Language Generation via DAG Transduction

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

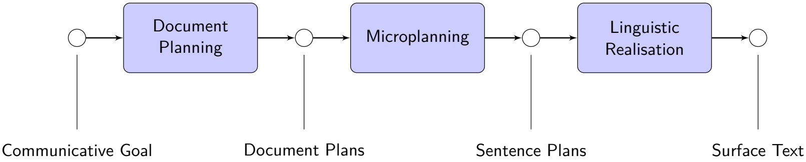
Institute of Computer Science and Technology
Peking University

July 17, 2018

Overview

Outline

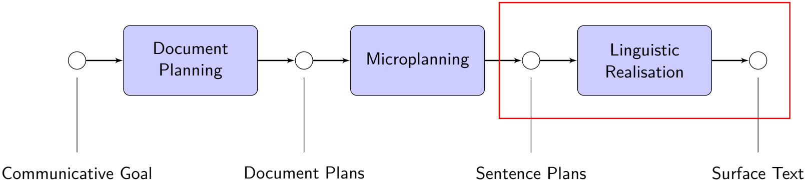
A NLG system Architecture



Reference

Ehud Reiter and Robert Dale, Building Natural Language Generation Systems, Cambridge University Press, 2000.

A NLG system Architecture



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[1] and Stephan Oepen[2]

[1] University of Sussex
[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

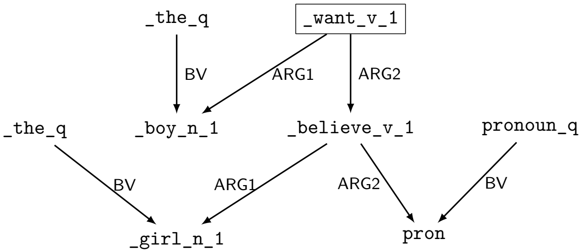
Different kinds of graph-structured semantic representations:

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

Graph-Structured Meaning Representation

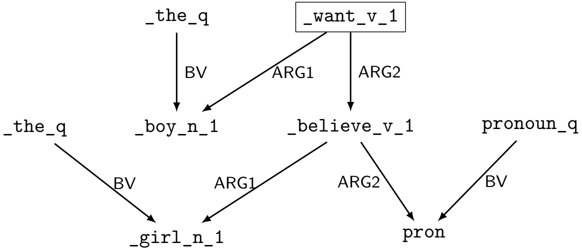
Different kinds of graph-structured semantic representations:

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



Type-Logical Semantic Graph

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. Seqence-to-seqence Models. (AMR-to-text)

Reference

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation.

Previous Work

1. Seqence-to-seqence 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. Seqence-to-seqence 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

Formalisms for Strings, Trees and Graphs

Chomsky hierarchy	Grammar	Abstract machines
Type-0	-	Turing machine
Type-1	Context-sensitive	Linear-bounded
-	Tree-adjoining	Embedded pushdown
Type-2	Context-free	Nondeterministic pushdown
Type-3	Regular	Finite

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

Existing System

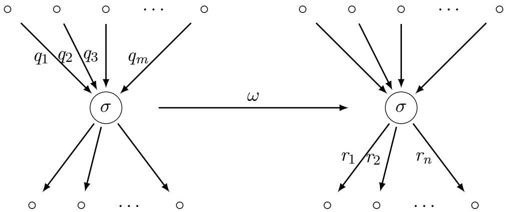
David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez and Giorgio Satta. Weighted DAG Automata for Semantic Graphs.

the longest NLP paper that I've ever read

DAG Automata

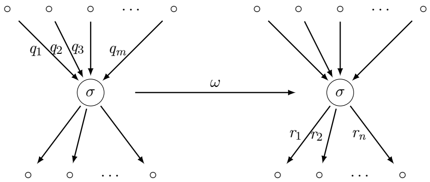
A weighted DAG automaton is a tuple

$$M = \langle \Sigma, Q, \delta, \mathbb{K} \rangle$$



$$\{q_1, \cdots, q_m\} \xrightarrow{\sigma/\omega} \{r_1, \cdots, r_n\}$$

DAG Automata



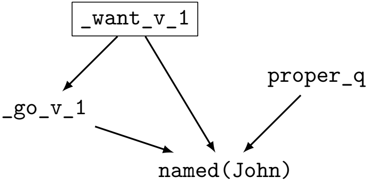
- A run of M on DAG $D = \langle V, E, \ell \rangle$ is an edge labeling function $\rho : E \to Q$.
- The weight of ρ is the product of all weight of local transitions:

