Towards a Compositional Typed Semantics for Universal Dependencies

Siva Reddy

School of Informatics
The University of Edinburgh

People

Mirella Lapata

Mark Steedman

People

Mirella Lapata

Mark Steedman

Oscar Täckström

Dipanjan Das

Tom Kwiatkowski

Michael Collins

Slav Petrov

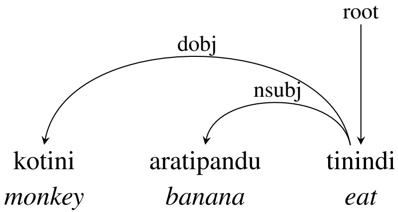
Syntax helps Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Syntax helps Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Syntax helps Semantics



Syntax helps Semantics

root

dobj

nsubj

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Syntax in humans?

Studies on Peruvian Indian bilinguals indicate Quechua word order influences the local varieties of Spanish [Odlin, 1989]

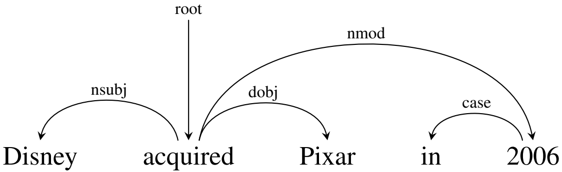
Syntax in humans?

Studies on Peruvian Indian bilinguals indicate Quechua word order influences the local varieties of Spanish [Odlin, 1989]

Arabs show strong preference for SVO in Dutch, whereas Turks for SOV [Jansen et al., 1981; Appel, 1984]

Universal Dependencies

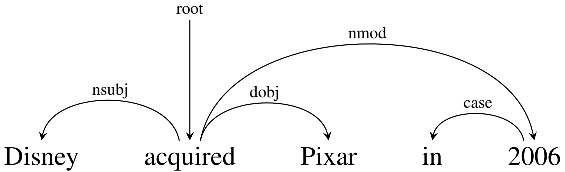
Homogeneous syntactic representation across languages

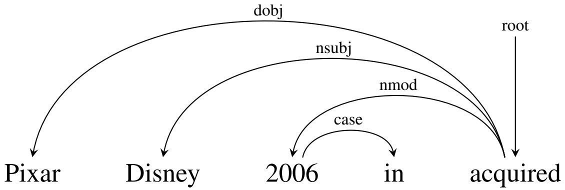


Pixar Disney 2006 in acquired

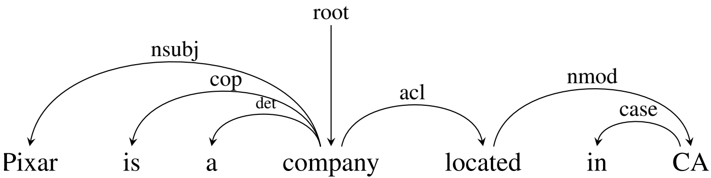
Universal Dependencies

Homogeneous syntactic representation across languages



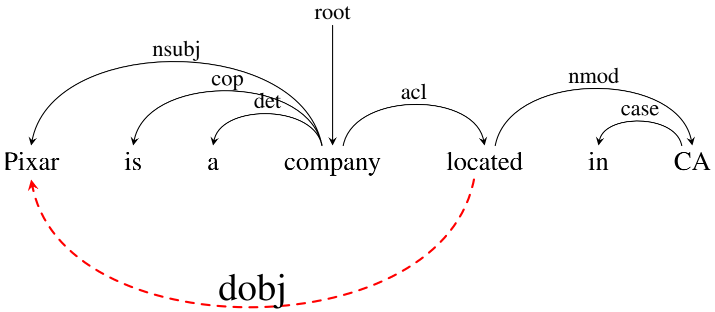


Dependencies to Logical Forms



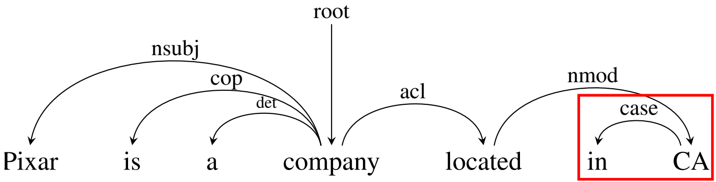
$$\exists z.\mathrm{company}(\mathrm{Pixar}) \wedge \mathrm{located}(z_e) \wedge \mathrm{arg}_2(z_e, \mathrm{Pixar}) \wedge \mathrm{arg}_{\mathrm{in}}(z_e, \mathrm{CA})$$

Dependencies to Logical Forms



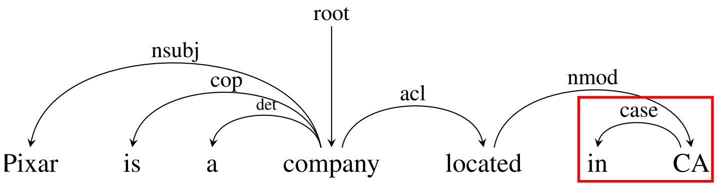
$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \text{arg}_2(z_e, \text{Pixar}) \wedge \text{arg}_{\text{in}}(z_e, \text{CA})$$

Dependencies to Logical Forms



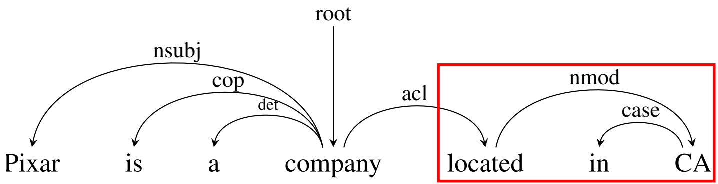
$(\mathit{case} \quad \mathrm{CA} \quad \mathrm{in})$

Dependencies to Logical Forms



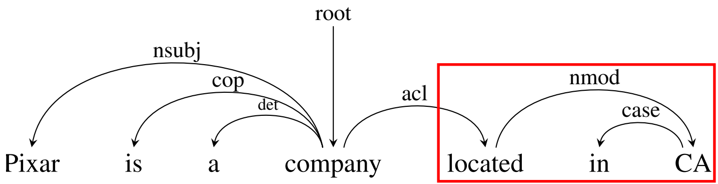
$$\frac{\begin{array}{cccc} & (\mathit{case} & \text{CA} & \text{in}) \\ {}_e) & \lambda f g x. f(x) & \lambda x.\text{CA}(x_a) & \lambda x.\text{empty}(x \end{array}}{\lambda x.\ \text{CA}(x_a)}$$

Dependencies to Logical Forms



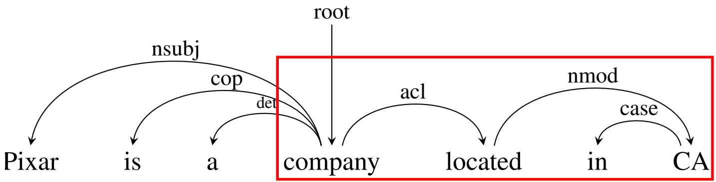
$(nmod \quad \text{located} \quad \text{in CA}) \quad).$

Dependencies to Logical Forms



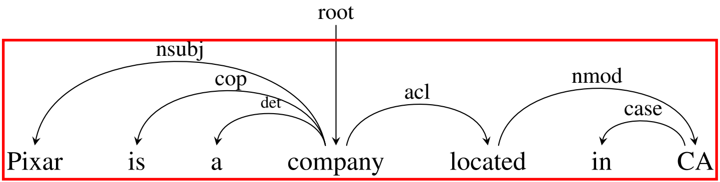
$$\frac{\begin{array}{c}(\mathit{nmod} \\ \lambda f g z.\exists x. \\ f(z) \wedge g(x) \wedge \\ \wedge \mathrm{arg}_{\mathrm{in}}(z_e, x_a)\end{array} \qquad \begin{array}{c}\text{located} \\ \lambda x.\text{located}(x_e)\end{array} \qquad \dfrac{\text{in CA})}{\lambda x.\ \text{CA}(x_a)}}{\lambda z.\text{located}(z_e) \wedge \text{CA}(x_a) \wedge \mathrm{arg}_{\mathrm{in}}(z_e, x_a)}$$

Dependencies to Logical Forms



$$\dfrac{\begin{array}{c}(acl \\ \lambda f g x.\exists z. \\ f(x) \wedge g(x) \wedge \\ \text{arg}_2(z_e, x_a)\end{array} \qquad \begin{array}{c}\text{company} \\ \lambda x.\text{compay}(x_a)\end{array} \qquad \dfrac{\text{located in CA})}{\lambda g z.\exists x.\text{located}(z_e) \wedge \text{CA}(x_a) \wedge \text{arg}_{\text{in}}(z_e, x_a)}}{\lambda x.\exists y z.\text{company}(x_a) \wedge \text{located}(z_e) \wedge \text{CA}(y_a) \wedge \text{arg}_2(z_e, x_a) \wedge \text{arg}_{\text{in}}(z_e, y_a)}$$

Dependencies to Logical Forms



$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \text{arg}_2(z_e, \text{Pixar}) \wedge \text{arg}_{\text{in}}(z_e, \text{CA})$$

Synchronous Syntax-Semantics Interfaces

Derive semantics from syntactic derivation

- CCG [Bos et al., 2004; Lewis & Steedman, 2013]
- HPSG [Copestake et al., 2001]
- LFG [Dalrymple et al., 1995]
- TAG [Joshi et al., 1995]

CCG has been a popular choice

- No treebanks for many languages
- Syntactic categories differ in each language

Why from Universal Dependencies?

