6] {A};
  \node[main node] (9) [above of=7] {P};
  \node[main node] (10) [right of=9] {C};
  \node[main node] (11) [right of=5] {F};

  \path
  (1) edge node {A} (6)
  (6) edge node {P} (7)
  (7) edge node {A} (8)
  (4) edge node {C} (10)
  (10) edge node {F} (11)
  ;
\end{tikzpicture}
Example

\[ \Rightarrow \text{RIGHT-EDGE}_D \]

<table>
<thead>
<tr>
<th>stack</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="black" alt="You" /></td>
<td><img src="red" alt="bath" /></td>
</tr>
<tr>
<td><img src="blue" alt="long" /></td>
<td></td>
</tr>
</tbody>
</table>

```
graph

You \(\rightarrow\) want \(\rightarrow\) You

You \(\rightarrow\) want \(\rightarrow\) to

take \(\rightarrow\) a \(\rightarrow\) long

A \(\rightarrow\) P

C \(\rightarrow\) F

P \(\rightarrow\) A

D \(\rightarrow\) F
```


Example

⇒ REDUCE

stack

buffer

You

bath

A

P

You

want

A

F

to

P

take

A

C

long

F

D

A

P
Example

⇒ SWAP

can you take a long bath?

You want to take a long bath.

You⇒ Stack

buffer

You bath

Graph:

You want to take a long bath.
Example

⇒ \textsc{Right-Edge}_A

stack

\begin{tabular}{c|c}
\text{You} & \text{bath} \\
\end{tabular}

buffer

graph
Example

⇒ REDUCE

stack

buffer

You

bath

You

want

take

a

long

D

C

F

A

P

A

P

F

A

You

want

to

take

A

P

A

P

D
Example

⇒ REDUCE

You want to take a long bath.

You

⇒ REDUCE

You

want

to

take

a

long

bath

You

buffer

stack

drawn

graph
Example

⇒ SHIFT

You want to take a long bath

stack

buffer

You

bath
Example

⇒ REDUCE

stack

graph

buffer

You want to take a long bath.
Example

⇒ SHIFT

You want to take a long bath.

Graph:

- Stack
- Buffer
- Graph

Nodes:
- You
- Want
- To
- Take
- A
- Long
- P
- A
- F
- D

Edges:
- A
- P
- A
- F
- P
- C
- F
- D

- You to Want
- Want to To
- To to Take
- Take to A
- A to Long
Example

$\Rightarrow \text{RIGHT-EDGE}_C$

stack

buffer

You want to take a long bath

diagram

graph
Example

⇒ FINISH

You want to take a long bath.

Graph:

- Stack: bath
- Buffer: stack

Diagram:

You --> want --> to take a long bath.
Training

An oracle provides the transition sequence given the correct graph:

\[ \text{You want to take a long bath} \]

\[ \text{Shift, Right-Edge}_A, \text{Shift, Swap, Right-Edge}_P, \text{Reduce, Shift, Shift, Node}_F, \text{Reduce, Left-Remote}_A, \text{Shift, Shift, Node}_C, \text{Reduce, Shift, Right-Edge}_P, \text{Shift, Right-Edge}_F, \text{Reduce, Shift, Swap, Right-Edge}_D, \text{Reduce, Swap, Right-Edge}_A, \text{Reduce, Reduce, Shift, Reduce, Shift, Right-Edge}_C, \text{Finish} \]
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

Sparse  Perceptron with sparse features (Zhang and Nivre, 2011).
MLP    Embeddings + feedforward NN (Chen and Manning, 2014).
BiLSTM Embeddings + deep bidirectional LSTM + MLP
         (Kiperwasser and Goldberg, 2016).

Features: words, POS, syntactic dependencies, existing edge labels
from the stack and buffer + parents, children, grandchildren;
ordinal features (height, number of parents and children)
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

**Sparse** Perceptron with sparse features (Zhang and Nivre, 2011).
**MLP** Embeddings + feedforward NN (Chen and Manning, 2014).
**BiLSTM** Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).

Effective “lookahead” encoded in the representation.
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

**Sparse** Perceptron with sparse features (Zhang and Nivre, 2011).

**MLP** Embeddings + feedforward NN (Chen and Manning, 2014).

**BiLSTM** Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).
TUPA Model

Learn to greedily predict transition based on current state.

Experimenting with three classifiers:

- **Sparse** Perceptron with sparse features (Zhang and Nivre, 2011).
- **MLP** Embeddings + feedforward NN (Chen and Manning, 2014).
- **BiLSTM** Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).
TUPA Model

Learn to greedily predict transition based on current state.
Experimenting with three classifiers:

- **Sparse** Perceptron with sparse features (Zhang and Nivre, 2011).
- **MLP** Embeddings + feedforward NN (Chen and Manning, 2014).
- **BiLSTM** Embeddings + **deep bidirectional LSTM** + MLP (Kiperwasser and Goldberg, 2016).
You take a long bath.
Experiments
Experimental Setup

- UCCA Wikipedia corpus \(4268 + 454 + 503\) sentences.
Baselines

No existing UCCA parsers $\Rightarrow$ conversion-based approximation.

