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

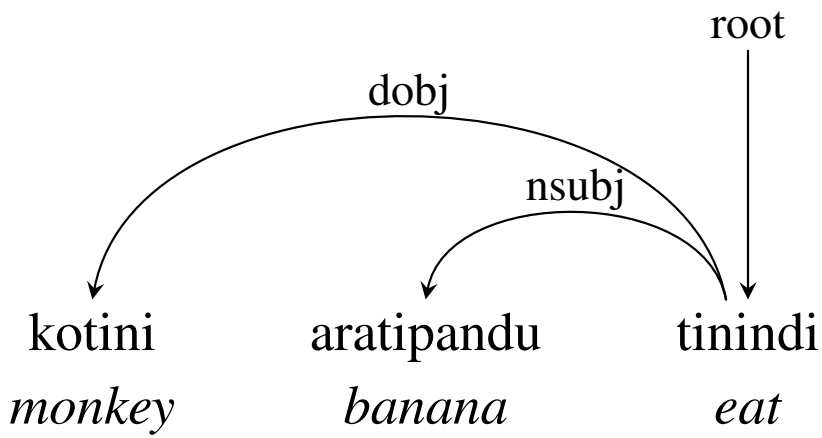
Syntax helps Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

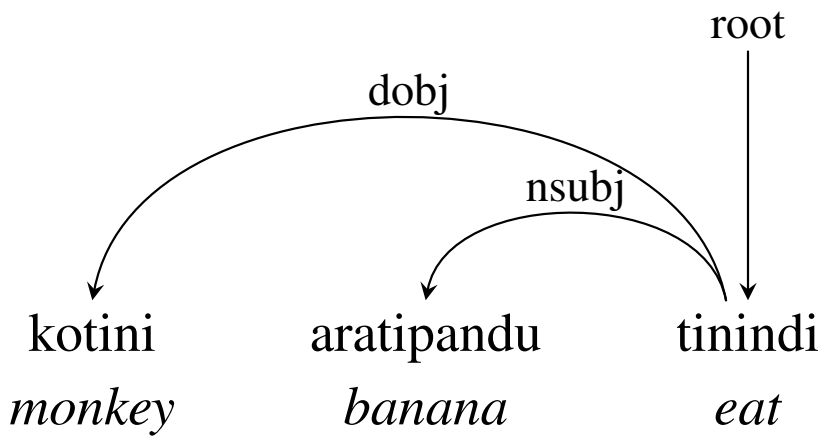
Syntax helps Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Syntax helps Semantics



Syntax helps Semantics



Syntax in humans?

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

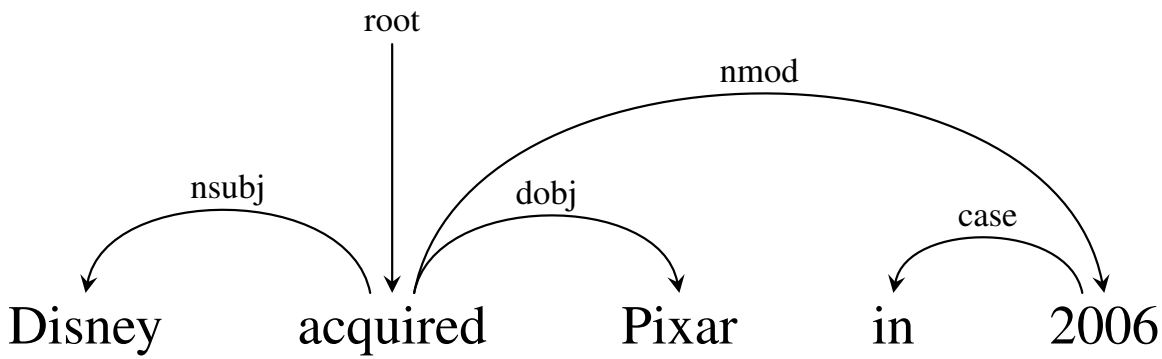
Syntax in humans?

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

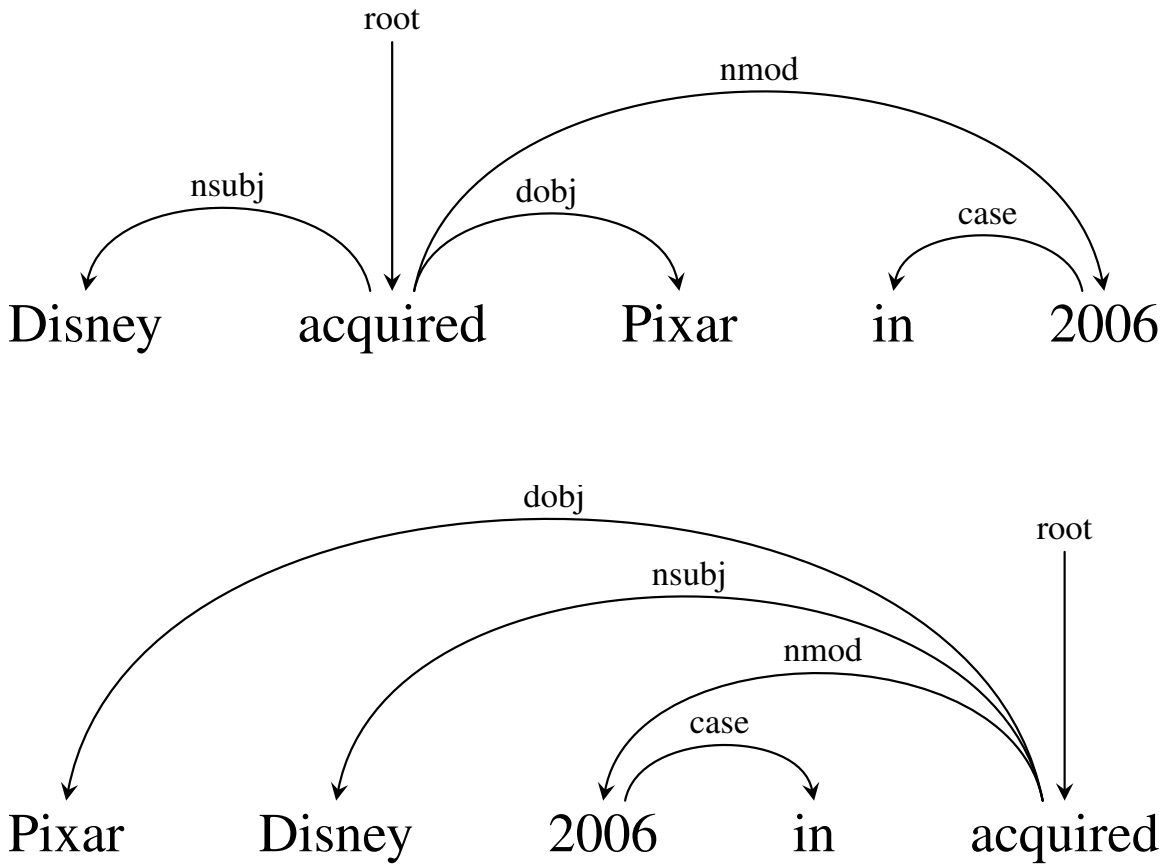
Homogeneous syntactic representation across languages



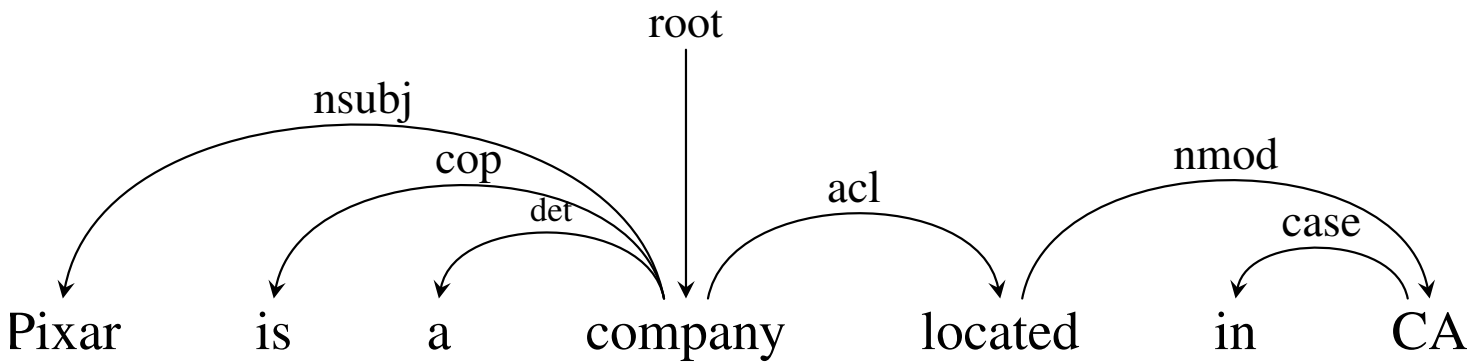
Pixar Disney 2006 in acquired

Universal Dependencies

Homogeneous syntactic representation across languages

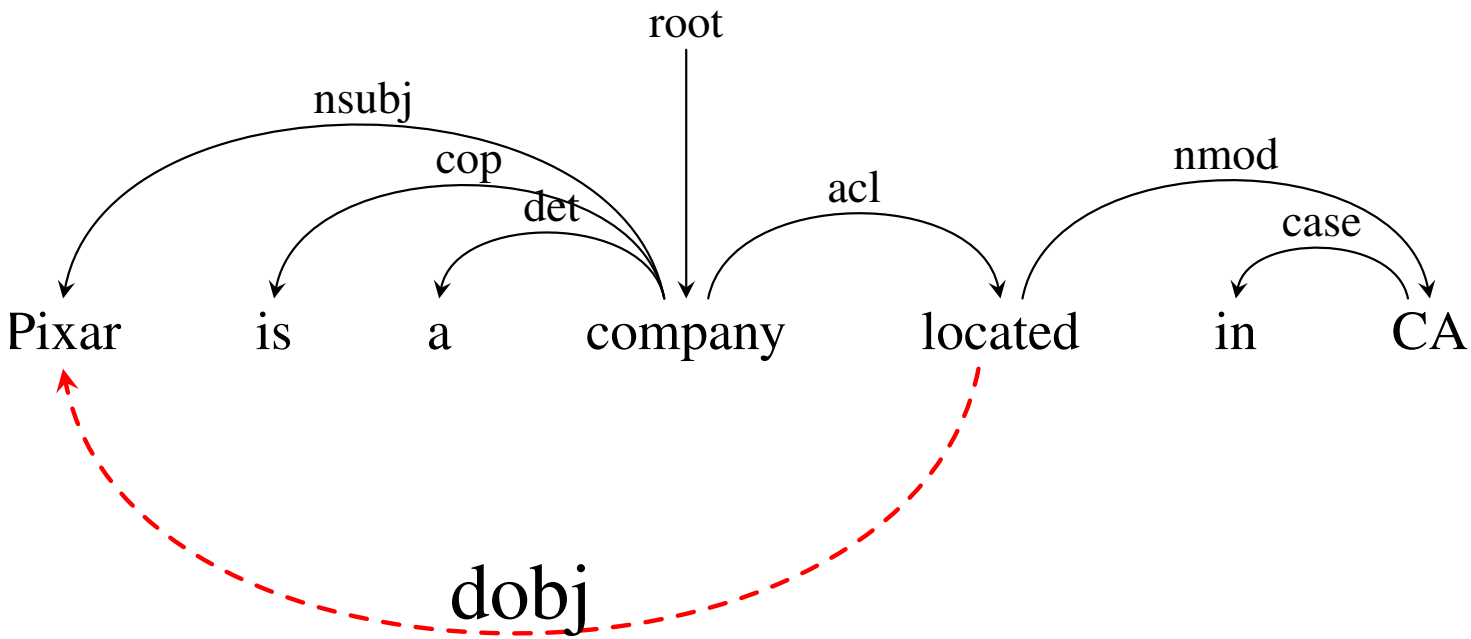


Dependencies to Logical Forms



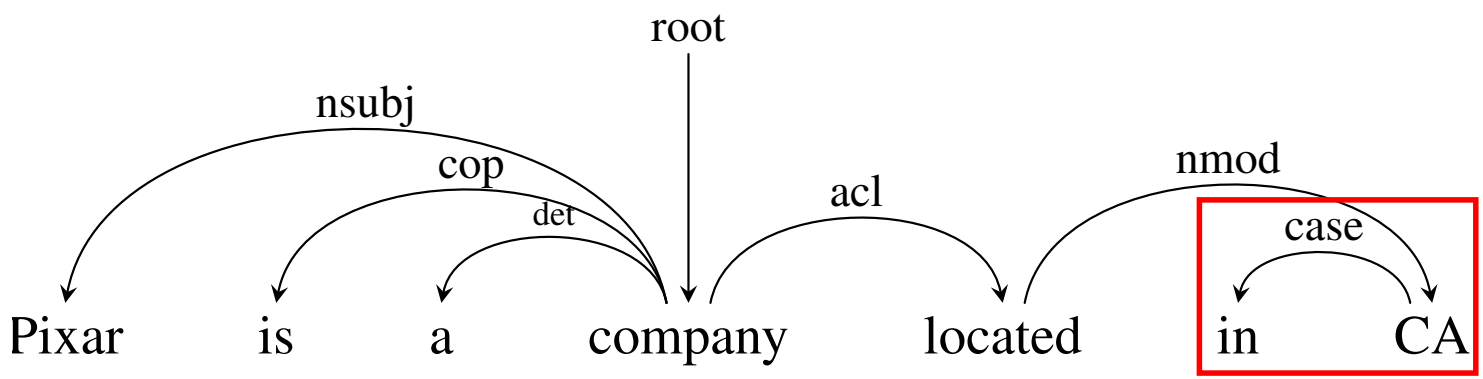
$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \text{arg}_2(z_e, \text{Pixar}) \wedge \text{arg}_{\text{in}}(z_e, \text{CA})$$

Dependencies to Logical Forms



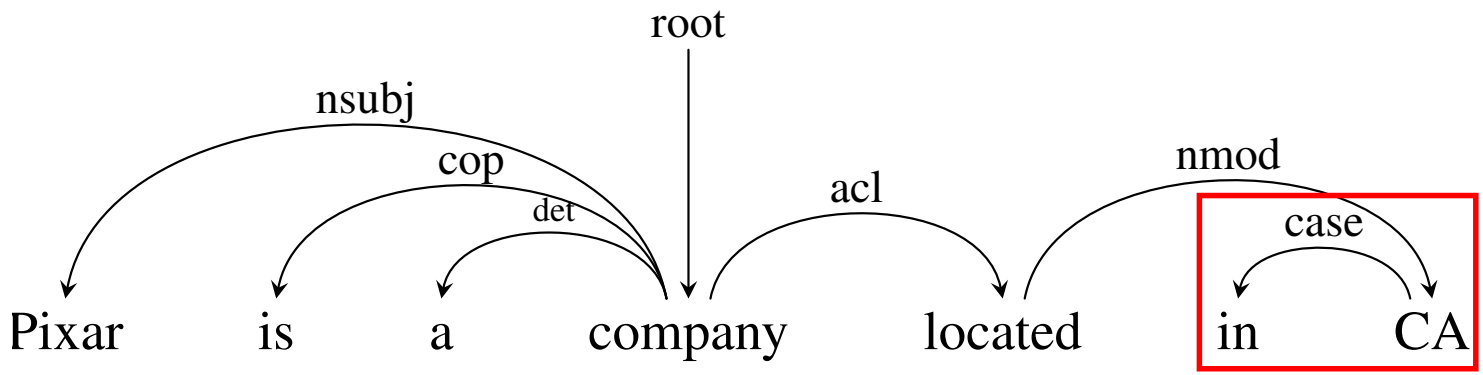
$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \text{arg}_2(z_e, \text{Pixar}) \wedge \text{arg}_{\text{in}}(z_e, \text{CA})$$

Dependencies to Logical Forms



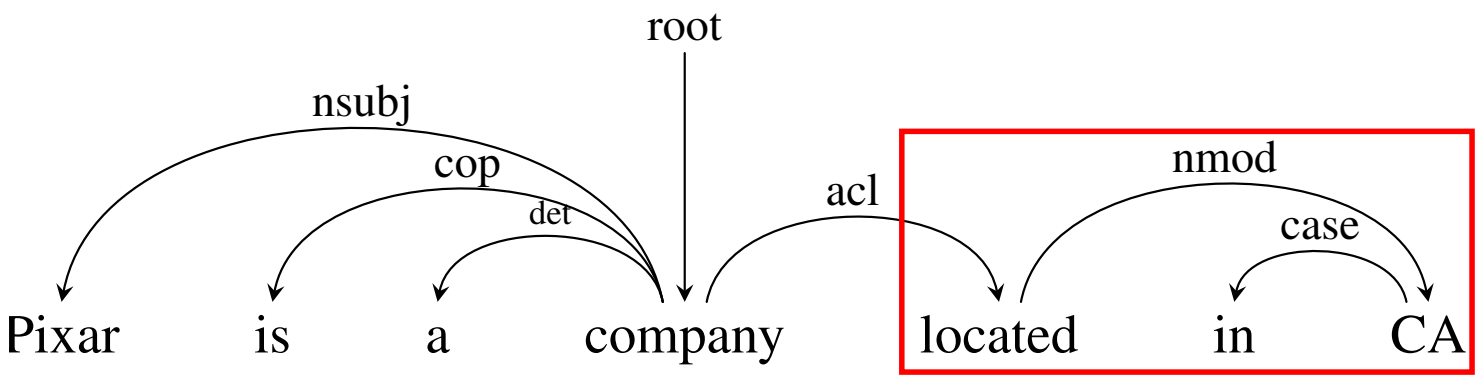
$(case \quad \text{CA} \quad \text{in})$

Dependencies to Logical Forms



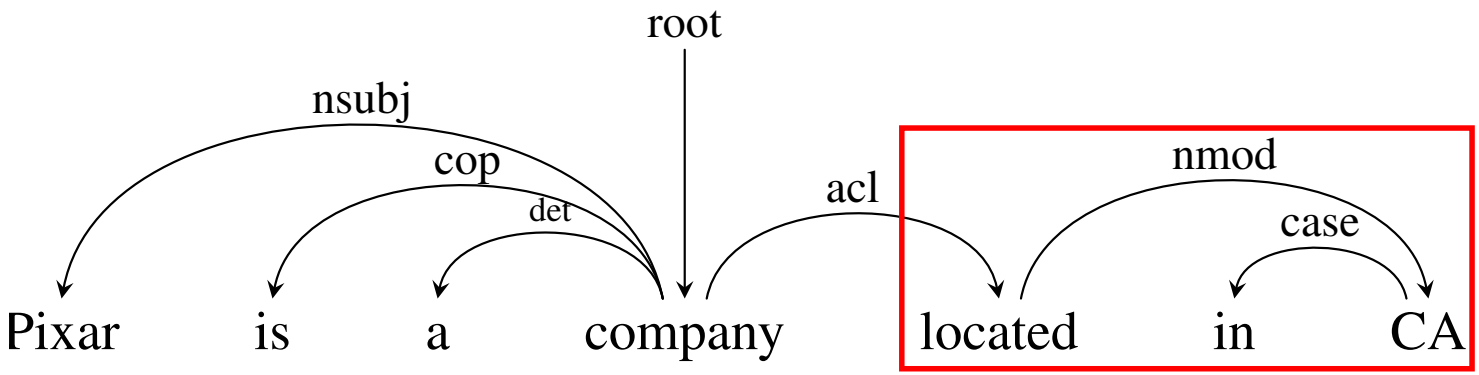
$$\frac{\begin{array}{cccc} & (case & \text{CA} & \text{in}) \\ {}_e) & \lambda f g x.f(x) & \lambda x.\text{CA}(x_a) & \lambda x.\text{empty}(x \end{array}}{\lambda x.\ \text{CA}(x_a)}$$

Dependencies to Logical Forms



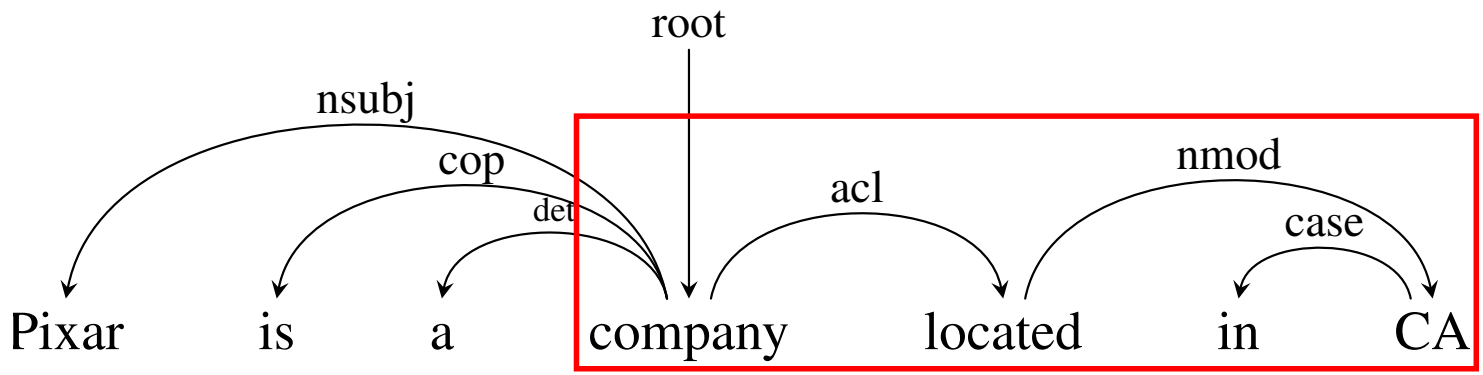
$(nmod \quad \text{located} \quad \text{in CA}) \quad).$

Dependencies to Logical Forms



$$\dfrac{\dfrac{(nmod \quad \lambda f g z.\exists x. \quad f(z) \wedge g(x) \wedge \wedge \mathrm{arg_{in}}(z_e, x_a) \qquad \text{located} \quad \lambda x.\mathrm{located}(x_e) \qquad \dfrac{\text{in CA})}{\lambda x.\ \mathrm{CA}(x_a)}}{}}{\lambda z.\mathrm{located}(z_e) \wedge \mathrm{CA}(x_a) \wedge \mathrm{arg_{in}}(z_e, x_a)}$$

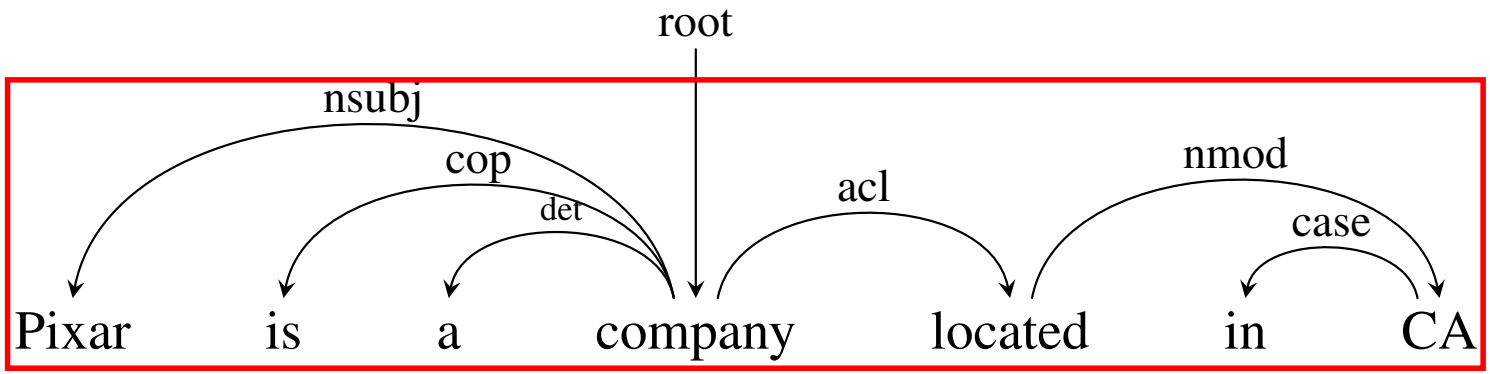
Dependencies to Logical Forms



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

$$f(x) \wedge g(x) \wedge \text{arg}_2(z_e, x_a)$$

Dependencies to Logical Forms



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

Reddy, Täckström, Kwiatkowski et al. (TACL 2016)

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?