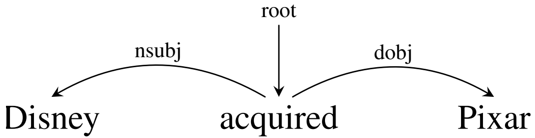
Treebanks in more than 40 languages

Very accurate parsers

[Andor et al., 2016; Chen & Manning, 2014]

Universal Types, both syntactic and semantic

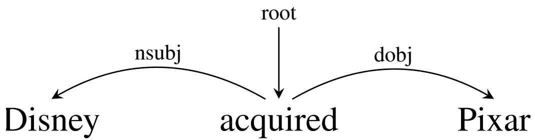
Dependencies to Logical Forms



$$\lambda z.\exists xy.\text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \text{Disney}(x_a) \wedge$$
$$\text{arg}_1(z_e, x_a) \wedge \text{arg}_2(z_e, y_a)$$

Dependencies to Logical Forms

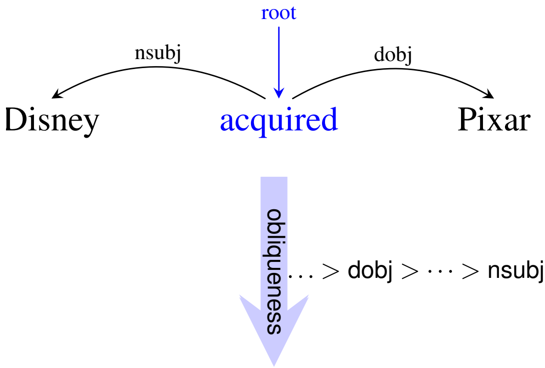
Our Approach



Let dependency labels drive the composition

Dependencies to Logical Forms

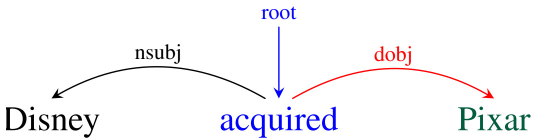
Our Approach

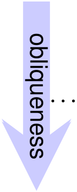


$\ldots > \text{dobj} > \cdots > \text{nsubj} > \ldots$

Dependencies to Logical Forms

Our Approach



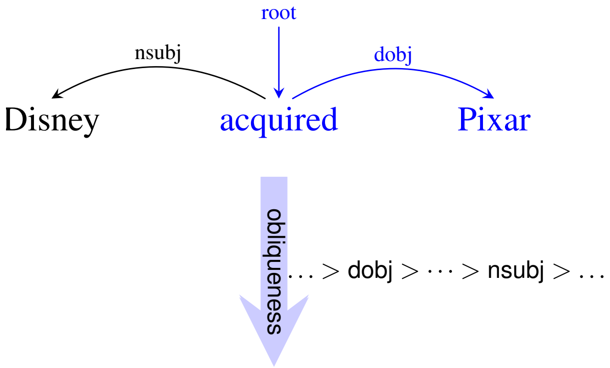


$\ldots >$ dobj $> \cdots >$ nsubj $> \ldots$

(dobj acquired Pixar)

Dependencies to Logical Forms

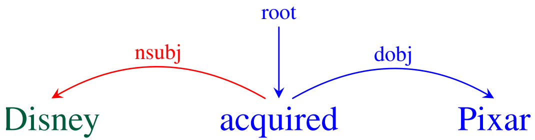
Our Approach

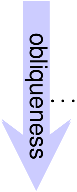


(dobj acquired Pixar)

Dependencies to Logical Forms

Our Approach



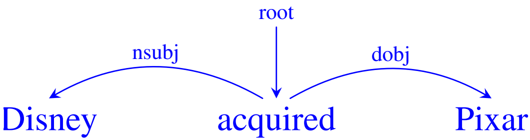


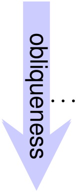
$\ldots >$ dobj $> \cdots >$ nsubj $> \ldots$

(nsubj (dobj acquired Pixar) Disney)

Dependencies to Logical Forms

Our Approach



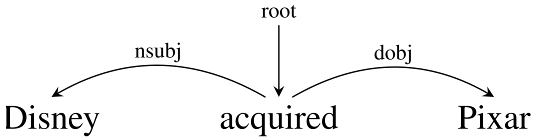


$\ldots >$ dobj $> \cdots >$ nsubj $> \ldots$

(nsubj (dobj acquired Pixar) Disney)

Dependencies to Logical Forms

Our Approach

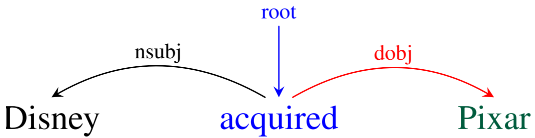


(nsubj (dobj acquired Pixar) Disney)

$$\lambda z.\exists xy.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{Disney}(x_a) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

Dependencies to Logical Forms

Lambda Calculus

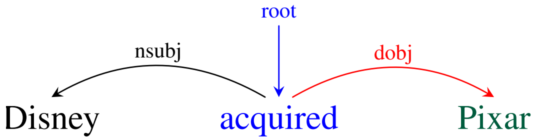


Lambda Calculus Basic Types

- Individuals: $\mathbf{Ind}$ (also denoted by $\cdot_a$)
- Events: $\mathbf{Event}$ (also denoted by $\cdot_e$)
- Truth values: $\mathbf{Bool}$

Dependencies to Logical Forms

Lambda Calculus



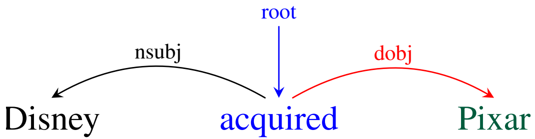
Lambda Expression for words

$$\text{acquired} \Rightarrow \lambda x.\,\mathsf{acquired}(x_e)$$

$$\text{Pixar} \Rightarrow \lambda x.\,\mathsf{Pixar}(x_a)$$

Dependencies to Logical Forms

Lambda Calculus

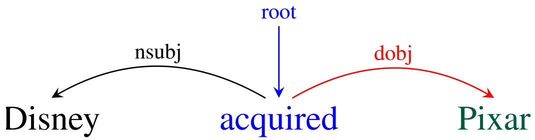


Lambda Expression for dependency labels

$$\text{dobj} \Rightarrow \lambda \mathbf{f}\ \mathbf{g}\ \mathbf{z}\ .\ \exists \mathbf{x}\ .\ \mathbf{f}(\mathbf{z}) \quad \wedge \quad \mathbf{g}(\mathbf{x}) \quad \wedge \quad \mathbf{arg_2}(\mathbf{z_e}, \mathbf{x_a})$$

Dependencies to Logical Forms

Lambda Calculus



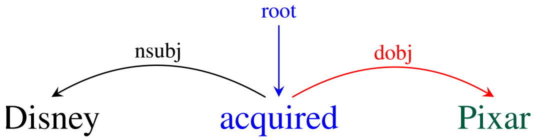
Lambda Expression for dependency labels

$$\text{dobj} \Rightarrow \lambda \mathbf{f}\ \mathbf{g}\ \mathbf{z}\ .\ \exists \mathbf{x}\ .\ \mathbf{f}(\mathbf{z}) \quad \wedge \quad \mathbf{g}(\mathbf{x}) \quad \wedge \quad \mathbf{arg_2}(\mathbf{z_e}, \mathbf{x_a})$$

This operation mirrors the tree structure

Dependencies to Logical Forms

Composition



(**dobj**	**acquired**	**Pixar**)
$\lambda f g z . \exists y . f(z) \wedge g(y) \wedge \arg_2(z_e, y_a)$	$\lambda z.\text{acquired}(z_e)$	$\lambda y.\text{Pixar}(y_a)$

Dependencies to Logical Forms

Composition

root

nsubj

dobj

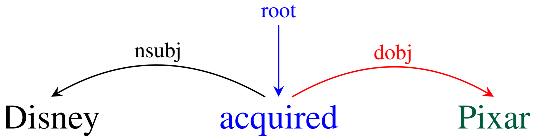
Disney acquired Pixar

$$
\begin{array}{ccc}
(\textbf{dobj} & \textbf{acquired} & \textbf{Pixar}) \\
\lambda f g z . \exists y . & \lambda z . \text{acquired}(z_e) & \lambda y . \text{Pixar}(y_a) \\
f(z) \wedge g(y) \wedge & & \\
\text{arg}_2(z_e, y_a) & &
\end{array}
$$

$$
\frac{}{\lambda g z .\ \exists y .\ \text{acquired}(z_e) \wedge g(y) \wedge \text{arg}_2(z_e, y_a)}
$$

Dependencies to Logical Forms

Composition



(**dobj** **acquired** **Pixar**)

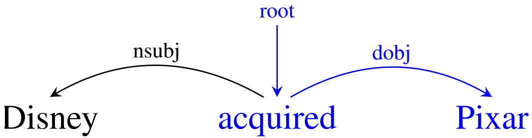
$$\lambda f g z. \exists y. f(z) \wedge g(y) \wedge \mathrm{arg}_2(z_e, y_a) \qquad \lambda z.\mathrm{acquired}(z_e) \qquad \lambda y.\mathrm{Pixar}(y_a)$$

$$\lambda g z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge g(y) \wedge \mathrm{arg}_2(z_e, y_a)$$

$$\lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

Dependencies to Logical Forms

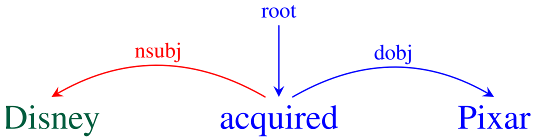
Composition



$$\frac{(\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar})}{\lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \\ \wedge\ \mathrm{arg}_2(z_e, y_a)}$$

Dependencies to Logical Forms

Composition



$$
\begin{array}{ccc}
(\textbf{nsubj} & (\textbf{dobj} \quad \textbf{acquired} \qquad \textbf{Pixar}) & \textbf{Disney}) \\
\lambda f g z.\ \exists x. & \overline{\phantom{\lambda z.\ \exists y.\ \text{acquired}(z_e) \wedge \text{Pixar}(y_a)}} & \lambda x.\text{Disney}(x_a) \\
f(z) \wedge g(x) \wedge & \lambda z.\ \exists y.\ \text{acquired}(z_e) \wedge \text{Pixar}(y_a) & \\
\text{arg}_1(z_e, x_a) & \wedge\ \text{arg}_2(z_e, y_a) &
\end{array}
$$