Bilexical DAG parsers (allow reentrancy):

- DAGParser (Ribeyre et al., 2014): transition-based.

Tree parsers (all transition-based):

- MaltParser (Nivre et al., 2007): bilexical tree parser.
- Stack LSTM Parser (Dyer et al., 2015): bilexical tree parser.
- uparse (Maier, 2015): allows non-terminals, discontinuity.

UCCA bilexical DAG approximation (for tree, delete remote edges).
Bilexical Graph Approximation

1. Convert UCCA to bilexical dependencies.
2. Train bilexical parsers and apply to test sentences.
3. Reconstruct UCCA graphs and compare with gold standard.

After graduation, Joe moved to Paris.
Evaluation

Comparing graphs over the same sequence of tokens,

- Match edges by their terminal yield and label.
- Calculate **labeled precision, recall and F1** scores.
- Separate primary and remote edges.

```
gold

After graduation, Joe moved to Paris.

predicted

After graduation, Joe moved to Paris.

Primary: \( \frac{6}{9} = 67\% \quad \frac{6}{10} = 60\% \quad \frac{64}{100} = 64\% \)

Remote: \( \frac{1}{2} = 50\% \quad \frac{1}{1} = 100\% \quad \frac{67}{100} = 67\% \)
```
## Results

**TUPA\textsubscript{BiLSTM}** obtains the highest F-scores in all metrics:

<table>
<thead>
<tr>
<th></th>
<th>Primary edges</th>
<th></th>
<th>Remote edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
<td>LP</td>
</tr>
<tr>
<td>TUPA\textsubscript{Sparse}</td>
<td>64.5</td>
<td>63.7</td>
<td>64.1</td>
<td>19.8</td>
</tr>
<tr>
<td>TUPA\textsubscript{MLP}</td>
<td>65.2</td>
<td>64.6</td>
<td>64.9</td>
<td>23.7</td>
</tr>
<tr>
<td>TUPA\textsubscript{BiLSTM}</td>
<td>74.4</td>
<td>72.7</td>
<td><strong>73.5</strong></td>
<td>47.4</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td></td>
<td></td>
<td></td>
<td>(91)</td>
</tr>
<tr>
<td>DAGParser</td>
<td>61.8</td>
<td>55.8</td>
<td>58.6</td>
<td>9.5</td>
</tr>
<tr>
<td>TurboParser</td>
<td>57.7</td>
<td>46</td>
<td>51.2</td>
<td>77.8</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td></td>
<td></td>
<td></td>
<td>(91)</td>
</tr>
<tr>
<td>MaltParser</td>
<td>62.8</td>
<td>57.7</td>
<td>60.2</td>
<td>–</td>
</tr>
<tr>
<td>Stack LSTM</td>
<td>73.2</td>
<td>66.9</td>
<td>69.9</td>
<td>–</td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td></td>
<td></td>
<td>(100)</td>
</tr>
<tr>
<td>UPARSE</td>
<td>60.9</td>
<td>61.2</td>
<td>61.1</td>
<td>–</td>
</tr>
</tbody>
</table>

Results on the Wiki test set.
## Results

Comparable on out-of-domain test set:

<table>
<thead>
<tr>
<th></th>
<th>Primary edges</th>
<th></th>
<th>Remote edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
<td>LP</td>
</tr>
<tr>
<td><strong>TUPA\textsubscript{Sparse}</strong></td>
<td>59.6</td>
<td>59.9</td>
<td>59.8</td>
<td>22.2</td>
</tr>
<tr>
<td><strong>TUPA\textsubscript{MLP}</strong></td>
<td>62.3</td>
<td>62.6</td>
<td>62.5</td>
<td>20.9</td>
</tr>
<tr>
<td><strong>TUPA\textsubscript{BiLSTM}</strong></td>
<td>68.7</td>
<td>68.5</td>
<td><strong>68.6</strong></td>
<td>38.6</td>
</tr>
<tr>
<td>Bilexical DAG</td>
<td>(91.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DAGParser</strong></td>
<td>56.4</td>
<td>50.6</td>
<td>53.4</td>
<td></td>
</tr>
<tr>
<td><strong>TurboParser</strong></td>
<td>50.3</td>
<td>37.7</td>
<td>43.1</td>
<td>100</td>
</tr>
<tr>
<td>Bilexical tree</td>
<td>(91.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MaltParser</strong></td>
<td>57.8</td>
<td>53</td>
<td>55.3</td>
<td></td>
</tr>
<tr>
<td><strong>Stack LSTM</strong></td>
<td>66.1</td>
<td>61.1</td>
<td>63.5</td>
<td></td>
</tr>
<tr>
<td>Tree</td>
<td>(100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>UPARSE</strong></td>
<td>52.7</td>
<td>52.8</td>
<td>52.8</td>
<td></td>
</tr>
</tbody>
</table>

Results on the 20K Leagues out-of-domain set.
Conclusion

• UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
• TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
• Outperforms strong conversion-based baselines.