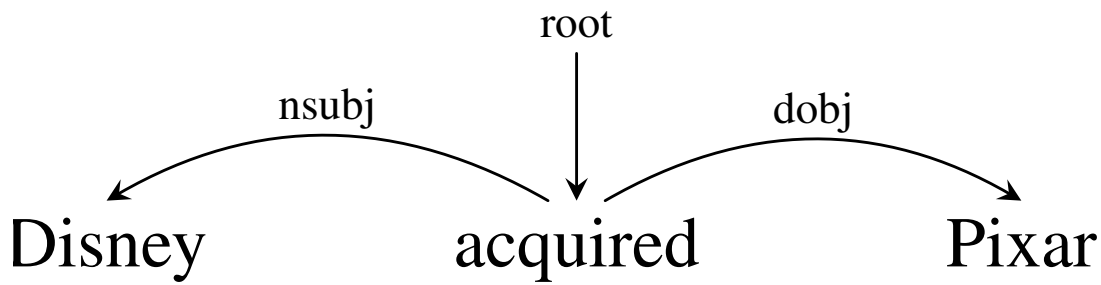
Treebanks in more than 40 languages

Very accurate parsers

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

Universal Types, both syntactic and semantic

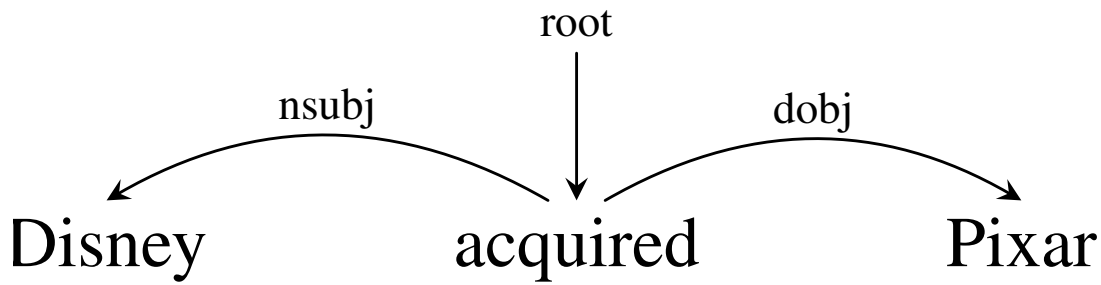
Dependencies to Logical Forms



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

Dependencies to Logical Forms

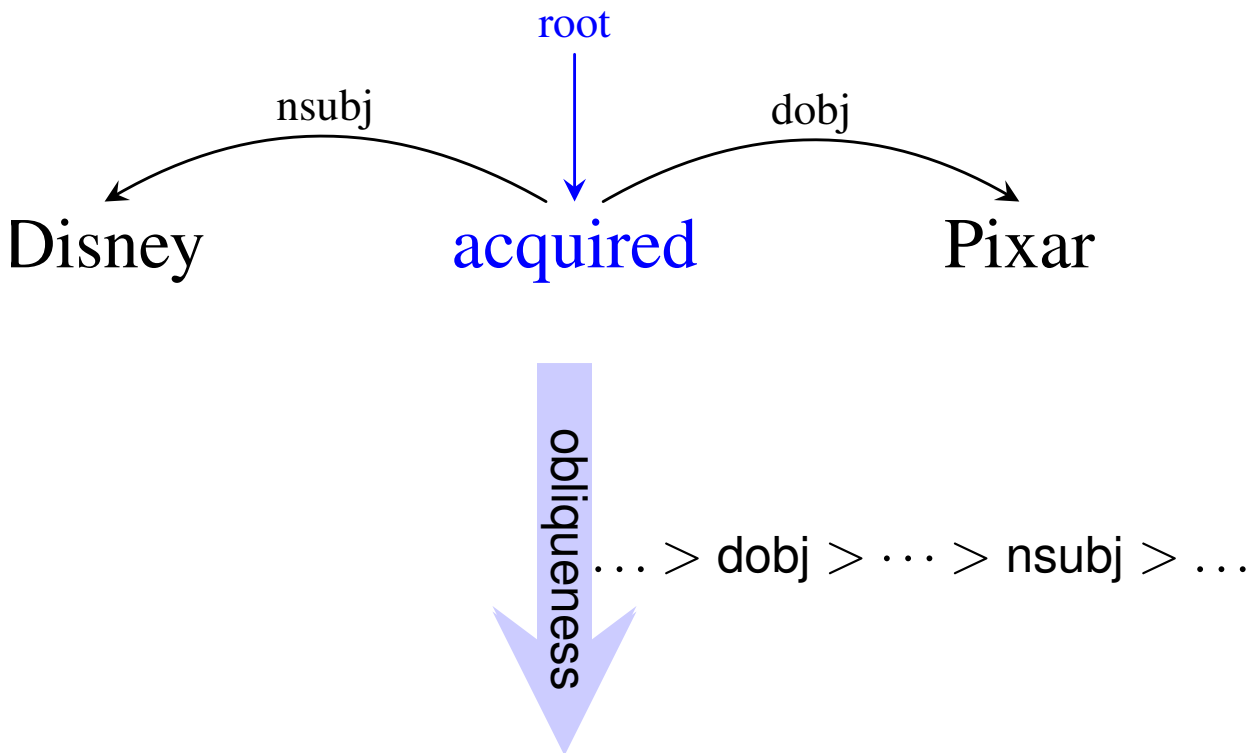
Our Approach



Let dependency labels drive the composition

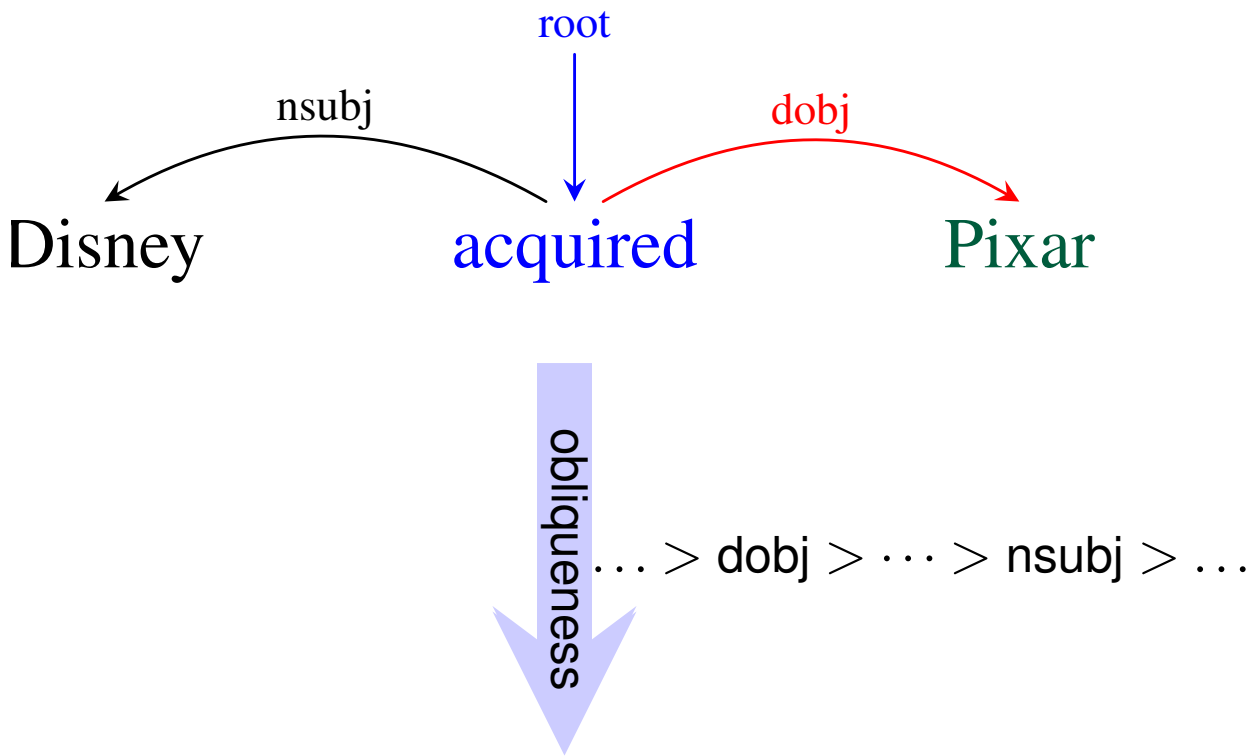
Dependencies to Logical Forms

Our Approach



Dependencies to Logical Forms

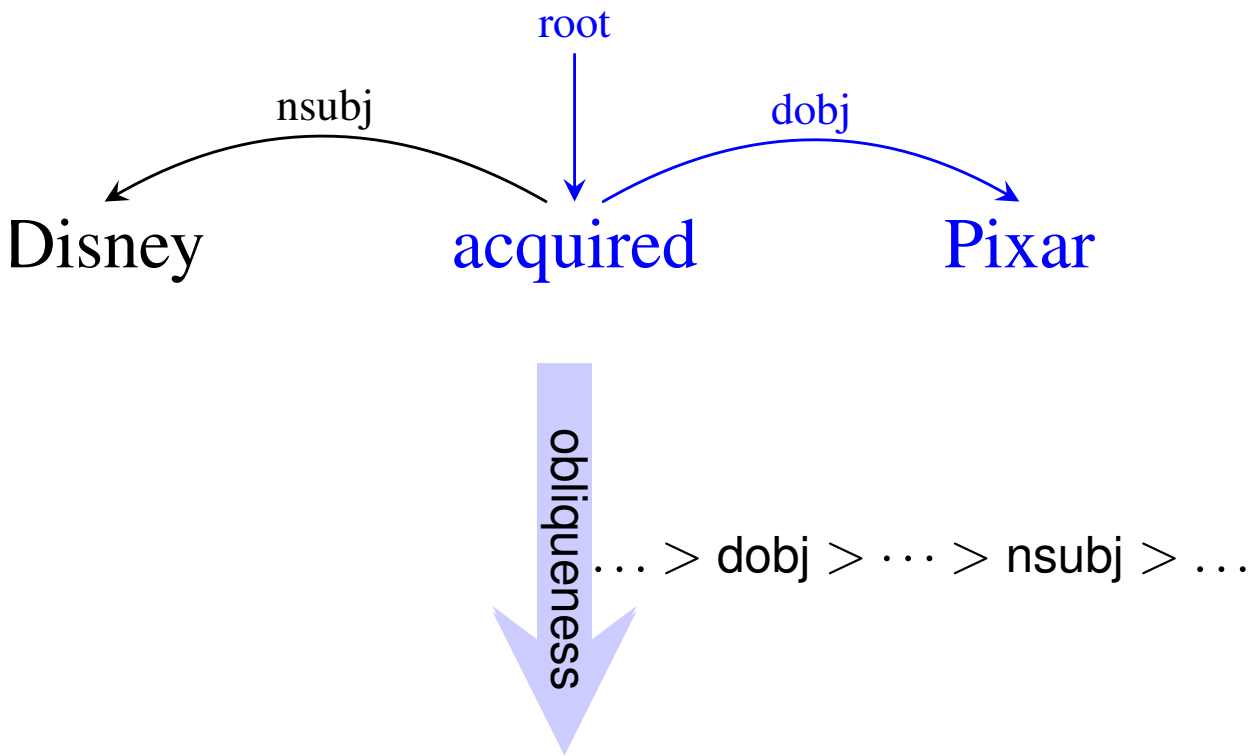
Our Approach



(dobj acquired Pixar)

Dependencies to Logical Forms

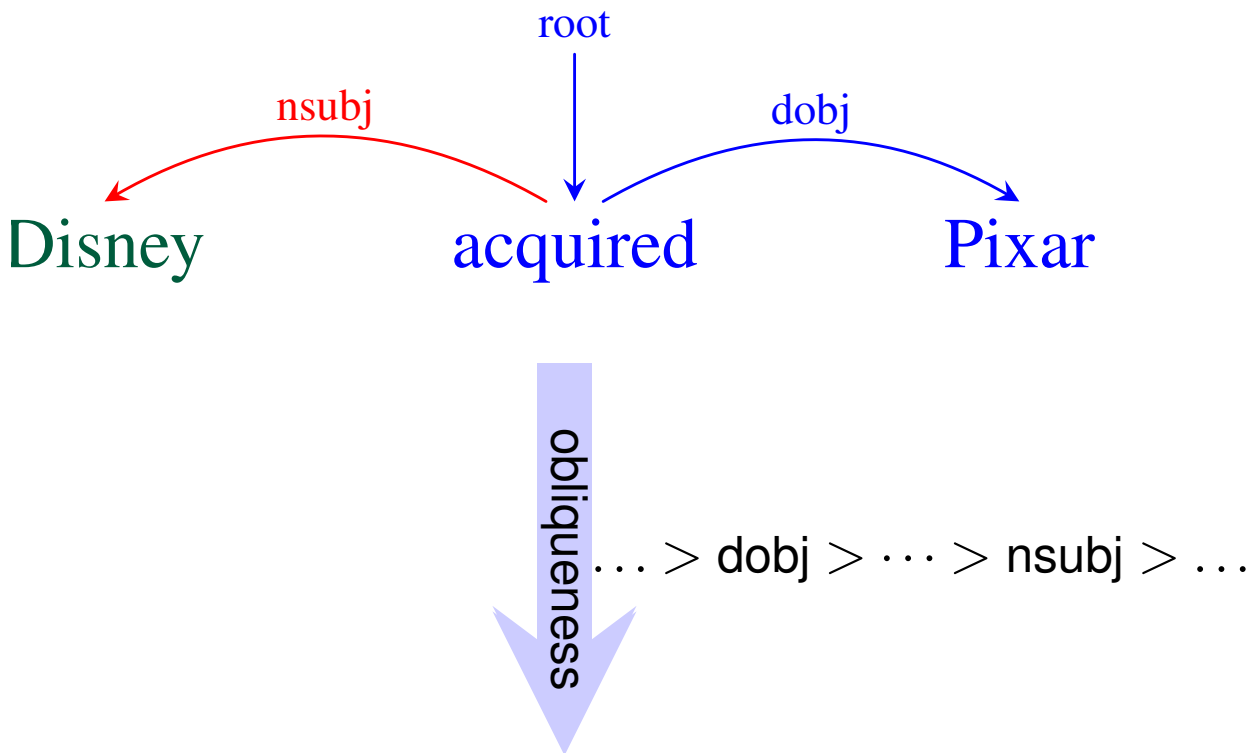
Our Approach



(dobj acquired Pixar)

Dependencies to Logical Forms

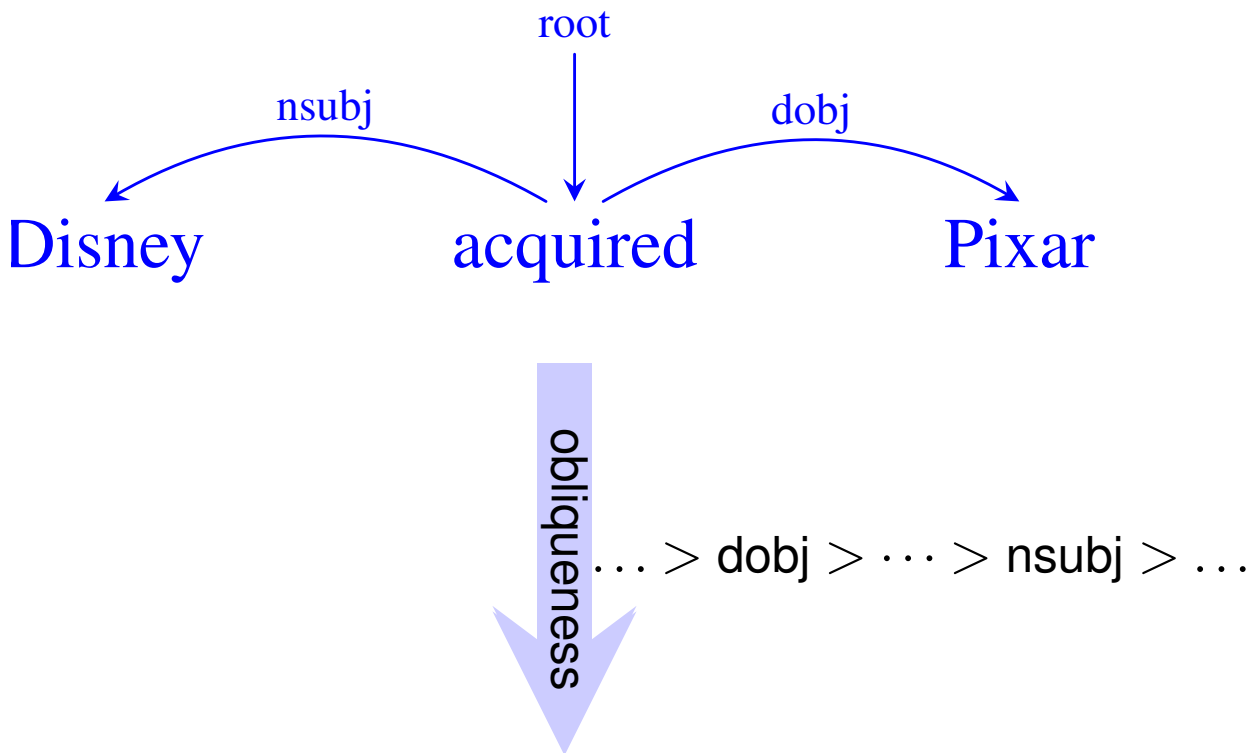
Our Approach



(nsubj (dobj acquired Pixar) Disney)

Dependencies to Logical Forms

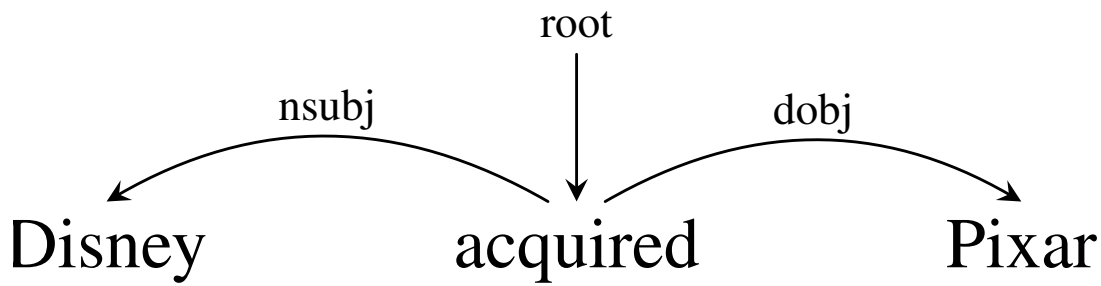
Our Approach



(nsubj (dobj acquired Pixar) Disney)

Dependencies to Logical Forms

Our Approach

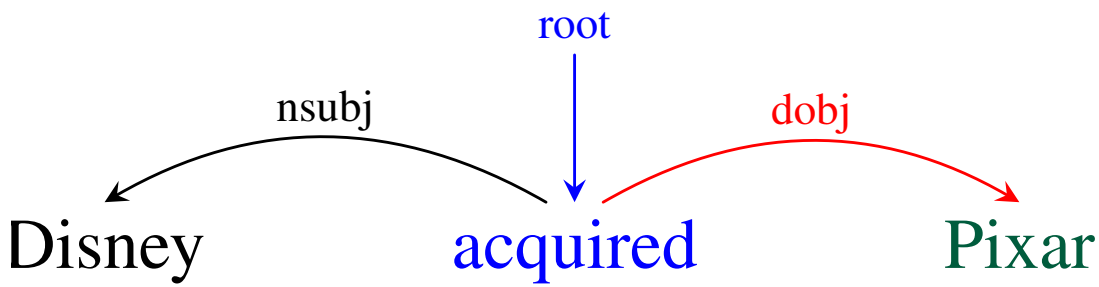


(nsubj (dobj acquired Pixar) Disney)

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

Dependencies to Logical Forms

Lambda Calculus

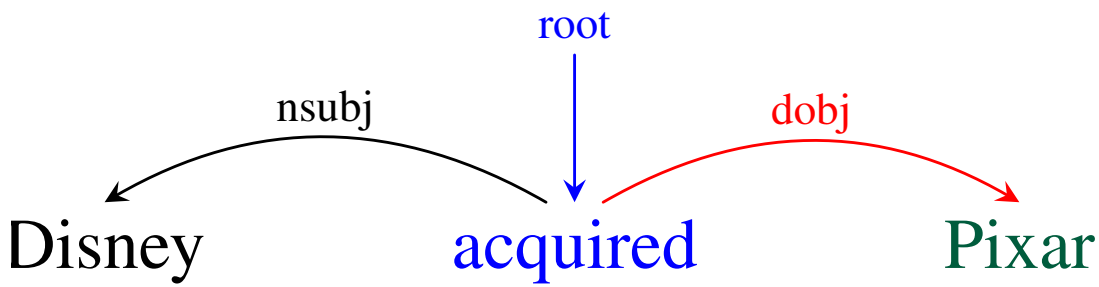


Lambda Calculus Basic Types

- Individuals: **Ind** (also denoted by $._a$)
- Events: **Event** (also denoted by $._e$)
- Truth values: **Bool**

Dependencies to Logical Forms

Lambda Calculus



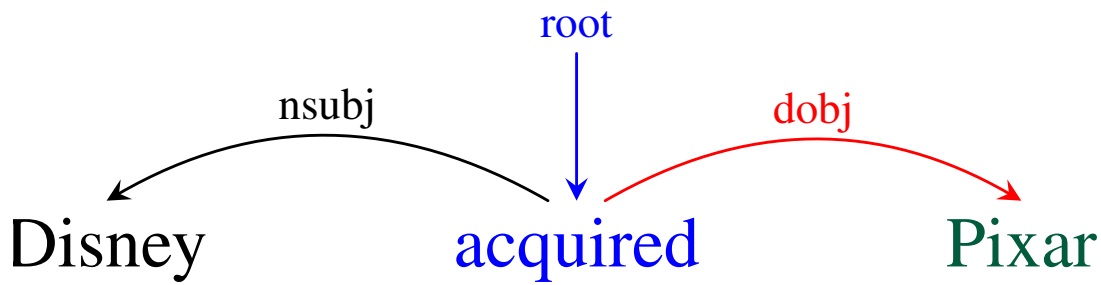
Lambda Expression for words

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

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

Dependencies to Logical Forms

Lambda Calculus

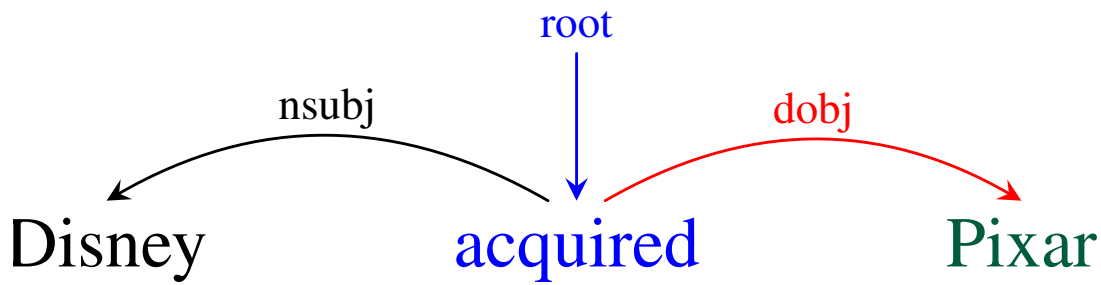


Lambda Expression for dependency labels

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

Dependencies to Logical Forms

Lambda Calculus



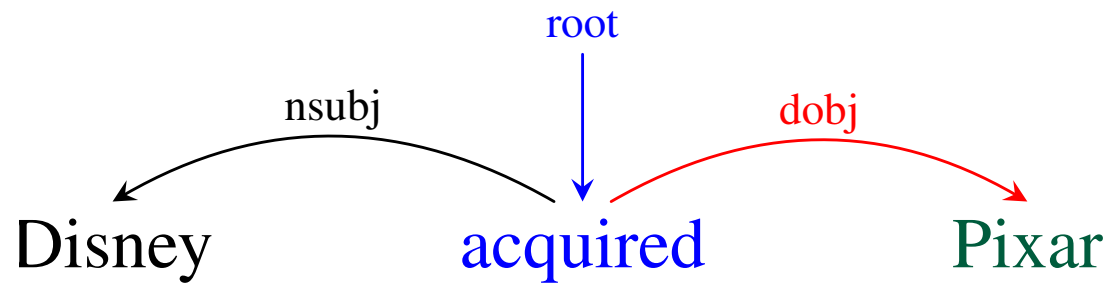
Lambda Expression for dependency labels

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

This operation mirrors the tree structure

Dependencies to Logical Forms

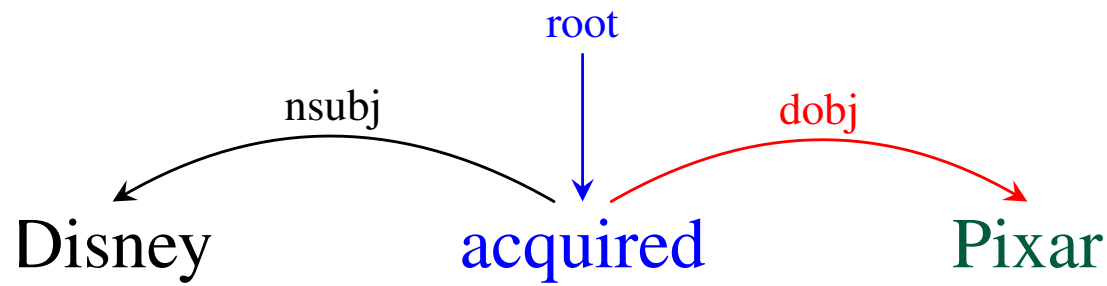
Composition



(**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



$$(\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar})$$

$$\lambda f g z.\exists y. \quad \lambda z.\text{acquired}(z_e) \quad \lambda y.\text{Pixar}(y_a)$$

$$f(z) \wedge g(y) \wedge$$

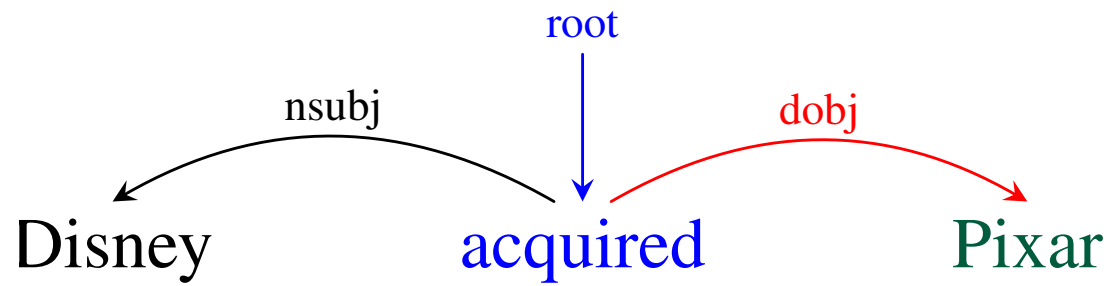
$$\text{arg}_2(z_e, y_a)$$

$$\lambda g z.\ \exists y.\ \text{acquired}(z_e) \wedge g(y)$$

$$\wedge\ \text{arg}_2(z_e, y_a)$$

Dependencies to Logical Forms

Composition



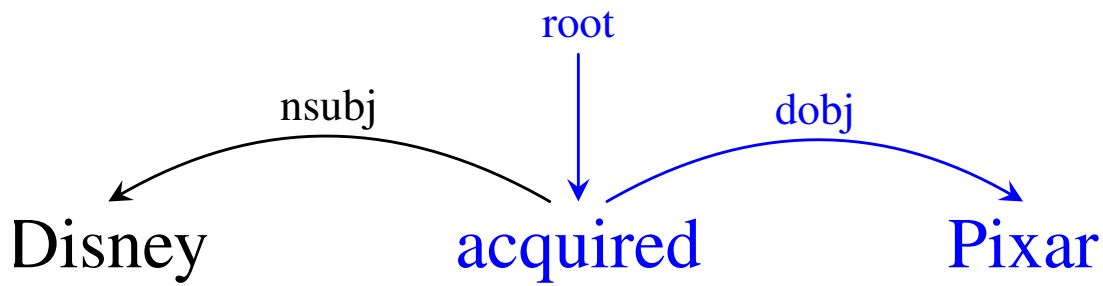
(**dobj**	**acquired**	**Pixar**)
$\lambda f g z.\exists y.$ $f(z) \wedge g(y) \wedge$ $\arg_2(z_e, y_a)$	$\lambda z.\text{acquired}(z_e)$	$\lambda y.\text{Pixar}(y_a)$

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

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

Dependencies to Logical Forms

Composition

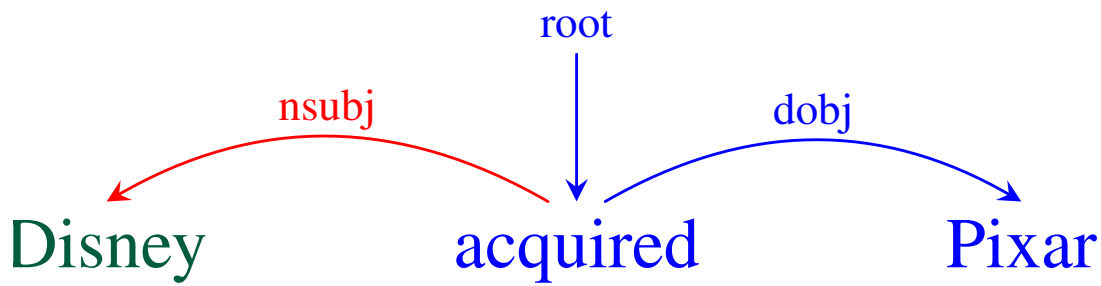


$$(\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar})$$

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

Dependencies to Logical Forms

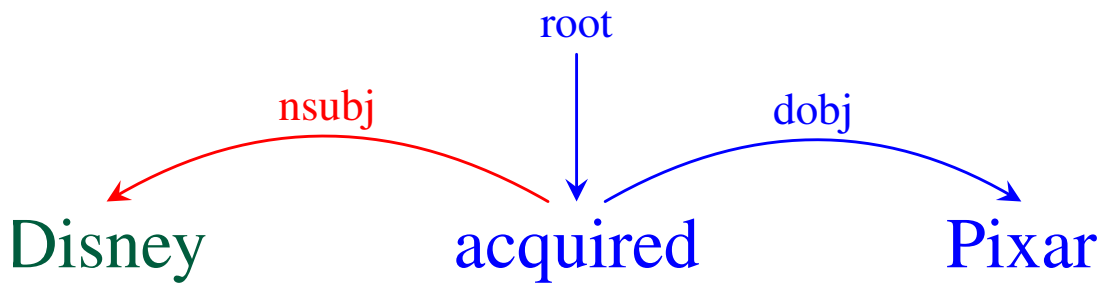
Composition



$$\frac{(\textbf{nsubj} \quad \lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \arg_1(z_e, x_a)) \qquad (\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar})}{\lambda z.\ \exists y.\ \text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \arg_2(z_e, y_a)} \qquad \textbf{Disney}) \quad \lambda x.\text{Disney}(x_a)$$

Dependencies to Logical Forms

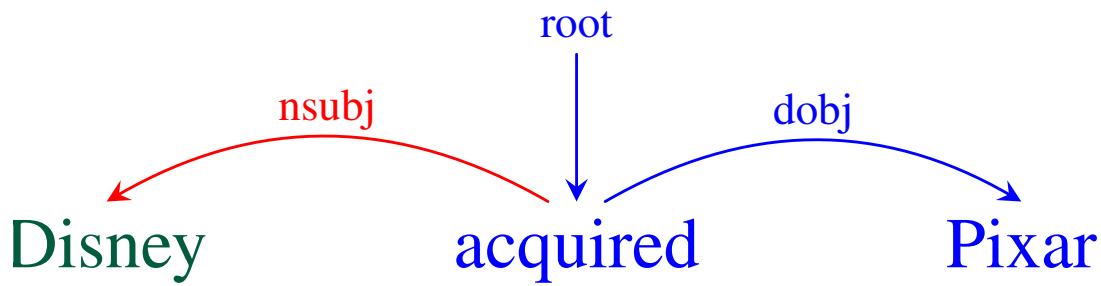
Composition



$$\frac{\dfrac{(\textbf{nsubj} \quad \lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \arg_1(z_e, x_a)) \qquad \dfrac{(\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar})}{\lambda z.\ \exists y.\ \text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \arg_2(z_e, y_a)} \qquad (\textbf{Disney} \quad \lambda x.\text{Disney}(x_a))}{\lambda g z.\exists x y.\text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \text{g}(x) \wedge \arg_1(z_e, x_a) \wedge \arg_2(z_e, y_a)}$$

Dependencies to Logical Forms

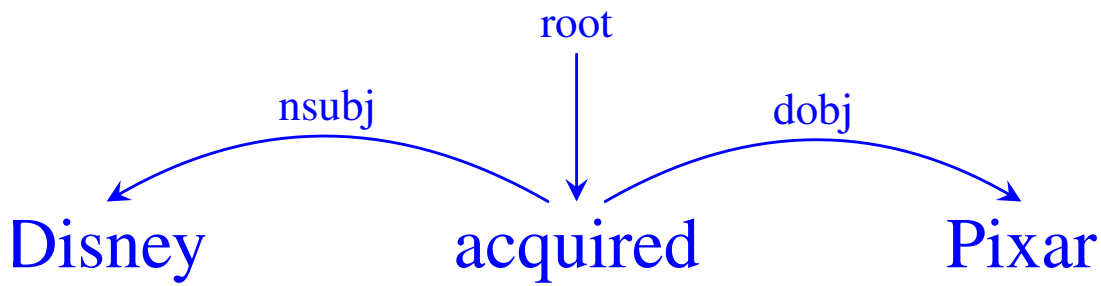
Composition



$$\frac{\dfrac{(\textbf{nsubj} \quad \lambda f g z.\ \exists x.\ f(z) \wedge g(x) \wedge \arg_1(z_e, x_a)) \qquad \dfrac{(\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar}) \qquad \textbf{Disney})}{\lambda z.\ \exists y.\ \text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \arg_2(z_e, y_a)} \quad \lambda x.\text{Disney}(x_a)}{\lambda g z.\exists x y.\text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \text{g}(x) \wedge \arg_1(z_e, x_a) \wedge \arg_2(z_e, y_a)}}{\lambda z.\exists x y.\text{acquired}(z_e) \wedge \text{Pixar}(y_a) \wedge \text{Disney}(x_a) \wedge \arg_1(z_e, x_a) \wedge \arg_2(z_e, y_a)}$$