Dependencies to Logical Forms

Composition

root

nsubj　　　　dobj

Disney　　acquired　　Pixar

(**nsubj** (**dobj** **acquired** **Pixar**) **Disney**)

$$\lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \mathrm{arg}_1(z_e, x_a)$$

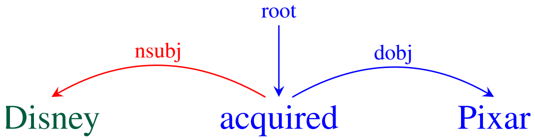
$$\frac{(\textbf{dobj}\ \textbf{acquired}\ \textbf{Pixar})}{\lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{arg}_2(z_e, y_a)}$$

$$\lambda x.\mathrm{Disney}(x_a)$$

$$\frac{\lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \mathrm{arg}_1(z_e, x_a) \qquad \lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{arg}_2(z_e, y_a)}{\lambda g z.\exists x y.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{g}(x) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)}$$

Dependencies to Logical Forms

Composition



(**nsubj** (**dobj** **acquired** **Pixar**) **Disney**)

$$\lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \mathrm{arg}_1(z_e, x_a)$$

$$\lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

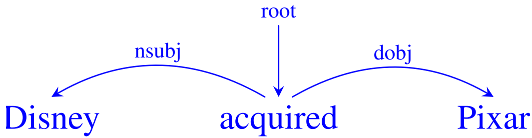
$$\lambda x.\mathrm{Disney}(x_a)$$

$$\lambda g z.\exists x y.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{g}(x) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

$$\lambda z.\exists x y.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{Disney}(x_a) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

Dependencies to Logical Forms

Composition



$$\frac{(\textbf{nsubj}\quad(\textbf{dobj}\quad\textbf{acquired}\quad\textbf{Pixar})\quad\textbf{Disney})}{\lambda z.\exists xy.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{Disney}(x_a) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)}$$

Dependencies to Logical Forms

Disney $\xrightarrow{\text{appos}}$ the company

$$\textit{appos} = \lambda fgx.f(x) \land g(x)$$

company $\xrightarrow{\text{partmod}}$ located in CA

$$\textit{partmod} = \lambda fgx.\exists z.f(x) \land g(z) \land \mathsf{arg}_2(z_e, x_a)$$

Disney $\xrightarrow{\text{conj}}$ and Pixar

$$\textit{conj} = \lambda fgz.\exists xy.f(x) \land g(y) \land \mathrm{coord}(z, x, y)$$

Comparison with CCG

$$
\dfrac{\dfrac{\text{Disney}}{NP} \quad \dfrac{\dfrac{\text{acquired}}{S\backslash NP/NP} \quad \dfrac{\text{Pixar}}{NP}}{S\backslash NP}>}{S}<
$$

Semantics:

- Disney: Disney
- acquired: $\lambda y \lambda x \lambda e.\ \text{acquired}(e) \wedge \text{arg}_1(e, x) \wedge \text{arg}_2(e, y)$
- Pixar: Pixar

$S\backslash NP$ (>): $\lambda x \lambda e.\ \text{acquired}(e) \wedge \text{arg}_1(e, x) \wedge \text{arg}_2(e, \text{Pixar})$

S (<): $\lambda e.\ \text{acquired}(e) \wedge \text{arg}_1(e, \text{Disney}) \wedge \text{arg}_2(e, \text{Pixar})$

CCG	DepLambda
Lexicalized semantics S\NP/NP : $\lambda y \lambda x \lambda e.\, \mathrm{acquired}(e) \wedge \mathrm{arg}_1(e,x) \wedge \mathrm{arg}_2(e,y)$	Simple lexical semantics $\lambda x.\, \mathrm{acquired}(x_e)$
Words drive composition	Dependencies drive composition
Language specific types	Mostly universal
Argument adjunct distinction $S\backslash NP/PP/NP$ vs. $(S\backslash NP)\backslash(S\backslash NP)/NP$	Every dependent is an adjunct

CCG	DepLambda
Lexicalized semantics S\NP/NP : $\lambda y \lambda x \lambda e.\mathrm{acquired}(e) \wedge \mathrm{arg}_1(e,x) \wedge \mathrm{arg}_2(e,y)$	Simple lexical semantics $\lambda x.\mathrm{acquired}(x_e)$
Words drive composition	Dependencies drive composition
Language specific types	Mostly universal
Argument adjunct distinction $S\backslash NP/PP/NP$ vs. $(S\backslash NP)\backslash(S\backslash NP)/NP$	Every dependent is an adjunct
With "complex types" comes power $(S[dcl]\backslash NP)/(S[to]\backslash NP_{\mathbf{x}})/NP_{\mathbf{x}}$	Single-type system is robust, but restricted $\lambda fgx\ldots$ i.e., $\eta \rightarrow \eta \rightarrow \eta$ for all labels

Relative clause in CCG

company	which	Disney	acquired
N	$(N_x\backslash N_x)/(S_{dcl}/NP_x)$	N	$(S_{dcl}\backslash NP)/NP$
$\lambda x.company(x)$	$\lambda p\lambda q\lambda x.q(x) \wedge \exists e[p(x,e)]$		$\lambda x\lambda y\lambda e.acquire(e) \wedge A0(y,e) \wedge A1(x,e)$

Disney:

$$\frac{N}{NP \quad disney}$$

$$\frac{NP}{S_X/(S_X\backslash NP) \quad \lambda p\lambda e.p(disney,e)}$$

Disney acquired:

$$S_{dcl}/NP$$

$$\lambda x\lambda e.acquire(e) \wedge A0(disney,e) \wedge A1(x,e)$$

which Disney acquired:

$$N\backslash N$$

$$\lambda p\lambda x.p(x) \wedge \exists e[acquire(e) \wedge A0(disney,e) \wedge A1(x,e)]$$

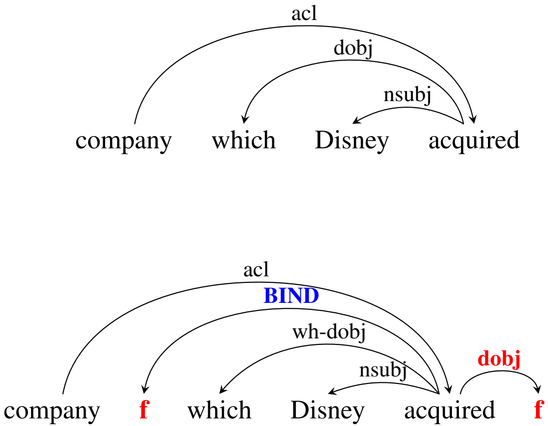
company which Disney acquired:

$$N$$

$$\lambda x.company(x) \wedge \exists e[acquire(e) \wedge A0(disney,e) \wedge A1(x,e)]$$

Relative Clause in DepLambda

following Carpenter (1998)



DepLambda in a nutshell

Dependency tree is a series of compositions

- Dependency label defines the composition function
- Each function takes two typed-semantic sub-expressions
- Returns typed-semantics of the larger expression

Function could be any computation

- e.g., a neural network

Richer context-sensitive types will allow richer composition functions

- e.g., neural networks with tensors/neural networks
- directly to the target-application semantics

DepLambda in a nutshell

Dependency tree is a series of compositions

- Dependency label defines the composition function
- Each function takes two typed-semantic sub-expressions
- Returns typed-semantics of the larger expression

Function could be any computation

- e.g., a neural network

Richer context-sensitive types will allow richer composition functions

- e.g., neural networks with tensors/neural networks
- directly to the target-application semantics

DepLambda in a nutshell

Dependency tree is a series of compositions

- Dependency label defines the composition function
- Each function takes two typed-semantic sub-expressions
- Returns typed-semantics of the larger expression

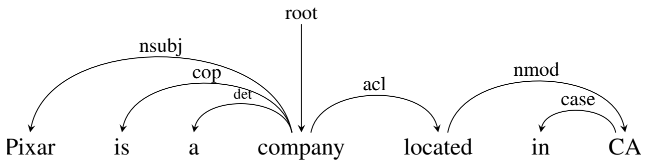
Function could be any computation

- e.g., a neural network

Richer context-sensitive types will allow richer composition functions

- e.g., neural networks with tensors/neural networks
- directly to the target-application semantics

Recap



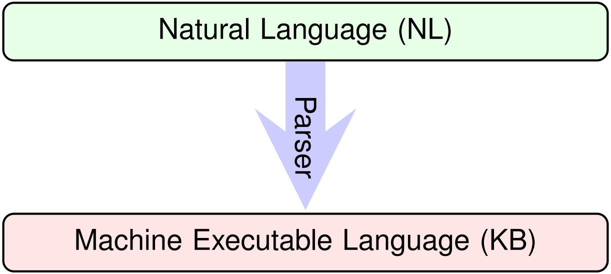
$\ldots >$ dobj $> \cdots >$ nsubj $> \ldots$

$$
\begin{array}{ccccccccc}
\ldots & (acl & \text{company} & (nmod & \text{located} & (case & \text{CA} & \text{in}) &))\ldots \\
 & \lambda fgx.\exists z. & \lambda x.\text{compay}(x_a) & \lambda fgz.\exists x. & \lambda x.\text{located}(x_e) & \lambda fgx.f(x) & \lambda x.\text{CA}(x_a) & \lambda x.\text{empty}(x) & \\
 & f(x) \wedge g(z) \wedge & & f(z) \wedge g(x) & & \multicolumn{3}{c}{\rule{6cm}{0.4pt}} & \\
 & \text{arg}_2(z_e, x_a) & & arg_{in}(z_e, x_a) & & \multicolumn{3}{c}{\lambda x.\ \text{CA}(x_a)} &
\end{array}
$$

lambda expression composition

$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \text{arg}_2(z_e, \text{Pixar}) \wedge \text{arg}_{\text{in}}(z_e, \text{CA})$$

Grounded Semantic Parsing



Freebase Semantic Parsing

Titanic

1997 · Drama film/Romance · 3h 30m

7.7/10 · IMDb
88% · Rotten Tomatoes

James Cameron's "Titanic" is an epic, action-packed romance set against the ill-fated maiden voyage of the R.M.S. Titanic; the pride and joy of the White Star Line and, at the time, the larg… More

Initial release: November 18, 1997 (London)

Director: James Cameron

Featured song: My Heart Will Go On

Cast

Leonardo DiCaprio	Kate Winslet	Billy Zane	Gloria Stuart	Kathy Bates
Jack Dawson	Rose DeWitt Bukater	Caledon Hockley	Rose DeWitt Bukater	Molly Brown

Freebase Semantic Parsing

NL

Leonardo DiCaprio starred as Jack in Titanic which was directed by James Cameron.