Code: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: cs.huji.ac.il/~oabend/ucca.html
Conclusion

- UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
- TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
- Outperforms strong conversion-based baselines.

Future Work:

- More languages (German corpus construction is underway).
- Parsing other schemes, such as AMR.
- Compare semantic representations through conversion.
- Text simplification, MT evaluation and other applications.

Code: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: cs.huji.ac.il/~oabend/ucca.html
Conclusion

- UCCA’s semantic distinctions require a graph structure including non-terminals, reentrancy and discontinuity.
- TUPA is an accurate transition-based UCCA parser, and the first to support UCCA and any DAG over the text tokens.
- Outperforms strong conversion-based baselines.

Future Work:

- More languages (German corpus construction is underway).
- Parsing other schemes, such as AMR.
- Compare semantic representations through conversion.
- Text simplification, MT evaluation and other applications.

Code: github.com/danielhers/tupa
Demo: bit.ly/tupademo
Corpora: cs.huji.ac.il/~oabend/ucca.html

Thank you!
Universal Conceptual Cognitive Annotation (UCCA).
In Proc. of ACL, pages 228–238.

The state of the art in semantic representation.
In Proc. of ACL, pages 77–89.

Uccaapp: Web-application for syntactic and semantic phrase-based annotation.

Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data.
In Proc. of SemEval, pages 970–973.

Abstract Meaning Representation for sembanking.
In Proc. of the Linguistic Annotation Workshop.


In Proc. of EMNLP, pages 740–750.
Transition-based dependence parsing with stack long short-term memory.

Simple and accurate dependency parsing using bidirectional LSTM feature representations.
TACL, 4:313–327.

Discontinuous incremental shift-reduce parsing.

Incrementality in deterministic dependency parsing.

MaltParser: A language-independent system for data-driven dependency parsing.
Natural Language Engineering, 13(02):95–135.

Towards comparability of linguistic graph banks for semantic parsing.
In Proc. of LREC.

Alpage: Transition-based semantic graph parsing with syntactic features.
In Proc. of SemEval, pages 97–103.

Backup
## UCCA Corpora

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Dev</td>
</tr>
<tr>
<td># passages</td>
<td>300</td>
<td>34</td>
</tr>
<tr>
<td># sentences</td>
<td>4268</td>
<td>454</td>
</tr>
<tr>
<td># nodes</td>
<td>298,993</td>
<td>33,704</td>
</tr>
<tr>
<td>% terminal</td>
<td>42.96</td>
<td>43.54</td>
</tr>
<tr>
<td>% non-term.</td>
<td>58.33</td>
<td>57.60</td>
</tr>
<tr>
<td>% discont.</td>
<td>0.54</td>
<td>0.53</td>
</tr>
<tr>
<td>% reentrant</td>
<td>2.38</td>
<td>1.88</td>
</tr>
<tr>
<td># edges</td>
<td>287,914</td>
<td>32,460</td>
</tr>
<tr>
<td>% primary</td>
<td>98.25</td>
<td>98.75</td>
</tr>
<tr>
<td>% remote</td>
<td>1.75</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Average per non-terminal node

| # children       | 1.67  | 1.68  | 1.66  | 1.61   |

Corpus statistics.
Evaluation

Mutual edges between predicted graph $G_p = (V_p, E_p, \ell_p)$ and gold graph $G_g = (V_g, E_g, \ell_g)$, both over terminals $W = \{w_1, \ldots, w_n\}$:

$$M(G_p, G_g) = \{(e_1, e_2) \in E_p \times E_g \mid y(e_1) = y(e_2) \land \ell_p(e_1) = \ell_g(e_2)\}$$

The yield $y(e) \subseteq W$ of an edge $e = (u, v)$ in either graph is the set of terminals in $W$ that are descendants of $v$. $\ell$ is the edge label.

Labeled precision, recall and F-score are then defined as:

$$LP = \frac{|M(G_p, G_g)|}{|E_p|}, \quad LR = \frac{|M(G_p, G_g)|}{|E_g|},$$

$$LF = \frac{2 \cdot LP \cdot LR}{LP + LR}.$$

Two variants: one for primary edges, and another for remote edges.