Natural Language Learning with Formal Semantics

Wenhu Chen

Outline

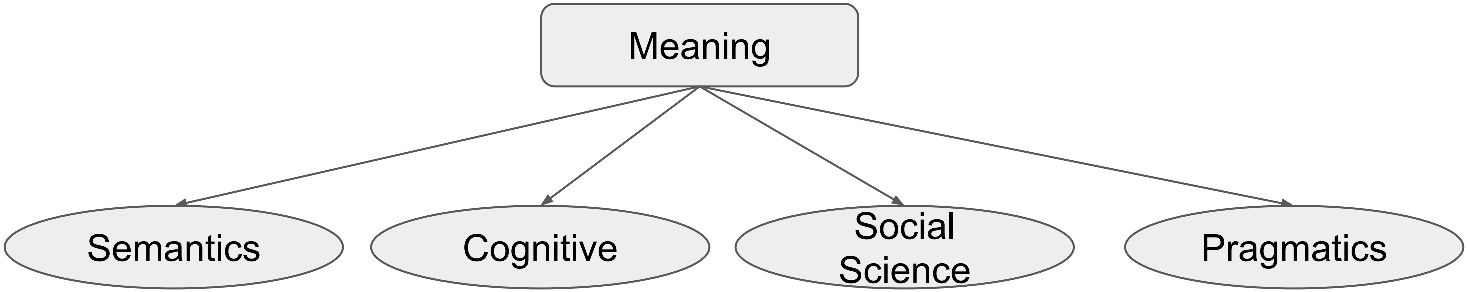
1. Introduction
2. Formal Semantics
3. Its Application in NLP
4. My Work & Future Direction

Meaning

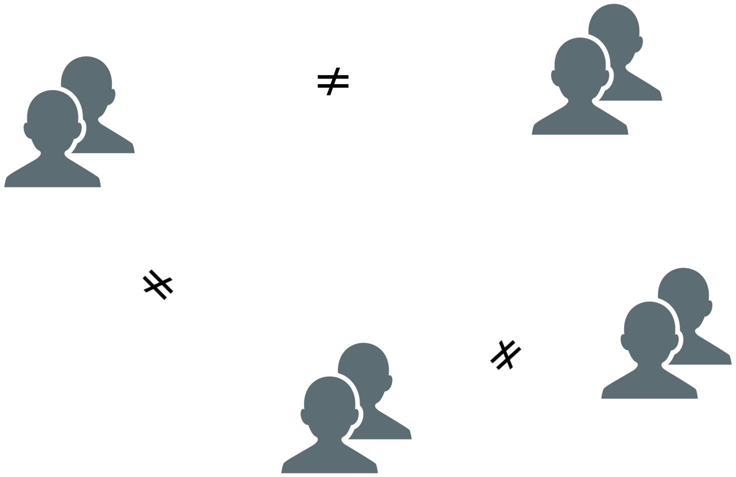
Meaning

Meaning


NLP


Meaning
Semantics
Cognitive
Social Science
Pragmatics

Meaning



Linguists can barely reach an agreement on the definition of “meaning”

Meaning

Raw-String

1. Simple
2. Straightforward

Bag of words
Unigram/Bigram

Meaning

Raw-String

1. Simple
2. Straightforward

Bag of words
Unigram/Bigram

Formal Semantics

1. Compositional
2. Explainable

Lambda Calculus
Montague Semantics

Meaning

Raw-String

1. Simple
2. Straightforward

Bag of words
Unigram/Bigram

Formal Semantics

1. Compositional
2. Explainable

Lambda Calculus
Montague Semantics

Real-valued Vector

1. Data-Drive
2. Computation Efficient

Word2Vec/Glove
BERT

Meaning

What aspects do you need the meaning representation to capture?

Meaning

What aspects do you need the meaning representation to capture?

Can it lean meaning or just similarity between meaning?

Can it lean grounded meaning or learn lexical meaning?

1. Raw-String Representation

- Cannot directly link to the external world knowledge.
- Cannot combine pieces of knowledge together to perform inference.

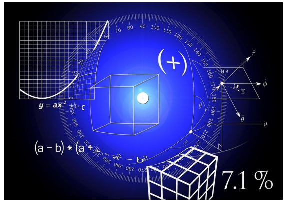
Utterance1: John has an apple
Utterance2: John has a banana

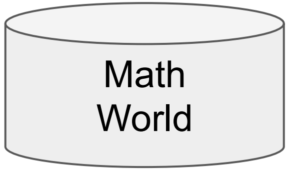
?