Freebase Semantic Parsing

NL Leonardo DiCaprio starred as Jack in Titanic which was directed by James Cameron.

Parser

KB $\text{cast}(\textsc{Titanic}, \textsc{DiCaprio}, \textsc{Jack}) \wedge \text{director}(\textsc{Titanic}, \textsc{Cameron})$

Freebase Semantic Parsing

NL Leonardo DiCaprio starred as Jack in Titanic which was directed by James Cameron.

Parser

KB $\text{cast}(\textsc{Titanic}, \textsc{DiCaprio}, \textsc{Jack}) \wedge \text{director}(\textsc{Titanic}, \textsc{Cameron})$

Execute

TRUE

Freebase Semantic Parsing

[Berant et al., 2013, Kwiatkowski et al., 2013]

Question

Who is the director of Titanic?

Answer

$\{$James Cameron$\}$

Freebase Semantic Parsing

[Berant et al., 2013, Kwiatkowski et al., 2013]

Question

Who is the director of Titanic?

Logical Form

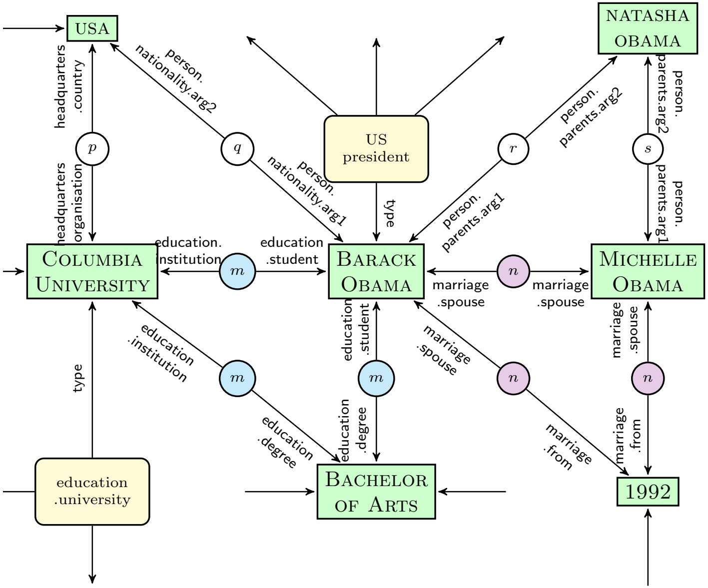
Latent

$\lambda x.$ film.directed_by$(Titanic, x)$

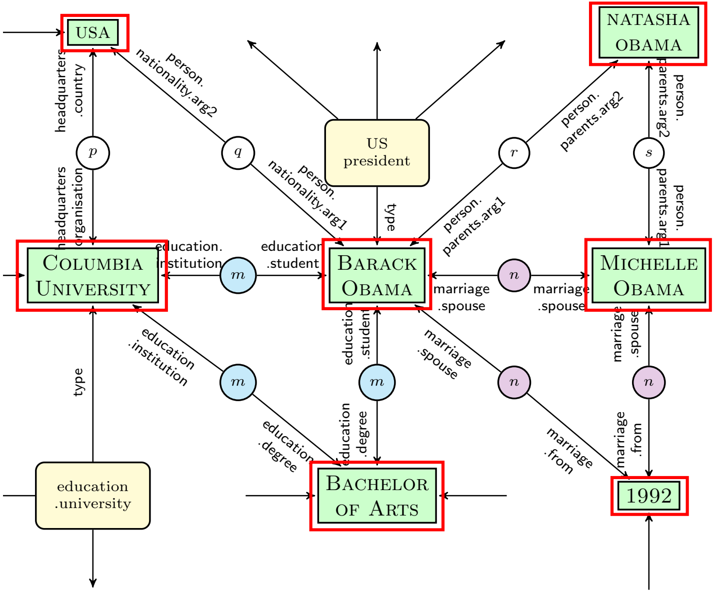
Answer

$\{$James Cameron$\}$

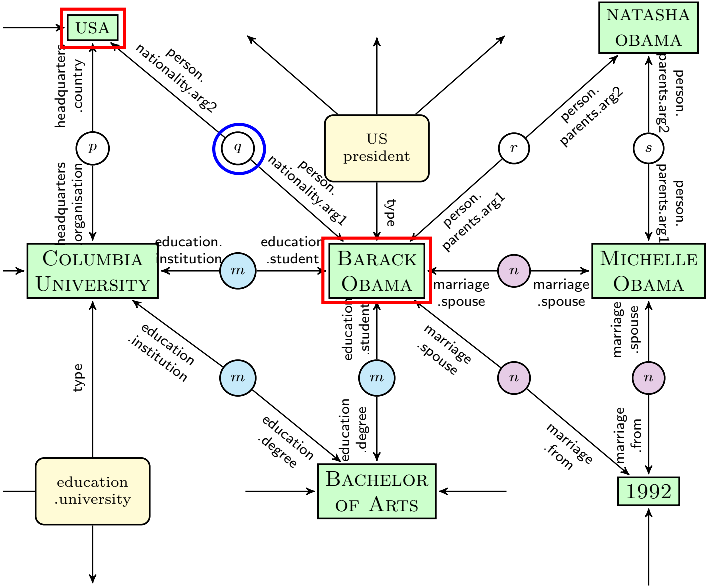
Freebase is a Graph



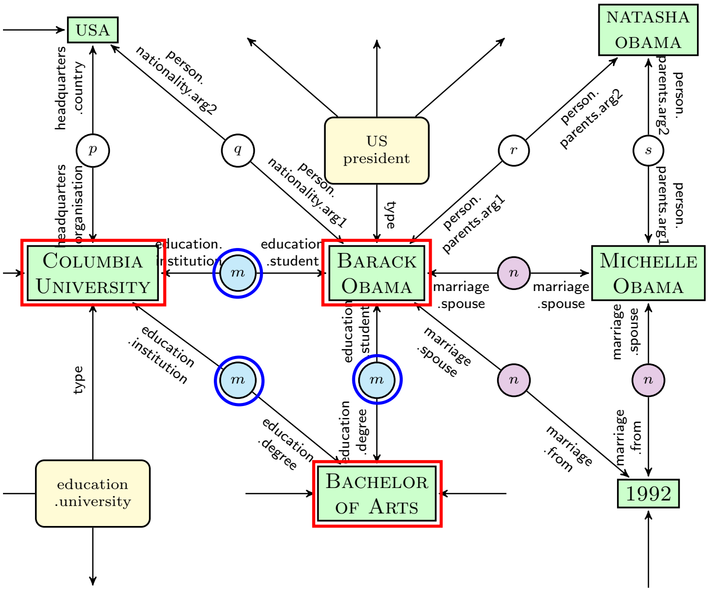
Freebase is a Graph



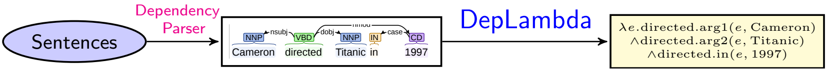
Freebase is a Graph



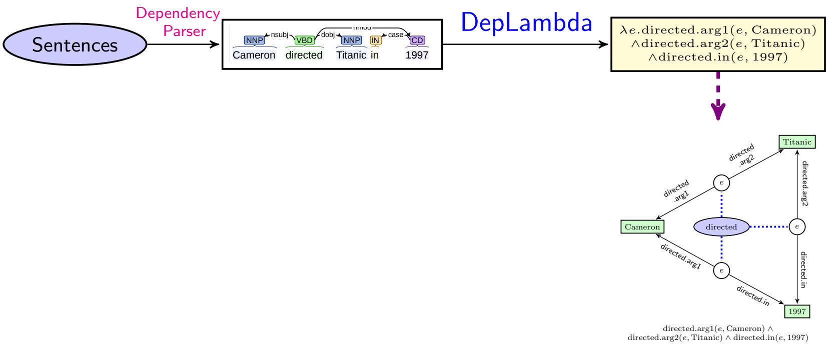
Freebase is a Graph



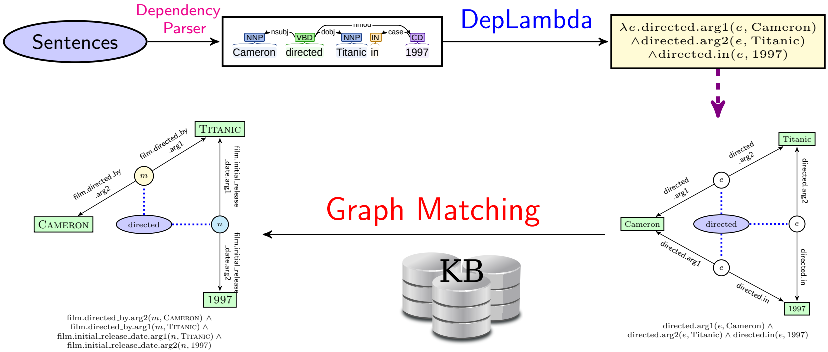
Semantic Parsing as Graph Matching



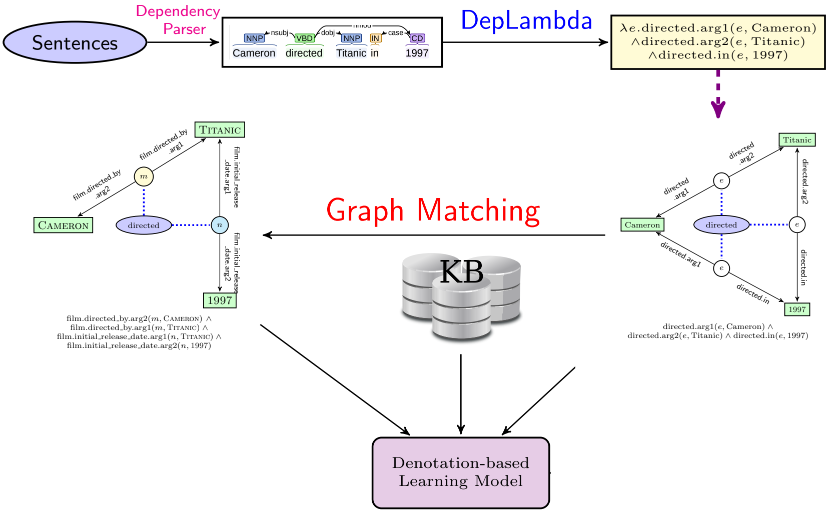
Semantic Parsing as Graph Matching



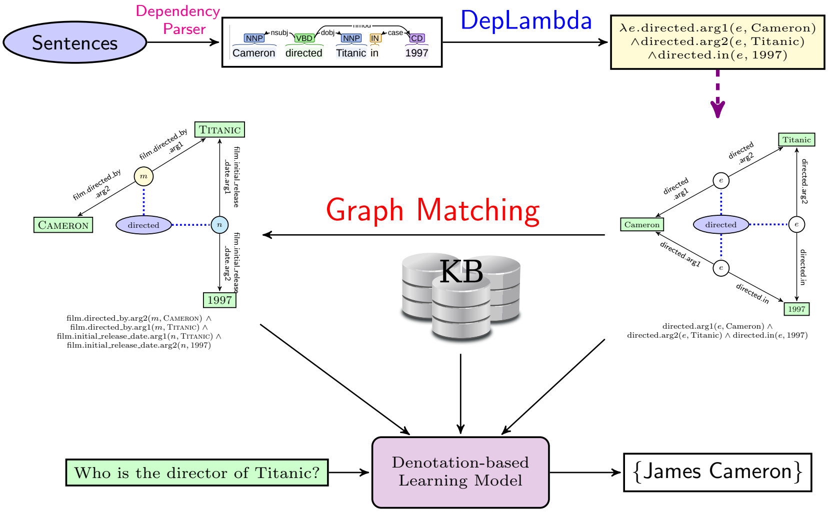
Semantic Parsing as Graph Matching



Semantic Parsing as Graph Matching



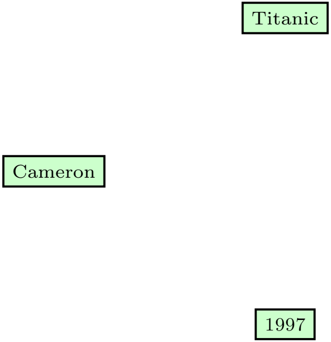
Semantic Parsing as Graph Matching



Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

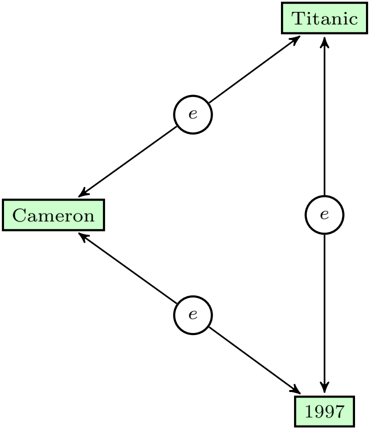