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

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Recognition Rules:

$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{$😄, 😎$\}$

$\{\} \xrightarrow{\texttt{proper\_q}} \{$🙂$\}$

$\{$😄$\} \xrightarrow{\texttt{\_go\_v\_1}} \{$😉$\}$

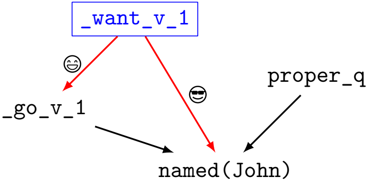
$\{$😄$\} \xrightarrow{\texttt{\_go\_v\_1}} \{$😵$\}$

$\{$😉, 😎, 🙂$\} \xrightarrow{\texttt{named(John)}} \{\}$

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Recognition Rules:

$\{\}$ —_want_v_1→ {😄, 😎}

$\{\}$ —proper_q→ {🙂}

{😄} —_go_v_1→ {😉}

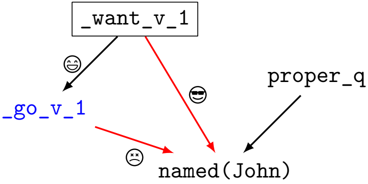
{😄} —_go_v_1→ {😵}

{😉, 😎, 🙂} —named(John)→ $\{\}$

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Recognition Rules:

{} —_want_v_1→ {😄, 😎}

{} —proper_q→ {🙂}

{😄} —_go_v_1→ {😉}

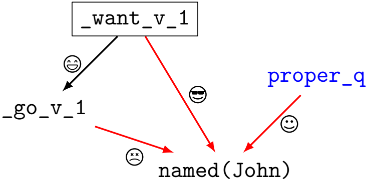
{😄} —_go_v_1→ {😵}

{😉, 😎, 🙂} —named(John)→ {}

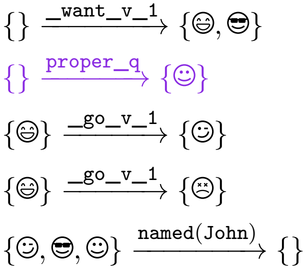
DAG Automata: Toy Example

States: 

John wants to go.



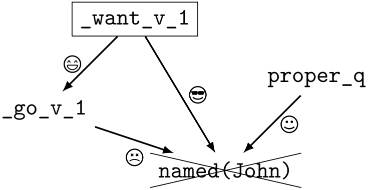
Recognition Rules:



DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Failed !

Recognition Rules:

{} —_want_v_1→ {😄, 😎}

{} —proper_q→ {🙂}

{😄} —_go_v_1→ {😉}

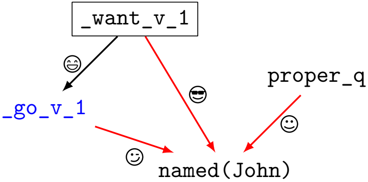
{😄} —_go_v_1→ {😵}

{😉, 😎, 🙂} —named(John)→ {}

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Recognition Rules:

{} --_want_v_1--> {😄, 😎}

{} --proper_q--> {🙂}

{😄} --_go_v_1--> {😉}

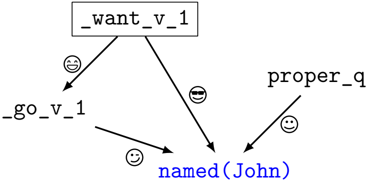
{😄} --_go_v_1--> {😵}

{😉, 😎, 🙂} --named(John)--> {}

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Recognition Rules:

{} —_want_v_1→ {😄, 😎}

{} —proper_q→ {🙂}

{😄} —_go_v_1→ {😉}

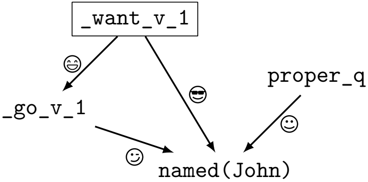
{😄} —_go_v_1→ {😵}

{😉, 😎, 🙂} —named(John)→ {}

DAG Automata: Toy Example

States: 😉 😄 😎 🙂 😵

John wants to go.