Dependencies to Logical Forms

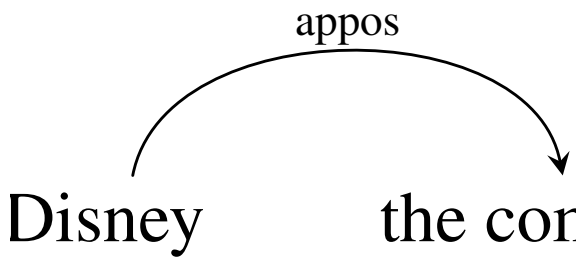
Composition



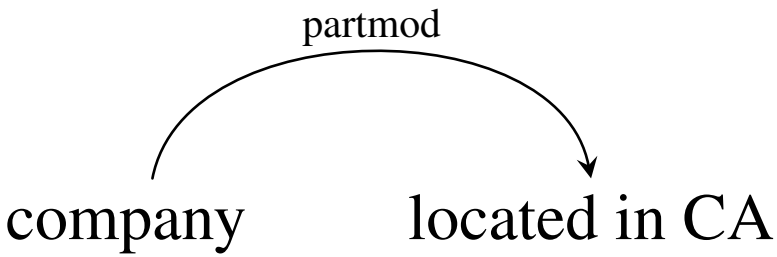
$$(\textbf{nsubj} \quad (\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar}) \quad \textbf{Disney})$$

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

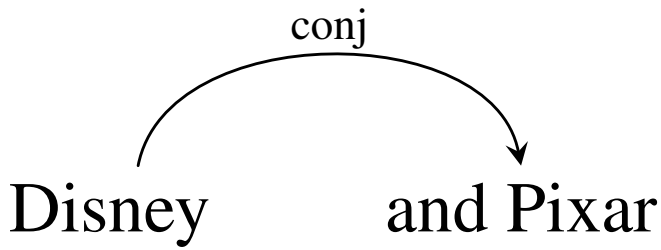
Dependencies to Logical Forms



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

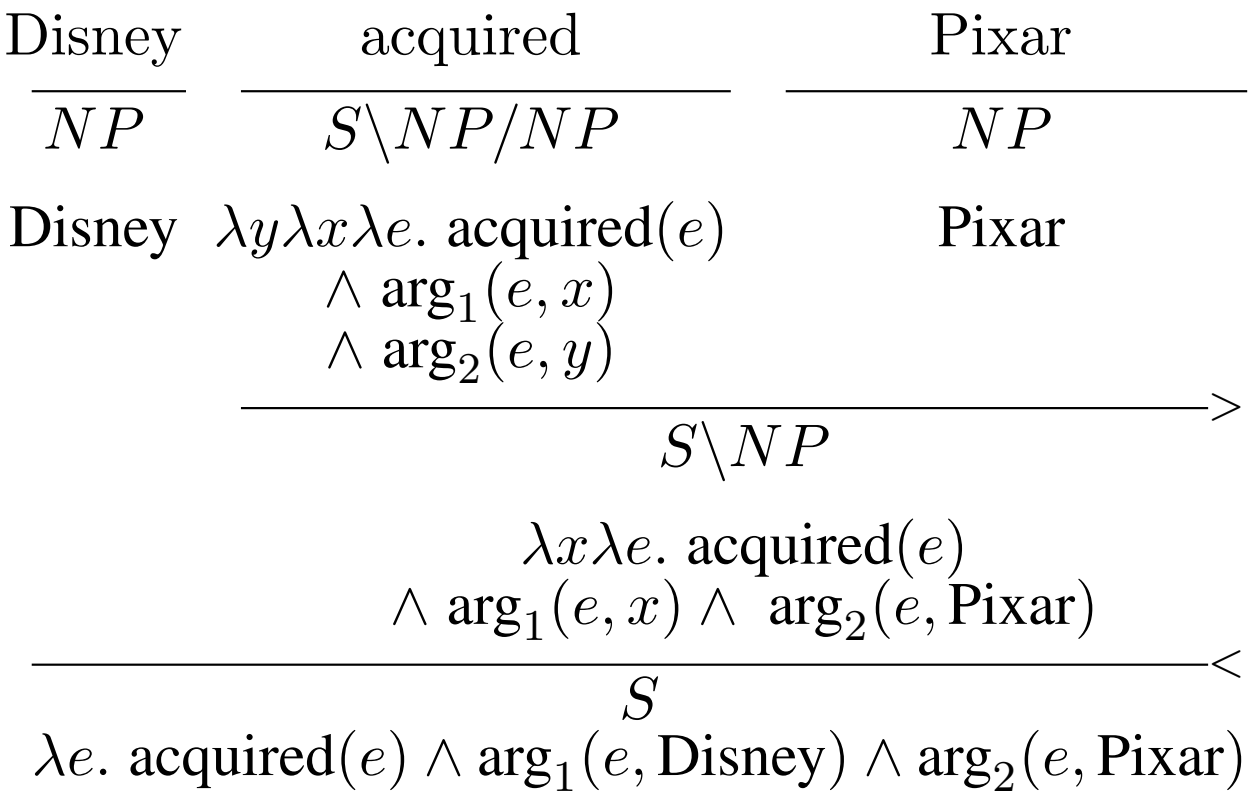


$$\textit{partmod} = \lambda fgx.\exists z.f(x) \wedge g(z) \wedge \text{arg}_2(z_e, x_a)$$



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

Comparison with CCG



Comparison with CCG

CCG	DepLambda
Lexicalized semantics S\NP/NP : $\lambda y \lambda x \lambda e.\, \text{acquired}(e) \wedge \text{arg}_1(e,x) \wedge \text{arg}_2(e,y)$	Simple lexical semantics $\lambda x.\, \text{acquired}(x_e)$
Words drive composition	Dependencies drive composition
Language specific types	Mostly universal

Comparison with CCG

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

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

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

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

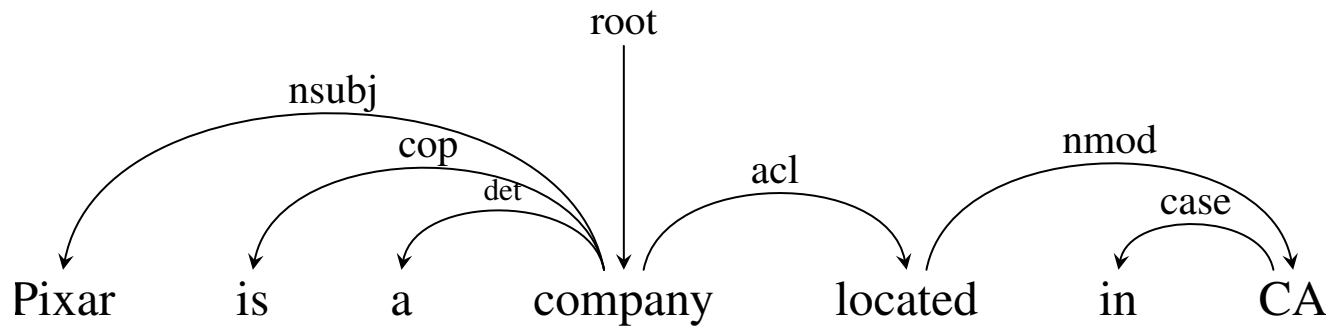
Function could be any computation

- 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



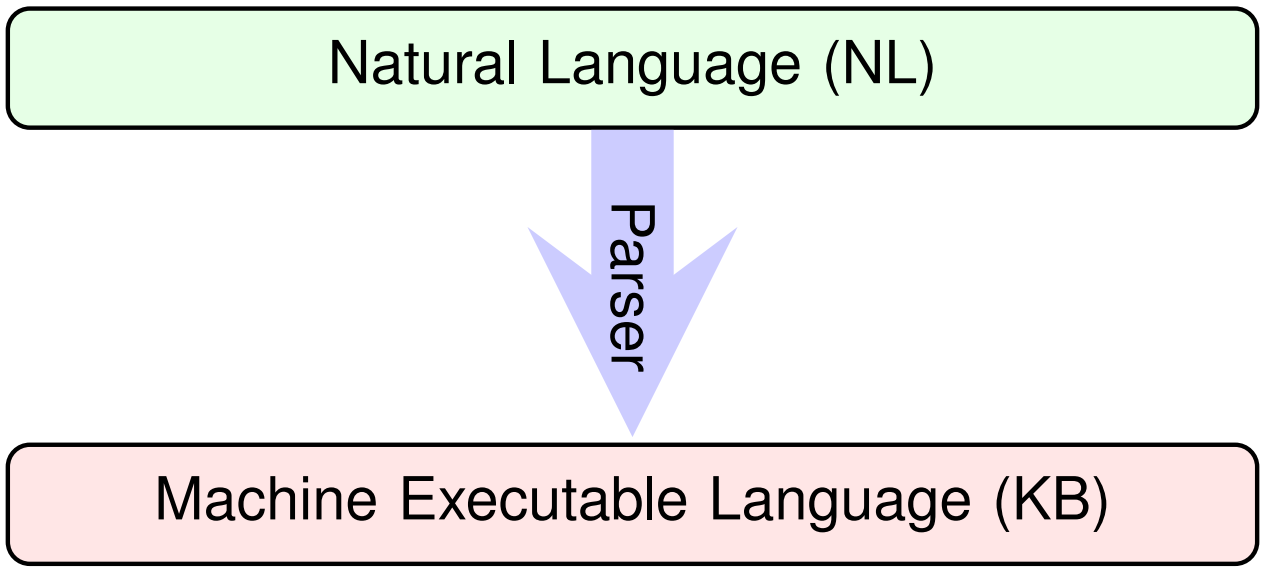
$\ldots > \text{dobj} > \cdots > \text{nsubj} > \ldots$

$$\ldots \quad \underset{\substack{\lambda f g x.\exists z.\\ f(x) \wedge g(z) \wedge \\ \arg_2(z_e, x_a)}}{(acl} \quad \underset{\lambda x.\text{compay}(x_a)}{\text{company}} \quad \underset{\substack{\lambda f g z.\exists x.\\ f(z) \wedge g(x) \\ arg_{in}(z_e, x_a)}}{(nmod} \quad \underset{\lambda x.\text{located}(x_e)}{\text{located}} \quad \frac{\underset{\lambda f g x.f(x)}{(case} \quad \underset{\lambda x.\text{CA}(x_a)}{\text{CA}} \quad \underset{\lambda x.\text{empty}(x)}{\text{in)}}}{\lambda x.\ \text{CA}(x_a)} \quad))\ldots$$

lambda expression composition

$$\exists z.\text{company}(\text{Pixar}) \wedge \text{located}(z_e) \wedge \arg_2(z_e, \text{Pixar}) \wedge \arg_{\text{in}}(z_e, \text{CA})$$

Grounded Semantic Parsing



Freebase Semantic Parsing

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

Question

Who is the director of Titanic?

Answer

{James Cameron}

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

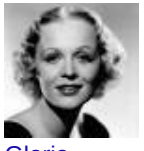
Featured song: My Heart Will Go On

Cast


Leonardo DiCaprio
Jack Dawson


Kate Winslet
Rose DeWitt Bukater


Billy Zane
Caledon Hockley


Gloria Stuart
Rose DeWitt Bukater


Kathy Bates
Molly Brown

Freebase Semantic Parsing

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

Question

Who is the director of Titanic?