$$\lambda e.\text{directed.arg1}(e, \text{Cameron}) \land \text{directed.arg2}(e, \text{Titanic}) \land \text{directed.in}(e, 1997)$$



Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

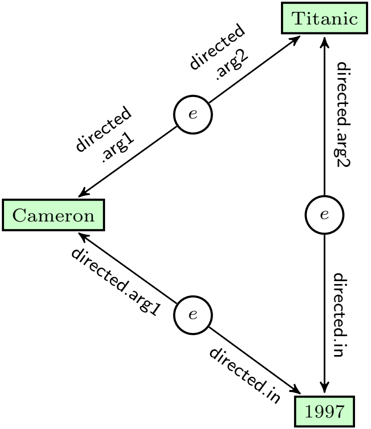
$$\lambda e.\text{directed.arg1}(\mathbf{e}, \texttt{Cameron}) \wedge \text{directed.arg2}(\mathbf{e}, \texttt{Titanic}) \wedge \text{directed.in}(\mathbf{e}, \texttt{1997})$$



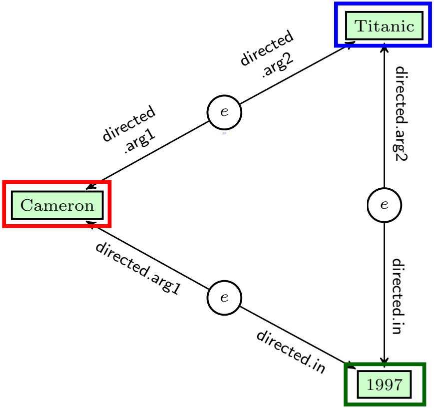
Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

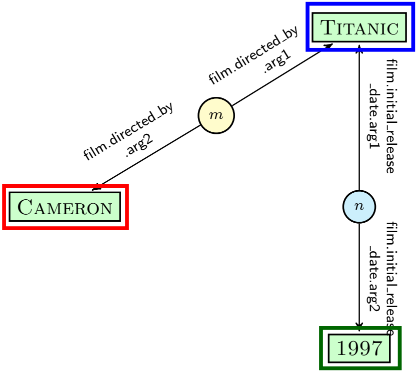
$$\lambda e.\textcolor{red}{\text{directed.arg1}}(e, \texttt{Cameron}) \wedge \textcolor{red}{\text{directed.arg2}}(e, \texttt{Titanic}) \wedge \textcolor{red}{\text{directed.in}}(e, \texttt{1997})$$



Graph Matching



directed.arg1(e, Cameron) $\wedge$
directed.arg2(e, Titanic) $\wedge$ directed.in(e, 1997)

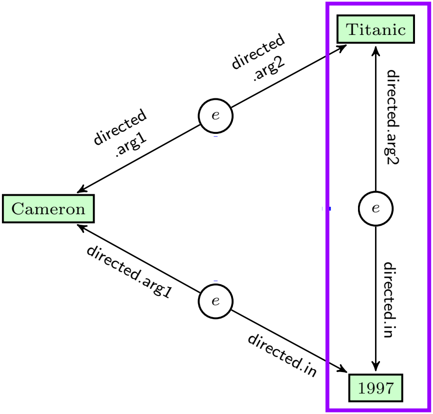


film.directed_by.arg2(m, CAMERON) $\wedge$
film.directed_by.arg1(m, TITANIC) $\wedge$
film.initial_release_date.arg1(n, TITANIC) $\wedge$
film.initial_release_date.arg2(n, 1997)

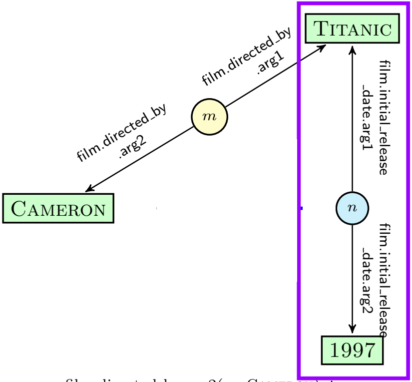
Ungrounded Graph

Grounded Graph

Graph Matching



$\text{directed.arg1}(e, \text{Cameron}) \wedge$
$\text{directed.arg2}(e, \text{Titanic}) \wedge \text{directed.in}(e, 1997)$

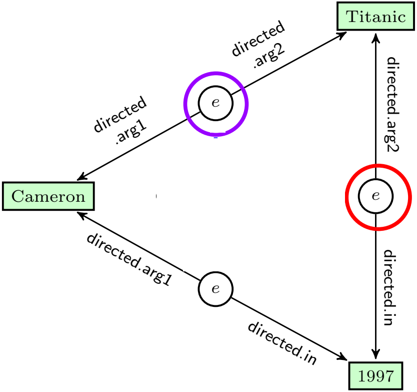


$\text{film.directed\_by.arg2}(m, \text{CAMERON}) \wedge$
$\text{film.directed\_by.arg1}(m, \text{TITANIC}) \wedge$
$\text{film.initial\_release\_date.arg1}(n, \text{TITANIC}) \wedge$
$\text{film.initial\_release\_date.arg2}(n, 1997)$

Ungrounded Graph

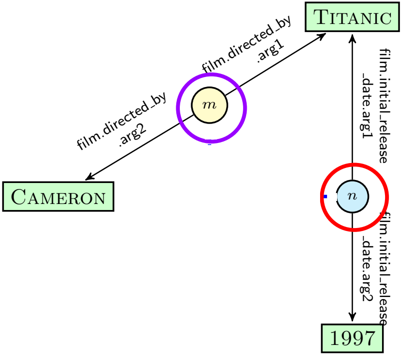
Grounded Graph

Graph Matching



$$\text{directed.arg1}(e, \text{Cameron}) \wedge \text{directed.arg2}(e, \text{Titanic}) \wedge \text{directed.in}(e, 1997)$$