Accept !

Recognition Rules:

$\{\} \xrightarrow{\text{\_want\_v\_1}}$ {😄, 😎}

$\{\} \xrightarrow{\text{proper\_q}}$ {🙂}

{😄} $\xrightarrow{\text{\_go\_v\_1}}$ {😉}

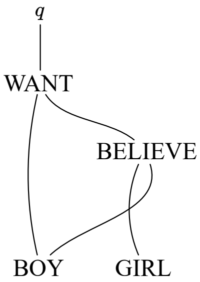
{😄} $\xrightarrow{\text{\_go\_v\_1}}$ {😵}

{😉, 😎, 🙂} $\xrightarrow{\text{named(John)}} \{\}$

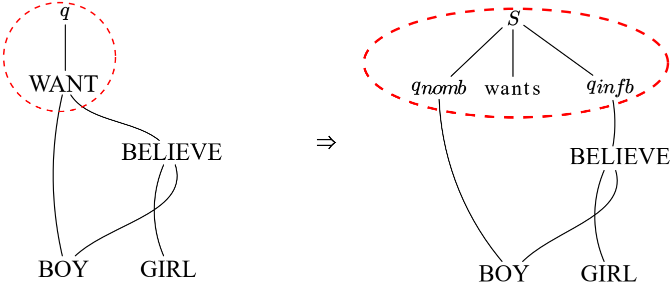
Existing System

Daniel Quernheim and Kevin Knight. 2012. Towards probabilistic acceptors and transducers for feature structures