Logical Form

$\lambda x.\ \text{film.directed\_by}(\textit{Titanic}, x)$

Latent

Answer

{James Cameron}

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

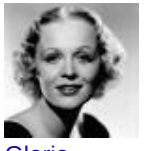
Featured song: My Heart Will Go On

Cast


Leonardo DiCaprio
Jack Dawson

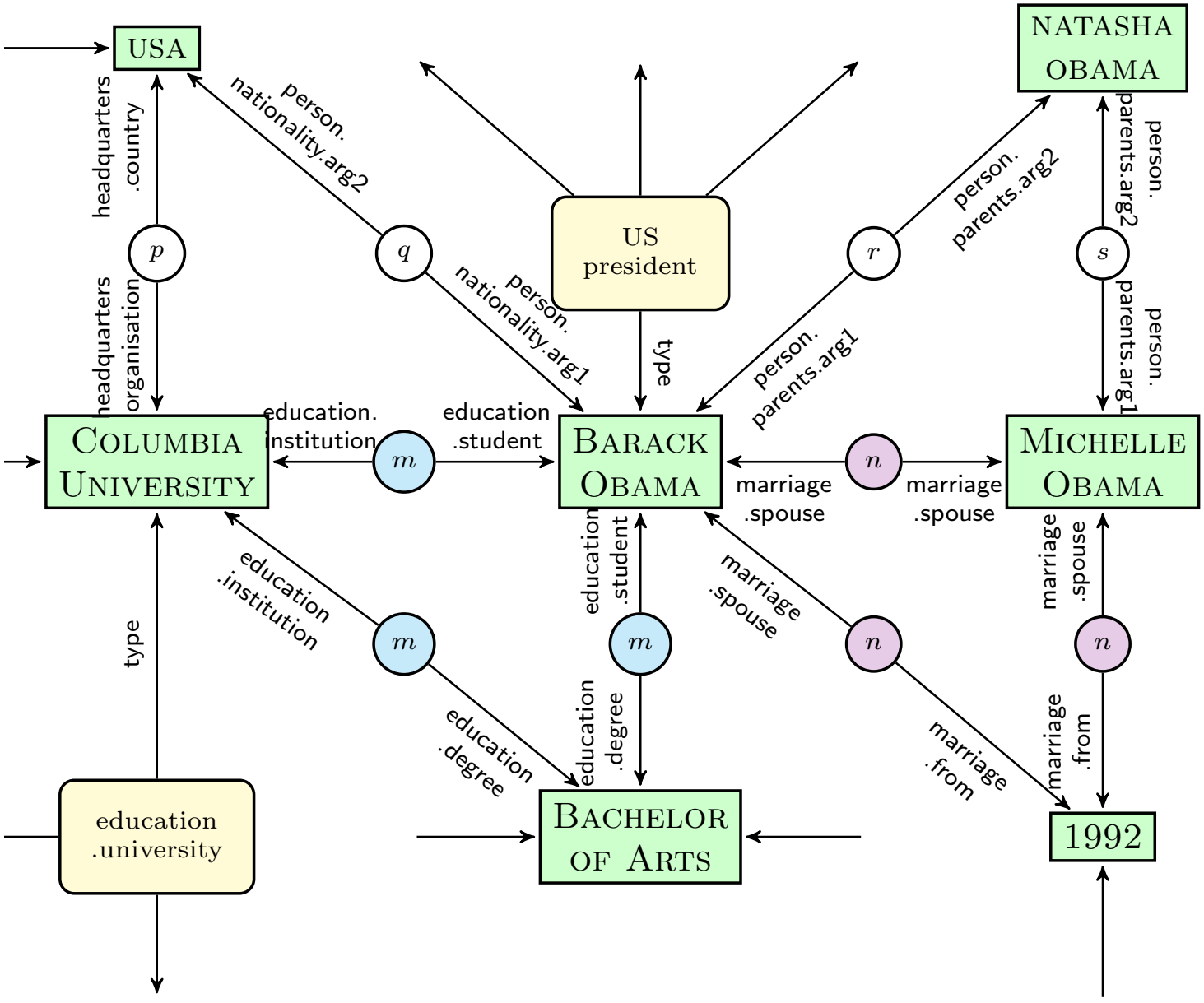

Kate Winslet
Rose DeWitt Bukater


Billy Zane
Caledon Hockley

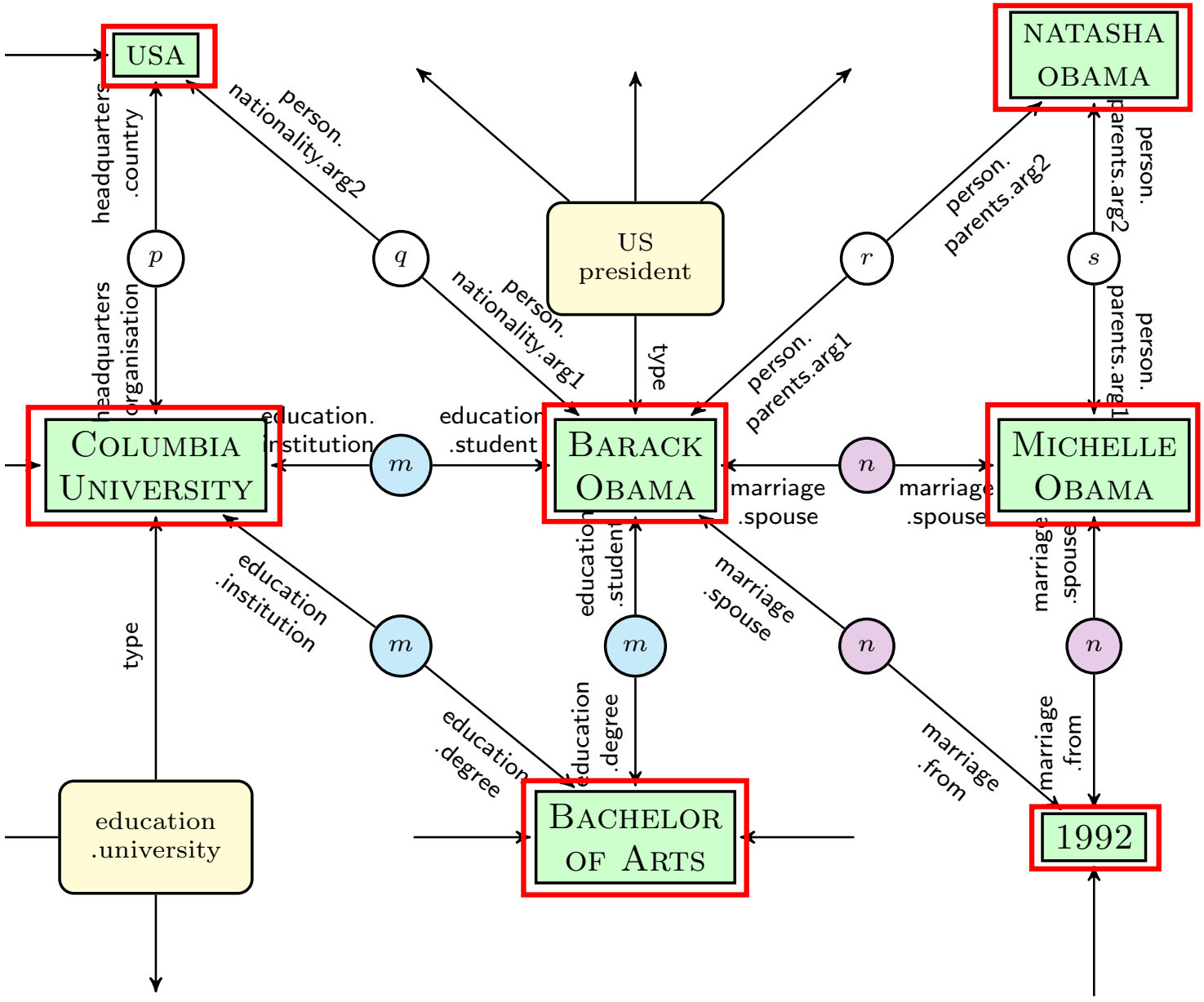

Gloria Stuart
Rose DeWitt Bukater


Kathy Bates
Molly Brown

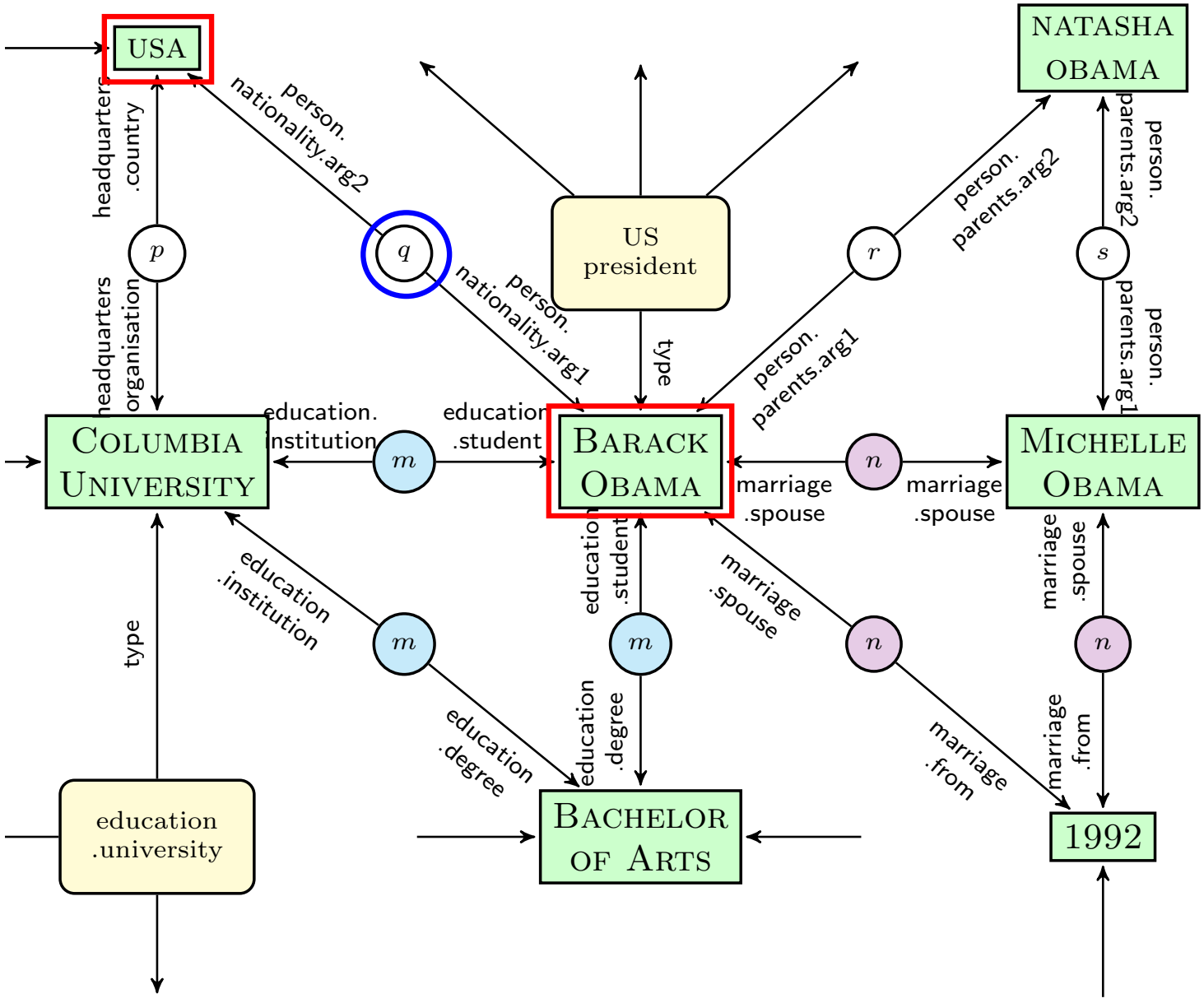
Freebase is a Graph



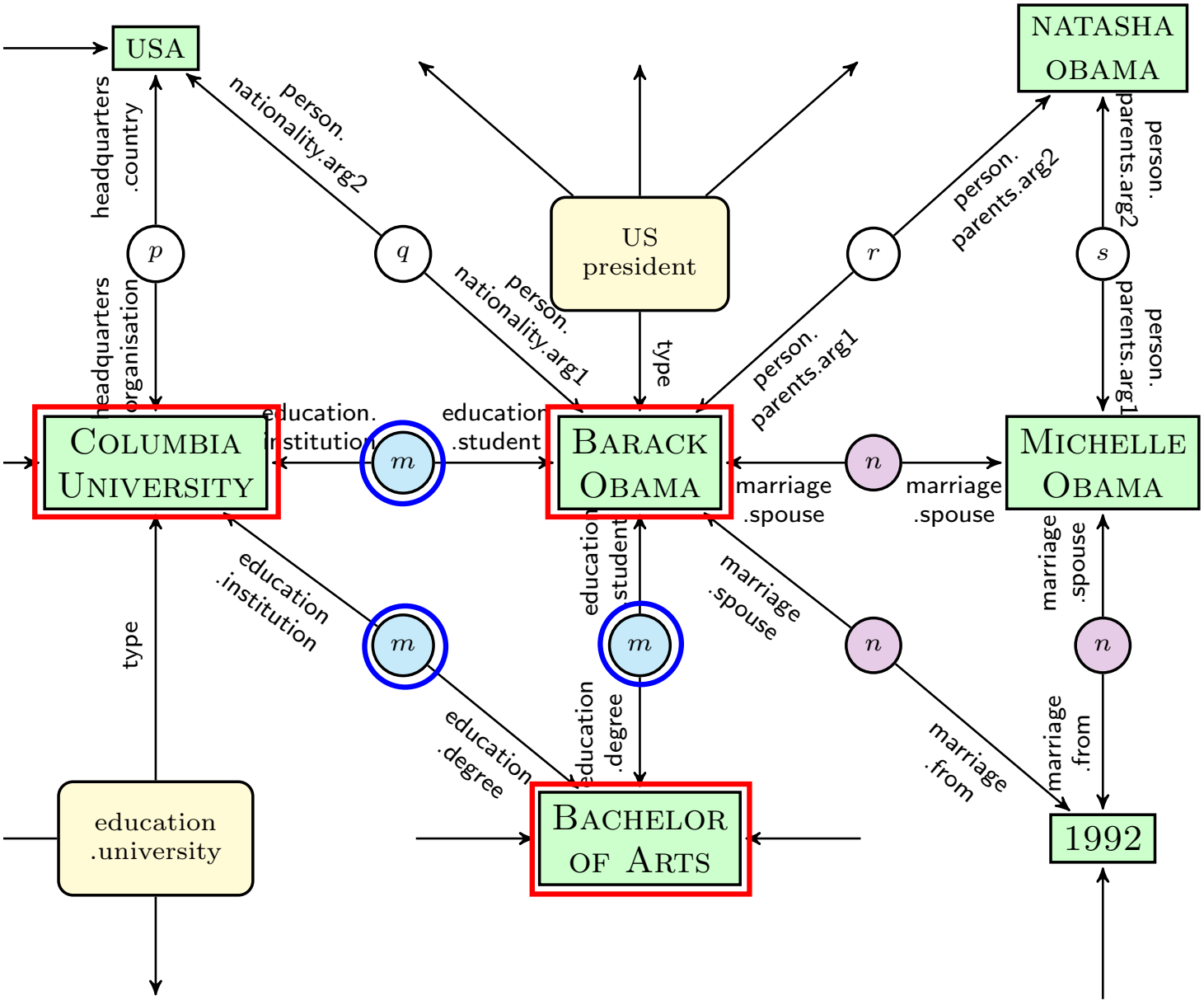
Freebase is a Graph



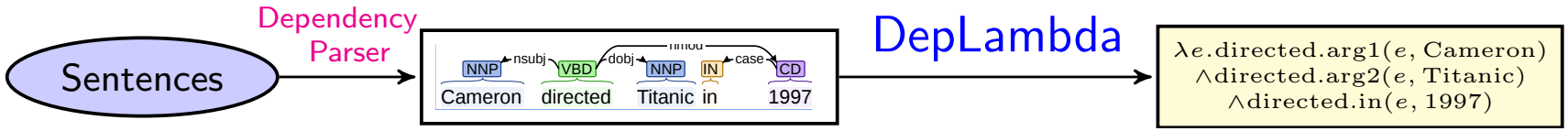
Freebase is a Graph



Freebase is a Graph

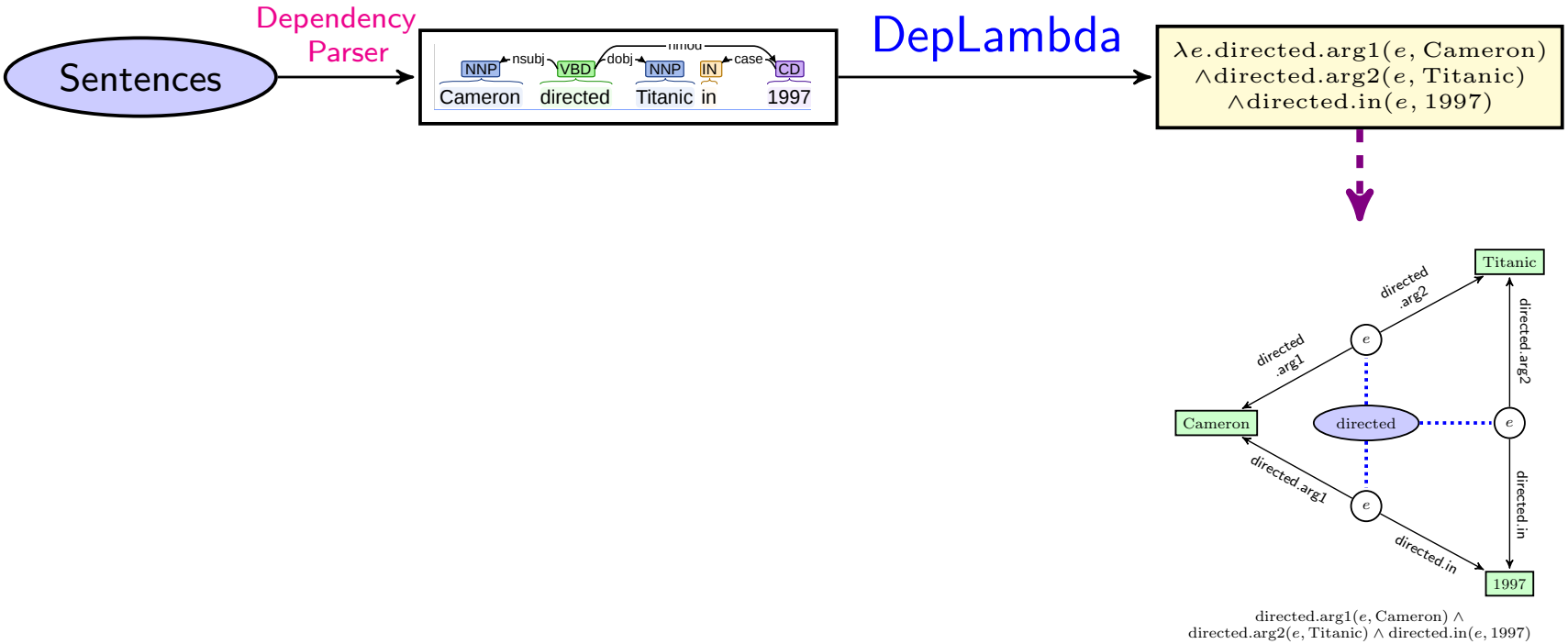


Semantic Parsing as Graph Matching



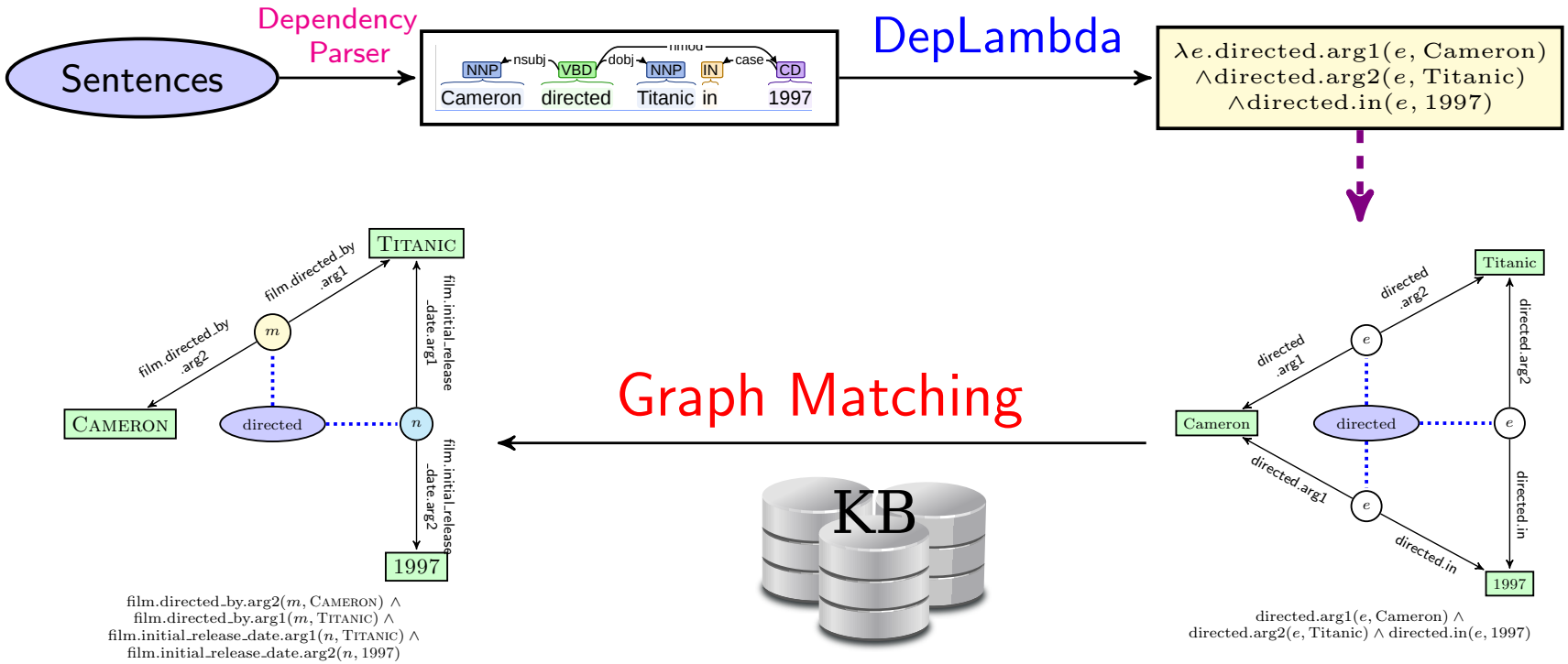
Reddy, Lapata, Steedman (TACL 2014)

Semantic Parsing as Graph Matching



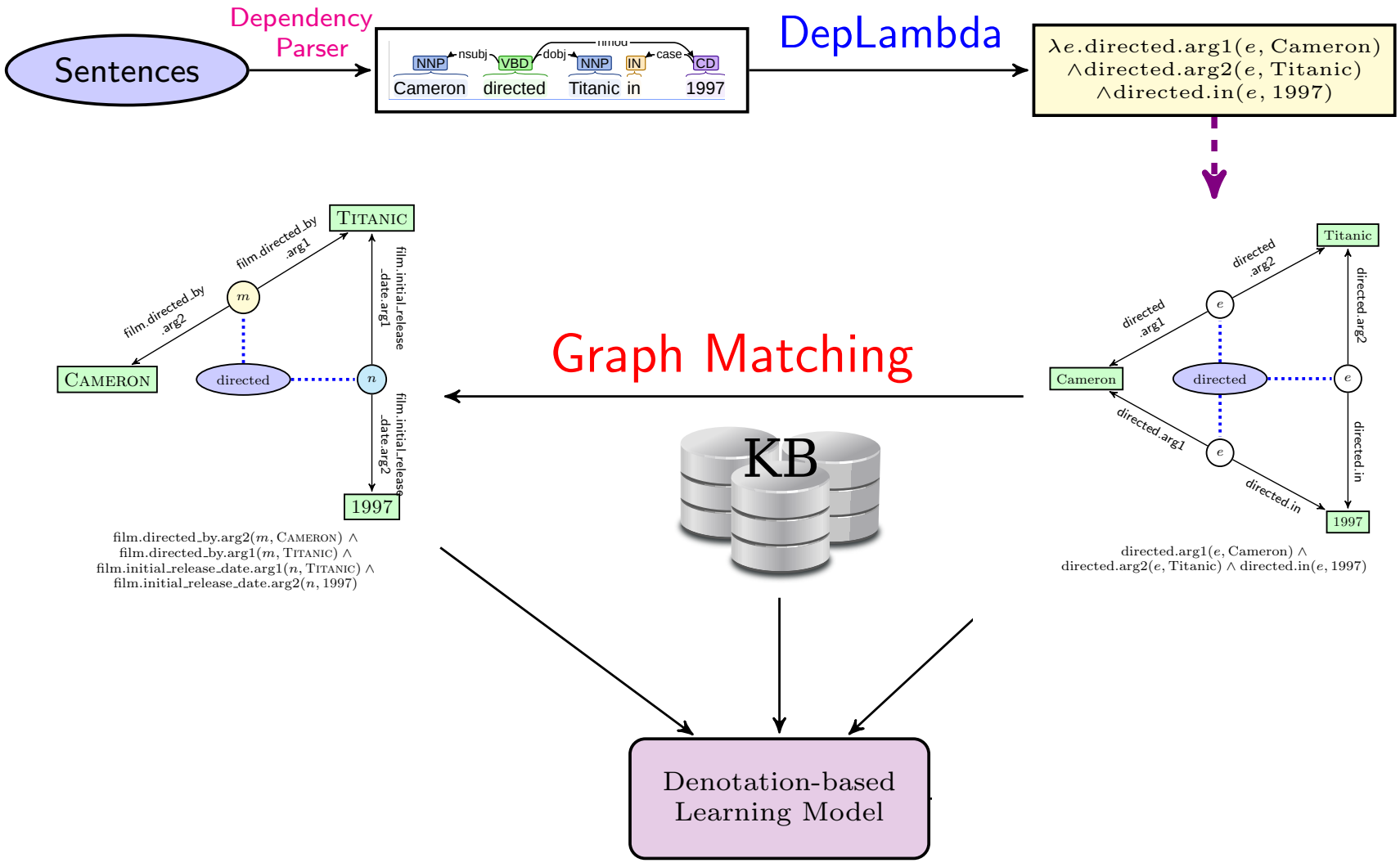
Reddy, Lapata, Steedman (TACL 2014)

Semantic Parsing as Graph Matching



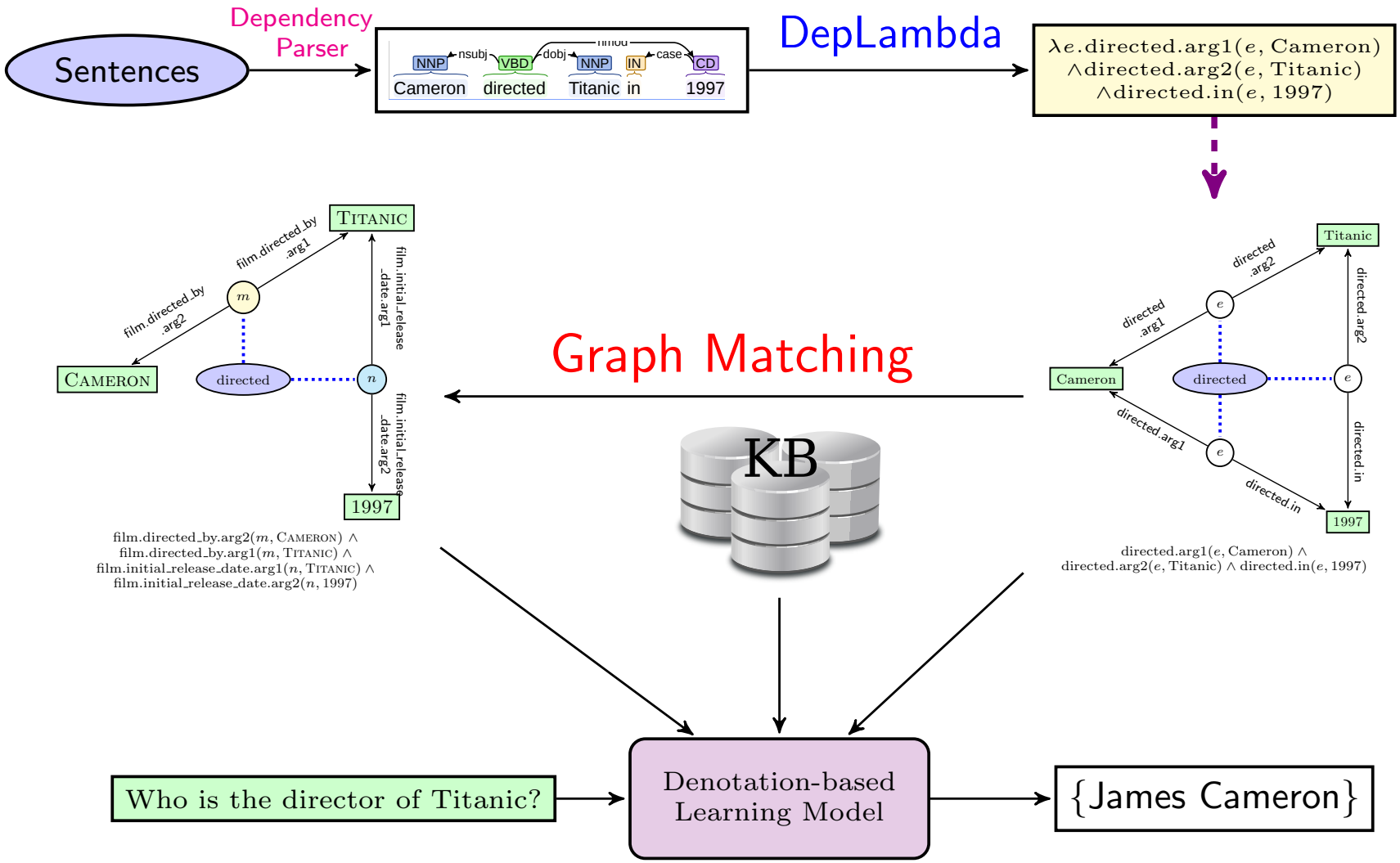
Reddy, Lapata, Steedman (TACL 2014)

Semantic Parsing as Graph Matching



Reddy, Lapata, Steedman (TACL 2014)

Semantic Parsing as Graph Matching

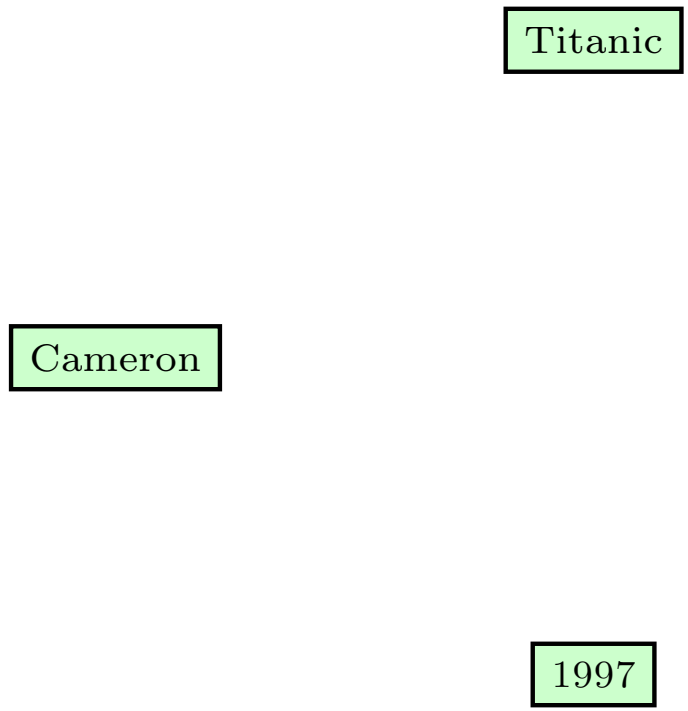


Reddy, Lapata, Steedman (TACL 2014)

Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

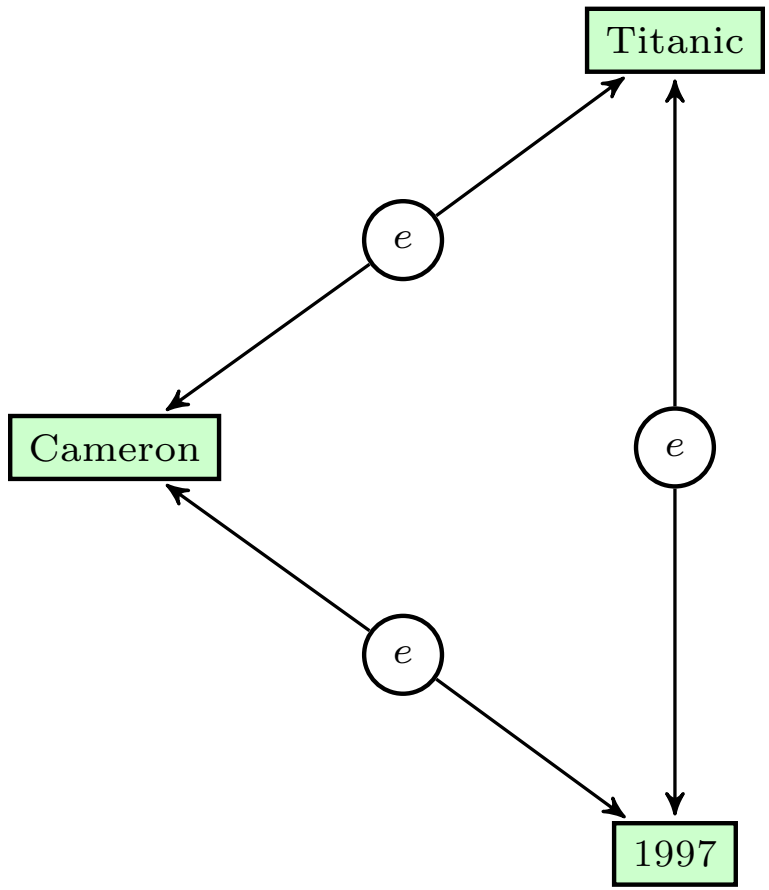
$$\lambda e.\text{directed.arg1}(e, \text{Cameron}) \wedge \text{directed.arg2}(e, \text{Titanic}) \wedge \text{directed.in}(e, 1997)$$



Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

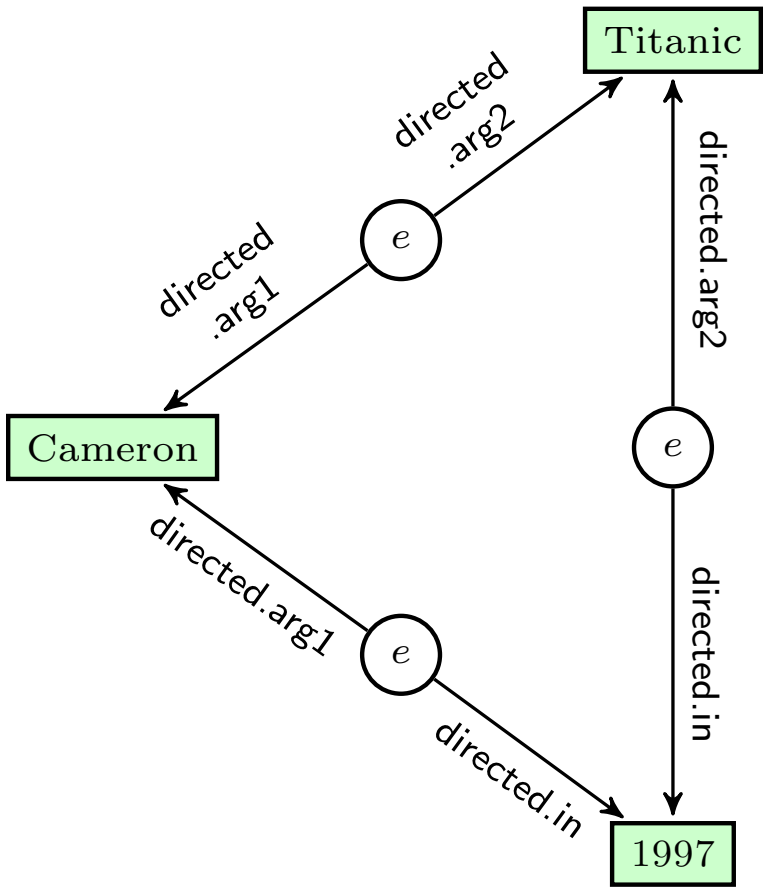
$$\lambda e.\text{directed.arg1}(\mathbf{e}, \texttt{Cameron}) \wedge \text{directed.arg2}(\mathbf{e}, \texttt{Titanic}) \wedge \text{directed.in}(\mathbf{e}, 1997)$$