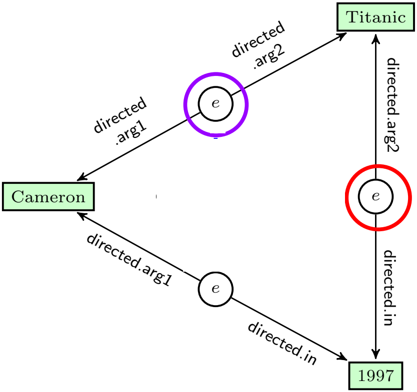
Ungrounded Graph



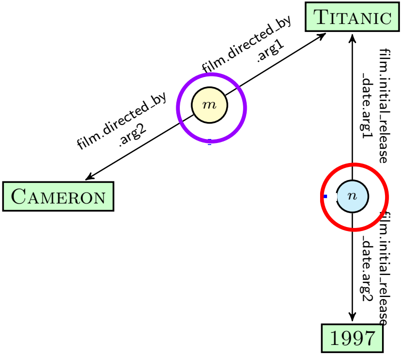
$$\text{film.directed\_by.arg2}(m, \textsc{Cameron}) \wedge \text{film.directed\_by.arg1}(m, \textsc{Titanic}) \wedge \text{film.initial\_release\_date.arg1}(n, \textsc{Titanic}) \wedge \text{film.initial\_release\_date.arg2}(n, 1997)$$

Grounded Graph

Graph Matching

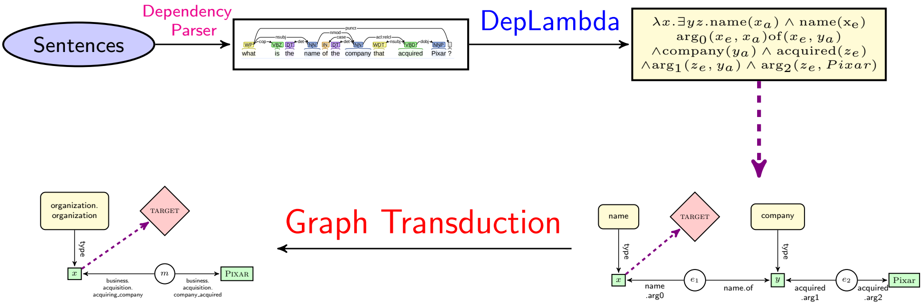


$$\text{directed.arg1}(e, \text{Cameron}) \wedge \\ \text{directed.arg2}(e, \text{Titanic}) \wedge \text{directed.in}(e, 1997)$$

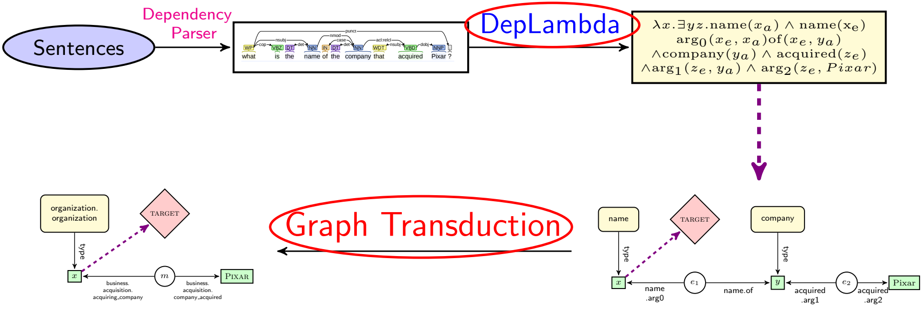


$$\text{film.directed\_by.arg2}(m, \textsc{Cameron}) \wedge \\ \text{film.directed\_by.arg1}(m, \textsc{Titanic}) \wedge \\ \text{film.initial\_release\_date.arg1}(n, \textsc{Titanic}) \wedge \\ \text{film.initial\_release\_date.arg2}(n, 1997)$$

Freebase Semantic Parsing



Freebase Semantic Parsing



Baselines

SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

CCGGRAPH: CCG logical forms

DEPTREE: Directly transduce a dependency tree to target graph

Baselines

SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

CCGGRAPH: CCG logical forms

DEPTREE: Directly transduce a dependency tree to target graph

Baselines

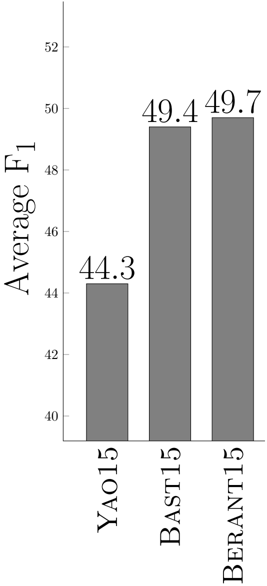
SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

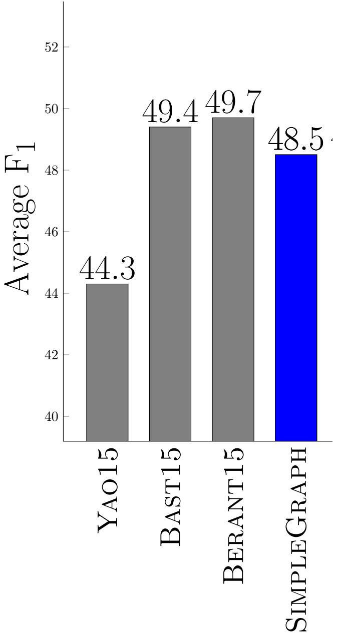
CCGGRAPH: CCG logical forms

DEPTREE: Directly transduce a dependency tree to target graph

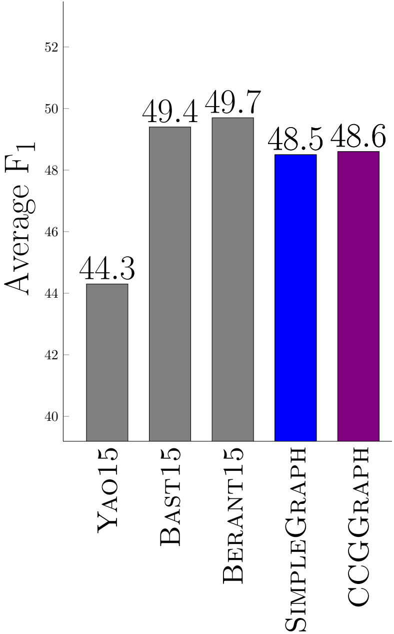
Results on WebQuestions



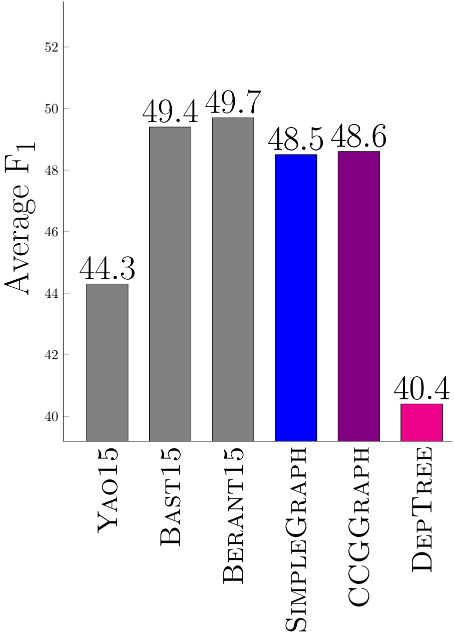
Results on WebQuestions



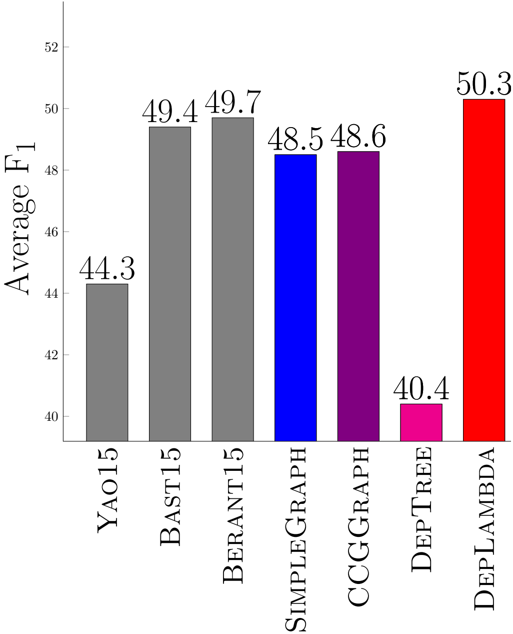
Results on WebQuestions



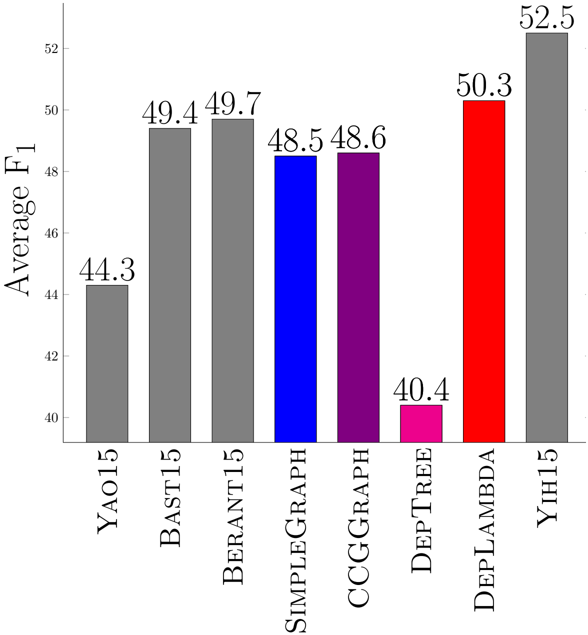
Results on WebQuestions



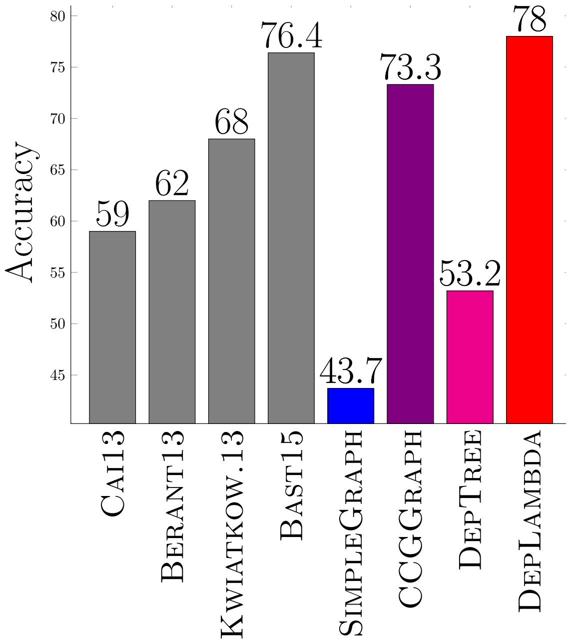
Results on WebQuestions



Results on WebQuestions



Results on Free917



Summary

- Lambda Calculus for converting Dependencies to Logical Forms
- Semantic parsing as Graph Transduction
- State-of-the-art results on Free917 and competitive results on WebQuestions
- Work in progress: DepLambda for multiple languages

Rules available from

https://github.com/sivareddyg/deplambda

Thank You!

