Language Generation via DAG Transduction

Yajie Ye, Weiwei Sun and Xiaojun Wan
{yeyajie,ws,wanxiaojun}@pku.edu.cn

Institute of Computer Science and Technology
Peking University

July 17, 2018
Overview

1. Background
2. Formal Models
3. Our DAG Transducer
4. Evaluation
Outline

1. Background
2. Formal Models
3. Our DAG Transducer
4. Evaluation
A NLG system Architecture

Reference
In this paper, we study surface realization, i.e. mapping meaning representations to natural language sentences.
Meaning Representation

- Logic form, e.g. lambda calculus

A Probabilistic Forest-to-String Model for Language Generation from Typed Lambda Calculus Expressions

Wei Lu and Hwee Tou Ng
Department of Computer Science
School of Computing
National University of Singapore
{luwei,nght}@comp.nus.edu.sg
Meaning Representation

- Logic form, e.g. lambda calculus
- Feature structures

High Efficiency Realization for a Wide-Coverage Unification Grammar*

John Carroll\textsuperscript{1} and Stephan Oepen\textsuperscript{2}

\textsuperscript{1} University of Sussex
\textsuperscript{2} University of Oslo and Stanford University
Meaning Representation

- Logic form, e.g. lambda calculus
- Feature structures
- This paper: Graphs!
Graph-Structured Meaning Representation

Different kinds of graph-structured semantic representations:

- **Semantic Dependency Graphs (SDP)**
- Abstract Meaning Representations (AMR)
- Dependency-based Minimal Recursion Semantics (DMRS)
- Elementary Dependency Structures (EDS)
Graph-Structured Meaning Representation

Different kinds of graph-structured semantic representations:

- Semantic Dependency Graphs (SDP)
- Abstract Meaning Representations (AMR)
- Dependency-based Minimal Recursion Semantics (DMRS)
- Elementary Dependency Structures (EDS)
Graph-Structured Meaning Representation

Different kinds of graph-structured semantic representations:

- Semantic Dependency Graphs (SDP)
- Abstract Meaning Representations (AMR)
- Dependency-based Minimal Recursion Semantics (DMRS)
- Elementary Dependency Structures (EDS)
Graph-Structured Meaning Representation

Different kinds of graph-structured semantic representations:

- Semantic Dependency Graphs (SDP)
- Abstract Meaning Representations (AMR)
- Dependency-based Minimal Recursion Semantics (DMRS)
- Elementary Dependency Structures (EDS)
EDS graphs are grounded under type-logical semantics. They are usually very flat and multi-rooted graphs.

The boy wants the girl to believe him.
Previous Work

1. Sequence-to-sequence Models. (AMR-to-text)

Reference
Previous Work

1. Sequence-to-sequence Models. (AMR-to-text)
2. Synchronous Node Replacement Grammar. (AMR-to-text)

Reference
Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text generation with synchronous node replacement grammar.
Previous Work

1. Sequence-to-sequence Models. (AMR-to-text)
2. Synchronous Node Replacement Grammar. (AMR-to-text)
3. Other Unification grammar-based methods

Reference
Carroll, John and Oepen, Stephan 2005. High efficiency realization for a wide-coverage unification grammar
Outline

1. Background
2. Formal Models
3. Our DAG Transducer
4. Evaluation
Formalisms for Strings, Trees and Graphs

<table>
<thead>
<tr>
<th>Chomsky hierarchy</th>
<th>Grammar</th>
<th>Abstract machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-0</td>
<td>-</td>
<td>Turing machine</td>
</tr>
<tr>
<td>Type-1</td>
<td>Context-sensitive</td>
<td>Linear-bounded</td>
</tr>
<tr>
<td>-</td>
<td>Tree-adjoining</td>
<td>Embedded pushdown</td>
</tr>
<tr>
<td>Type-2</td>
<td>Context-free</td>
<td>Nondeterministic pushdown</td>
</tr>
<tr>
<td>Type-3</td>
<td>Regular</td>
<td>Finite</td>
</tr>
</tbody>
</table>

Manipulating Graphs: **Graph Grammar** and **DAG Automata**.
Existing System


the longest NLP paper that I’ve ever read
A weighted DAG automaton is a tuple

\[ M = \langle \Sigma, Q, \delta, K \rangle \]
A run of $M$ on DAG $D = \langle V, E, \ell \rangle$ is an edge labeling function $\rho : E \to Q$.