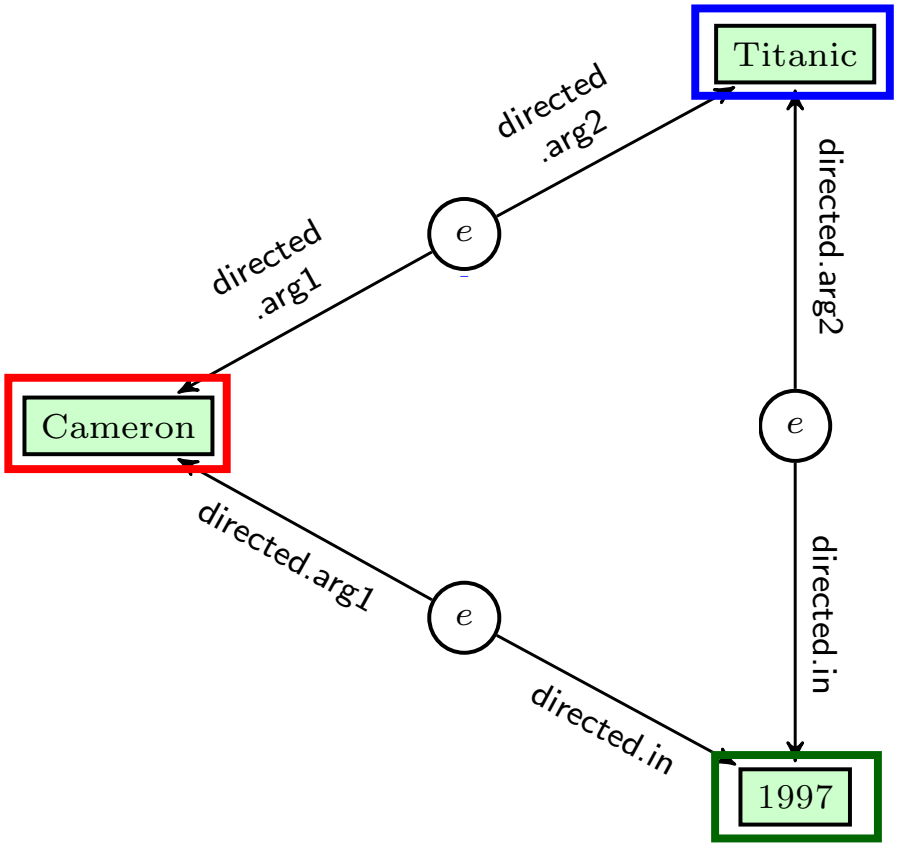
Logical Form to Ungrounded Graph

Cameron directed Titanic in 1997

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

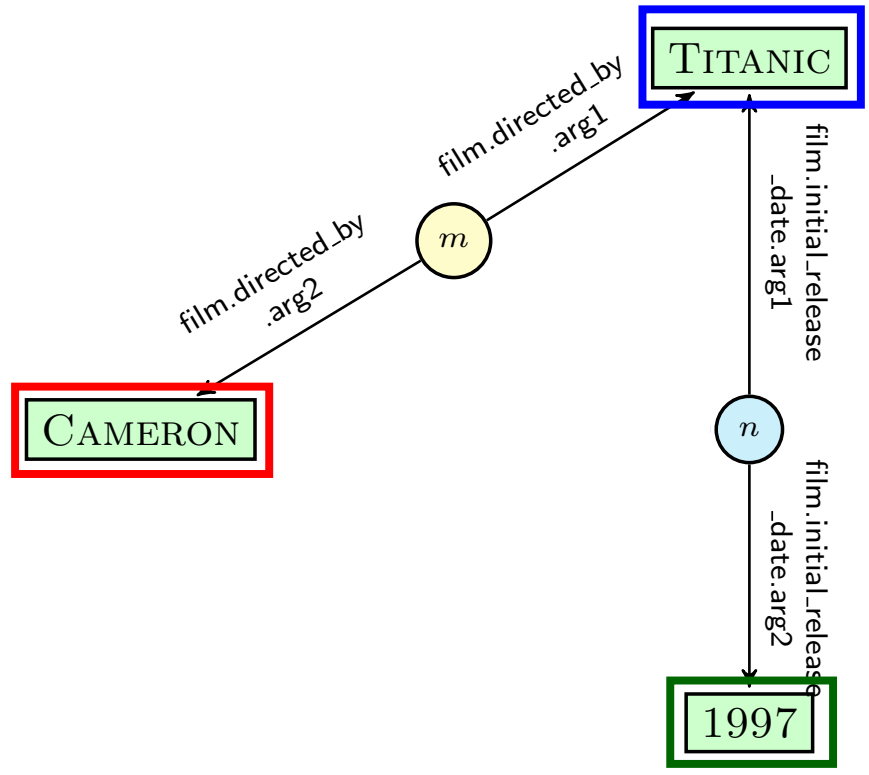


Graph Matching



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

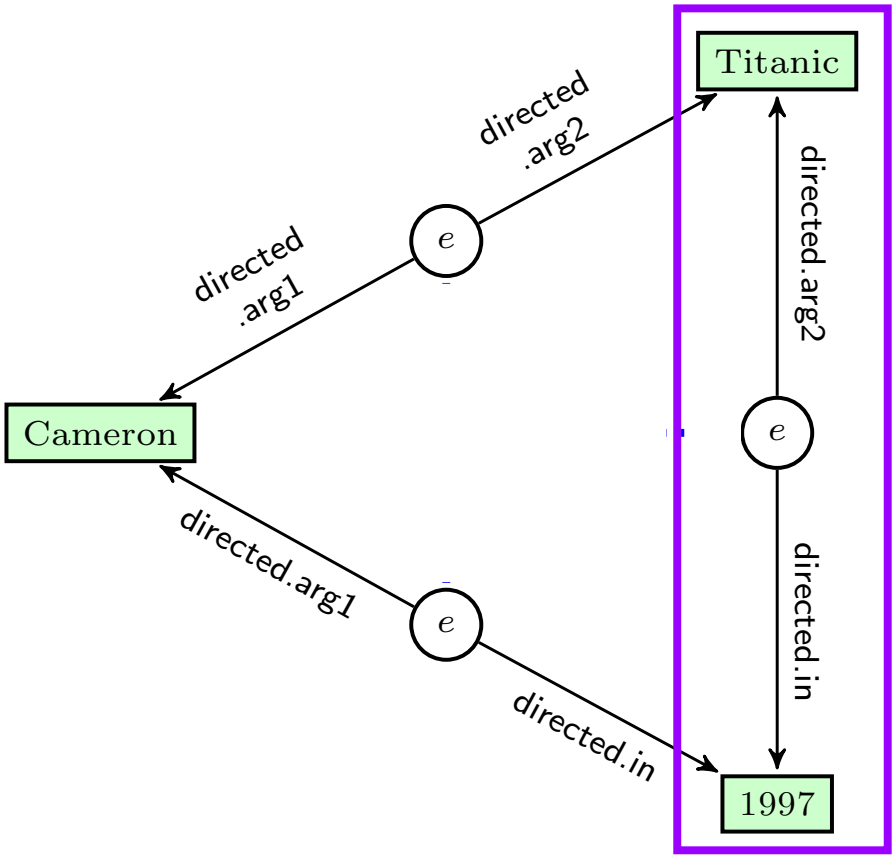
Ungrounded Graph



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

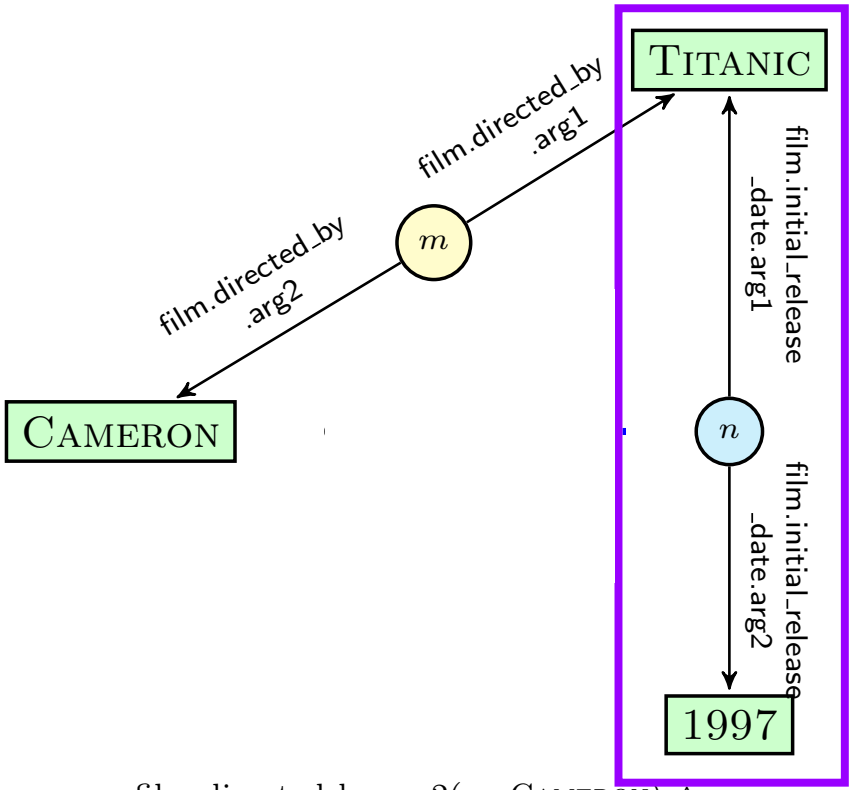
Grounded Graph

Graph Matching



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

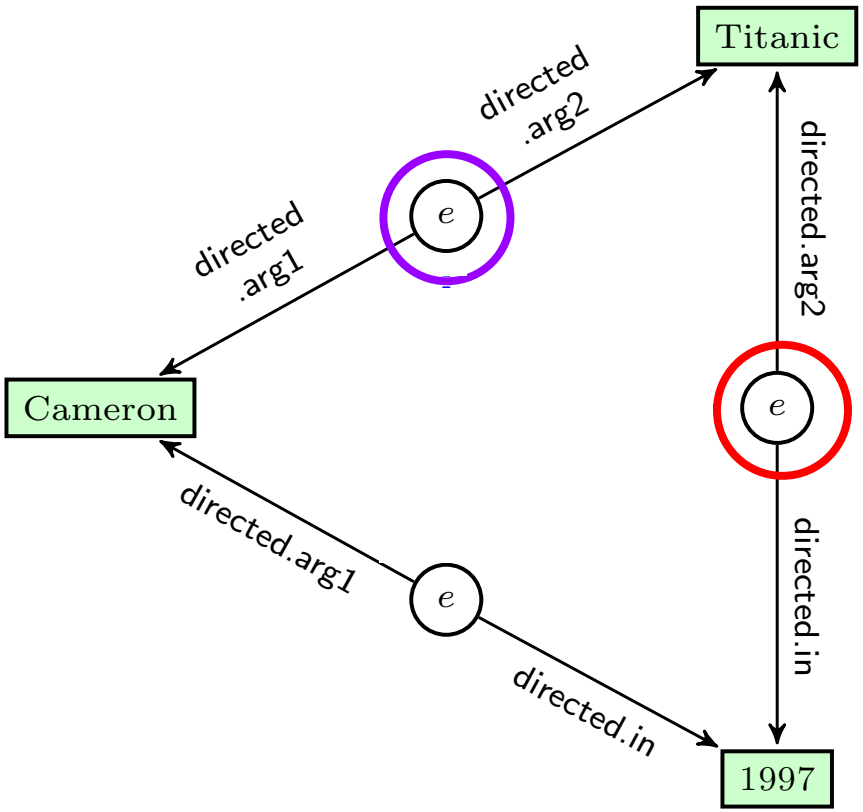
Ungrounded Graph



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

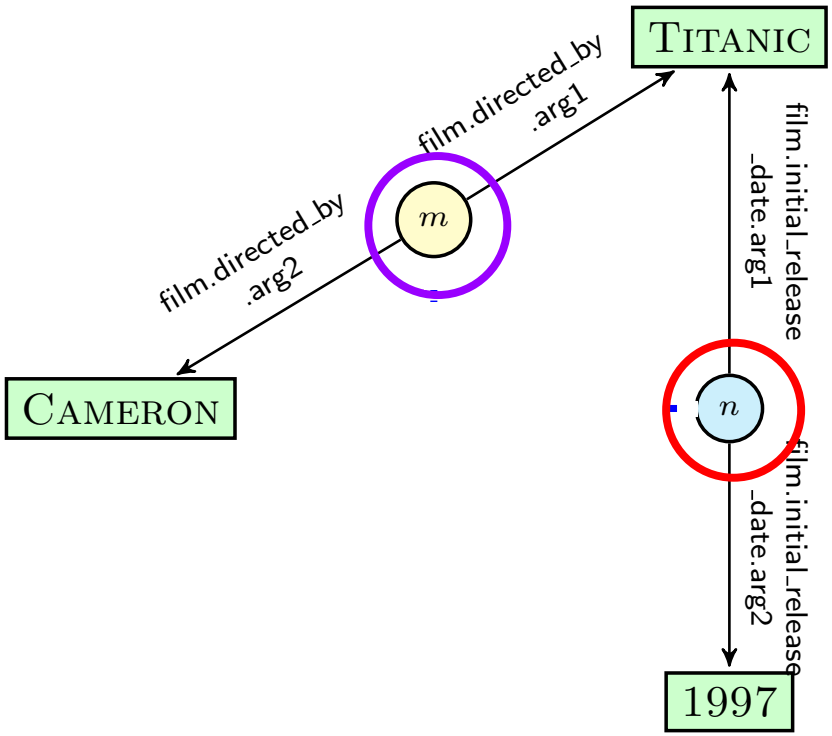
Grounded Graph

Graph Matching



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

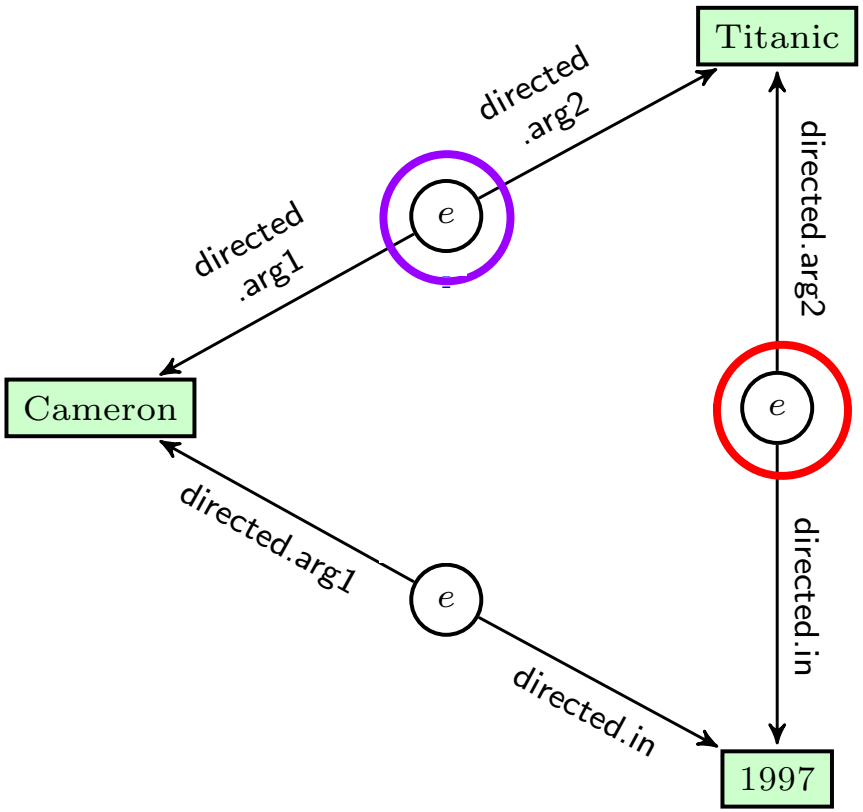
Ungrounded Graph



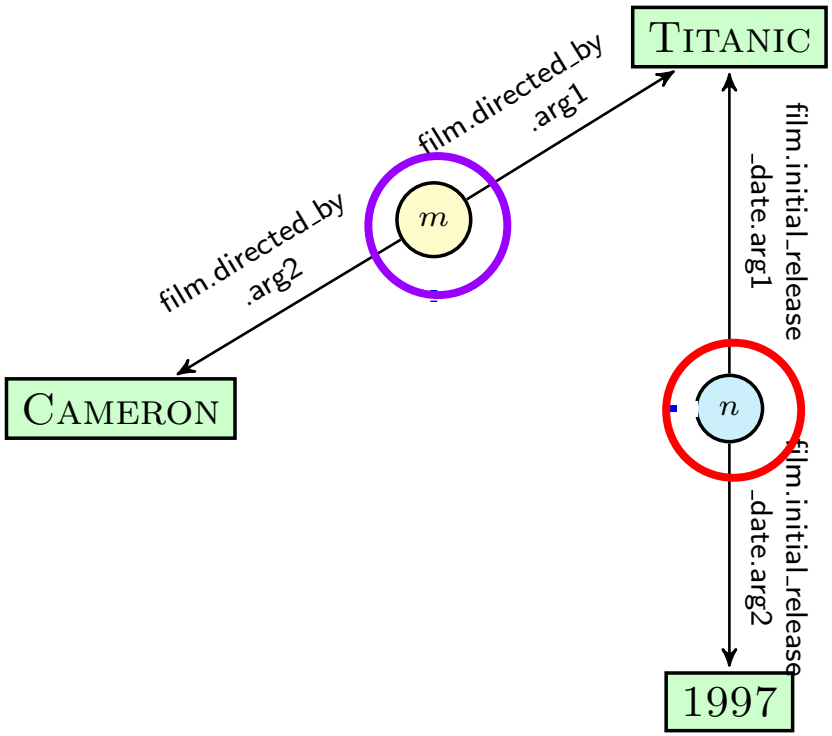
$\text{film.directed\_by.arg2}(m, \textsc{Cameron}) \wedge$
$\text{film.directed\_by.arg1}(m, \textsc{Titanic}) \wedge$
$\text{film.initial\_release\_date.arg1}(n, \textsc{Titanic}) \wedge$
$\text{film.initial\_release\_date.arg2}(n, 1997)$

Grounded Graph

Graph Matching

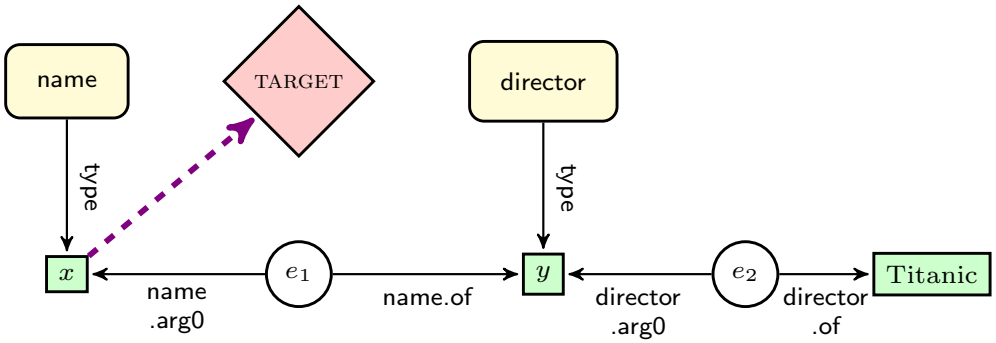


$\text{directed.arg1}(e, \text{Cameron}) \wedge$
$\text{directed.arg2}(e, \text{Titanic}) \wedge \text{directed.in}(e, 1997)$

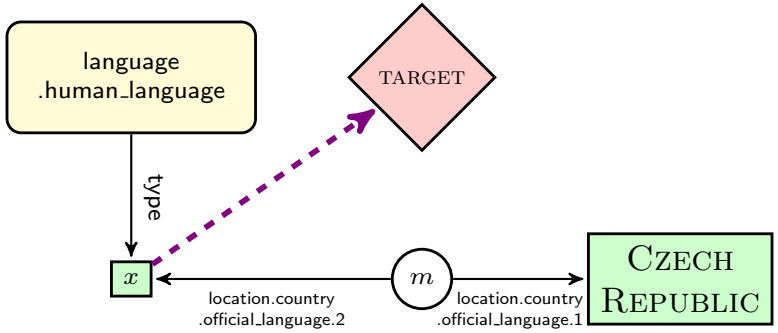


$\text{film.directed\_by.arg2}(m, \textsc{Cameron}) \wedge$
$\text{film.directed\_by.arg1}(m, \textsc{Titanic}) \wedge$
$\text{film.initial\_release\_date.arg1}(n, \textsc{Titanic}) \wedge$
$\text{film.initial\_release\_date.arg2}(n, 1997)$

Graph Mismatch

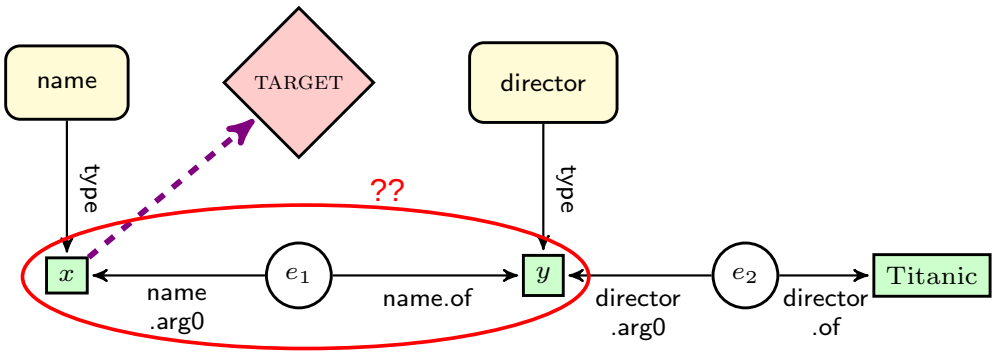


Ungrounded graph

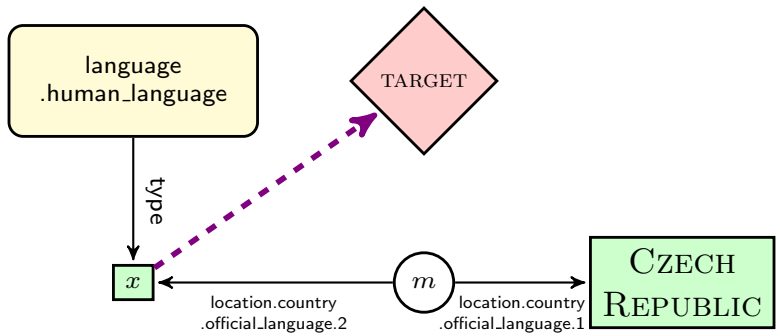


Freebase graph

Graph Mismatch

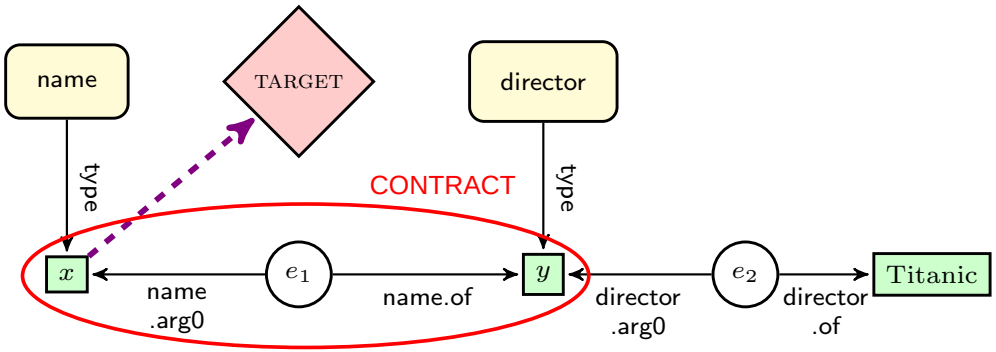


Ungrounded graph

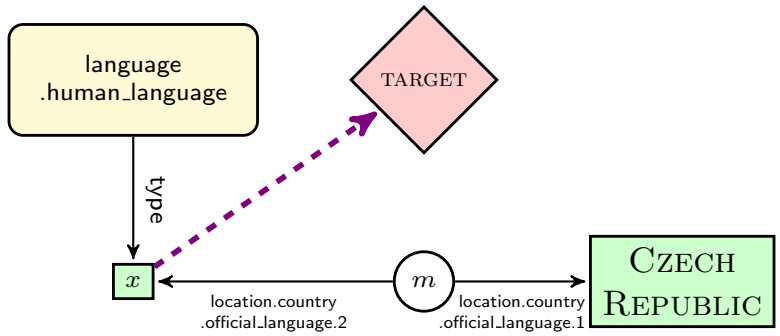


Freebase graph

Graph Mismatch



Ungrounded graph

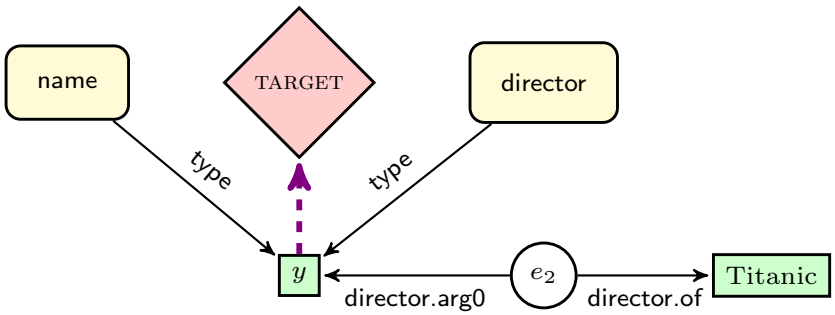


Freebase graph

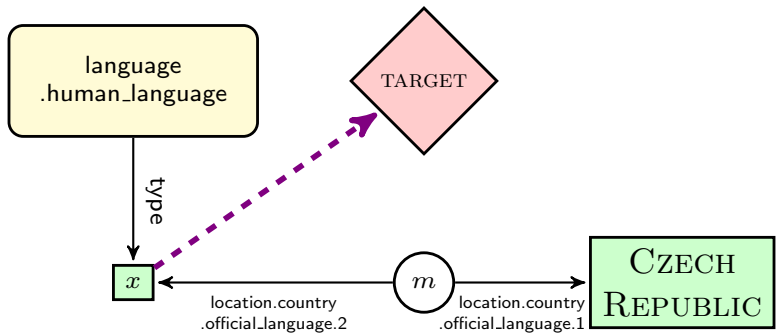
- Paraphrasing is an alternative†

†Narayan, Reddy, Cohen (INLG 2016)

Graph Mismatch



Ungrounded graph



Freebase graph

Learning Model

Structured Perceptron: Ranks grounded and ungrounded graph pairs

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

Learning Model

Structured Perceptron: Ranks grounded and ungrounded graph pairs

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

Features Φ: Defined over question, grounded and ungrounded graph

Learning Model

Structured Perceptron: Ranks grounded and ungrounded graph pairs

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

Features Φ: Defined over question, grounded and ungrounded graph

Training: Use gold graph to update weights

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

Learning Model

Structured Perceptron: Ranks grounded and ungrounded graph pairs

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

Features Φ: Defined over question, grounded and ungrounded graph

Training: Use gold graph to update weights

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

- But we do not have access to gold graphs
- Access only to the answers rather than the query

Learning Model

Oracle Graphs:

Get grounded graphs matching an ungrounded graph

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

Surrogate Gold Graph:

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

Beam Search:

Limit the predictions to 100 graph pairs

Baselines

SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

Baselines

SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

CCGGRAPH: CCG logical forms

Baselines

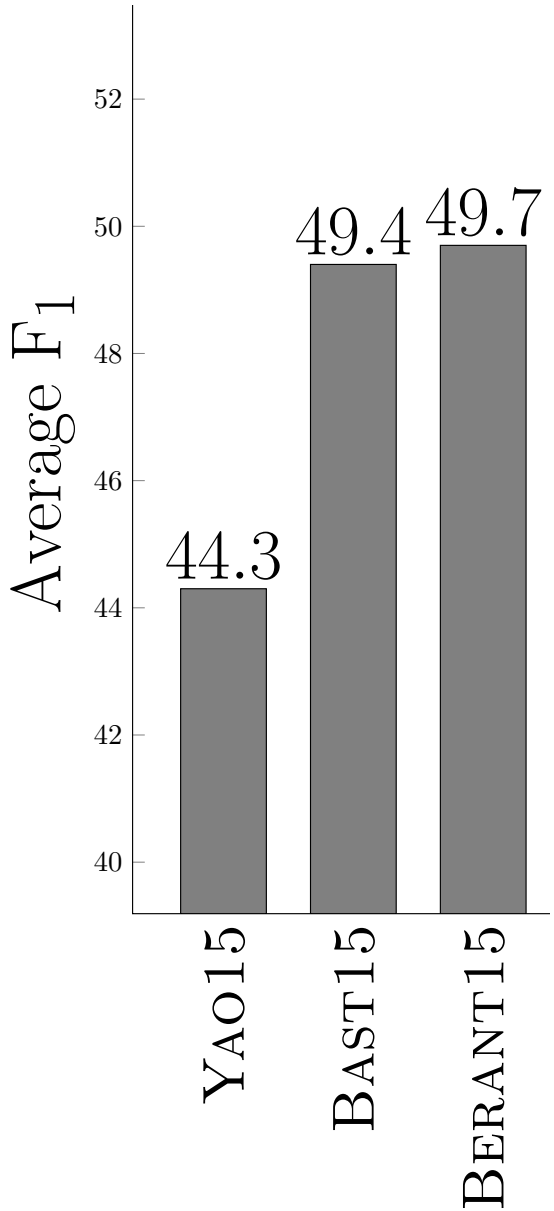
SIMPLEGRAPH: All entities connected to a single event