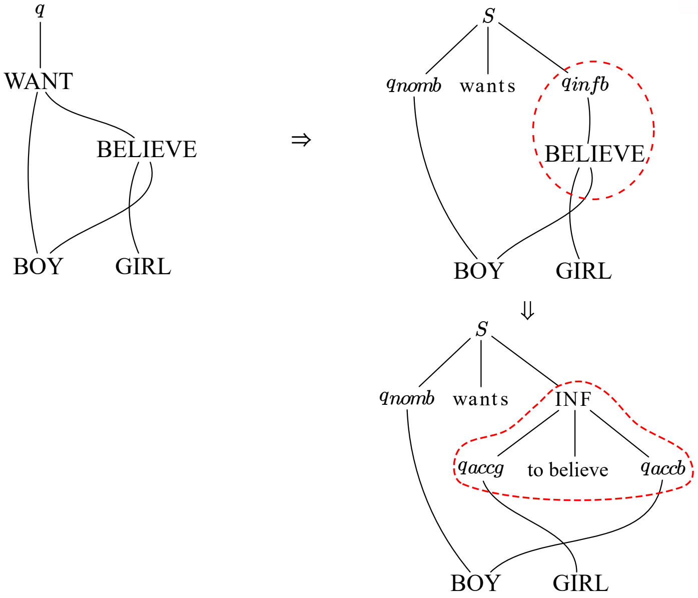
DAG-to-Tree Transducer



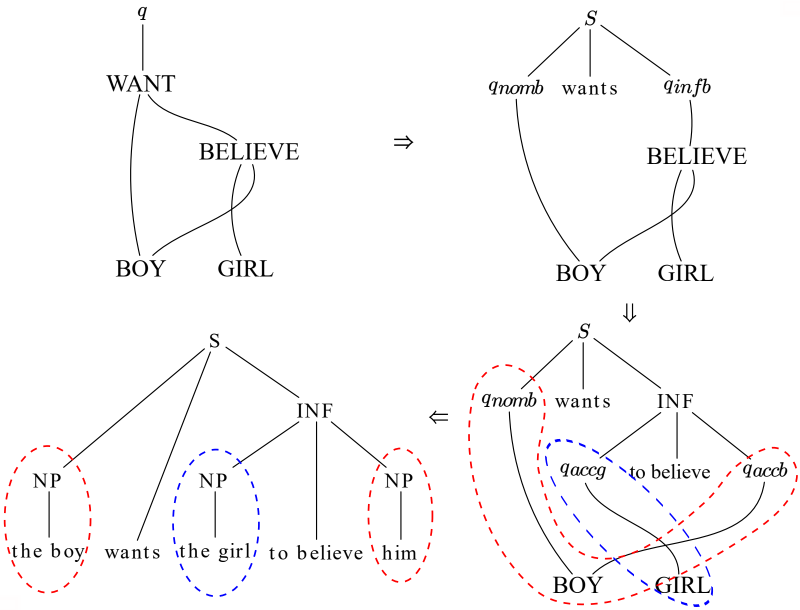
DAG-to-Tree Transducer



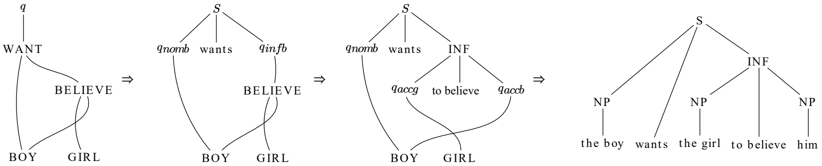
DAG-to-Tree Transducer



DAG-to-Tree Transducer



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

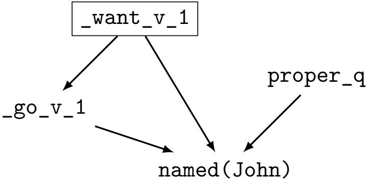
Outline

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: ☺ 😄 😎 ☺



John wants to go.

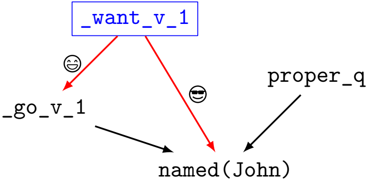
The output of our transducer is a *program*:

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: 🙂 😄 😎 🙂



John wants to go.

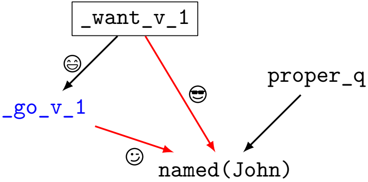
The output of our transducer is a *program*:

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: 😉 😄 😎 🙂



John wants to go.

The output of our transducer is a *program*:

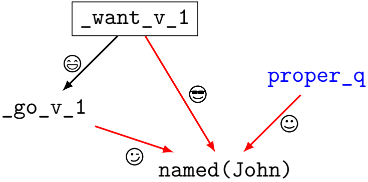
$$S = x_{21} + \mathtt{want} + x_{11}$$

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: 😉 😄 😎 🙂



John wants to go.

The output of our transducer is a *program*:

$$S = x_{21} + \texttt{want} + x_{11}$$

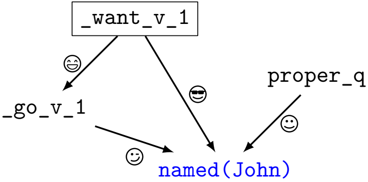
$$x_{11} = \texttt{to} + \texttt{go}$$

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: 😉 😄 😎 🙂



John wants to go.

The output of our transducer is a *program*:

$$S = x_{21} + \texttt{want} + x_{11}$$

$$x_{11} = \texttt{to} + \texttt{go}$$

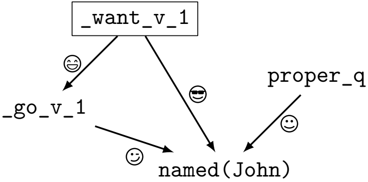
$$x_{41} = \epsilon$$

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: 😉 😄 😎 🙂



John wants to go.

The output of our transducer is a *program*:

$$S = x_{21} + \texttt{want} + x_{11}$$

$$x_{11} = \texttt{to} + \texttt{go}$$

$$x_{41} = \epsilon$$

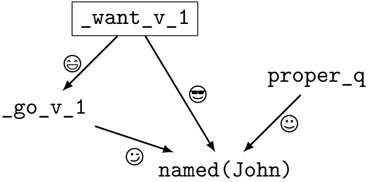
$$x_{21} = x_{41} + \texttt{John}$$

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: 😉 😄 😎 🙂



John wants to go.