The weight of $\rho$ is the product of all weight of local transitions:

$$\delta(\rho) = \bigotimes_{v \in V} \delta \left[ \rho(in(v)) \xrightarrow{\ell(v)} \rho(out(v)) \right]$$
DAG Automata: Toy Example

States: 😊😊😊😊😊

John wants to go.

 Recognition Rules:

{ } \xrightarrow{\_want_v_1} \{😊, 😎\}
{ } \xrightarrow{\_go_v_1} \{😊\}
{😊} \xrightarrow{\_go_v_1} \{😊\}
{😊} \xrightarrow{\_go_v_1} \{😊\}
{😊, 😎, 😍} \xrightarrow{nominated(John)} \{\}
DAG Automata: Toy Example

States: 😊😊😊😊😊

John wants to go.

_want_v_1

😊

_proper_q_

_go_v_1

😊

(named(John))

Recognition Rules:

{} ➞ _want_v_1 ➞ {😊, 😊}

{} ➞ proper_q ➞ {😊}

{} ➞ _go_v_1 ➞ {😊}

{} ➞ _go_v_1 ➞ {😊}

{} ➞ _go_v_1 ➞ {😊}

{} ➞ _go_v_1 ➞ {😊}

{} ➞ named(John) ➞ {}
DAG Automata: Toy Example

States: 🙃 ☺ ☁ ☀ ☝️

John wants to go.

Recognition Rules:

\[
\begin{align*}
\{\} & \xrightarrow{\_\text{want\_v\_1}} \{☺, ☄️\} \\
\{\} & \xrightarrow{\text{proper\_q}} \{☺\} \\
\{☺\} & \xrightarrow{\_\text{go\_v\_1}} \{☺\} \\
\{☺\} & \xrightarrow{\text{go\_v\_1}} \{☺\} \\
\{☺, ☺, ☀\} & \xrightarrow{\text{named(John)}} \{\} \\
\end{align*}
\]
DAG Automata: Toy Example

States: 😊😊😊😊😊

John wants to go.

Recognition Rules:

\[
\begin{align*}
\{\} & \xrightarrow{\_\text{want}_v\_1} \{😊, 😌\} \\
\{\} & \xrightarrow{\_\text{proper}_q} \{😊\} \\
\{😊\} & \xrightarrow{\_\text{go}_v\_1} \{😊\} \\
\{😊\} & \xrightarrow{\_\text{go}_v\_1} \{😊\} \\
\{😊, 😌, 😊\} & \xrightarrow{\text{named}(\text{John})} \{\}\n\end{align*}
\]
DAG Automata: Toy Example

States: 😊😊😊😊😊

John wants to go.

Recognition Rules:

- \( \{ \} \xrightarrow{\text{want}_v_1} \{😊, 😊\} \)
- \( \{ \} \xrightarrow{\text{proper}_q} \{😊\} \)
- \( \{😊\} \xrightarrow{\text{go}_v_1} \{😊\} \)
- \( \{😊\} \xrightarrow{\text{go}_v_1} \{😊\} \)
- \( \{😊, 😊, 😊\} \xrightarrow{\text{named}(John)} \{\} \)

Failed!
DAG Automata: Toy Example

States: 😊😊😊😊😊

John wants to go.

Recognition Rules:

{} → want_v_1 → {😊, 😞}

{} → proper_q → {😊}

{😊} → go_v_1 → {😊}

{😊} → go_v_1 → {😊}

{😊, 😞, 😊} → named(John) → {}
DAG Automata: Toy Example

States: ☺ ☻ ☼ ☺ ☼

John wants to go.