- Does not handle compositional questions

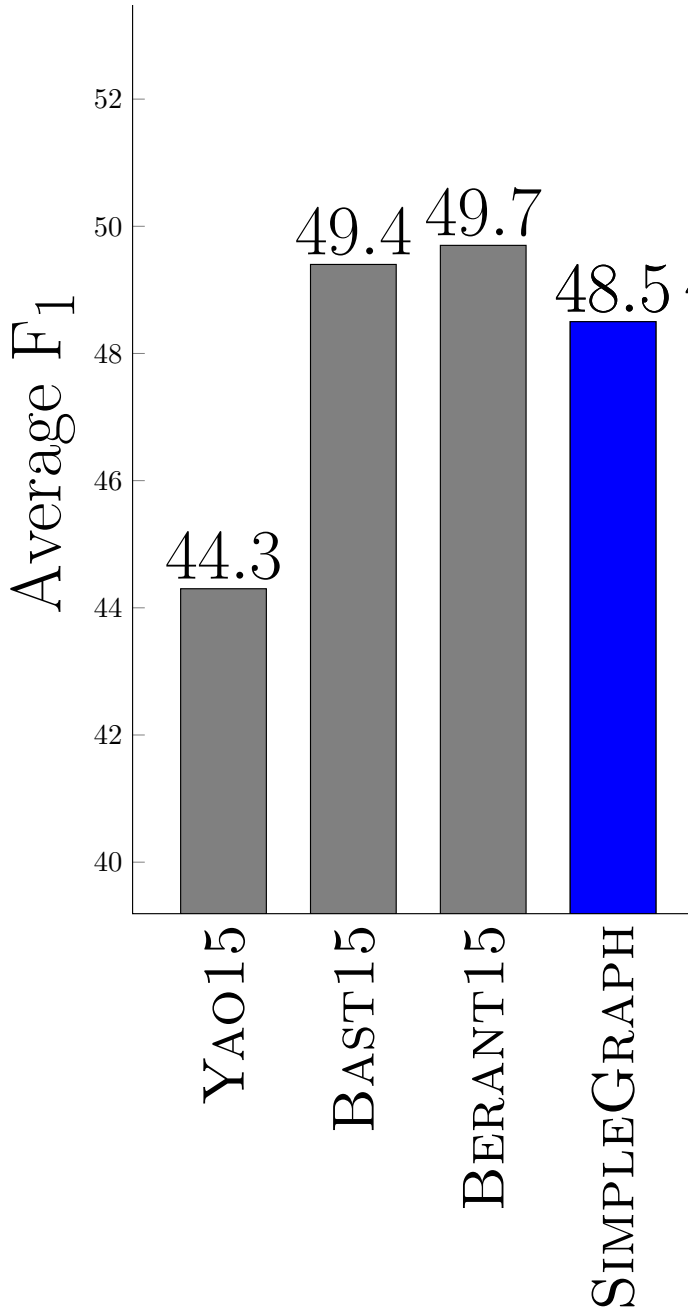
CCGGRAPH: CCG logical forms

DEPTREE: Directly transduce a dependency tree to target graph

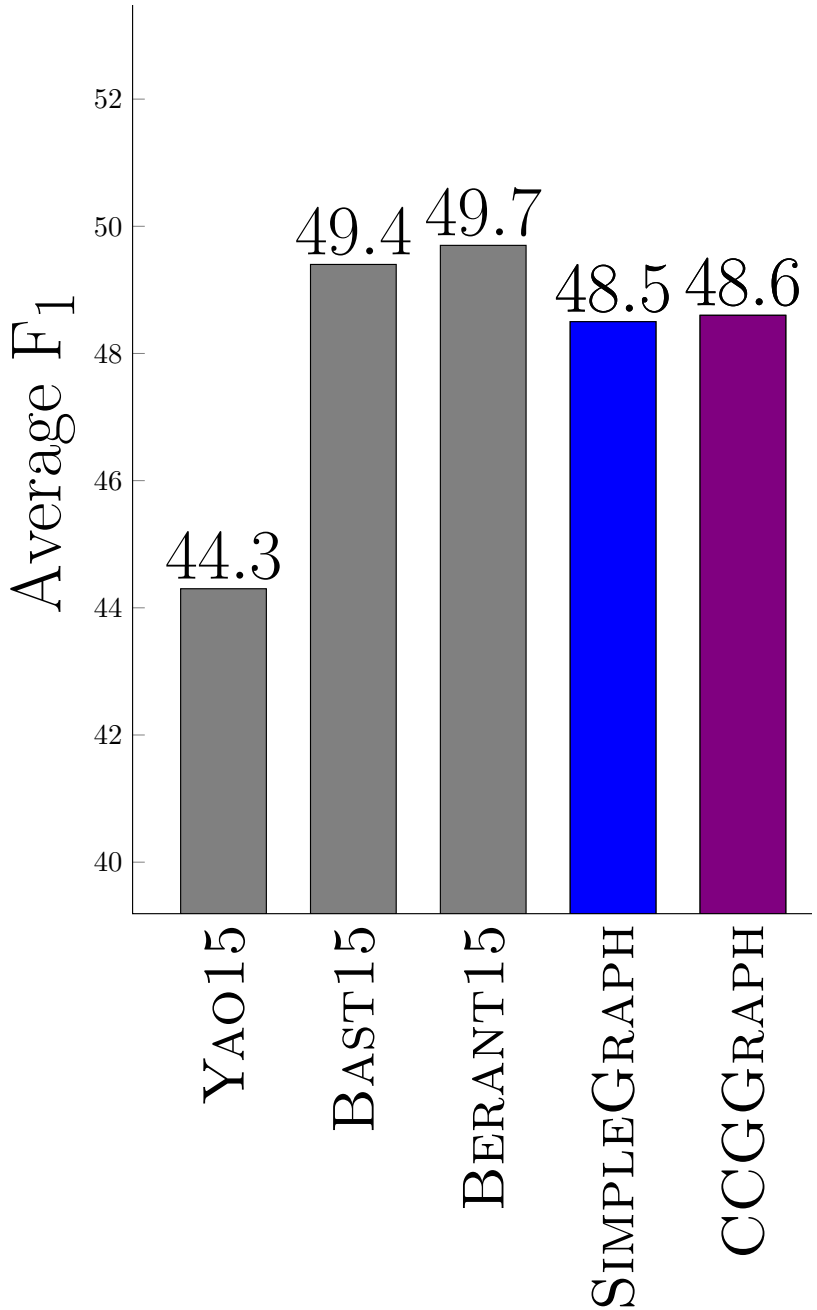
Results on WebQuestions



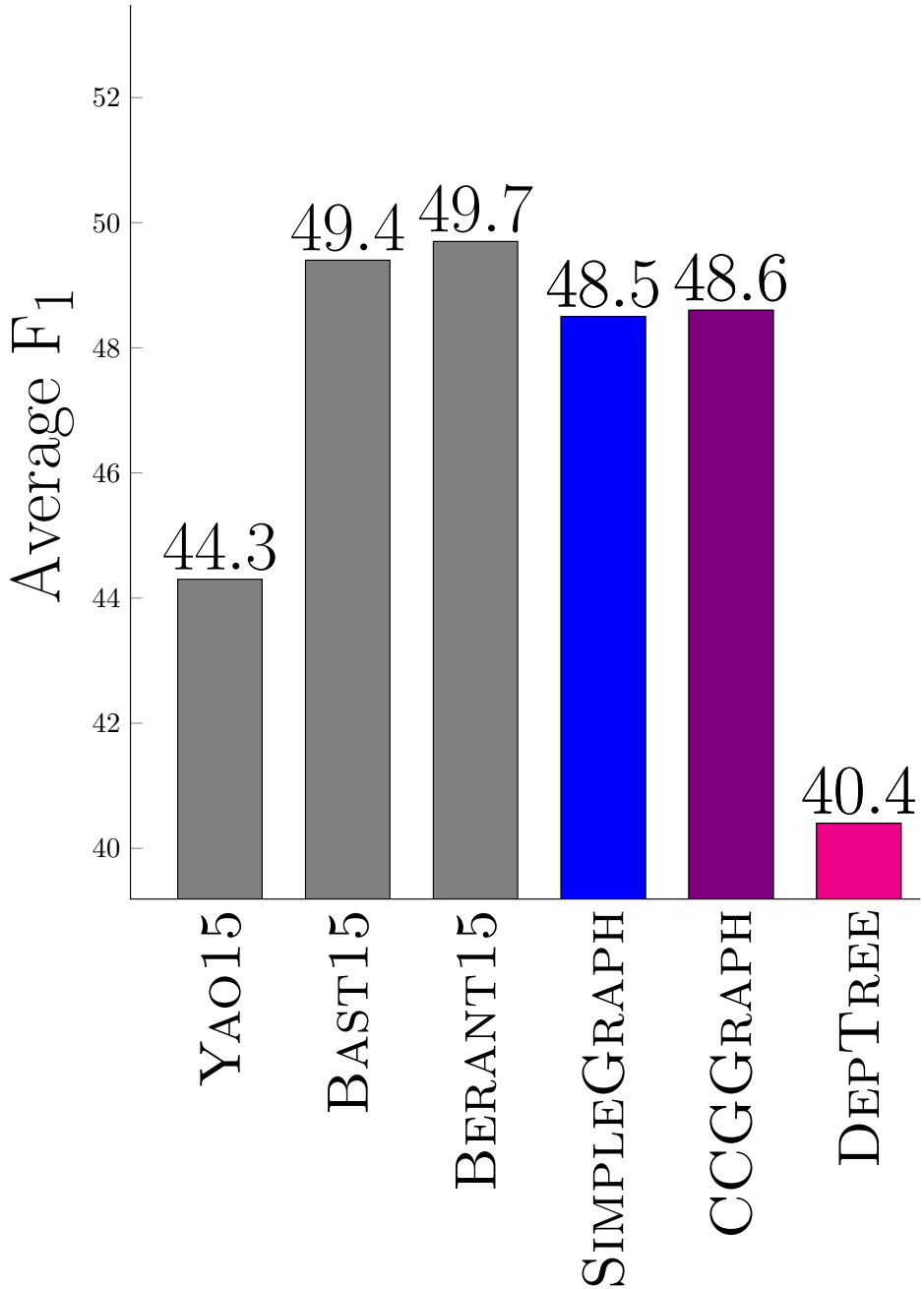
Results on WebQuestions



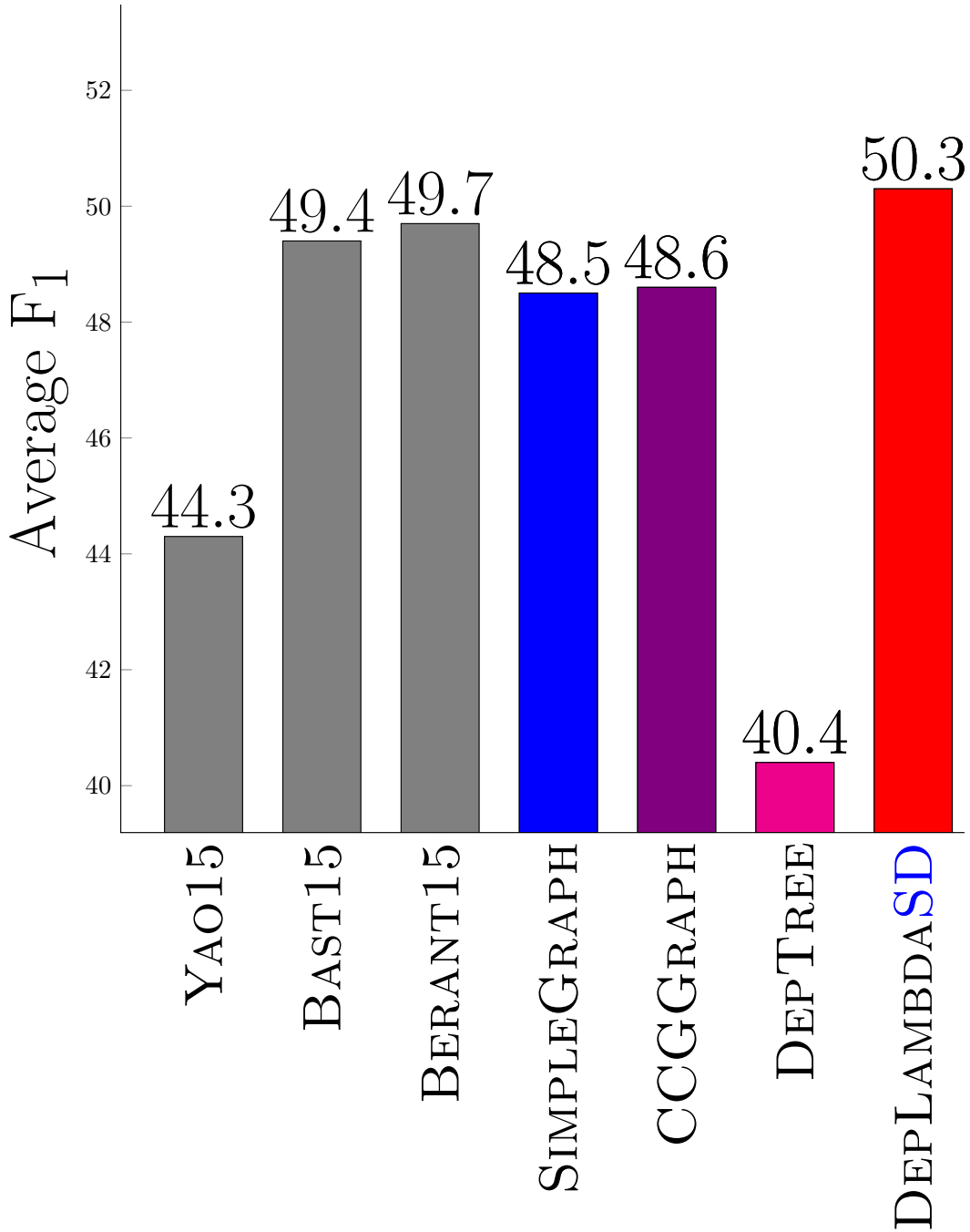
Results on WebQuestions



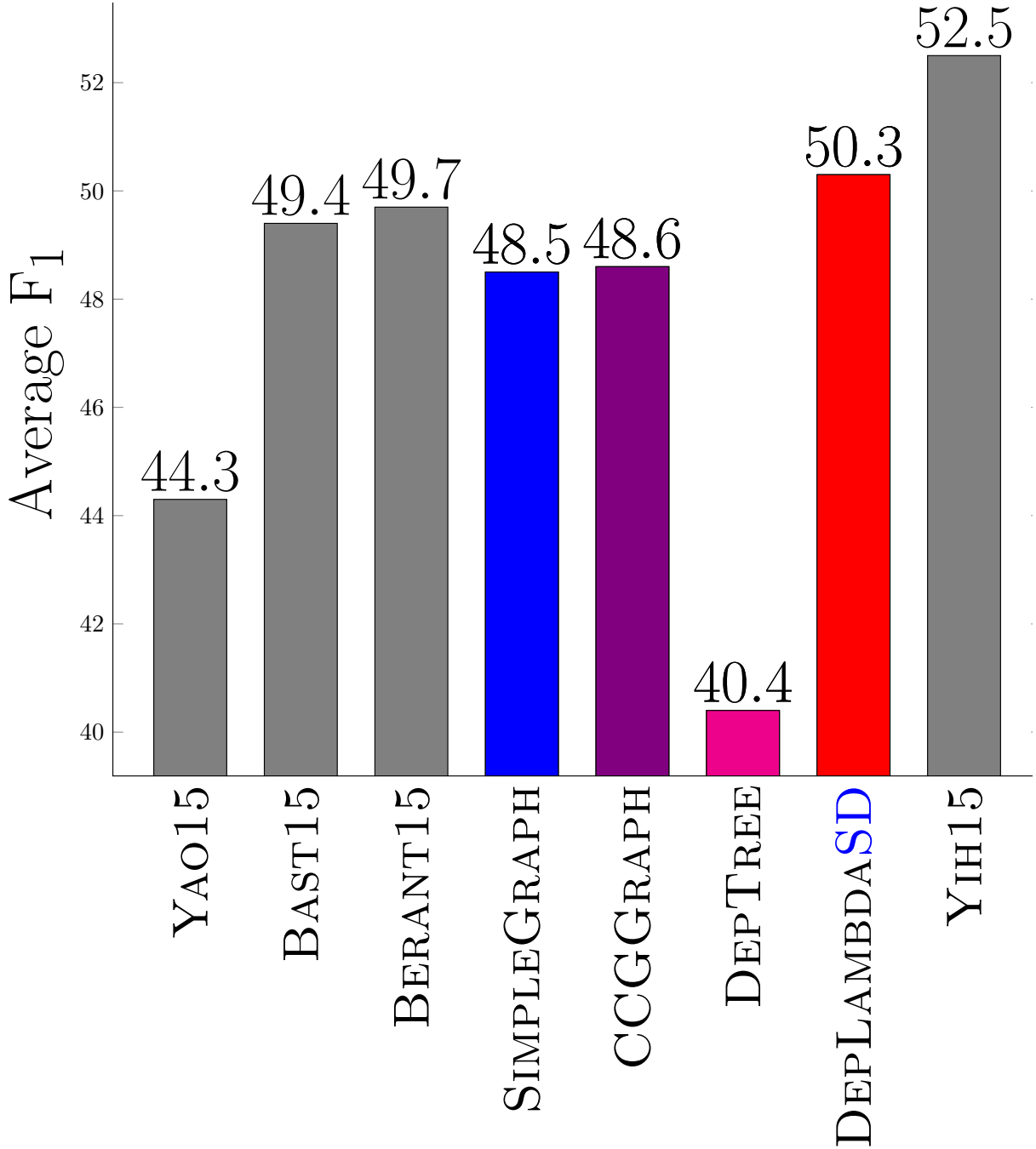
Results on WebQuestions



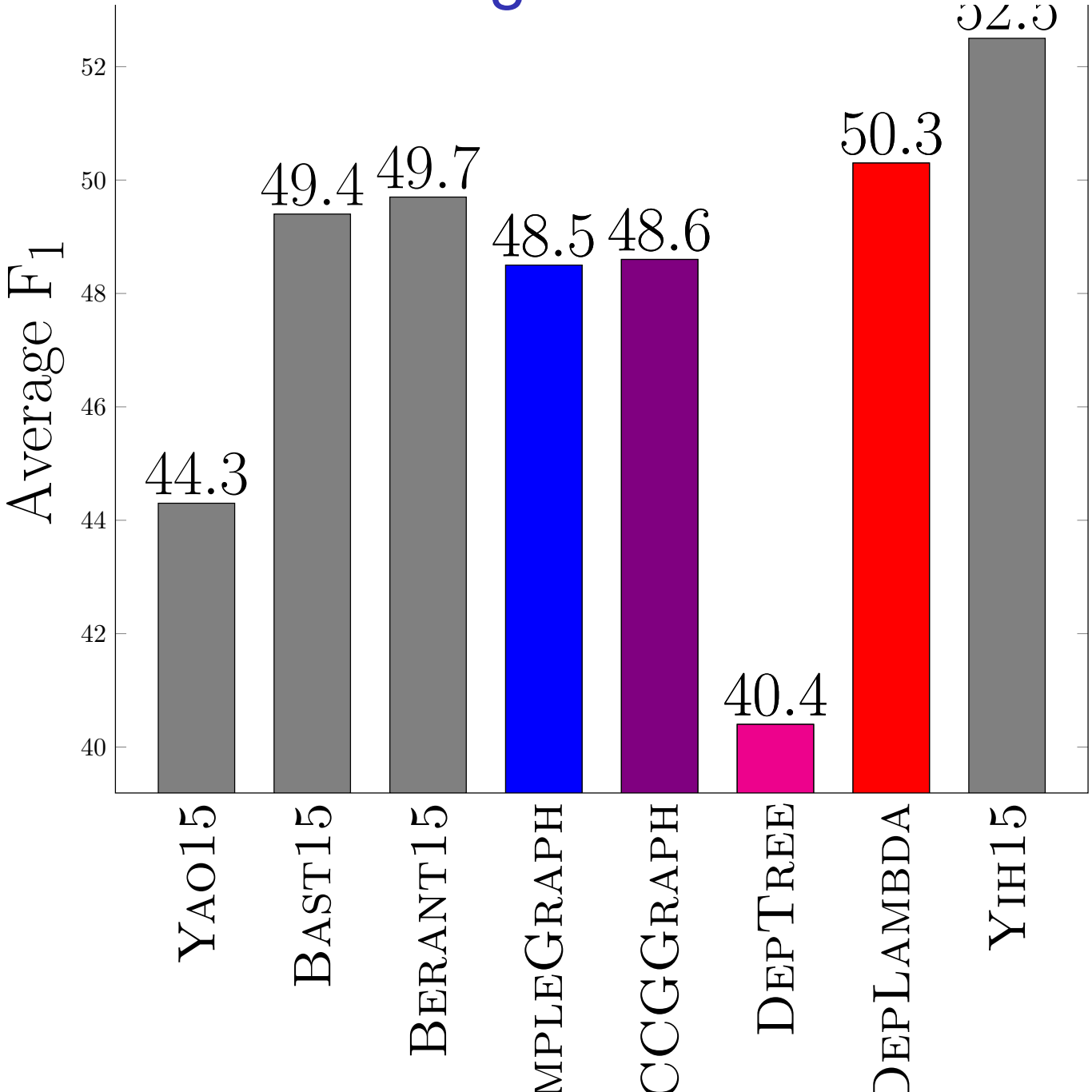
Results on WebQuestions



Results on WebQuestions



Results on Multilingual WebQuestions



Error Analysis

CCGGRAPH fails to produce logical forms for 4.5%.

- Sensitive to grammatical errors
- e.g., *what nestle owns?*

DepLambda fails only for 0.9%

- e.g., *what to do washington december*

Semantic Parsing without QA pairs

Question

Who is the director of Titanic?

Logical Form

$\lambda x.\ \text{film.directed\_by}(\textit{Titanic}, x)$

Latent

Answer

{James Cameron}

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
Jack Dawson

Kate Winslet
Rose DeWitt Bukater

Billy Zane
Caledon Hockley

Gloria Stuart
Rose DeWitt Bukater

Kathy Bates
Molly Brown

Semantic Parsing without QA pairs

Question

Who is the director of Titanic?

Logical Form

$\lambda x.\ \text{film.directed\_by}(Titanic, x)$

Latent

Answer

{James Cameron}

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
Jack Dawson

Kate Winslet
Rose DeWitt Bukater

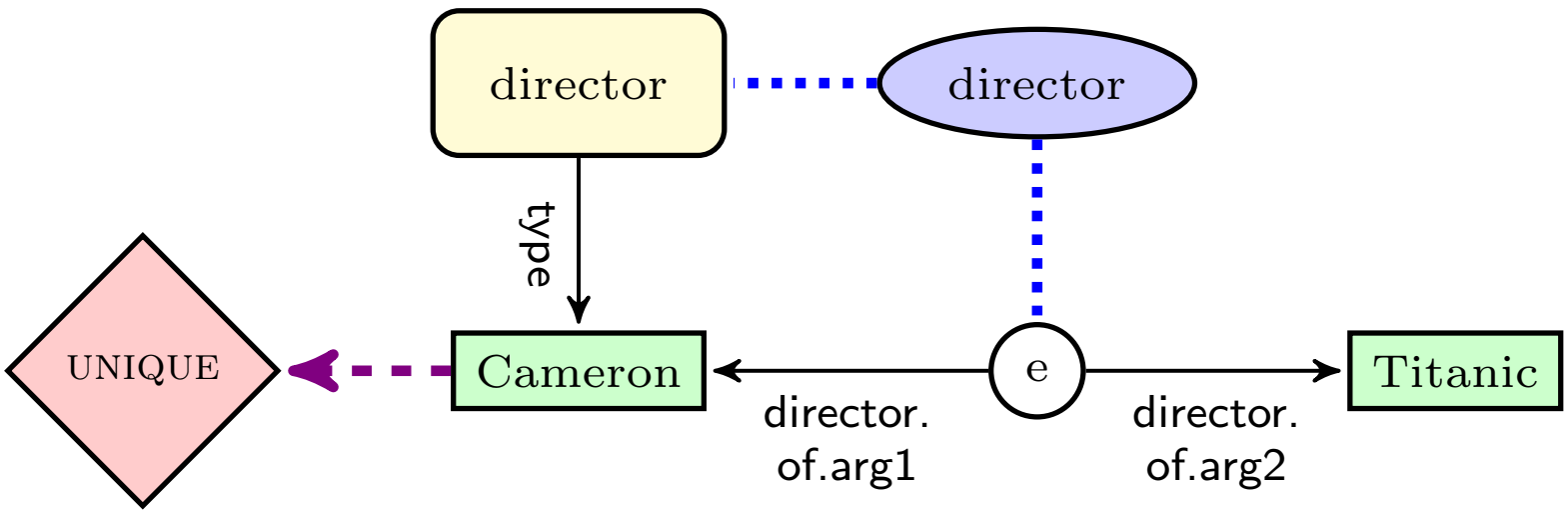
Billy Zane
Caledon Hockley

Gloria Stuart
Rose DeWitt Bukater

Kathy Bates
Molly Brown

Learning from Text

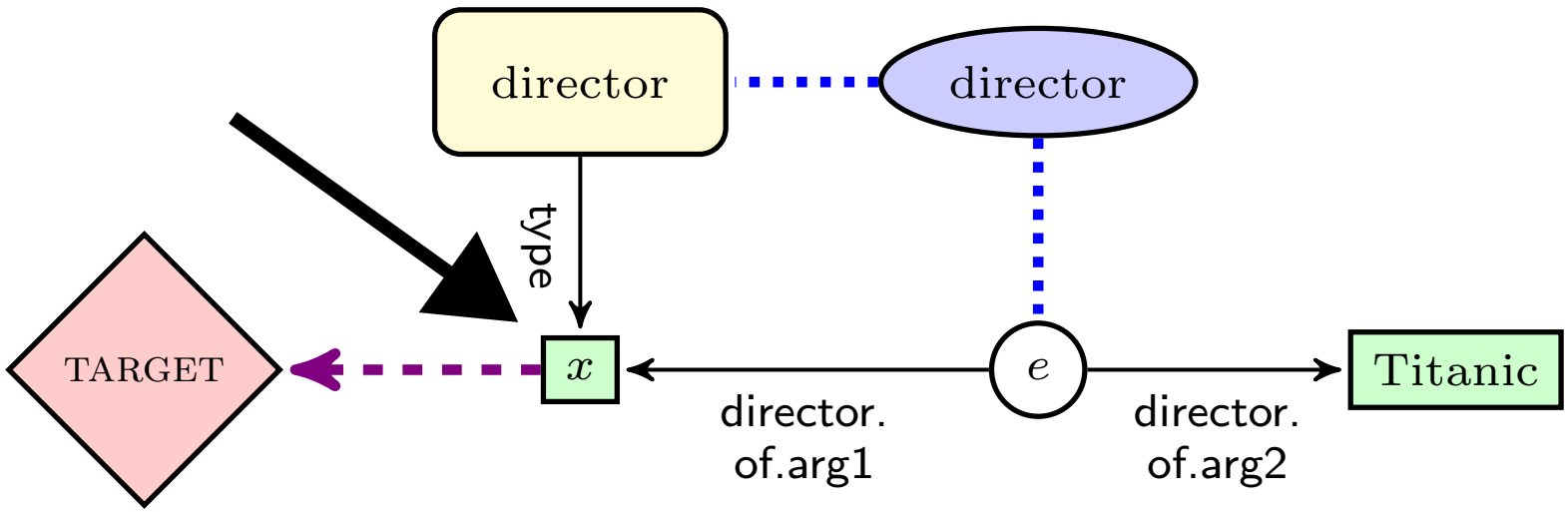
Cameron is the director of Titanic.



$$\text{UNIQUE}(\text{Cameron}) \wedge \text{director}(\text{Cameron}) \wedge$$
$$\text{director.of.arg1}(e, \text{Cameron}) \wedge \text{director.of.arg2}(e, \text{Titanic})$$

Learning from Text

Cameron is the director of Titanic.

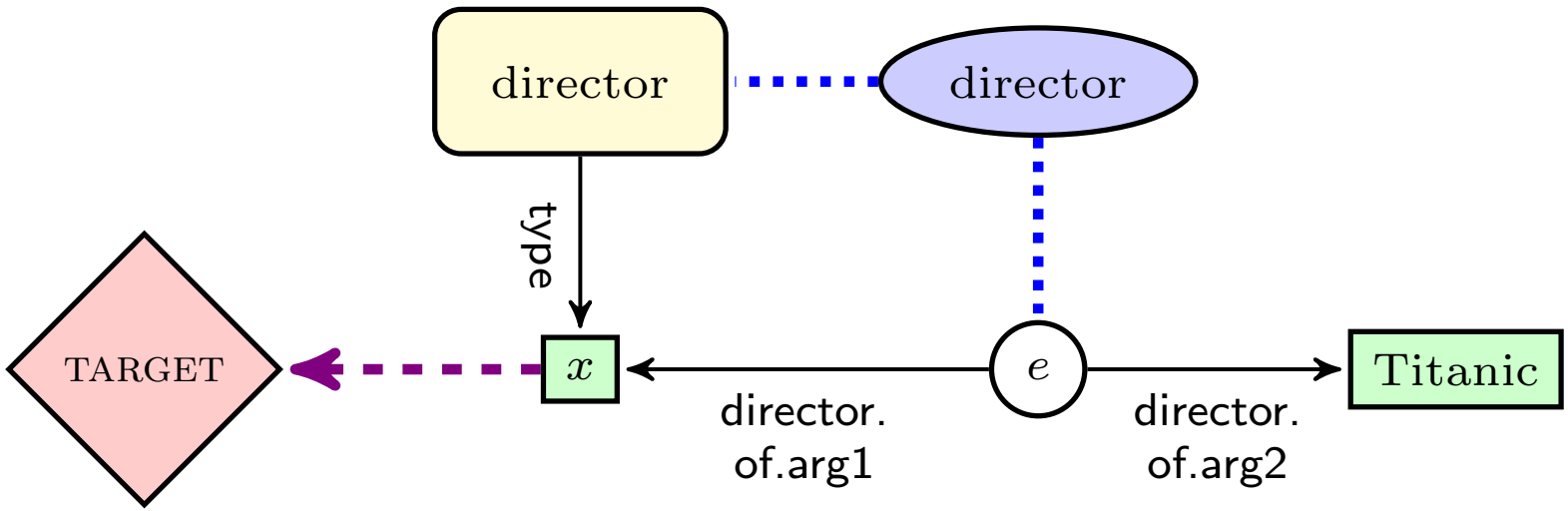


$$\text{TARGET}(x) \wedge \text{director}(x) \wedge \text{director.of.arg1}(e, x) \wedge \text{director.of.arg2}(e, \text{Titanic})$$

Select one of the entities as target and replace it with x (here *Cameron*).

Learning from Text

Who is the director of Titanic?

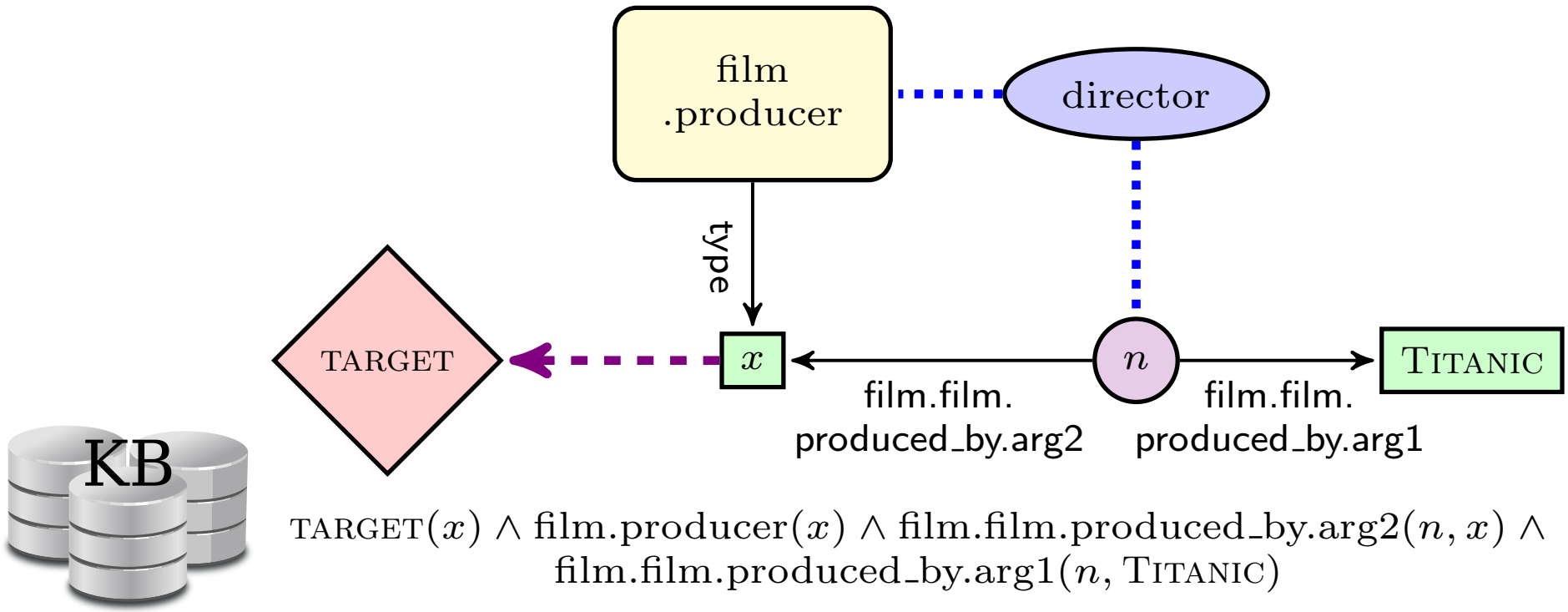


$$\text{TARGET}(x) \wedge \text{director}(x) \wedge \text{director.of.arg1}(e, x) \wedge \text{director.of.arg2}(e, \text{Titanic})$$

Build all knowledge base subgraphs with x uninstantiated.

Learning from Text

Who is the director of Titanic?

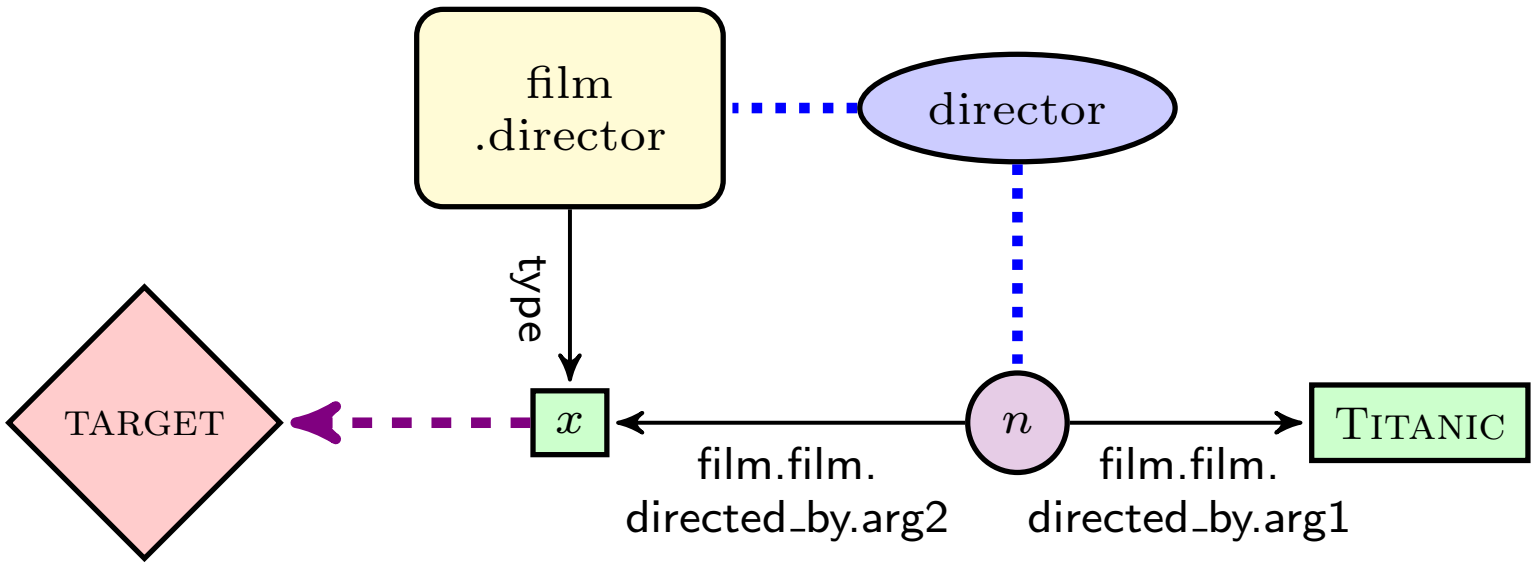


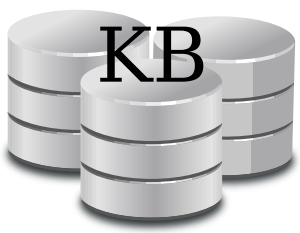
$$\text{TARGET}(x) \wedge \text{film.producer}(x) \wedge \text{film.film.produced\_by.arg2}(n, x) \wedge \text{film.film.produced\_by.arg1}(n, \text{TITANIC})$$

Build all knowledge base subgraphs with x uninstantiated.

Learning from Text

Who is the director of Titanic?