The output of our transducer is a *program*:

$$S = x_{21} + \texttt{want} + x_{11}$$

$$x_{11} = \texttt{to} + \texttt{go}$$

$$x_{41} = \epsilon$$

$$x_{21} = x_{41} + \texttt{John}$$

$\Longrightarrow$ $S = \texttt{John want to go}$

Transducation Rules

Recognition Part	**Generation Part**
A valid DAG Automata transition	Statement template(s)
$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\text{☺}, \text{😎}\}$	$S = v_{\text{☺}} + L + v_{\text{😎}}$

Transducation Rules

Recognition Part	**Generation Part**
A valid DAG Automata transition	Statement template(s)
$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\text{☺}, \text{😎}\}$	$S = v_{\text{☺}} + L + v_{\text{😎}}$

We use parameterized states:

`label(number,direction)`

The range of direction: `unchanged`, `empty`, `reversed`.

Transducation Rules

Recognition Part	**Generation Part**
A valid DAG Automata transition	Statement template(s)
$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{$☺, 😎$\}$	$S = v_{☺} + L + v_{😎}$

We use parameterized states:

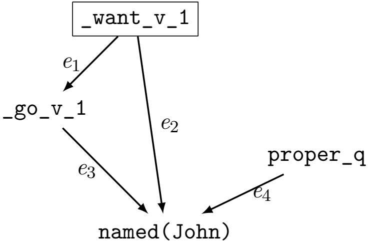
`label(number,direction)`

The range of `direction`: `unchanged`, `empty`, `reversed`.

$$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\} \quad S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$$

Toy Example

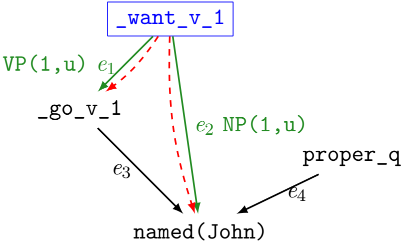
$Q = \{\texttt{DET(1,r)}, \texttt{Empty(0,e)}, \texttt{VP(1,u)}, \texttt{NP(1,u)}\}$		
Rule	For Recognition	For Generation
1	$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$	$v_{\texttt{DET(1,r)}} = \epsilon$
2	$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$	$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$
3	$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$	$v_{\texttt{VP(1,u)}} = \texttt{to} + L$
4	$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$	$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$



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

Toy Example

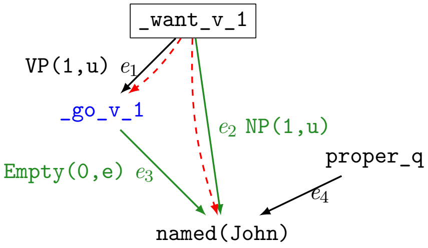
<table>
<tr><td colspan="3">$Q = \{\texttt{DET(1,r)}, \texttt{Empty(0,e)}, \texttt{VP(1,u)}, \texttt{NP(1,u)}\}$</td></tr>
<tr><td>Rule</td><td>For Recognition</td><td>For Generation</td></tr>
<tr><td>1</td><td>$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$</td><td>$v_{\texttt{DET(1,r)}} = \epsilon$</td></tr>
<tr><td>2</td><td>$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$</td><td>$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$</td></tr>
<tr><td>3</td><td>$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$</td><td>$v_{\texttt{VP(1,u)}} = \text{to} + L$</td></tr>
<tr><td>4</td><td>$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$</td><td>$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$</td></tr>
</table>



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

Toy Example

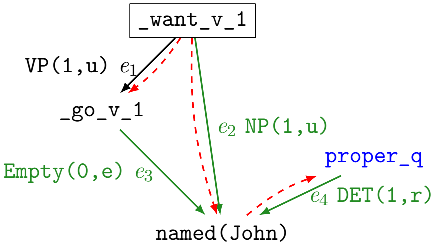
$Q = \{\texttt{DET(1,r)}, \texttt{Empty(0,e)}, \texttt{VP(1,u)}, \texttt{NP(1,u)}\}$		
Rule	For Recognition	For Generation
1	$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$	$v_{\texttt{DET(1,r)}} = \epsilon$
2	$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$	$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$
3	$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$	$v_{\texttt{VP(1,u)}} = \text{to} + L$
4	$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$	$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$