Recognition Rules:

\[
\emptyset \xrightarrow{\text{want}_v_1} \{☺, ☻\}
\]

\[
\emptyset \xrightarrow{\text{proper}_q} \{☺\}
\]

\[
\{☺\} \xrightarrow{\text{go}_v_1} \{☺\}
\]

\[
\{☺\} \xrightarrow{\text{go}_v_1} \{☺\}
\]

\[
\{☺, ☻, ☼\} \xrightarrow{\text{named}(\text{John})} \emptyset
\]
DAG Automata: Toy Example

States: 😄😄😄😄😄

John wants to go.

Accept!

Recognition Rules:

\[
\begin{align*}
\emptyset & \xrightarrow{\text{want}_v_1} \{😃, 😞\} \\
\emptyset & \xrightarrow{\text{proper}_q} \{😃\} \\
\{😃\} & \xrightarrow{\text{go}_v_1} \{😃\} \\
\{😃\} & \xrightarrow{\text{go}_v_1} \{😃\} \\
\{😃, 😃, 😃\} & \xrightarrow{\text{named}(John)} \{\}\ 
\end{align*}
\]
Existing System

Daniel Quernheim and Kevin Knight. 2012. *Towards probabilistic acceptors and transducers for feature structures*
DAG-to-Tree Transducer

\[
q
\]

\[
\text{WANT}
\]

\[
\text{BELIEVE}
\]

\[
\text{BOY}
\]

\[
\text{GIRL}
\]
DAG-to-Tree Transducer

\( q \Rightarrow S \)

WANT

BELIEVE

BOY GIRL

\( q_{nomb} \) wants \( q_{inf} \)

BELIEVE

BOY GIRL
DAG-to-Tree Transducer

\[
\begin{array}{c}
q \\
\text{WANT} \\
\text{BELIEVE} \\
\text{BOY} \\
\text{GIRL}
\end{array}
\quad \Rightarrow
\begin{array}{c}
S \\
q_{\text{nomb}} \quad \text{wants} \\
q_{\text{infb}} \\
\text{BOY} \\
\text{GIRL} \\
\text{BELIEVE} \\
q_{\text{acgb}} \\
\text{INF} \\
\text{q_{acgb}} \quad \text{to believe} \\
\text{BOY} \\
\text{GIRL}
\end{array}
\]
DAG-to-Tree Transducer

\[ q \Rightarrow S \]

\[ q_{nom} \text{ \textbf{WANT} } \]

\[ q_{inf} \text{ \textbf{BELIEVE} } \]

\[ q_{accg} \text{ } \Rightarrow \text{ \textbf{BELIEVE} } q_{accb} \]

\[ S \quad \Rightarrow \quad \]

\[ NP \quad \Rightarrow \quad \text{the boy} \quad \text{wants} \quad \text{the girl} \quad \text{to believe} \quad \text{him} \]

\[ \Rightarrow \quad \text{WANT} \quad \text{BELIEVE} \quad \text{BELIEVE} \quad \text{BOY} \quad \text{GIRL} \]

\[ \downarrow \]

\[ S \quad \Rightarrow \quad \]

\[ q_{nom} \text{ \textbf{wants} } q_{accg} \text{ } \quad \Rightarrow \quad \text{\textbf{to believe} q_{accb} } \]

\[ S \quad \Rightarrow \quad \]

\[ \text{BOY} \quad \text{GIRL} \]
DAG-to-Tree Transducer

Challenges for DAG-to-tree transduction on EDS graphs:

- Cannot easily reverse the directions of edges
- Cannot easily handle multiple roots
Outline

1. Background
2. Formal Models
3. Our DAG Transducer
4. Evaluation
Our DAG-to-program transducer

The basic idea:

- 🐐 Rewritting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- 😄 Obtaining target structures based on side effects of the DAG recognition.

States: 😃 😃 😃 😃

The output of our transducer is a program:

John wants to go.
Our DAG-to-program transducer

The basic idea:

- 😊 Rewriting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- 😊 Obtaining target structures based on side effects of the DAG recognition.

