Universal Semantic Parsing

Siva Reddy
Stanford University


Oscar Täckström


Slav Petrov


Mark Steedman


Mirella Lapata

Google and University of Edinburgh

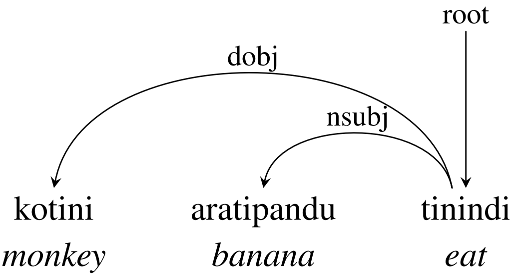
Dependency Trees help Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Dependency Trees help Semantics

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Dependency Trees help Semantics



Dependency Trees help Semantics

root

dobj

nsubj

kotini	aratipandu	tinindi
monkey	*banana*	*eat*

Universal Dependencies

Common syntactic representation in 50+ languages

Manning laws:

- Satisfactory linguistic analysis
- Easy to comprehend (e.g., 40 labels)
- Rapid and consistent annotations
- High accuracy parsing [Dozat et al. 2017]

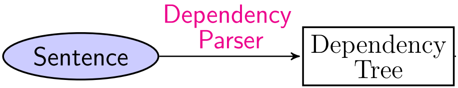
Dependency Tree to Semantics

Dependencies lack a formal theory of semantics

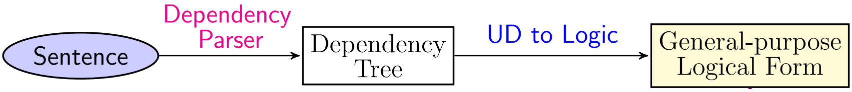
This Talk

Universal Semantic Parsing:
Language-agnostic conversion of
Universal Dependencies to Logical Forms

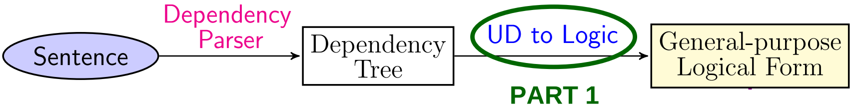
This Talk



This Talk

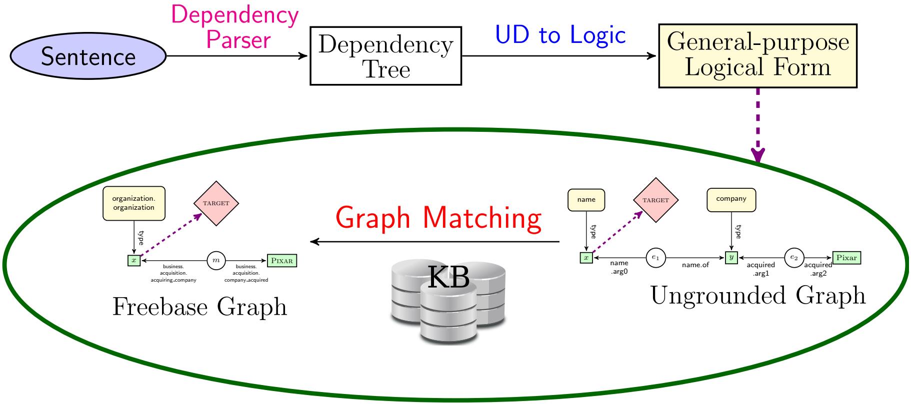

Sentence
Dependency Parser
Dependency Tree
UD to Logic
General-purpose Logical Form

This Talk



Sentence
Dependency
Parser
Dependency
Tree
UD to Logic
PART 1
General-purpose
Logical Form

This Talk



PART 2: Freebase QA

This Talk: Contributions

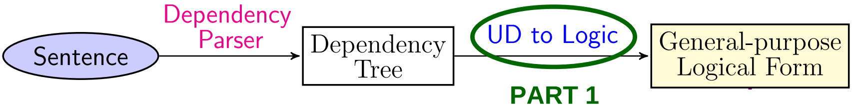
Universal Dependencies to **general-purpose** logical forms

A general solution that also works for **Dependency Graphs**

Multilingual evaluation of logical forms on **Freebase QA**

WebQuestions and GraphQuestions QA datasets in **German** and **Spanish**

Part 1: Universal Semantic Parsing



Dependency Tree to Semantics

Principle of Compositionality: the semantics of a complex expression is determined by the semantics of its constituent expressions and the rules used to combine them

Dependency Tree to Semantics

Principle of Compositionality: the semantics of a complex expression is determined by the semantics of its constituent expressions and the rules used to combine them

Complex expression is the dependency tree

Dependency Tree to Semantics

Principle of Compositionality: the semantics of a complex expression is determined by the semantics of its constituent expressions and the rules used to combine them

Complex expression is the dependency tree

Constituent expressions are subtrees

Dependency Tree to Semantics

Principle of Compositionality: the semantics of a complex expression is determined by the semantics of its constituent expressions and the rules used to combine them

Complex expression is the dependency tree

Constituent expressions are subtrees

Rules are the dependency labels

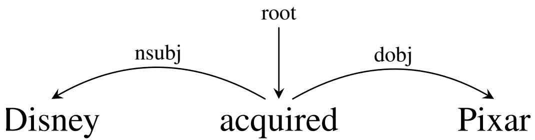
Universal Semantic Parsing: Objectives

Logical form must be built

1. **compositionally** from the dependency tree

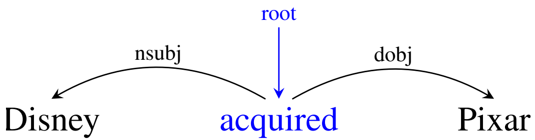
2. in a **language-agnostic** manner
   - Dependency labels and postags dictate the semantics, **not** the words