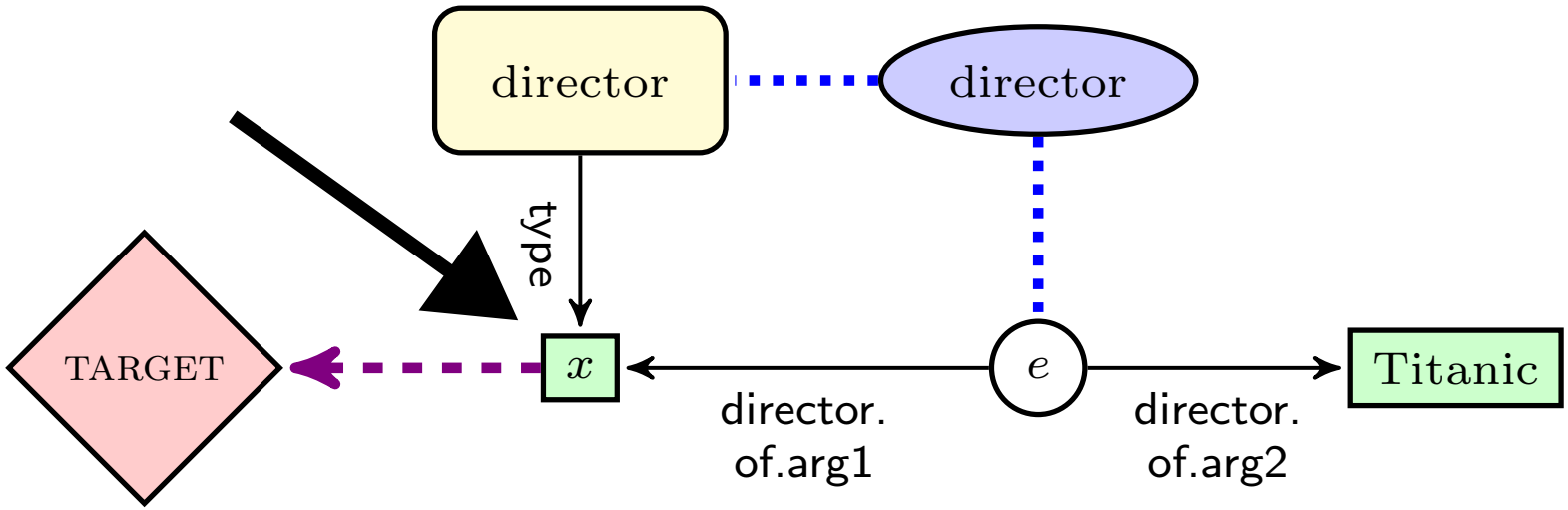


$$\text{TARGET}(x) \wedge \text{film.director}(x) \wedge \text{film.film.directed\_by.arg2}(n, x) \wedge \text{film.film.directed\_by.arg1}(n, \text{TITANIC})$$

Build all knowledge base subgraphs with x uninstantiated.

Learning from Text

Cameron is the director of Titanic.



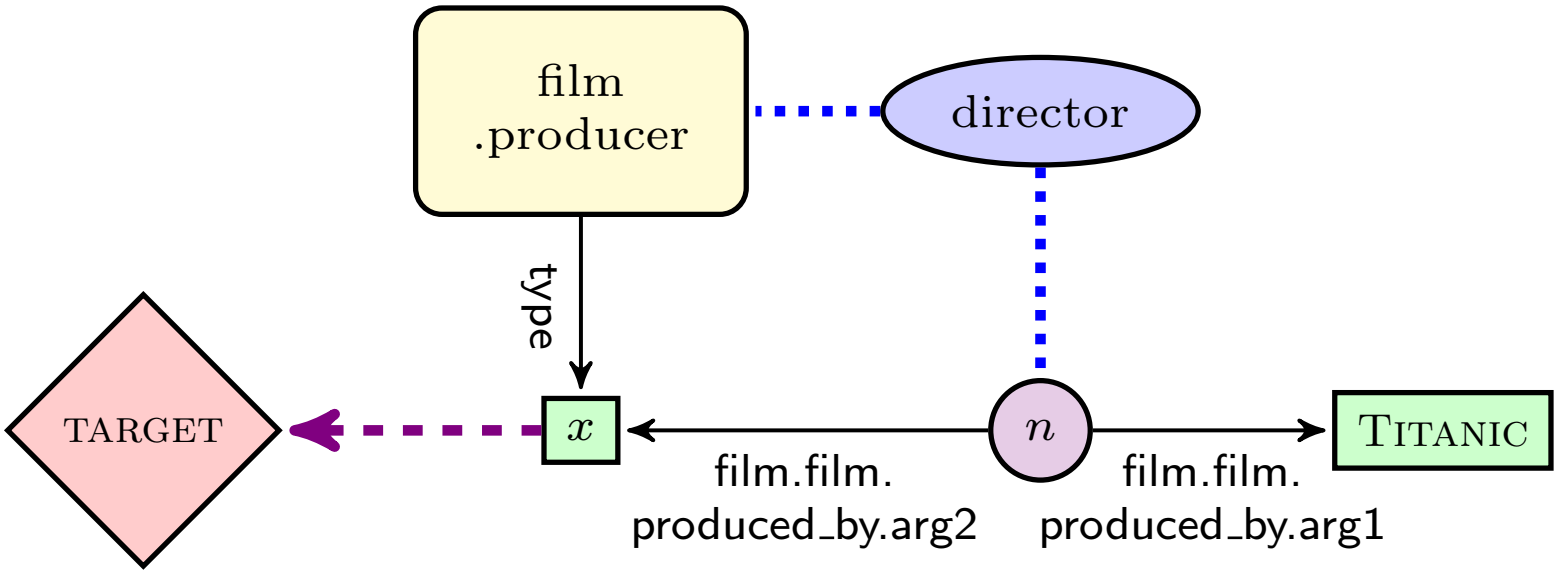
$$\text{TARGET}(x) \wedge \text{director}(x) \wedge \text{director.of.arg1}(e, x) \wedge \text{director.of.arg2}(e, \text{Titanic})$$

$$[\![x]\!]_{NL} = \{\text{CAMERON}\}$$

Use denotations in NL and Freebase to select a surrogate gold graph.

Learning from Text

Who is the director of Titanic?



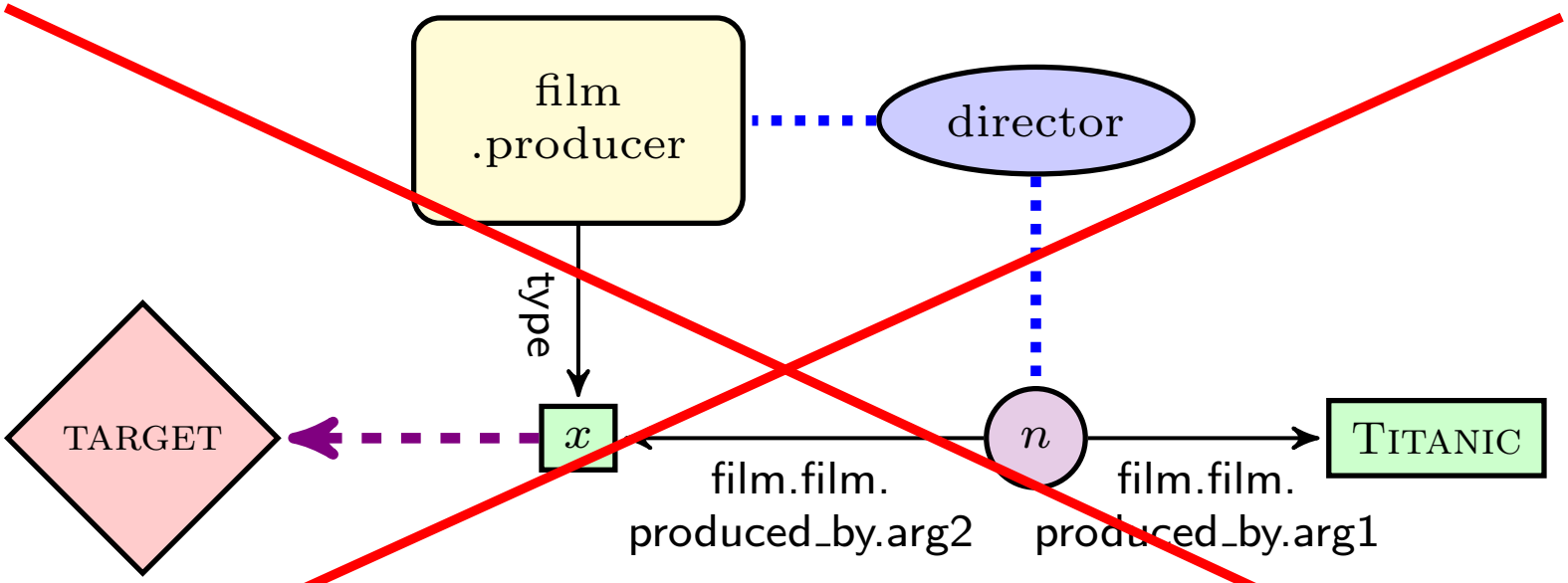
$$\text{TARGET}(x) \wedge \text{film.producer}(x) \wedge \text{film.film.produced\_by.arg2}(n, x) \wedge \text{film.film.produced\_by.arg1}(n, \text{TITANIC})$$

$$[\![x]\!]_{KB} = \{\text{CAMERON}, \text{JONLANDAU}\}$$

Use denotations in NL and Freebase to select a surrogate gold graph.

Learning from Text

Who is the director of Titanic?



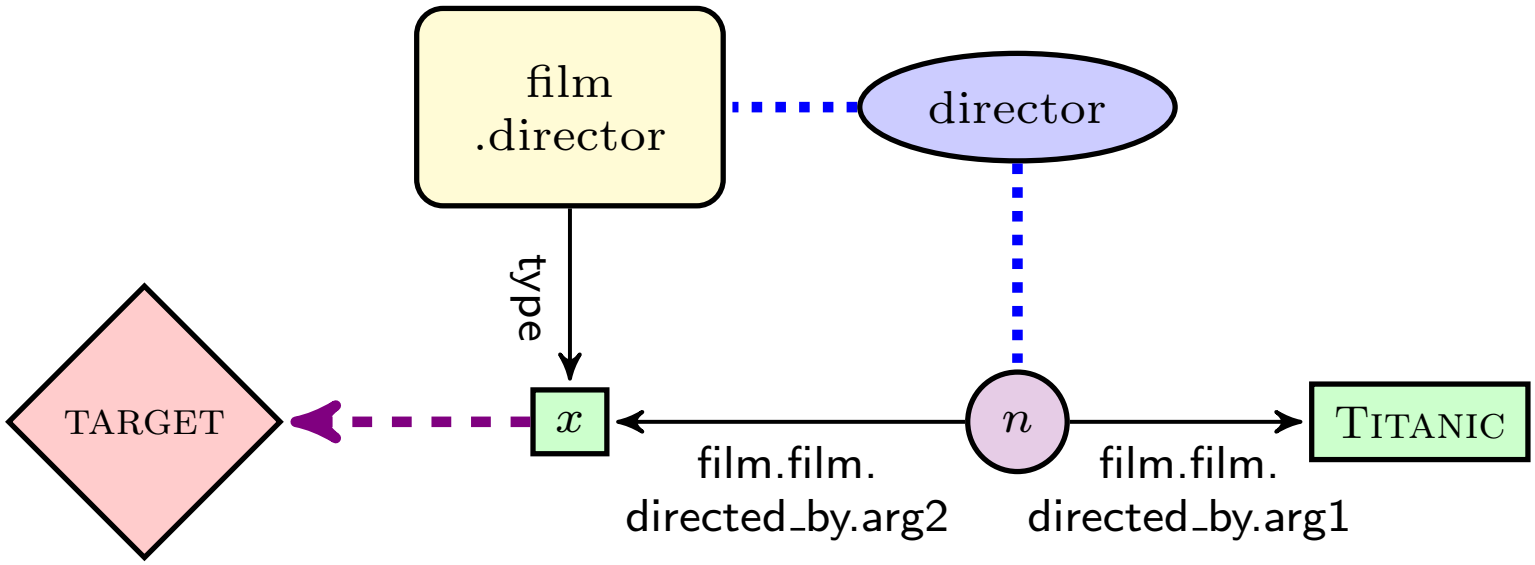
$$\text{TARGET}(x) \wedge \text{film.producer}(x) \wedge \text{film.film.produced\_by.arg2}(n, x) \wedge \text{film.film.produced\_by.arg1}(n, \text{TITANIC})$$

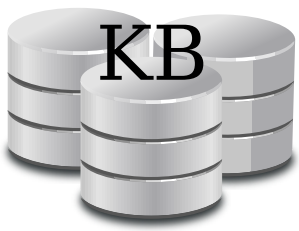
$$[\![x]\!]_{KB} = \{\text{CAMERON}, \text{JONLANDAU}\} \neq [\![x]\!]_{NL}$$

Use denotations in NL and Freebase to select a surrogate gold graph.

Learning from Text

Who is the director of Titanic?





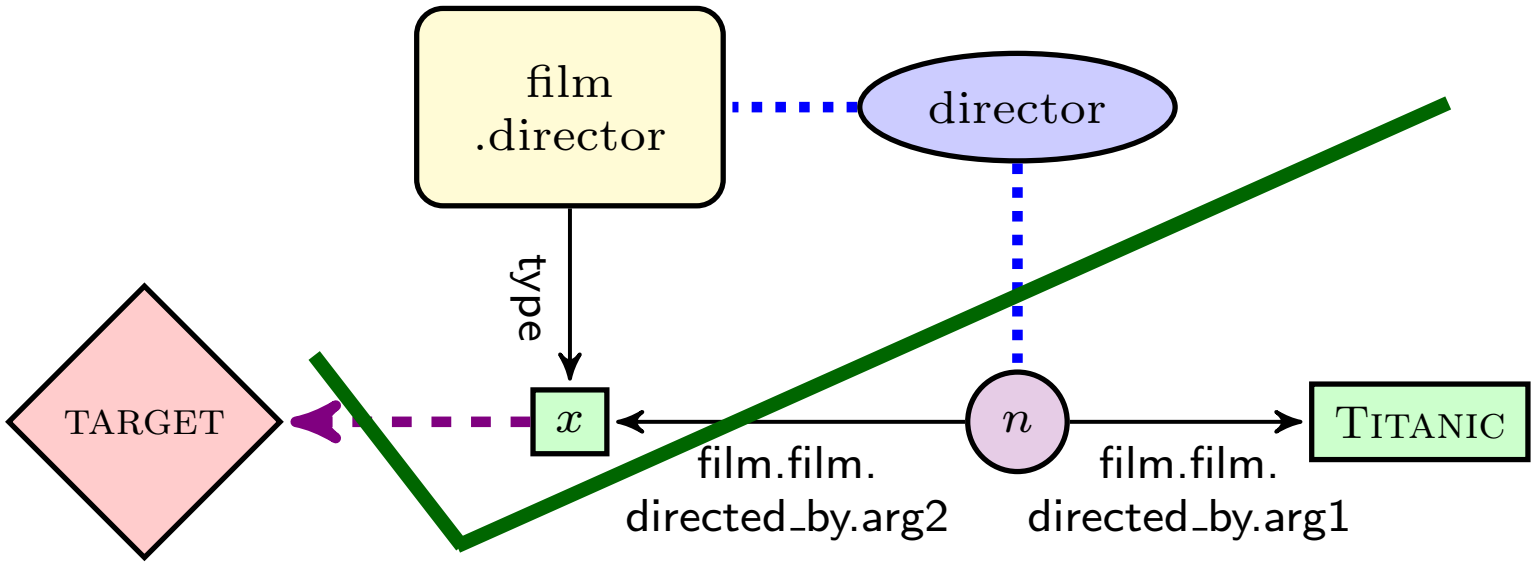
$$\text{TARGET}(x) \wedge \text{film.director}(x) \wedge \text{film.film.directed\_by.arg2}(n, x) \wedge \text{film.film.directed\_by.arg1}(n, \text{TITANIC})$$

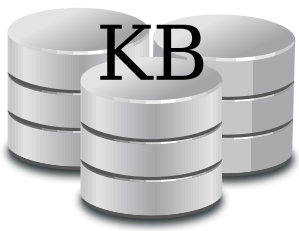
$$[\![x]\!]_{KB} = \{\text{CAMERON}\}$$

Use denotations in NL and Freebase to select a surrogate gold graph.

Learning from Text

Who is the director of Titanic?





$$\text{TARGET}(x) \wedge \text{film.director}(x) \wedge \text{film.film.directed\_by.arg2}(n, x) \wedge \text{film.film.directed\_by.arg1}(n, \text{TITANIC})$$

$$[\![x]\!]_{KB} = \{\text{CAMERON}\} = [\![x]\!]_{NL}$$

Use denotations in NL and Freebase to select a surrogate gold graph.

Learning from Text: Summary

Learning proceeds by creating question-like graphs by replacing named entities in logical forms mined from web text with a variable.

Then try to find the subgraph of the knowledge graph with the the most similar denotation.

Experiments

Target Domains: Business, Film, People

- Largest domains of Freebase
- 120m triples, 411 relations and 210 types.

Training data: 99K sentences from ClueWeb09

Experiments

Test data:

- Free917 [Cai and Yates, 2013]
  - Syntactically well-formed queries
  - 124 queries for our target domains
- WebQuestions [Berant et al., 2013]
  - Google search queries starting with *wh* question words.
  - 570 queries for our target domains

Results on Free917

System	Precision	Recall	F1	
MWG	52.6	49.1	50.8	
KCAZ13	72.6	66.1	69.2	
This Work	**81.9**	**76.6**	**79.2**	**+10.0**

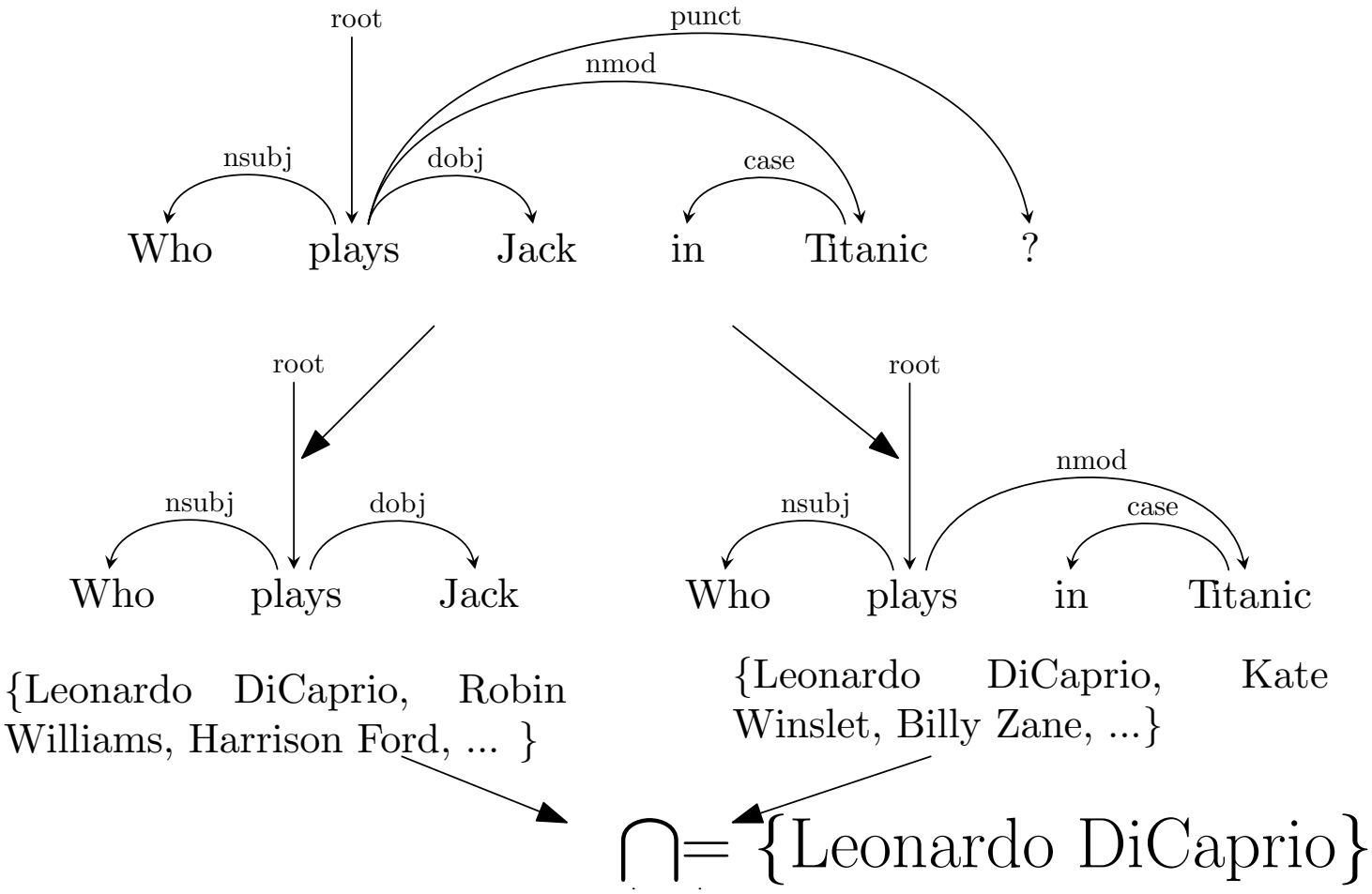
- MWG: Greedy Maximum Weighted Graph
- KCAZ13: Kwiatkowski et al. 2013
  - Manually annotated QA pairs (612 pairs)
  - Wiktionary

Results on WebQuestions

System	Precision	Recall	F1	$AvgF1$ (BL14)	
MWG	39.4	34.0	36.5	41.6	
BL14	—	—	—	37.5	
This Work	**41.9**	**37.0**	**39.3**	**47.7**	**+10.2**

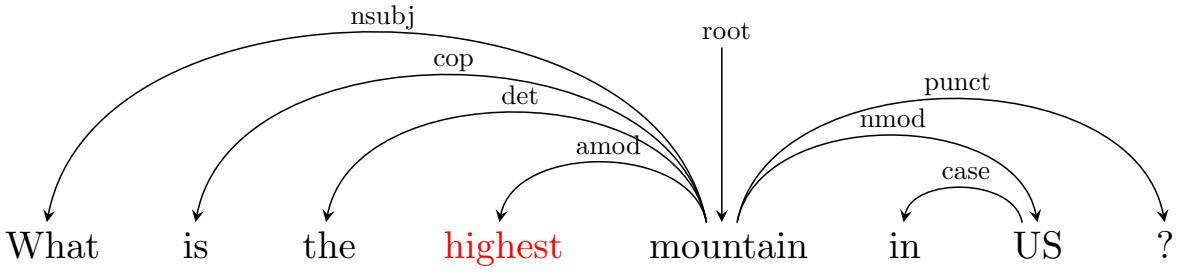
- MWG: Greedy Maximum Weighted Graph
- BL14: Berant and Liang, 2014
  - ClueWeb09
  - Manually annotated QA pairs (1115 pairs)
  - Paraphrases

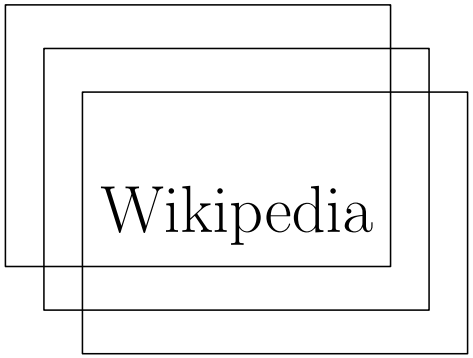
Semantic Parsing with Natural Logic and IE

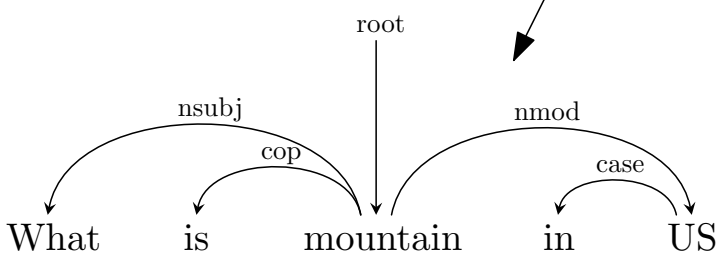


Kun, Reddy, Feng et al. (ACL 2016)

Natural-Logic styled Semantic Parsing







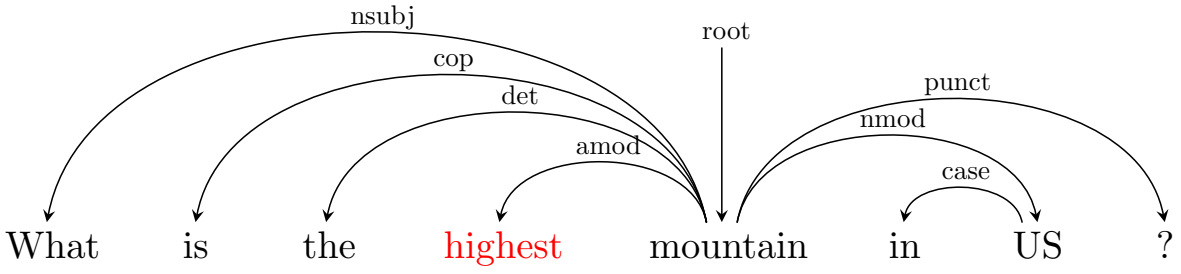
{Rainer, Whitney, Denali, Elbert, Saint Elias, Longs Peak, Pikes Peak, ...}

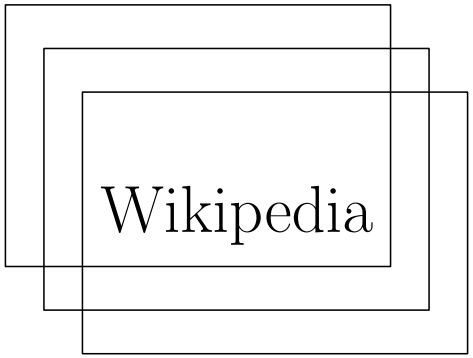
Denali (also known as Mount McKinley, its former official name) is the highest mountain peak in North America, with a summit elevation of 20,310 feet (6,190 m) above sea level

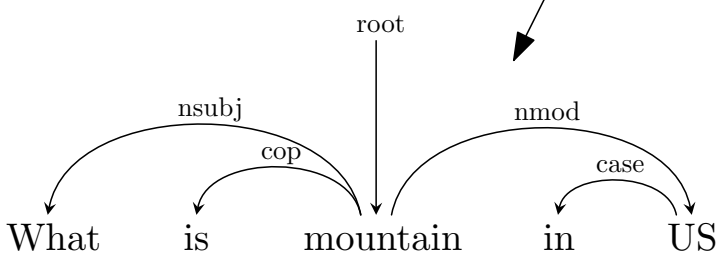
SVM

{Denali}

Natural-Logic styled Semantic Parsing







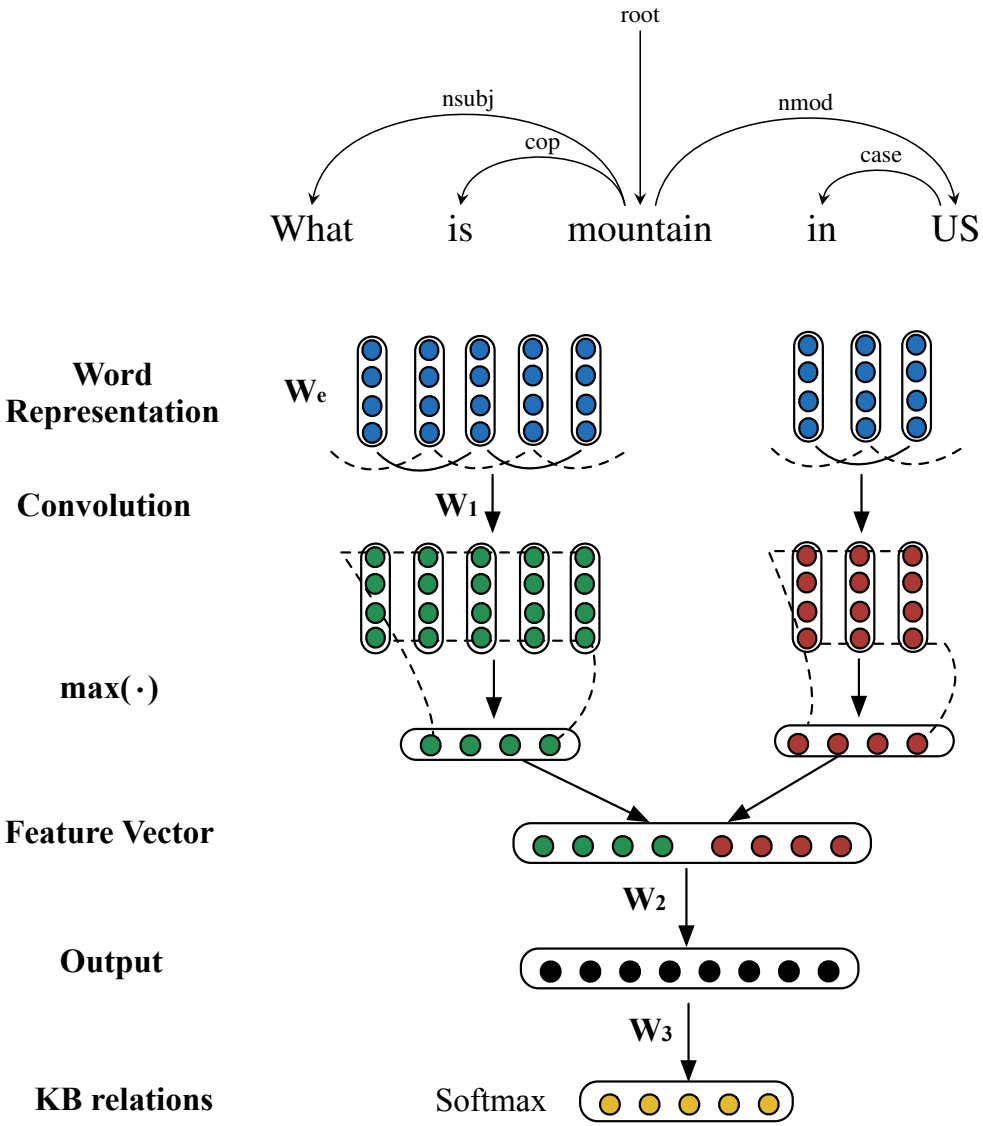
{Rainer, Whitney, Denali, Elbert, Saint Elias, Longs Peak, Pikes Peak, ...}

Denali (also known as Mount McKinley, its former official name) is the highest mountain peak in North America, with a summit elevation of 20,310 feet (6,190 m) above sea level

SVM

{Denali}

Natural-Logic styled Semantic Parsing



Summary

- Lambda Calculus for converting Dependencies to Logical Forms
- Semantic parsing as Graph Transduction
- Semantic Parsing with Natural Logic and IE

Summary

- Lambda Calculus for converting Dependencies to Logical Forms
- Semantic parsing as Graph Transduction
- Semantic Parsing with Natural Logic and IE

Demo at
https://sivareddy.in/deplambda.html

Learning Model

Structured Perceptron: Ranks a pair of grounded and ungrounded graph

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

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

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.