States: 😊😊😊😊

The output of our transducer is a program:

John wants to go.
Our DAG-to-program transducer

The basic idea:

• 😊 Rewriting: directly generating a new data structure piece by piece, during recognizing an input DAG.
• 😊 Obtaining target structures based on side effects of the DAG recognition.

States: 😊😊😊😊

The output of our transducer is a program:

\[ S = x_{21} + \text{want} + x_{11} \]

John wants to go.
Our DAG-to-program transducer

The basic idea:

- Rewriting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- Obtaining target structures based on side effects of the DAG recognition.

States: 😄 😄 😄 😄

The output of our transducer is a program:

\[ S = x_{21} + \text{want} + x_{11} \]
\[ x_{11} = \text{to} + \text{go} \]

John wants to go.
Our DAG-to-program transducer

The basic idea:

- 😊 Rewriting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- 😊 Obtaining target structures based on side effects of the DAG recognition.

States: 😊😊😊😊

The output of our transducer is a program:

\[ S = x_{21} + \text{want} + x_{11} \]
\[ x_{11} = \text{to} + \text{go} \]
\[ x_{41} = \epsilon \]

John wants to go.
Our DAG-to-program transducer

The basic idea:

- 😊 Rewriting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- 😊 Obtaining target structures based on side effects of the DAG recognition.

States: 😊😊😊😊

_Saved_\_\_want_v_\_1_  

/go_v_\_1_  

proper_q  

_named(John)_

John wants to go.

The output of our transducer is a _program_:

\[
S = x_{21} + \text{want} + x_{11}
\]

\[
x_{11} = \text{to} + \text{go}
\]

\[
x_{41} = \epsilon
\]

\[
x_{21} = x_{41} + \text{John}
\]
Our DAG-to-program transducer

The basic idea:

- 😊 Rewritting: directly generating a new data structure piece by piece, during recognizing an input DAG.
- 😊 Obtaining target structures based on side effects of the DAG recognition.

States: 😊😊😊😊

The output of our transducer is a program:

\[
S = x_{21} + \text{want} + x_{11} \\
\]
\[
x_{11} = \text{to} + \text{go} \\
\]
\[
x_{41} = \epsilon \\
\]
\[
x_{21} = x_{41} + \text{John} \\
\]

John wants to go.

\[\implies S = \text{John want to go}\]
Transduction Rules

Recognition Part
A valid DAG Automata transition

\{ \} \xrightarrow{-\text{want}_v \_1} \{ \!\!, \!\!\! \}

Generation Part
Statement template(s)

\[ S = v_{\!\!\!} + L + v_{\!\!\!} \]
Transduction Rules

**Recognition Part**
A valid DAG Automata transition

\[
\{ \} \xrightarrow{\text{want}_v} \{ \mathbb{D}, \mathbb{R} \}
\]

**Generation Part**
Statement template(s)

\[
S = v_{\mathbb{D}} + L + v_{\mathbb{R}}
\]

We use **parameterized** states:

\[
\text{label(number,direction)}
\]

The range of direction: **unchanged**, **empty**, **reversed**.
Transduction Rules

Recognition Part
A valid DAG Automata transition

\[
\{ \} \xrightarrow{\text{want\_v\_1}} \{ \text{ ALIGN}, \text{ ALIGN} \}
\]

Generation Part
Statement template(s)

\[
S = v_{\text{ ALIGN}} + L + v_{\text{ ALIGN}}
\]

We use parameterized states:

\text{label}(number,direction)

The range of direction: unchanged, empty, reversed.

\[
\{ \} \xrightarrow{\text{want\_v\_1}} \{ \text{VP(1,u)}, \text{NP(1,u)} \} \quad S = v_{\text{NP(1,u)}} + L + v_{\text{VP(1,u)}}
\]
**Toy Example**