Compositional



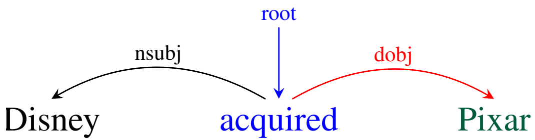
$$\lambda z.\exists xy.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{Disney}(x_a) \wedge \\ \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)$$

Compositional



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

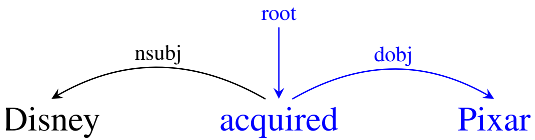
Compositional



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

(dobj acquired Pixar)

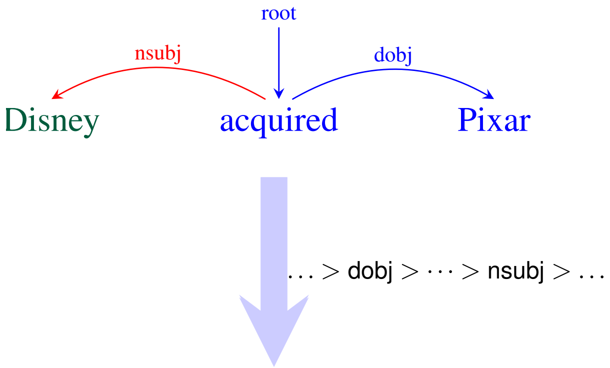
Compositional



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

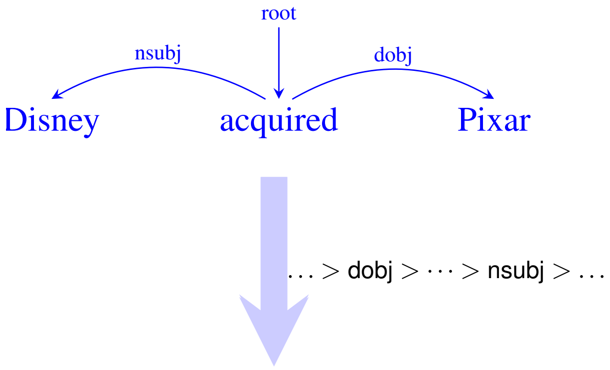
(dobj acquired Pixar)

Compositional



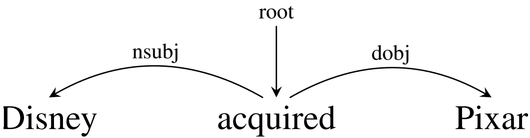
(nsubj (dobj acquired Pixar) Disney)

Compositional



(nsubj (dobj acquired Pixar) Disney)

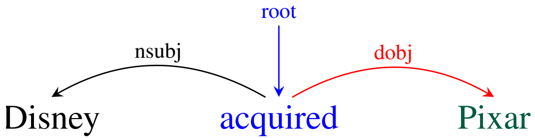
Compositional



(nsubj (dobj acquired Pixar) Disney)

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

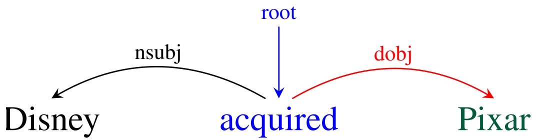
Language-agnostic Conversion



Lambda Expression for words

$$VERB \Rightarrow \lambda x.\mathsf{word}(x_e)$$
$$PROPN \Rightarrow \lambda x.\mathsf{word}(x_a)$$

Language-agnostic Conversion

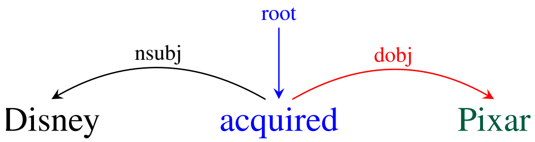


Lambda Expression for words

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

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

Language-agnostic Conversion

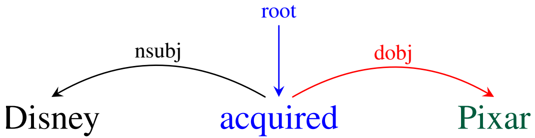


Lambda Expression for dependency labels

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

Dependencies to Logical Forms

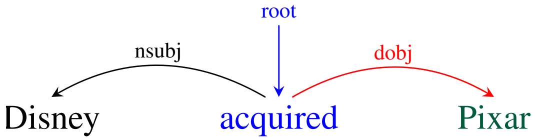
Composition



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

Dependencies to Logical Forms

Composition



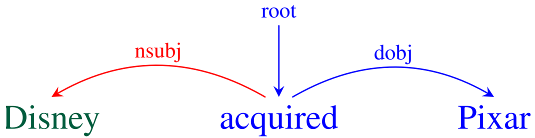
$$
\begin{array}{ccc}
(\textbf{dobj} & \textbf{acquired} & \textbf{Pixar}) \\
\lambda f \lambda g \lambda z . \exists y . & \lambda z . \text{acquired}(z_e) & \lambda y . \text{Pixar}(y_a) \\
f(z) \wedge g(y) \wedge & & \\
\text{arg}_2(z_e, y_a) & &
\end{array}
$$