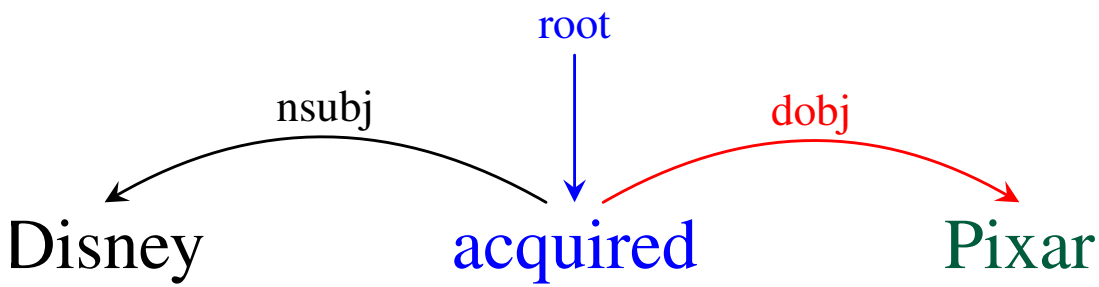
Surrogate Gold Graph:

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

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

Dependencies to Logical Forms

Single Type System

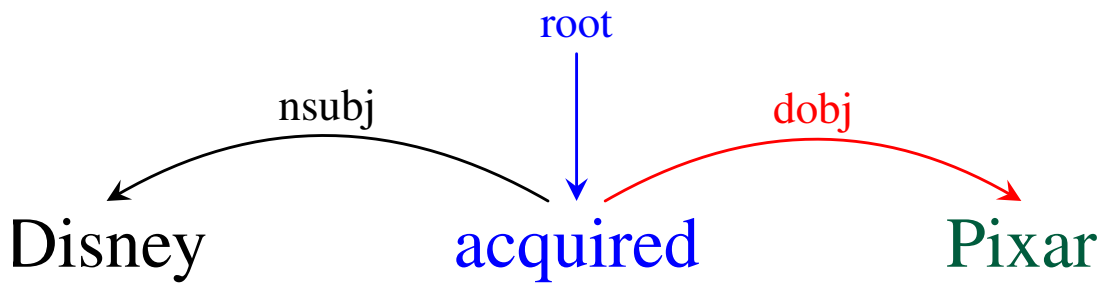


All constituents are of the same lambda expression type

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

Dependencies to Logical Forms

Single Type System

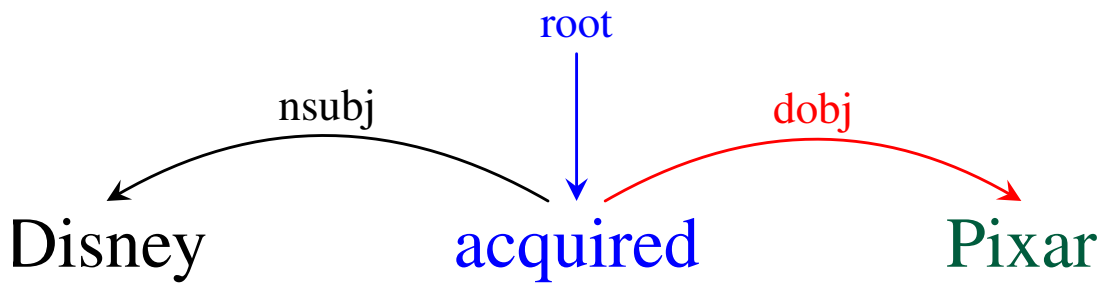


All **words** have a *lambda expression* of type η

- **TYPE**[acquired] = η
- **TYPE**[Pixar] = η

Dependencies to Logical Forms

Single Type System

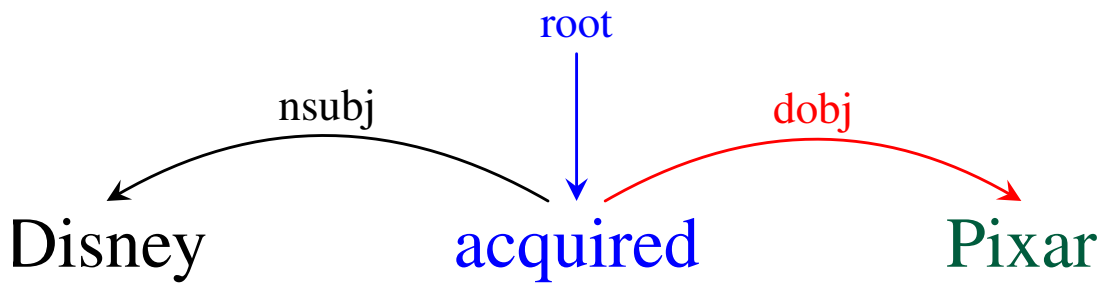


All constituents have a *lambda expression* of type η

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

Dependencies to Logical Forms

Single Type System



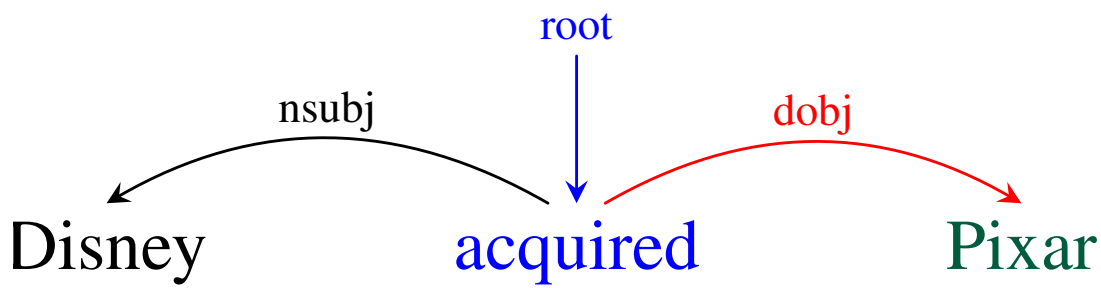
All constituents have a *lambda expression* of type η

- **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



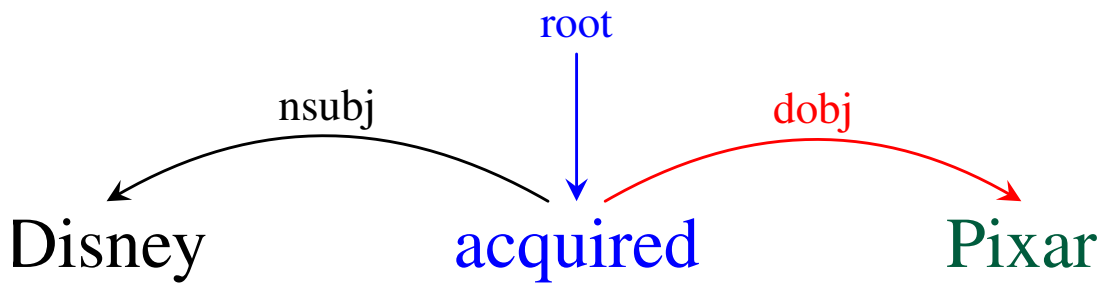
Lambda Expression for words

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

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

Dependencies to Logical Forms

Lambda Calculus for Single Type System



Lambda Expression for words

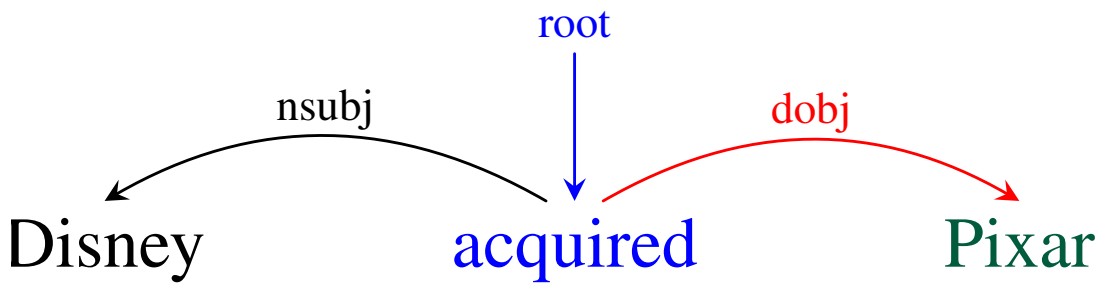
$$\text{acquired} \Rightarrow \lambda x_e.\,\text{acquired}(x_e) \quad \Rightarrow \textbf{TYPE} = \textbf{Event} \rightarrow \textbf{Bool}$$

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

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

Dependencies to Logical Forms

Lambda Calculus for Single Type System



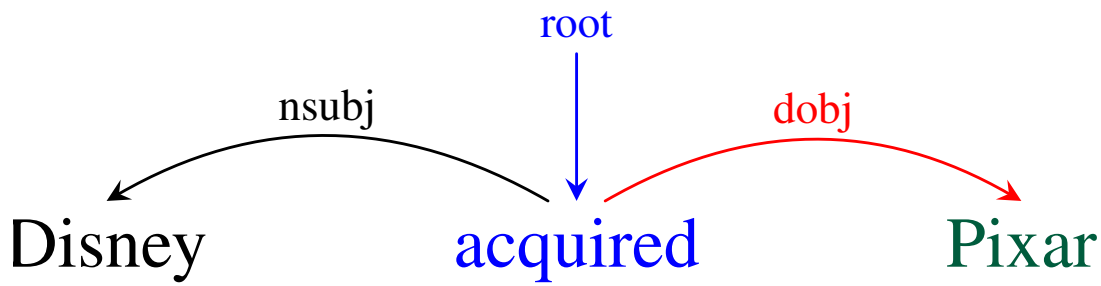
Lambda Expression for words

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

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

Dependencies to Logical Forms

Lambda Calculus for Single Type System



Lambda Expression for words

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

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

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

Lambda calculus for "*Single Type*" System

More dependency labels

- (appos Disney the_company)

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

This function unifies two nodes

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)

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

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)

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

This function creates a struct with two variables

Lambda calculus for "*Single Type*" System

More dependency labels

- (conj Disney_and Pixar)

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

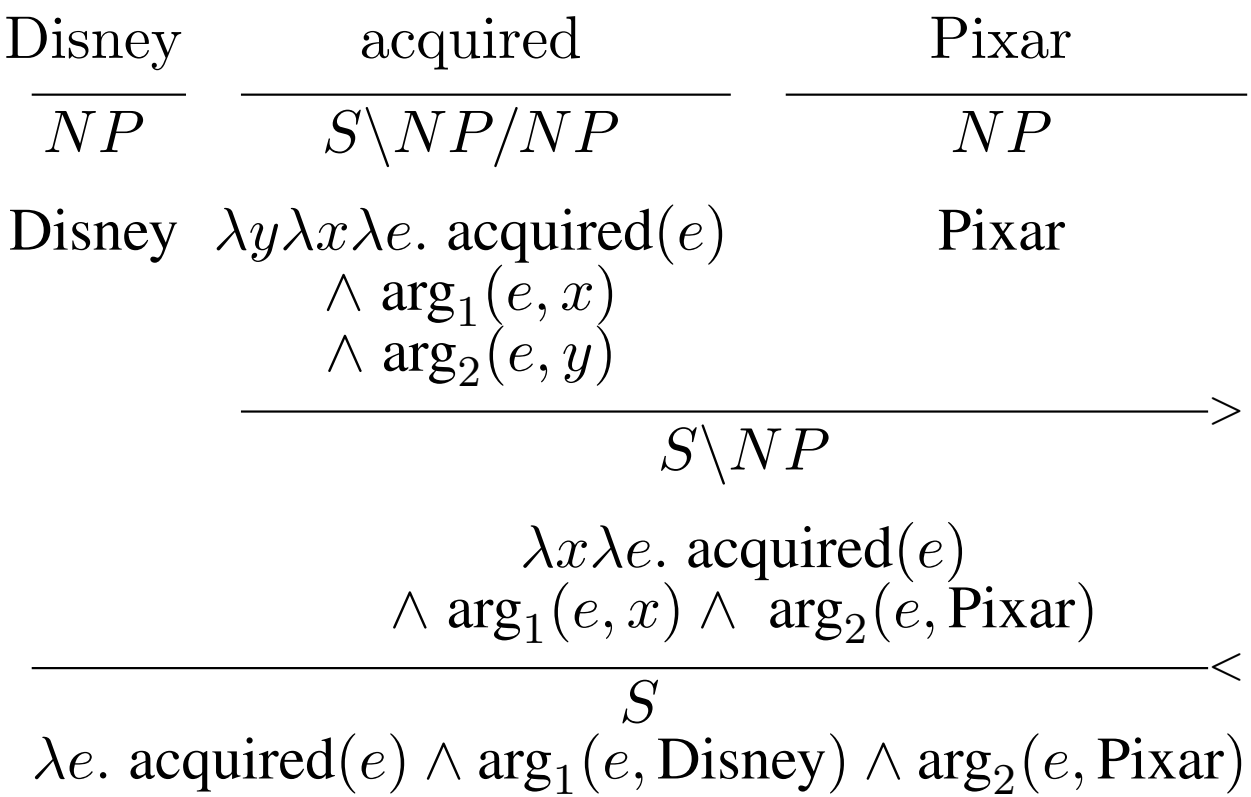
This function creates a struct with two variables

- (rcmod Disney which_acquired_Pixar)

$$\begin{array}{l}(\text{rcmod Disney} \\ \quad (\text{wh-dobj } (\textsc{bind}\ f\ (\text{nsubj } (\text{dobj acquired } f)\ \text{Pixar})) \\ \quad\quad \text{which}))\end{array}$$

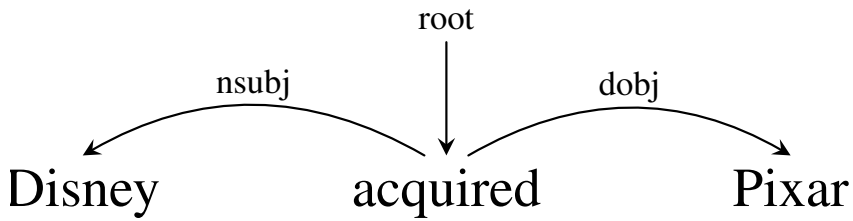
CCG to Logical Forms

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



Dependencies to Logical Forms

Challenges

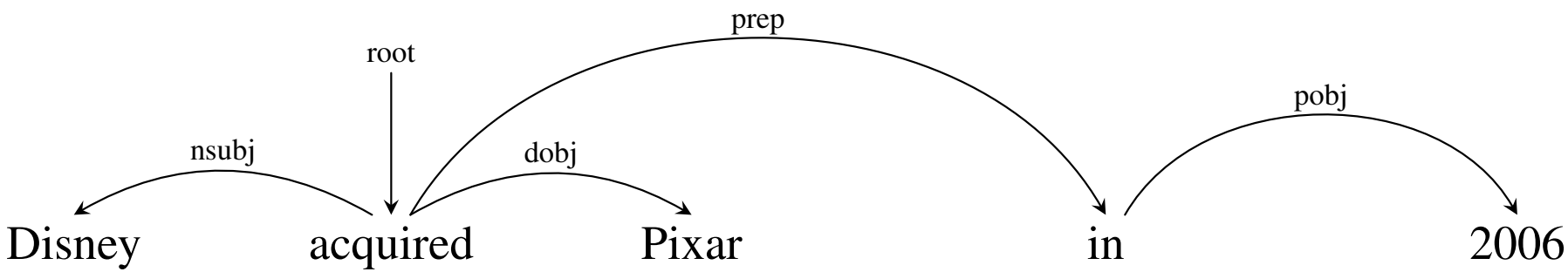


The obvious idea

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

Dependencies to Logical Forms

Challenges



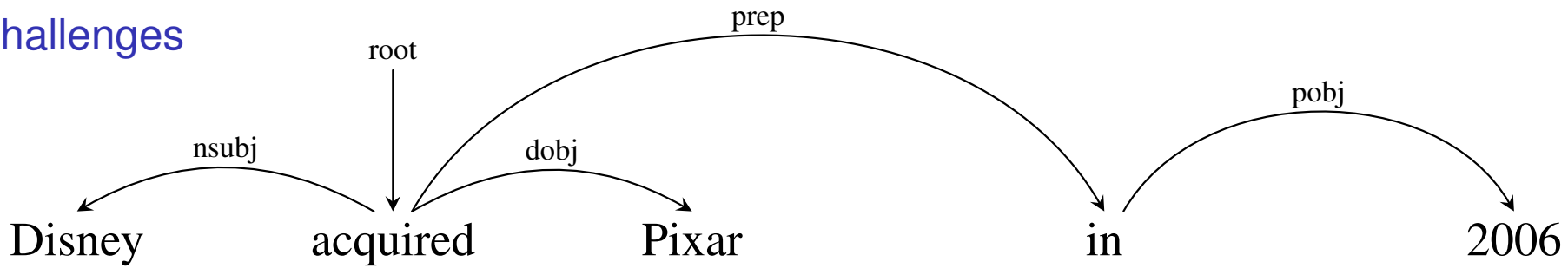
The obvious idea

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

But, what about?

Dependencies to Logical Forms

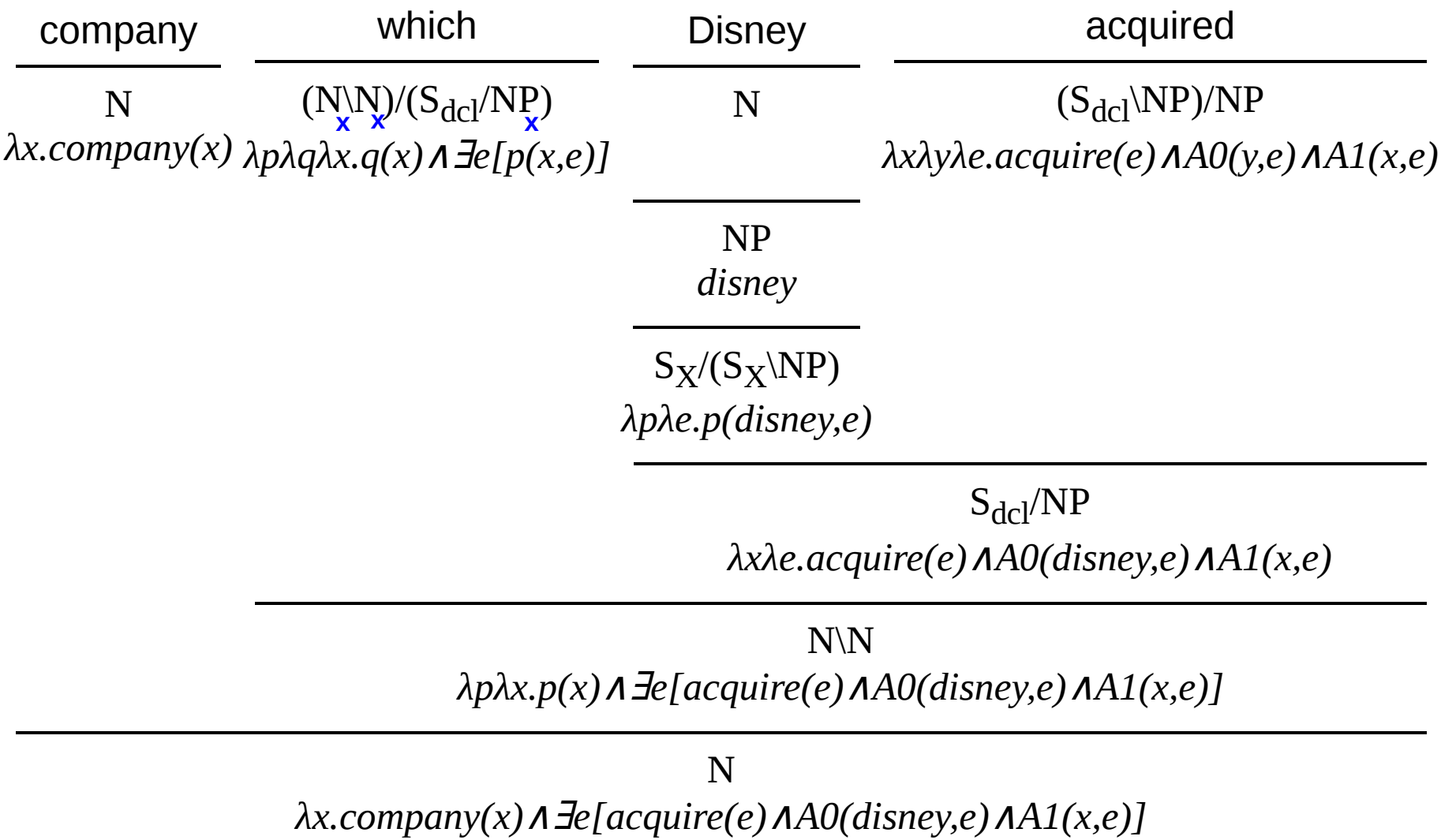
Challenges



Problems

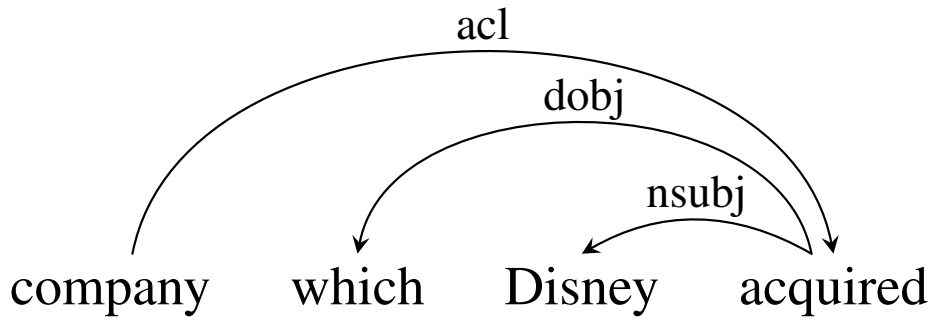
1. Rules $\propto$ dependency label permutations
2. Complex lexical semantics
3. Highly sensitive to parse errors
4. Prone to type collisions

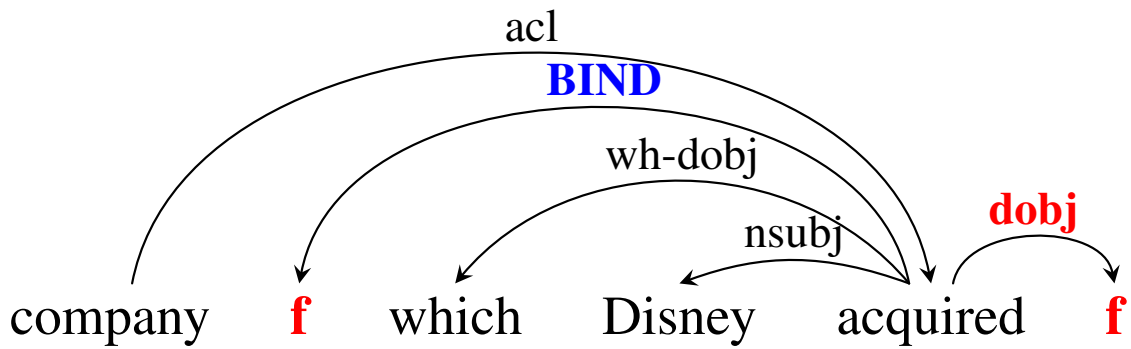
Relative Clause in CCG



Relative Clause in DepLambda

following Carpenter (1998)





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)

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)

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:

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

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:

(rcmod Apple
(wh-dobj (BIND f (nsubj (dobj founded f) Jobs))
which))

Relative Clause

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

Sentence:

Apple which Jobs founded

Binarized tree:

$$\begin{array}{l}(\text{rcmod Apple}\\ \quad (\text{wh-dobj } (\textsc{bind}\ f\ (\text{nsubj } (\text{dobj founded } f)\ \text{Jobs}))\\ \quad\quad \text{which}))\end{array}$$

Substitution:

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

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

Logical Expression:

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

Expressivity

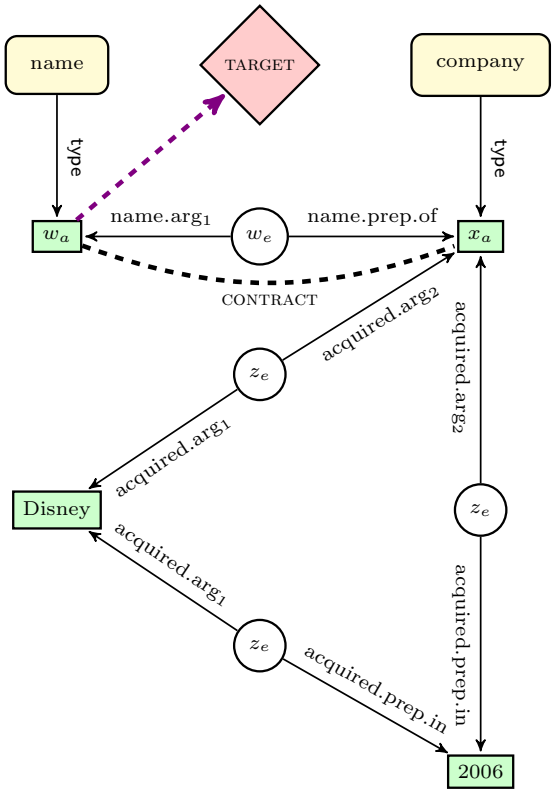
How isomorphic are the representations compared to Knowledge Graph?
Average Oracle F_1 Table.

Search Space

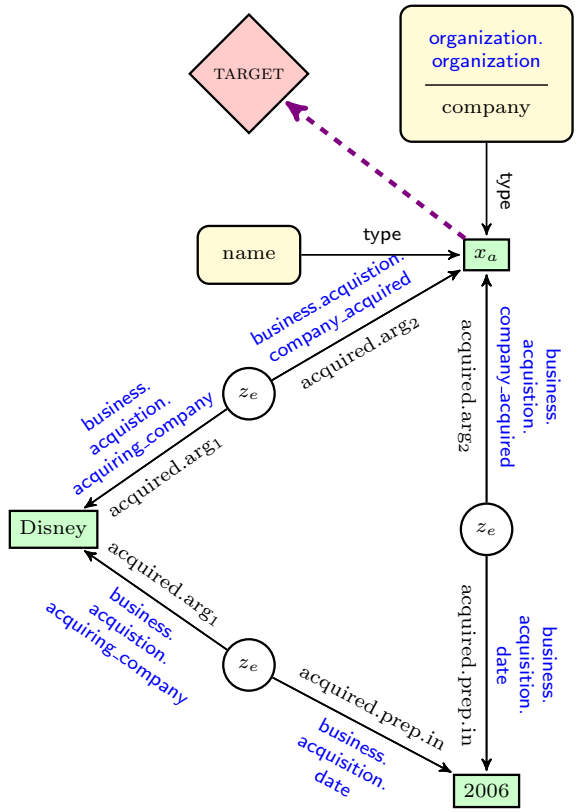
How many ways to reach an answer?
Average Oracle F_1 Table.

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

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