\[ Q = \{ \text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u) \} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( { } \xrightarrow{\text{proper}_q} { \text{DET}(1,r) } )</td>
<td>( v_{\text{DET}(1,r)} = \epsilon )</td>
</tr>
<tr>
<td>2</td>
<td>( { } \xrightarrow{\text{want}_v_1} { \text{VP}(1,u), \text{NP}(1,u) } )</td>
<td>( S = v_{\text{NP}(1,u)} + L + v_{\text{VP}(1,u)} )</td>
</tr>
<tr>
<td>3</td>
<td>( { \text{VP}(1,u) } \xrightarrow{\text{go}_v_1} { \text{Empty}(0,e) } )</td>
<td>( v_{\text{VP}(1,u)} = \text{to} + L )</td>
</tr>
<tr>
<td>4</td>
<td>( { \text{NP}(1,u), \text{DET}(1,r) } \xrightarrow{\text{named}} { } )</td>
<td>( v_{\text{NP}(1,u)} = v_{\text{DET}(1,r)} + L )</td>
</tr>
</tbody>
</table>

**Recognition:** To find an edge labeling function \( \rho \). The red dashed edges make up an intermediate graph \( T(\rho) \).
Toy Example

\[ Q = \{ \text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u) \} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>({} \xrightarrow{\text{proper}_q} {\text{DET}(1,r)})</td>
<td>(\nu_{\text{DET}(1,r)} = \epsilon)</td>
</tr>
<tr>
<td>2</td>
<td>({} \xrightarrow{_\text{want}_v_1} {\text{VP}(1,u), \text{NP}(1,u)})</td>
<td>(S = \nu_{\text{NP}(1,u)} + L + \nu_{\text{VP}(1,u)})</td>
</tr>
<tr>
<td>3</td>
<td>({\text{VP}(1,u)} \xrightarrow{_\text{go}_v_1} {\text{Empty}(0,e)})</td>
<td>(\nu_{\text{VP}(1,u)} = \text{to} + L)</td>
</tr>
<tr>
<td>4</td>
<td>({\text{NP}(1,u), \text{DET}(1,r)} \xrightarrow{\text{named}} {})</td>
<td>(\nu_{\text{NP}(1,u)} = \nu_{\text{DET}(1,r)} + L)</td>
</tr>
</tbody>
</table>

**Recognition:** To find an edge labeling function \(\rho\). The red dashed edges make up an intermediate graph \(T(\rho)\).
### Toy Example

\[ Q = \{ \text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u) \} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( { } \overset{\text{proper}_q}{\rightarrow} { \text{DET}(1,r) } )</td>
<td>( \nu_{\text{DET}(1,r)} = \epsilon )</td>
</tr>
<tr>
<td>2</td>
<td>( { } \overset{_\text{want}_v_1}{\rightarrow} { \text{VP}(1,u), \text{NP}(1,u) } )</td>
<td>( S = \nu_{\text{NP}(1,u)} + L + \nu_{\text{VP}(1,u)} )</td>
</tr>
<tr>
<td>3</td>
<td>( { \text{VP}(1,u) } \overset{\text{-go}_v_1}{\rightarrow} { \text{Empty}(0,e) } )</td>
<td>( \nu_{\text{VP}(1,u)} = \text{to} + L )</td>
</tr>
<tr>
<td>4</td>
<td>( { \text{NP}(1,u), \text{DET}(1,r) } \overset{\text{named}_q}{\rightarrow} { } )</td>
<td>( \nu_{\text{NP}(1,u)} = \nu_{\text{DET}(1,r)} + L )</td>
</tr>
</tbody>
</table>

**Recognition:** To find an edge labeling function \( \rho \). The red dashed edges make up an intermediate graph \( T(\rho) \).
Toy Example

\[ Q = \{\text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u)\} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{} \xrightarrow{\text{proper}_q} {\text{DET}(1,r)}</td>
<td>\nu_{\text{DET}(1,r)} = \epsilon</td>
</tr>
<tr>
<td>2</td>
<td>{} \xrightarrow{_\text{want}_v_1} {\text{VP}(1,u), \text{NP}(1,u)}</td>
<td>S = \nu_{\text{NP}(1,u)} + L + \nu_{\text{VP}(1,u)}</td>
</tr>
<tr>
<td>3</td>
<td>{\text{VP}(1,u)} \xrightarrow{_\text{go}_v_1} {\text{Empty}(0,e)}</td>
<td>\nu_{\text{VP}(1,u)} = \text{to} + L</td>
</tr>
<tr>
<td>4</td>
<td>{\text{NP}(1,u), \text{DET}(1,r)} \xrightarrow{\text{named}} {}</td>
<td>\nu_{\text{NP}(1,u)} = \nu_{\text{DET}(1,r)} + L</td>
</tr>
</tbody>
</table>