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

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

Dependencies to Logical Forms

Composition

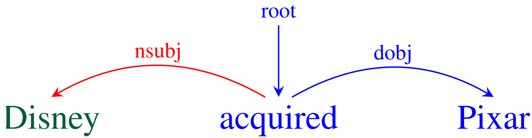


$$(\textbf{nsubj} \quad (\textbf{dobj} \quad \textbf{acquired} \quad \textbf{Pixar}) \quad \textbf{Disney})$$

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

Dependencies to Logical Forms

Composition



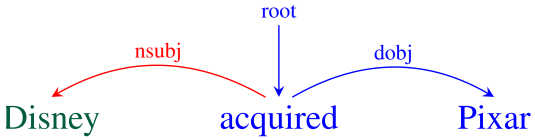
$$
\begin{array}{cc}
(\textbf{nsubj} & (\textbf{dobj}\quad \textbf{acquired}\quad \textbf{Pixar}) \quad\quad \textbf{Disney}) \\
\begin{array}{c}\lambda f \lambda g \lambda z.\ \exists x. \\ f(z) \wedge g(x) \wedge \\ \mathrm{arg}_1(z_e, x_a)\end{array} & \dfrac{}{\begin{array}{c}\lambda z.\ \exists y.\ \mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \\ \wedge\ \mathrm{arg}_2(z_e, y_a)\end{array}} \quad \lambda x.\mathrm{Disney}(x_a)
\end{array}
$$

$$
\overline{\begin{array}{c}\lambda g \lambda z.\exists xy.\mathrm{acquired}(z_e) \wedge \mathrm{Pixar}(y_a) \wedge \mathrm{g}(x) \wedge \\ \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{arg}_2(z_e, y_a)\end{array}}
$$

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

Dependencies to Logical Forms

Composition



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

Dependencies to Logical Forms

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

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

running $\xleftarrow{\text{amod}}$ horse

amod $=$
$$\lambda f \lambda g \lambda x. \exists z. f(x) \wedge g(z) \wedge \mathrm{amod}^{i}(z_e, x_a)$$

UD labels are insufficient in few cases

UD may conflate different semantic phenomenon

- DET could mean a determiner or a question word
  e.g., *what* vs *the*

UD labels are insufficient in few cases

UD may conflate different semantic phenomenon

- DET could mean a determiner or a question word
  e.g., *what* vs *the*

UD does not have long-distance dependencies
e.g., in control constructions

UD labels are insufficient in few cases

UD may conflate different semantic phenomenon

- DET could mean a determiner or a question word
  e.g., *what* vs *the*

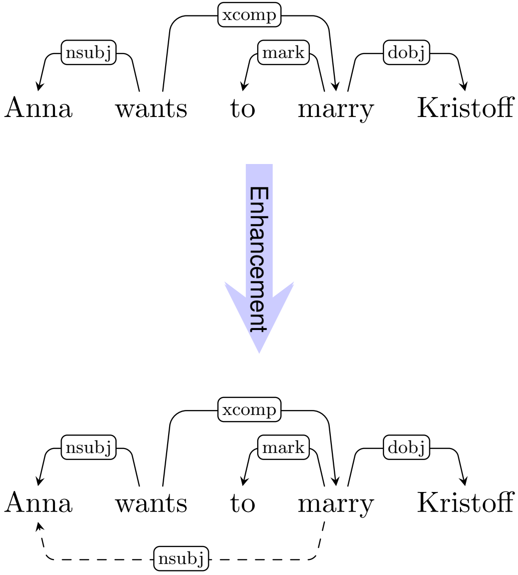
UD does not have long-distance dependencies
e.g., in control constructions

Solution: **Enhancement step**, a lightweight preprocessing

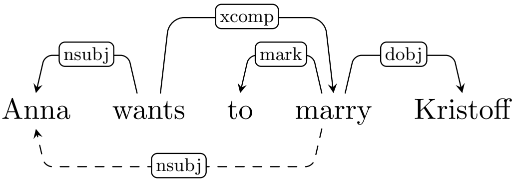
[Schuster and Manning 2016]