Summary

- Lambda Calculus for converting Dependencies to Logical Forms
- Semantic parsing as Graph Transduction
- State-of-the-art results on Free917 and competitive results on WebQuestions
- Work in progress: DepLambda for multiple languages

Rules available from

https://github.com/sivareddyg/deplambda

Thank You!

Summary

- Lambda Calculus for converting Dependencies to Logical Forms
- Semantic parsing as Graph Transduction
- State-of-the-art results on Free917 and competitive results on WebQuestions
- Work in progress: DepLambda for multiple languages

Rules available from
https://github.com/sivareddyg/deplambda
Thank You!

Learning Model

Structured Perceptron: Ranks a pair of grounded and ungrounded graph

$$(\hat{g}, \hat{u}) = \arg\max_{g,u} \Phi(g, u, q, KB) \cdot \theta$$

Features: Φ is defined over sentence, grounded and ungrounded graph

Training: Use a surrogate graph (dynamic oracle) to update weights

$$\theta \leftarrow \theta + \Phi(g^+, u^+, q, KB) - \Phi(\hat{g}, \hat{u}, q, KB)$$

Learning Model

Structured Perceptron: Ranks a pair of grounded and ungrounded graph

$$(\hat{g}, \hat{u}) = \arg\max_{g,u} \Phi(g, u, q, KB) \cdot \theta$$

Features: Φ is defined over sentence, grounded and ungrounded graph

Training: Use a surrogate graph (dynamic oracle) to update weights

$$\theta \leftarrow \theta + \Phi(g^{+}, u^{+}, q, KB) - \Phi(\hat{g}, \hat{u}, q, KB)$$

Learning Model

Structured Perceptron: Ranks a pair of grounded and ungrounded graph

$$(\hat{g}, \hat{u}) = \arg\max_{g,u} \Phi(g, u, q, KB) \cdot \theta$$

Features: Φ is defined over sentence, grounded and ungrounded graph

Training: Use a surrogate graph (dynamic oracle) to update weights

$$\theta \leftarrow \theta + \Phi(g^+, u^+, q, KB) - \Phi(\hat{g}, \hat{u}, q, KB)$$

Learning Model

Oracle Graphs: Search for all the grounded graphs reachable via the ungrounded graph.

Pick all the graphs with minimal F_1-loss against the gold answer.

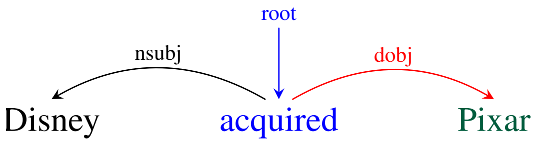
Surrogate Gold Graph:

$$(u^+, g^+) = \underset{(u,g) \in O_{KB,A}(q)}{\arg\max} \; \theta^t \cdot \Phi(u, g, q, KB),$$

Beam Search: Limit the predictions to 100 graph pairs, and choose the best prediction for update.

Dependencies to Logical Forms

Single Type System

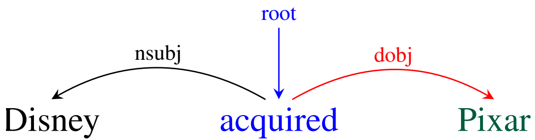


All constituents are of the same lambda expression type

TYPE[acquired] = **TYPE**[Pixar] = **TYPE**[(dobj acquired Pixar)]

Dependencies to Logical Forms

Single Type System

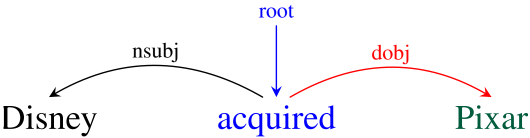


All **words** have a *lambda expression* of type η

- **TYPE**[acquired] = η
- **TYPE**[Pixar] = η
- **TYPE**[(dobj acquired Pixar)] = η

Dependencies to Logical Forms

Single Type System

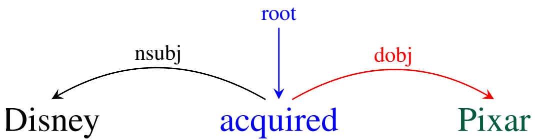


All **constituents** have a *lambda expression* of type η

- **TYPE**[acquired] = η
- **TYPE**[Pixar] = η
- **TYPE**[(dobj acquired Pixar)] = η

Dependencies to Logical Forms

Single Type System



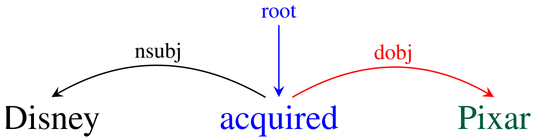
All constituents have a *lambda expression* of type η

- **TYPE**[acquired] = η
- **TYPE**[Pixar] = η
- **TYPE**[(dobj acquired Pixar)] = η

$\Longrightarrow$ **TYPE**[dobj] = $\eta \rightarrow \eta \rightarrow \eta$

Dependencies to Logical Forms

Lambda Calculus for Single Type System



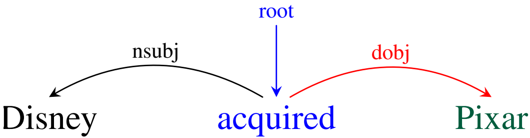
Lambda Expression for words

$$\text{acquired} \Rightarrow \lambda x_e.\,\mathsf{acquired}(x_e)$$

$$\text{Pixar} \Rightarrow \lambda x_a.\,\mathsf{Pixar}(x_a)$$

Dependencies to Logical Forms

Lambda Calculus for Single Type System



Lambda Expression for words

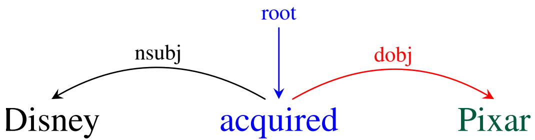
$$\text{acquired} \Rightarrow \lambda x_e.\,\text{acquired}(x_e) \qquad \Rightarrow \textsf{TYPE} = \textbf{Event} \rightarrow \textbf{Bool}$$

$$\text{Pixar} \Rightarrow \lambda x_a.\,\text{Pixar}(x_a) \qquad \Rightarrow \textsf{TYPE} = \textbf{Ind} \rightarrow \textbf{Bool}$$

Here **TYPE**[acquired] $\neq$ **TYPE**[Pixar] ✗

Dependencies to Logical Forms

Lambda Calculus for Single Type System



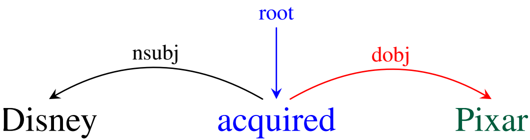
Lambda Expression for words

$$\text{acquired} \Rightarrow \lambda \mathbf{x_a} x_e . \text{acquired}(x_e)$$

$$\text{Pixar} \Rightarrow \lambda x_a \mathbf{x_e} . \text{Pixar}(x_a)$$

Dependencies to Logical Forms

Lambda Calculus for Single Type System



Lambda Expression for words

$$\text{acquired} \Rightarrow \lambda \mathbf{x_a} x_e . \text{acquired}(x_e) \quad \Rightarrow \textbf{TYPE} = \mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Bool}$$

$$\text{Pixar} \Rightarrow \lambda x_a \mathbf{x_e} . \text{Pixar}(x_a) \quad \Rightarrow \textbf{TYPE} = \mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Bool}$$

Here η = **TYPE**[acquired] = **TYPE**[Pixar] ✓

Lambda calculus for “*Single Type*” System

More dependency labels

- (appos Disney the_company)

$$\textit{appos} = \lambda fgx.f(x) \wedge g(x)$$

This function unifies two nodes

- (partmod a_company acquired_by_Disney)

$$\textit{partmod} = \lambda fgx.\exists z.f(x) \wedge g(z) \wedge \mathrm{arg}_1(z_e, x_a)$$

This function reverses the dependency arc direction, but still returns the head

Lambda calculus for "*Single Type*" System

More dependency labels

- (appos Disney the_company)

$$\textit{appos} = \lambda fgx.f(x) \wedge g(x)$$

This function unifies two nodes

- (partmod a_company acquired_by_Disney)

$$\textit{partmod} = \lambda fgx.\exists z.f(x) \wedge g(z) \wedge \mathrm{arg}_1(z_e, x_a)$$

This function reverses the dependency arc direction, but still returns the head

Lambda calculus for "*Single Type*" System

More dependency labels

- (conj Disney_and Pixar)

$$\mathit{conj} = \lambda fgz.\, \exists xy.\, f(x) \wedge g(y) \wedge \textbf{coord}(z, x, y)$$

This function creates a struct with two variables

- (rcmod Disney which_acquired_Pixar)

(rcmod Disney
(wh-dobj (BIND f (nsubj (dobj acquired f) Pixar))
which))