**Recognition**: To find an edge labeling function \( \rho \). The red dashed edges make up an intermediate graph \( T(\rho) \).
Toy Example

\[ Q = \{ \text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u) \} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( { } \xrightarrow{\text{proper}_q} { \text{DET}(1,r) } )</td>
<td>( v_{\text{DET}(1,r)} = \epsilon )</td>
</tr>
<tr>
<td>2</td>
<td>( { } \xrightarrow{\text{want}_v} { \text{VP}(1,u), \text{NP}(1,u) } )</td>
<td>( S = v_{\text{NP}(1,u)} + L + v_{\text{VP}(1,u)} )</td>
</tr>
<tr>
<td>3</td>
<td>( { \text{VP}(1,u) } \xrightarrow{\text{go}_v} { \text{Empty}(0,e) } )</td>
<td>( v_{\text{VP}(1,u)} = \text{to} + L )</td>
</tr>
<tr>
<td>4</td>
<td>( { \text{NP}(1,u), \text{DET}(1,r) } \xrightarrow{\text{named}} { } )</td>
<td>( v_{\text{NP}(1,u)} = v_{\text{DET}(1,r)} + L )</td>
</tr>
</tbody>
</table>

**Recognition:** To find an edge labeling function \( \rho \). The red dashed edges make up an intermediate graph \( T(\rho) \).

Accept!
Toy Example

\[ Q = \{ \text{DET}(1,r), \text{Empty}(0,e), \text{VP}(1,u), \text{NP}(1,u) \} \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>For Recognition</th>
<th>For Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{ } → {DET(1,r)}</td>
<td>( \nu_{\text{DET}(1,r)} = \epsilon )</td>
</tr>
<tr>
<td>2</td>
<td>{ } → {VP(1,u), NP(1,u)}</td>
<td>( S = \nu_{\text{NP}(1,u)} + L + \nu_{\text{VP}(1,u)} )</td>
</tr>
<tr>
<td>3</td>
<td>{VP(1,u)} → {Empty(0,e)}</td>
<td>( \nu_{\text{VP}(1,u)} = \text{to} + L )</td>
</tr>
<tr>
<td>4</td>
<td>{NP(1,u), DET(1,r)} → {}</td>
<td>( \nu_{\text{NP}(1,u)} = \nu_{\text{DET}(1,r)} + L )</td>
</tr>
</tbody>
</table>

\[ S = \nu_{\text{NP}(1,u)} + L + \nu_{\text{VP}(1,u)} \]

\[ \downarrow \]

\[ S = x_{21} + \text{want} + x_{11} \]

**Instantiation**: replace \( \nu_{l(j,d)} \) of edge \( e_i \) with variable \( x_{ij} \) and \( L \) with the output string in the statement templates.
DAG Transduction based-NLG

A general framework for DAG transduction based-NLG:

- **Semantic Graph** → **DAG Transducer** → **Sequential Lemmas** → **Seq2seq Model** → **Surface string**
Outline

1. Background
2. Formal Models
3. Our DAG Transducer
4. Evaluation
“the decline is even steeper than in September”, he said.
“the decline is even steeper than in September”, he said.
“the decline is even steeper than in September”, he said.
“the decline is even steeper than in September”, he said.
Finding intermediate tree

Assigning spans

Assigning labels

Generating statement templates

_Inducing Transduction Rules_

_The decline is even steeper than in September_, he said.
Inducing Transduction Rules

“the decline is even steeper than in September”, he said.

\[
\{\text{ADV}(1,r)\} \xrightarrow{\text{comp}} \{\text{PP}(1,u), \text{ADV}_{\text{PP}}(2,r)\}
\]

\[v_{\text{ADV}_{\text{PP}}(1,r)} = v_{\text{ADV}}(1,r)\]

\[v_{\text{ADV}_{\text{PP}}(2,r)} = \text{than} + v_{\text{PP}}(1,u)\]
Inducing Transduction Rules