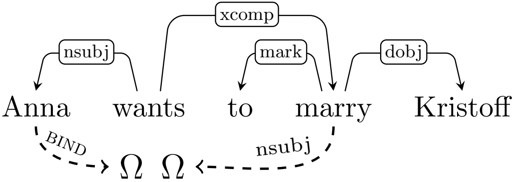
Enhancement Step

Question Words, Long-distance, Language-specific labels, Quantifiers

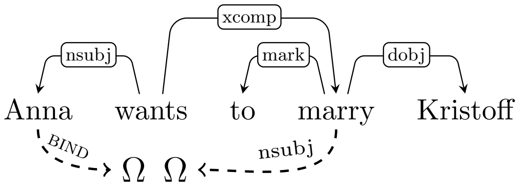


Dependency Graphs to Logical Forms





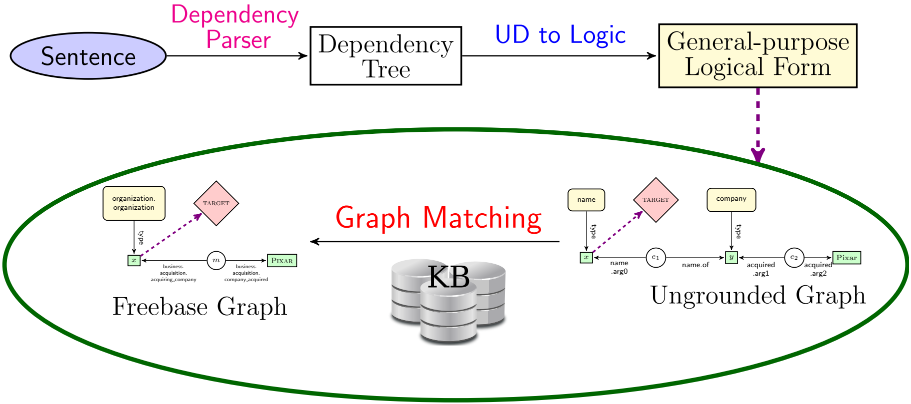
Dependency Graphs to Logical Forms



Lambda Expressions

BIND	=	$\lambda f \lambda g \lambda x. f(x) \wedge g(x)$
xcomp	=	$\lambda fgx. \exists y. f(x) \wedge g(y) \wedge \text{xcomp}(x_e, y_e)$
Ω	=	$\lambda x. \text{EQ}(x, \omega)$

Part 2: Question Answering



PART 2: Freebase QA

Knowledge-base Question Answering

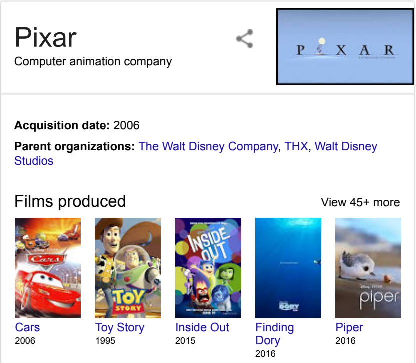
[Zelle and Mooney 1996, Berant et al. 2013, Kwiatkowski et al. 2013]

Question

Which company acquired Pixar?

Answer

{The Walt Disney Company}



Knowledge-base Question Answering

[Zelle and Mooney 1996, Berant et al. 2013, Kwiatkowski et al. 2013]

Question

Which company acquired Pixar?

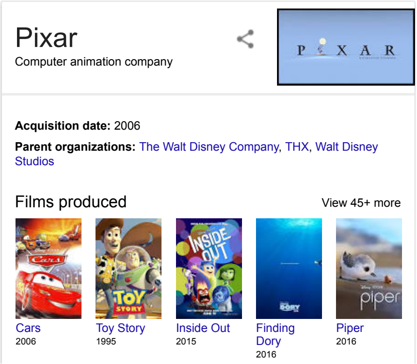
Grounded Logical Form

$\lambda x.\exists e.\ \text{organization}(x) \wedge$
$\text{acquisition}(e) \wedge$
$\text{acquiring\_company}(e, x) \wedge$
$\text{company\_acquired}(e, \text{Pixar})$

Latent

Answer

{The Walt Disney Company}



Freebase QA as Graph Matching [Reddy et al. 2014, 2016]

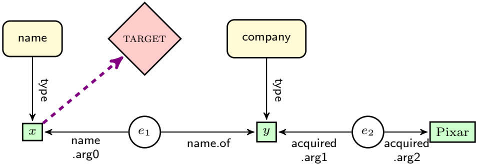
What is the name of the company that acquired Pixar?

$$\lambda x.\, \exists e_1 e_2 y.\, \mathrm{name}(x) \wedge \mathrm{name.arg}_0(e_1, x) \wedge \mathrm{name.of}(e_1, y)$$
$$\wedge\, \mathrm{company}(y) \wedge \mathrm{acquired.arg}_1(e_2, y) \wedge$$
$$\mathrm{acquired.arg}_2(e_2, \mathit{Pixar})$$

Freebase QA as Graph Matching [Reddy et al. 2014, 2016]

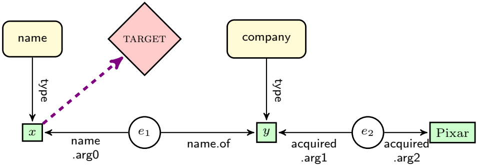
What is the name of the company that acquired Pixar?

$$\lambda x.\, \exists e_1 e_2 y.\, \text{name}(x) \wedge \text{name.arg}_0(e_1, x) \wedge \text{name.of}(e_1, y) \wedge \text{company}(y) \wedge \text{acquired.arg}_1(e_2, y) \wedge \text{acquired.arg}_2(e_2, \mathit{Pixar})$$

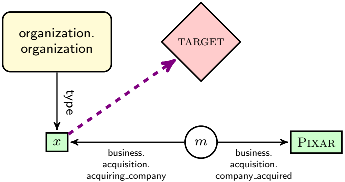


Ungrounded Graph

Freebase QA as Graph Matching [Reddy et al. 2014, 2016]

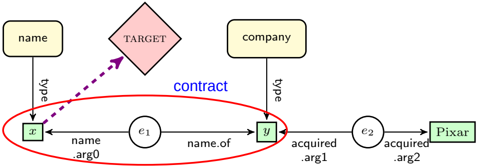


Ungrounded Graph

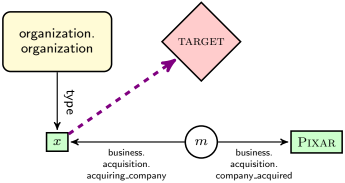


Freebase Graph

Freebase QA as Graph Matching [Reddy et al. 2014, 2016]