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

Toy Example

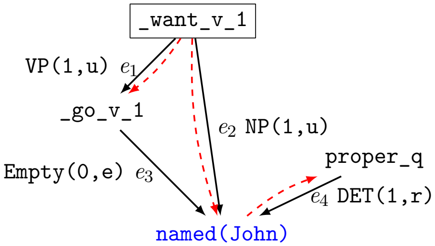
$Q = \{$DET(1,r), Empty(0,e), VP(1,u), NP(1,u)$\}$		
Rule	For Recognition	For Generation
1	$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$	$v_{\texttt{DET(1,r)}} = \epsilon$
2	$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$	$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$
3	$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$	$v_{\texttt{VP(1,u)}} = \text{to} + L$
4	$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$	$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$



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

Toy Example

$Q = \{$DET(1,r), Empty(0,e), VP(1,u), NP(1,u)$\}$		
Rule	For Recognition	For Generation
1	$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$	$v_{\texttt{DET(1,r)}} = \epsilon$
2	$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$	$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$
3	$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$	$v_{\texttt{VP(1,u)}} = \text{to} + L$
4	$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$	$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$



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

Accept !

Toy Example

$Q = \{\texttt{DET(1,r)}, \texttt{Empty(0,e)}, \texttt{VP(1,u)}, \texttt{NP(1,u)}\}$		
Rule	For Recognition	For Generation
1	$\{\} \xrightarrow{\texttt{proper\_q}} \{\texttt{DET(1,r)}\}$	$v_{\texttt{DET(1,r)}} = \epsilon$
2	$\{\} \xrightarrow{\texttt{\_want\_v\_1}} \{\texttt{VP(1,u)}, \texttt{NP(1,u)}\}$	$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$
3	$\{\texttt{VP(1,u)}\} \xrightarrow{\texttt{\_go\_v\_1}} \{\texttt{Empty(0,e)}\}$	$v_{\texttt{VP(1,u)}} = \text{to} + L$
4	$\{\texttt{NP(1,u)}, \texttt{DET(1,r)}\} \xrightarrow{\texttt{named}} \{\}$	$v_{\texttt{NP(1,u)}} = v_{\texttt{DET(1,r)}} + L$

$$S = v_{\texttt{NP(1,u)}} + L + v_{\texttt{VP(1,u)}}$$

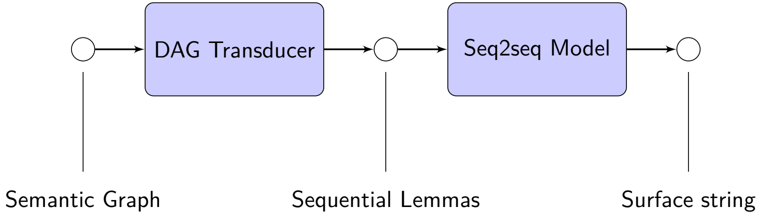
$$\Downarrow$$

$$S = x_{21} + \texttt{want} + x_{11}$$

Instantiation: replace $v_{l(j,d)}$ of edge e_i with variable x_{ij} and L with the output string in the statement templates.

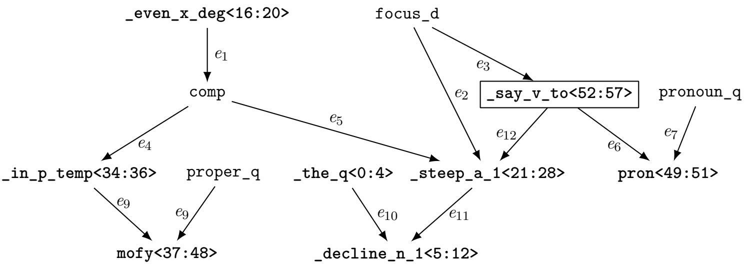
DAG Transduction based-NLG

A general framework for DAG transduction based-NLG:

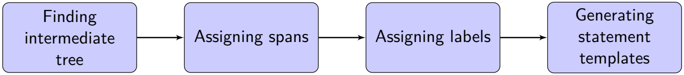


Outline

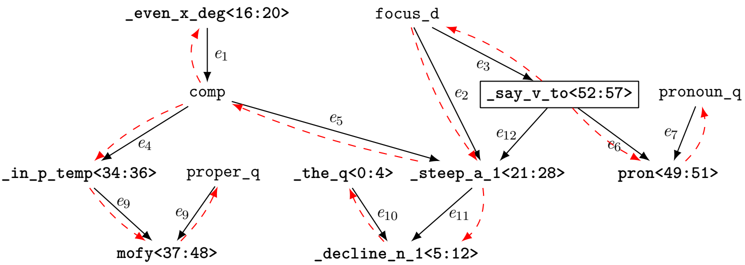
Inducing Transduction Rules



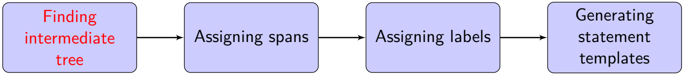
"the decline is even steeper than in September", he said.



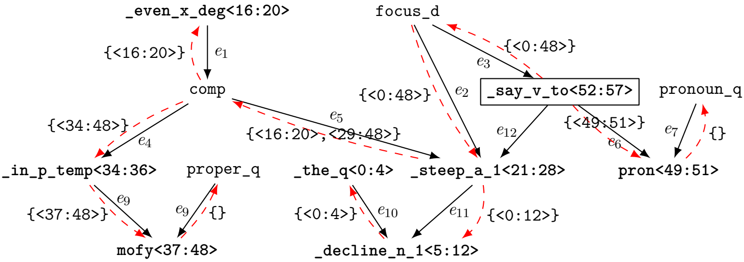
Inducing Transduction Rules



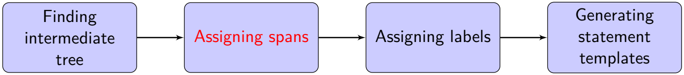
"the decline is even steeper than in September", he said.



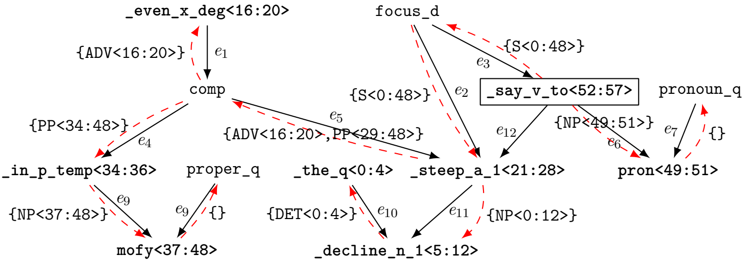
Inducing Transduction Rules



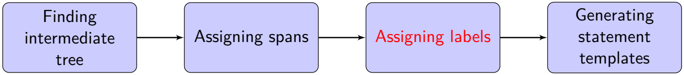
"the decline is even steeper than in September", he said.



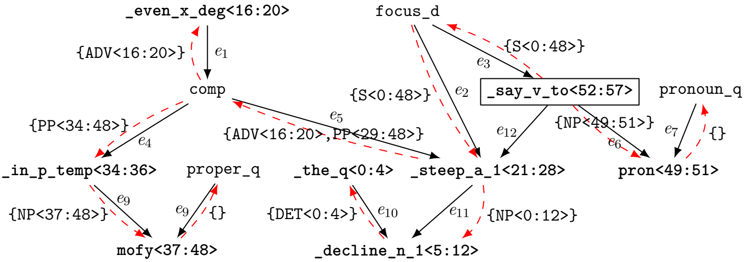
Inducing Transduction Rules



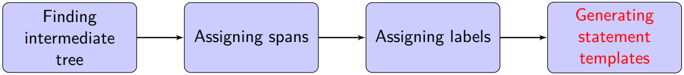
"the decline is even steeper than in September", he said.