"the decline is even steeper than in September", he said.
Experimental set-up

- Data: DeepBank + Wikiwoods
- Decoder: Beam search (beam size = 128)
- About 37,000 induced rules are directly obtained from DeepBank training dataset by a group of heuristic rules.
- Disambiguation: global linear model

<table>
<thead>
<tr>
<th>Transducer</th>
<th>Lemmas</th>
<th>Sentences</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>induced rules</td>
<td>89.44</td>
<td>74.94</td>
<td>67%</td>
</tr>
</tbody>
</table>
To deal with data sparseness problem, we use some heuristic rules to generate *extended rules* by slightly changing an induced rule. Given a induced rule:

\[
\{NP, \text{ADJ}\} \xrightarrow{X} \{\} \quad \nu_{NP} = \nu_{\text{ADJ}} + L
\]

New rule generated by deleting:

\[
\{NP\} \xrightarrow{X} \{\} \quad \nu_{NP} = L
\]
Fine-to-coarse Transduction

To deal with data sparseness problem, we use some heuristic rules to generate *extended rules* by slightly changing an induced rule. Given a induced rule:

\[
\{\text{NP, ADJ}\} \xrightarrow{X} \{\} \quad v_{\text{NP}} = v_{\text{ADJ}} + L
\]

New rule generated by copying:

\[
\{\text{NP, ADJ}_1, \text{ADJ}_2\} \xrightarrow{X} \{\} \quad v_{\text{NP}} = v_{\text{ADJ}_1} + v_{\text{ADJ}_2} + L
\]
Experimental set-up

- Data: DeepBank + Wikiwoods
- Decoder: Beam search (beam size = 128)
- About 37,000 **induced rules** and 440,000 **extended rules**
- Disambiguation: global linear model

<table>
<thead>
<tr>
<th>Transducer</th>
<th>Lemmas</th>
<th>Sentences</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>induced rules</td>
<td>89.44</td>
<td>74.94</td>
<td>67%</td>
</tr>
<tr>
<td>induced and extended rules</td>
<td>88.41</td>
<td>74.03</td>
<td>77%</td>
</tr>
</tbody>
</table>
Fine-to-coarse transduction

During decoding, when neither induced nor extended rule is applicable, we use markov model to create a dynamic rule on-the-fly:

\[
P(\{r_1, \cdots , r_n\} | C) = P(r_1 | C) \prod_{i=2}^{n} P(r_i | C) P(r_i | r_{i-1}, C)
\]

- \(C = \langle \{ q_1, \cdots , q_m \}, D \rangle\) represents the context.
- \(r_1, \cdots, r_n\) denotes the outgoing states.
NLG via DAG transduction

Experimental set-up

- Data: DeepBank + Wikiwoods
- Decoder: Beam search (beam size = 128)
- Other tool: OpenNMT

<table>
<thead>
<tr>
<th>Transducer</th>
<th>Lemmas</th>
<th>Sentences</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>induced rules</td>
<td>89.44</td>
<td>74.94</td>
<td>67%</td>
</tr>
<tr>
<td>induced and extended rules</td>
<td>88.41</td>
<td>74.03</td>
<td>77%</td>
</tr>
<tr>
<td>induced, extended and dynamic rules</td>
<td>82.04</td>
<td>68.07</td>
<td>100%</td>
</tr>
<tr>
<td>DFS-NN</td>
<td>50.45</td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>AMR-NN</td>
<td></td>
<td>33.8</td>
<td>100%</td>
</tr>
<tr>
<td>AMR-NRG</td>
<td></td>
<td>25.62</td>
<td>100%</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

English Resource Semantics is fantastic!

Conclusion

• Formalism works for graph-to-string mapping, not surprisingly or surprisingly

Future work

• Is the decoder perfect? No, not even close
• Is the disambiguation model a neural one? No, graph embedding is non-trivial.