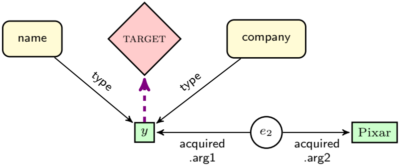


Ungrounded Graph

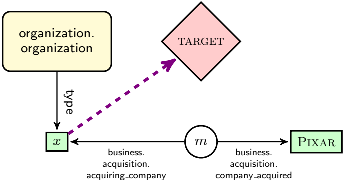


Freebase Graph

Freebase QA as Graph Matching [Reddy et al. 2014, 2016]



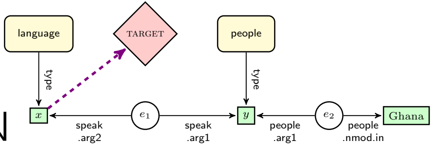
Ungrounded Graph



Freebase Graph

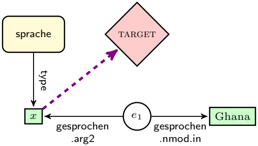
Multilingual Freebase QA

EN



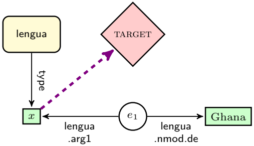
What language do people in Ghana speak?

DE

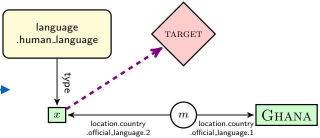


Welche Sprache wird in Ghana gesprochen?

ES



¿Cual es la lengua de Ghana?



Freebase Graph

Experimental Setup

69 lambda calculus rules

BiLSTM Parser [Kipperwiser and Goldberg 2016]

Multilingual WebQuestions and GraphQuestions

	WebQuestions
en	What language do the people in Ghana speak?
de	Welche Sprache wird in Ghana gesprochen?
es	¿Cuál es la lengua de Ghana?
	GraphQuestions
en	NASA has how many launch sites?
de	Wie viele Abschussbasen besitzt NASA?
es	¿Cuántos sitios de despegue tiene NASA?

Models

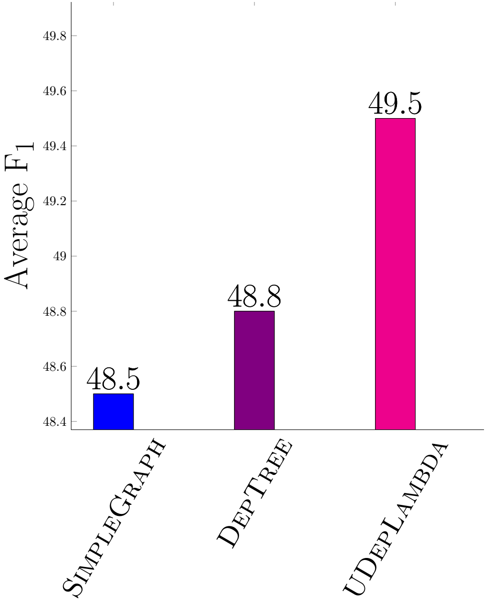
SIMPLEGRAPH : All entities connected to a single event
bag of words