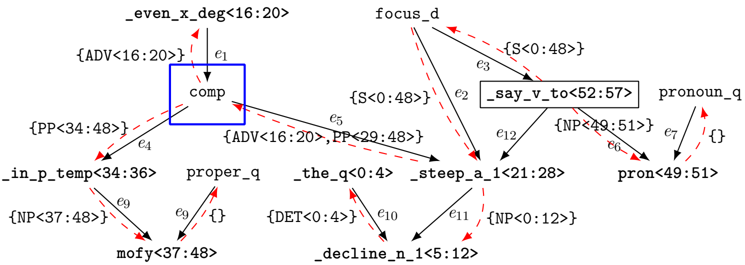
Inducing Transduction Rules



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



Inducing Transduction Rules



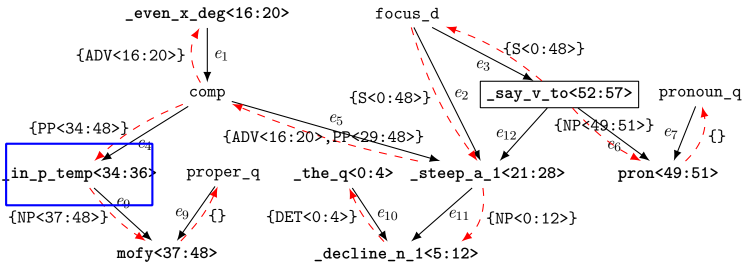
"the decline is even steeper than in September", he said.

$$\{\mathtt{ADV(1,r)}\} \xrightarrow{\mathtt{comp}} \{\mathtt{PP(1,u)}, \mathtt{ADV\_PP(2,r)}\}$$

$$v_{\mathtt{ADV\_PP(1,r)}} = v_{\mathtt{ADV(1,r)}}$$

$$v_{\mathtt{ADV\_PP(2,r)}} = \mathtt{than} + v_{\mathtt{PP(1,u)}}$$

Inducing Transduction Rules



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

$$\{\texttt{PP(1,u)}\} \xrightarrow{\texttt{\_in\_p\_temp}} \{\texttt{NP(1,u)}\}$$

$$v_{\texttt{PP(1,u)}} = \texttt{in} + v_{\texttt{NP(1,u)}}$$

NLG via DAG transduction

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

Transducer	Lemmas	Sentences	Coverage
induced rules	89.44	74.94	67%

Fine-to-coarse Transduction

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

$$\{\texttt{NP}, \texttt{ADJ}\} \xrightarrow{\texttt{X}} \{\} \qquad v_{\texttt{NP}} = v_{\texttt{ADJ}} + L$$

New rule generated by deleting:

$$\{\texttt{NP}\} \xrightarrow{\texttt{X}} \{\} \qquad v_{\texttt{NP}} = L$$

Fine-to-coarse Transduction

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

$$\{\mathtt{NP}, \mathtt{ADJ}\} \xrightarrow{\mathtt{X}} \{\} \qquad v_{\mathtt{NP}} = v_{\mathtt{ADJ}} + L$$

New rule generated by copying:

$$\{\mathtt{NP}, \mathtt{ADJ}_1, \mathtt{ADJ}_2\} \xrightarrow{\mathtt{X}} \{\} \qquad v_{\mathtt{NP}} = v_{\mathtt{ADJ}_1} + v_{\mathtt{ADJ}_2} + L$$

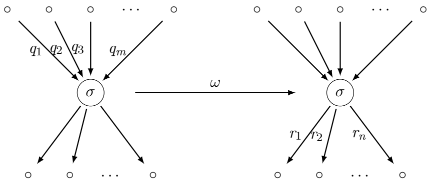
NLG via DAG transduction

Experimental set-up

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

Transducer	Lemmas	Sentences	Coverage
induced rules	89.44	74.94	67%
induced and exteneded rules	88.41	74.03	77%

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

Transducer	Lemmas	Sentences	Coverage
induced rules	89.44	74.94	67%
induced and exteneded rules	88.41	74.03	77%
induced, exteneded and dynamic rules	82.04	68.07	100%
DFS-NN	50.45		100%
AMR-NN		33.8	100%
AMR-NRG		25.62	100%

Conclusion and Future Work

English Resouce 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.

QUESTIONS?

COMMENTS?