Lambda calculus for “*Single Type*” System

More dependency labels

- (conj Disney_and Pixar)

$$\textit{conj} = \lambda fgz.\exists xy.f(x) \wedge g(y) \wedge \textbf{coord}(z,x,y)$$

This function creates a struct with two variables

- (rcmod Disney which_acquired_Pixar)

$$\begin{array}{l}(\text{rcmod Disney} \\ \qquad (\text{wh-dobj } (\textsf{BIND}\, f \ (\text{nsubj } (\text{dobj acquired}\, f) \text{ Pixar})) \\ \qquad\qquad \text{which}))\end{array}$$

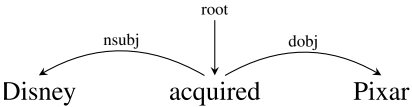
CCG to Logical Forms

Steedman, 2000, 2012; Bos et al., 2004; Lewis & Steedman, 2013; Reddy et al., 2014

$$
\begin{array}{ccc}
\text{Disney} & \text{acquired} & \text{Pixar} \\
\hline
NP & S\backslash NP/NP & NP \\
\text{Disney} & \begin{array}{c}\lambda y \lambda x \lambda e.\ \text{acquired}(e) \\ \wedge\ \text{arg}_1(e, x) \\ \wedge\ \text{arg}_2(e, y)\end{array} & \text{Pixar} \\
\cline{2-3}
 & \multicolumn{2}{c}{S\backslash NP \quad >} \\
 & \multicolumn{2}{c}{\begin{array}{c}\lambda x \lambda e.\ \text{acquired}(e) \\ \wedge\ \text{arg}_1(e, x) \wedge\ \text{arg}_2(e, \text{Pixar})\end{array}} \\
\hline
\multicolumn{3}{c}{S \quad <} \\
\multicolumn{3}{c}{\lambda e.\ \text{acquired}(e) \wedge \text{arg}_1(e, \text{Disney}) \wedge \text{arg}_2(e, \text{Pixar})}
\end{array}
$$

Dependencies to Logical Forms

Challenges

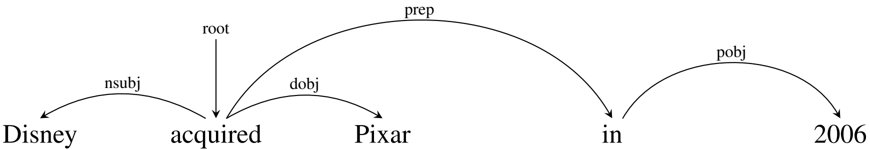


The obvious idea

if *proper noun* **then**
  assign $\lambda x.\mathrm{word}(x)$
else if *verb* with *subject* and *object* **then**
  assign $\lambda fge.\exists xy.f(x) \wedge g(y) \wedge \mathrm{word}(e) \wedge \mathrm{arg}_1(e,x) \wedge \mathrm{arg}_2(e,y)$
end if

Dependencies to Logical Forms

Challenges



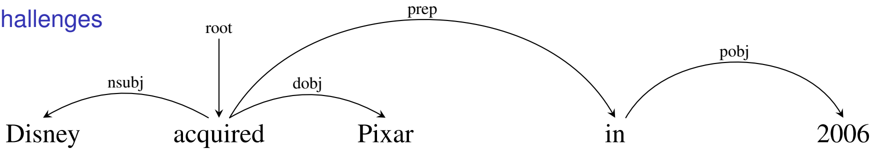
The obvious idea

if *proper noun* **then**
    assign $\lambda x.\mathrm{word}(x)$
else if *verb* with *subject* and *object* **then**
    assign $\lambda fge.\exists xy.f(x) \wedge g(y) \wedge \mathrm{word}(e) \wedge \mathrm{arg}_1(e,x) \wedge \mathrm{arg}_2(e,y)$
end if

But, what about?

Dependencies to Logical Forms

Challenges



Problems

1. Rules $\propto$ dependency label permutations
2. Complex lexical semantics
3. Highly sensitive to parse errors
4. Prone to type collisions

Comparison with CCG

Handling of control verbs is painful.

Sentence:
John persuaded Jim to acquire Pixar.

Binarized Tree:
(nsubj (xcomp (dobj persuaded Jim) to_acquire_Pixar) John)

Elegant handling in CCG
persuaded: $((S[dcl]\backslash NP)/(S[to]\backslash NP_x))/NP_x$

Comparison with CCG

Handling of control verbs is painful.

Sentence:
John persuaded Jim to acquire Pixar.

Binarized Tree:
(nsubj (xcomp (dobj persuaded Jim) to_acquire_Pixar) John)

Elegant handling in CCG
persuaded: $((S[dcl]\backslash NP)/(S[to]\backslash NP_x))/NP_x$

Conjunctions

Sentence:

Eminem signed to Interscope and discovered 50 Cent.

Binarized tree:

(nsubj (conj-vp (cc s_to_I and) d_50) Eminem)

Substitution:

conj-vp $\Rightarrow \lambda fgx.\exists yz.f(y) \wedge g(z) \wedge \mathrm{coord}(x,y,z)$

Logical Expression:

$$\lambda w.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{coord}(w,y,z) \\ \wedge \mathrm{arg}_1(w_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Post processing:

$$\lambda e.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{arg}_1(y_e,x_a) \\ \wedge \mathrm{arg}_1(z_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Conjunctions

Sentence:

Eminem signed to Interscope and discovered 50 Cent.

Binarized tree:

(nsubj (conj-vp (cc s_to_I and) d_50) Eminem)

Substitution:

conj-vp $\Rightarrow \lambda fgx.\exists yz.f(y) \wedge g(z) \wedge \mathrm{coord}(x,y,z)$

Logical Expression:

$$\lambda w.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{coord}(w,y,z) \\ \wedge \mathrm{arg}_1(w_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Post processing:

$$\lambda e.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{arg}_1(y_e,x_a) \\ \wedge \mathrm{arg}_1(z_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Conjunctions

Sentence:

Eminem signed to Interscope and discovered 50 Cent.

Binarized tree:

(nsubj (conj-vp (cc s_to_I and) d_50) Eminem)

Substitution:

conj-vp $\Rightarrow \lambda fgx.\exists yz.f(y) \wedge g(z) \wedge \text{coord}(x,y,z)$

Logical Expression:

$$\lambda w.\exists xyz.\text{Eminem}(x_a) \wedge \text{coord}(w,y,z) \\ \wedge \text{arg}_1(w_e,x_a) \wedge \text{s\_to\_I}(y) \wedge \text{d\_50}(z)$$

Post processing:

$$\lambda e.\exists xyz.\text{Eminem}(x_a) \wedge \text{arg}_1(y_e,x_a) \\ \wedge \text{arg}_1(z_e,x_a) \wedge \text{s\_to\_I}(y) \wedge \text{d\_50}(z)$$

Relative Clause

following Moortgat (1988); Pereira (1990); Carpenter (1998)

Sentence:

Apple which Jobs founded

Binarized tree:

$$
\begin{aligned}
&(\text{rcmod Apple} \\
&\qquad (\text{wh-dobj } (\text{BIND } f \ (\text{nsubj } (\text{dobj founded } f) \text{ Jobs})) \\
&\qquad\qquad \text{which}))
\end{aligned}
$$

Substitution:

wh-dobj $\Rightarrow \lambda fgz.f(z)$

rcmod $\Rightarrow \lambda fgz.f(z) \wedge g(z)$

Logical Expression:

$$
\begin{aligned}
&\lambda u.\, \exists xy.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \\
&\qquad \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, u_a) \wedge \text{Apple}(u_a)
\end{aligned}
$$

Relative Clause

following Moortgat (1988); Pereira (1990); Carpenter (1998)

Sentence:

Apple which Jobs founded

Binarized tree:

$$(\text{rcmod Apple} \\ \quad (\text{wh-dobj } (\textsf{BIND}\, f \; (\text{nsubj } (\text{dobj founded } f) \text{ Jobs})) \\ \qquad \text{which}))$$

Substitution:

wh-dobj $\Rightarrow \lambda fgz.f(z)$
rcmod $\Rightarrow \lambda fgz.f(z) \wedge g(z)$

Logical Expression:

$$\lambda u.\exists xy.\,\text{founded}(x_e) \wedge \text{Jobs}(y_a) \\ \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, u_a) \wedge \text{Apple}(u_a)$$

Expressivity

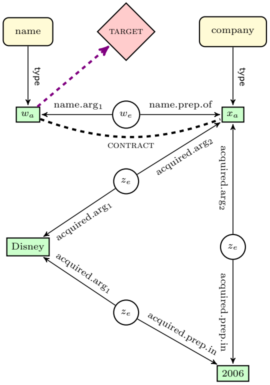
How isomorphic are the representations compared to Knowledge Graph?
Average Oracle F_1 Table.

Search Space

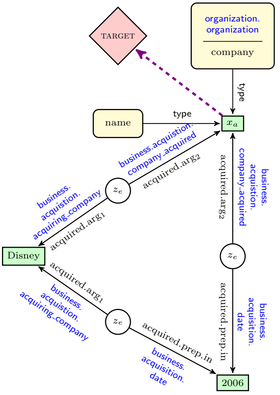
How many ways to reach an answer?
Average Oracle F_1 Table.

Graph Transformation: CONTRACT operation

What is the name of the company which Disney acquired in 2006?



Ungrounded graph



Grounded graph

Graph Mismatch: EXPAND operation

What to do Washington DC December?

Before EXPAND

- $\lambda z. \exists xyw. \text{TARGET}(x_a) \wedge \text{do}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{Washington\_DC}(y_a) \wedge \text{December}(w_a)$

After EXPAND

- $\lambda z. \exists xyw. \text{TARGET}(x_a) \wedge \text{do}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{Washington\_DC}(y_a) \wedge \text{dep}(z_e, y_a) \wedge \text{December}(w_a) \wedge \text{dep}(z_e, w_a)$