DEPTREE: Transduce a dependency tree to target graph

UDEPLAMBDA: Logical forms from Universal Dependencies

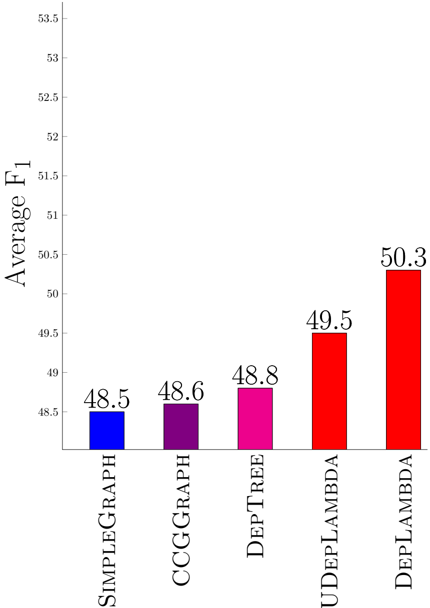
Results on Multilingual WebQuestions

English



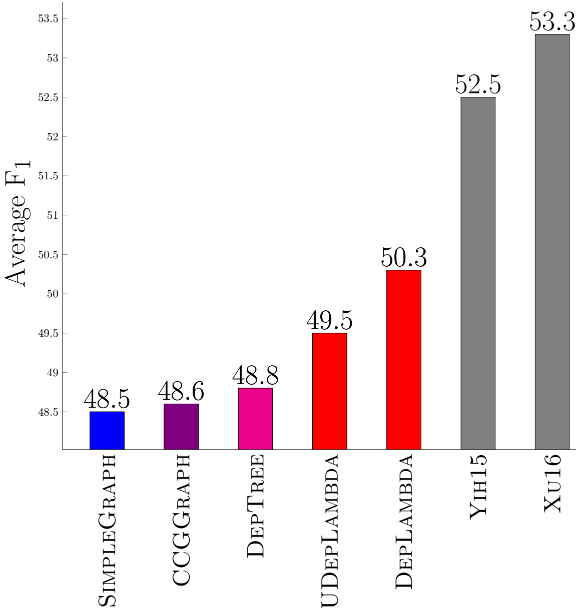
Results on Multilingual WebQuestions

English



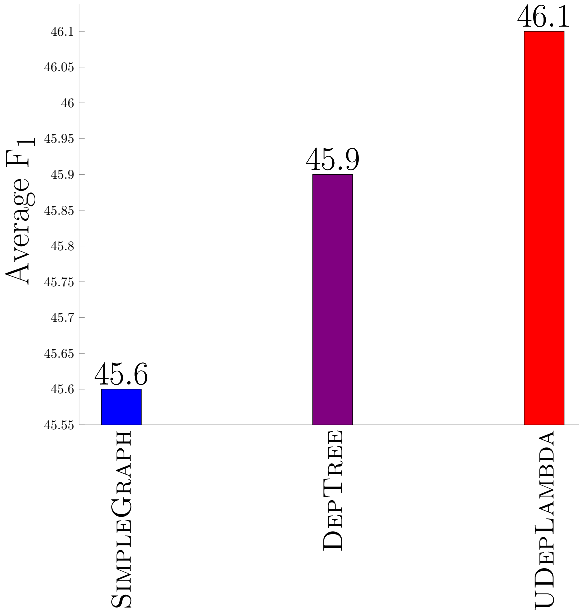
Results on Multilingual WebQuestions

English



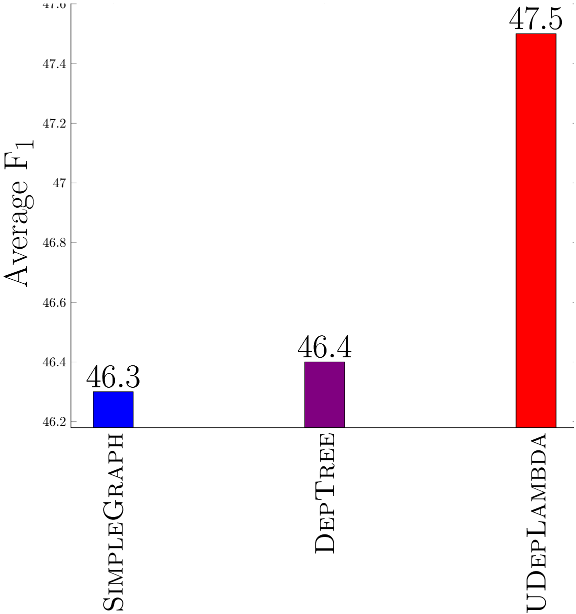
Results on Multilingual WebQuestions

German



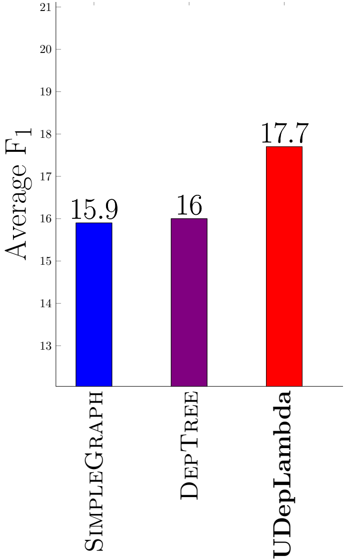
Results on Multilingual WebQuestions

Spanish



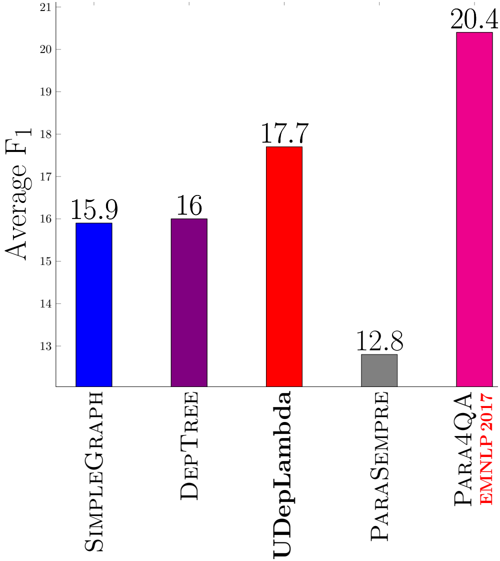
Results on Multilingual GraphQuestions

English



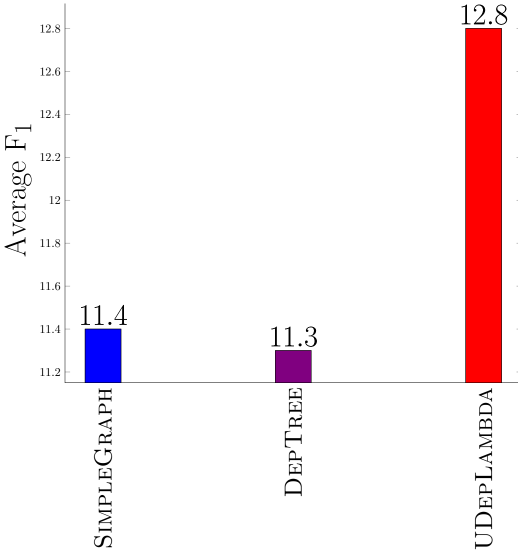
Results on Multilingual GraphQuestions

English



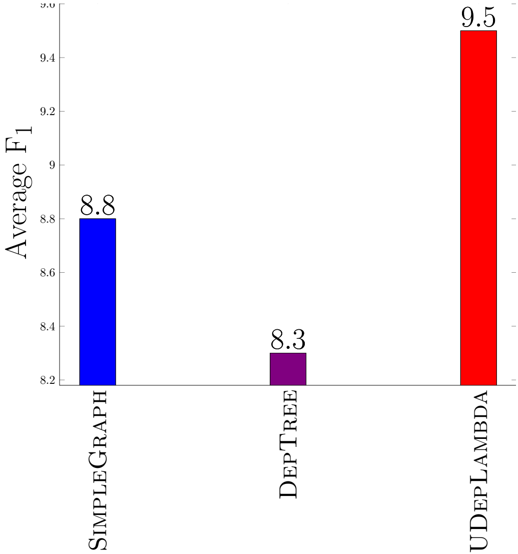
Results on Multilingual GraphQuestions

Spanish



Results on Multilingual GraphQuestions

German



Error Analysis / Limitations

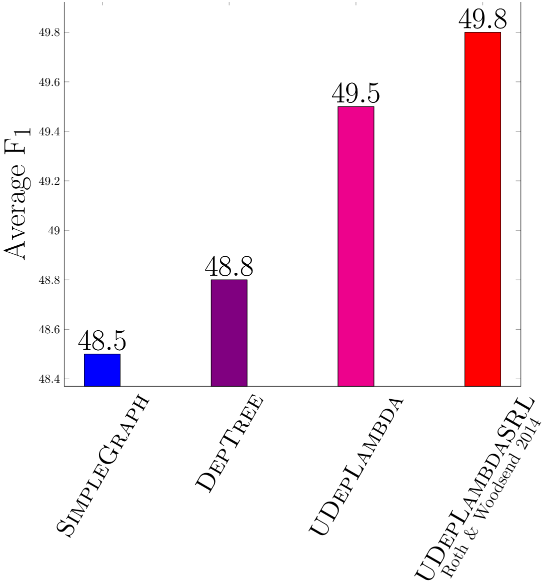
Context-sensitive semantics of dependency labels, e.g., *nsubj* is **not** always agent (arg_1)

- John broke the window ✓
- The window broke ✗

Solution: Semantic Role labeling [Palmer et al. 2010]

Results on Multilingual WebQuestions

English



Summary

Language-agnostic method for converting
Universal Dependencies to Logical forms

New Freebase evaluation datasets in German and Spanish

Ongoing Work: Richer Type System and Scoped Semantics

Code: `github.com/sivareddyg/UDepLambda`
Demo: `sivareddy.in/udeplambda.html`

Thank You!

Experimental Setup

69 lambda calculus rules

BiLSTM Parser [Kipperwiser and Goldberg 2016]

- English: 81.8
- German: 74.7
- Spanish: 82.2

Quantifiers and Negation Scope

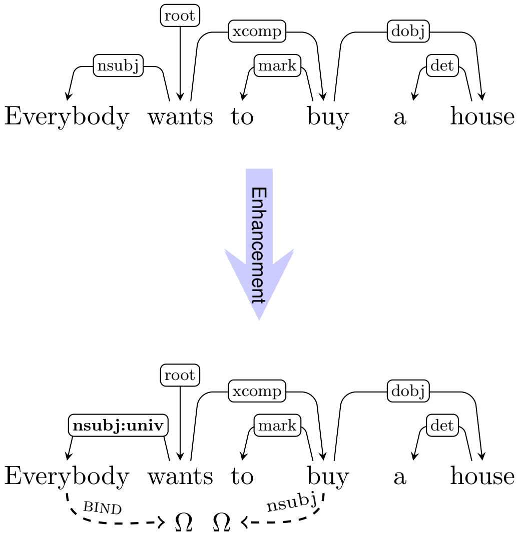
(Fancellu et al. 2017, Reddy et al. 2017)

Higher-order type system

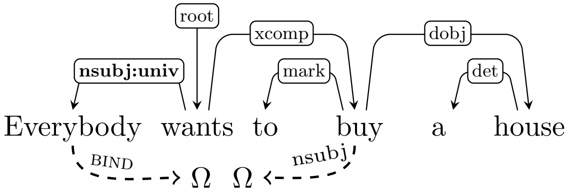
Fine-grained dependency labels

Quantifiers and Negation Scope

Fancellu et al. 2017, Reddy et al. 2017



Quantifiers and Negation Scope



Type System

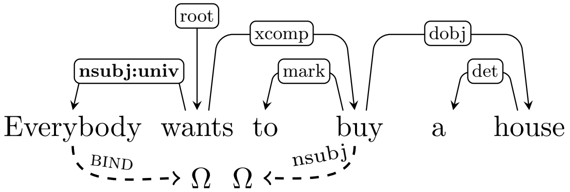
everybody $= \lambda x.\text{everybody}(x_a)$ [Old Type]

$= \lambda f.\forall x.\,\text{person}(x) \rightarrow f(x)$ [New Type]

wants $= \lambda x.\,\text{wants}(x_e)$ [Old Type]

$= \lambda f.\exists x.\,\text{wants}(x_e) \wedge f(x)$ [New Type]

Quantifiers and Negation Scope



Type System

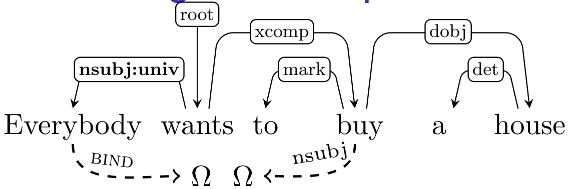
$$\text{nsubj} = \lambda fgx.\exists y.f(x) \wedge g(y) \wedge \text{arg}_1(x_e, y_a) \quad \text{[Old]}$$

$$\text{nsubj:univ} = \lambda PQf.Q(\lambda y.P(\lambda x.f(x) \wedge \text{arg}_1(x_e, y_a))) \quad \text{[New]}$$

$$\text{dobj} = \lambda fgx.\exists y.f(x) \wedge g(y) \wedge \text{arg}_2(x_e, y_a) \quad \text{[Old]}$$

$$= \lambda PQf.P(\lambda x.f(x) \wedge Q(\lambda y.\text{arg}_2(x_e, y_a))) \quad \text{[New]}$$

Quantifiers and Negation Scope



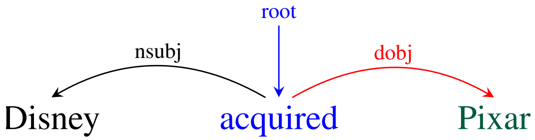
Old Expression:

(3) $$\lambda z.\exists xyw.\, \text{wants}(z_e) \wedge \text{everybody}(x_a) \wedge \text{arg}_1(z_e, x_a) \\ \wedge\, \text{buy}(y_e) \wedge \text{xcomp}(z_e, y_e) \wedge \text{arg}_1(y_e, x_a) \\ \wedge\, \text{arg}_1(x_e, y_a) \wedge \text{house}(w_a) \wedge \text{arg}_2(y_e, w_a)\,.$$

New Expression:

(6) $$\lambda f.\forall x\,.\,\text{person}(x_a) \rightarrow \\ [\exists zyw.f(z) \wedge \text{wants}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{buy}(y_e) \\ \wedge\, \text{xcomp}(z_e, y_e) \wedge \text{house}(w_a) \\ \wedge\, \text{arg}_1(z_e, x_a) \wedge \text{arg}_2(z_e, w_a)]\,.$$

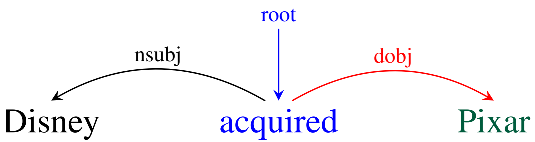
Single Type System



All constituents are of the same lambda expression type

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

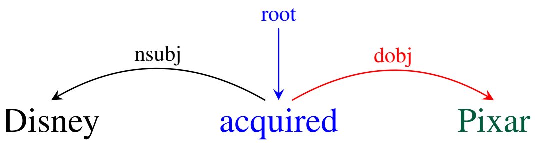
Single Type System



All words have a *lambda expression* of type η

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

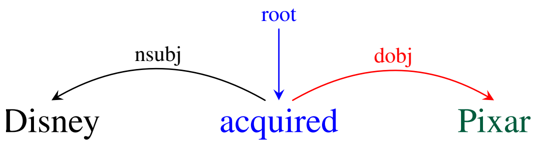
Single Type System



All constituents have a *lambda expression* of type η

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

Single Type System

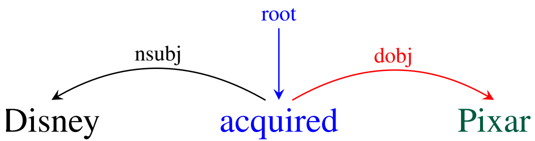


All constituents have a *lambda expression* of type η

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

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

Single Type System

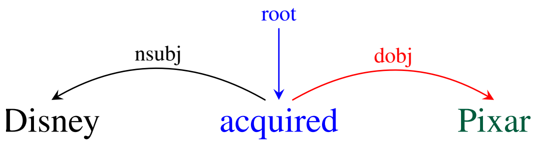


Lambda Expression for words

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

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

Single Type System



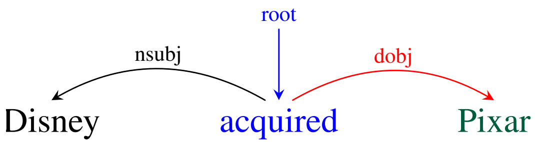
Lambda Expression for words

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

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

Here $\textsf{TYPE}[\text{acquired}] \neq \textsf{TYPE}[\text{Pixar}]$ ✗

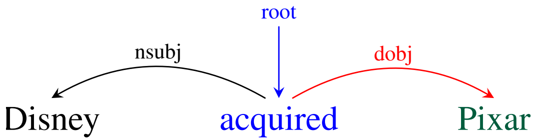
Single Type System



Lambda Expression for dependency labels

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

Single Type System

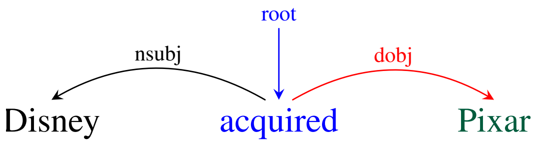


Lambda Expression for dependency labels

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

This operation mirrors the tree structure

Single Type System

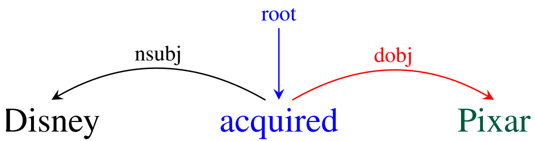


Lambda Expression for words

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

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

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] ✓

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

Logical Expression:

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

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:

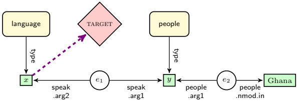
$$\lambda w.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{coord}(w,y,z) \\ \wedge \mathrm{arg}_1(w_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Post processing:

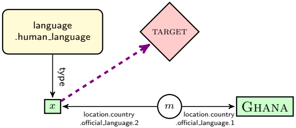
$$\lambda e.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{arg}_1(y_e,x_a) \\ \wedge \mathrm{arg}_1(z_e,x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Graph Transformation: CONTRACT operation

What language do the people in Ghana speak?



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

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