Utterance: John has an apple
and a banana.

1. Raw-String Representation

- Assuming we want to build an NLP system to understand math questions?
  a. Such system cannot leverage the mathematical knowledge.
  b. The sample complexity becomes extremely high.





?

Utterance: What is the largest prime less than 10?

2. Formal Semantics (Montague)

- We construct the meaning of natural language in terms of variables, predicates, arguments.

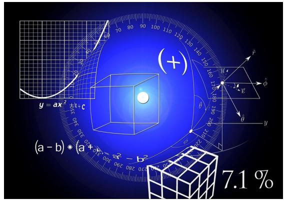
2. Formal Semantics (Montague)

- We construct the meaning of natural language in terms of variables, predicates, arguments.
- Formal Semantics aim to use explicit and grounded meaning representation to convey information.

2. Formal Semantics (Montague)

- We construct the meaning of natural language in terms of variables, predicates, arguments.
- Formal Semantics aim to use explicit and grounded meaning representation to convey information.
- The formal semantics can help us understand human language from a mathematical logic perspective.

2. Formal Semantics (Montague)



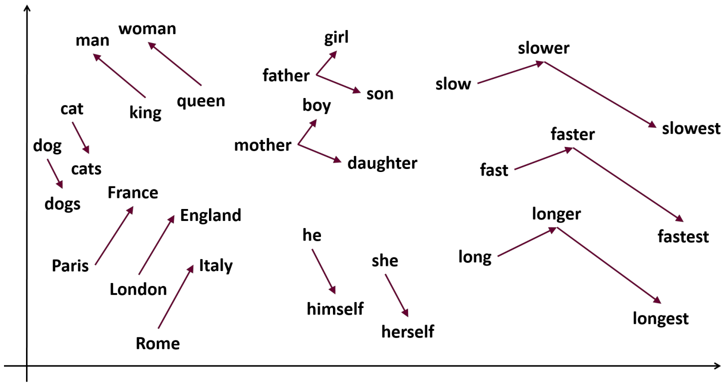
$$max(primes \cap (-\inf; 10))$$

Math World

Utterance: What is the largest prime less than 10?

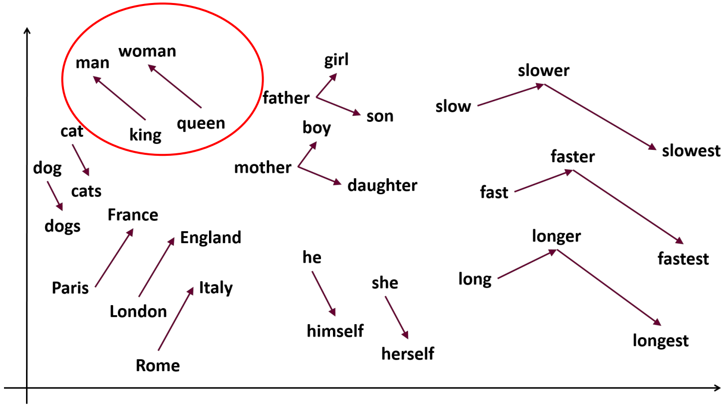
3. Vector Representation

- Data-driven learning process
- Suitable for deep learning framework



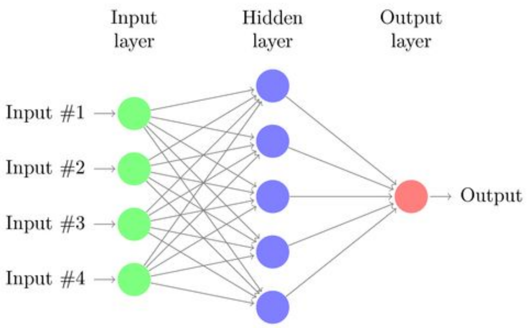
3. Vector Representation

- Data-driven learning process
- Suitable for deep learning framework



3. Vector Representation

Utterance

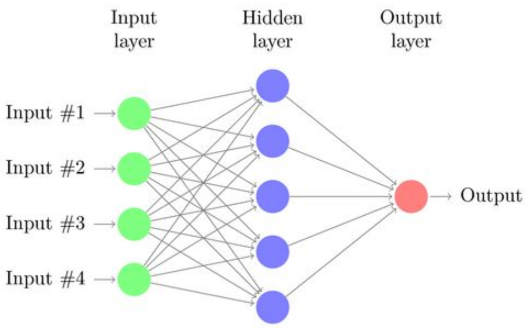


Behavior

We don’t need to worry about the structure of computation in the middle

3. Vector Representation

Utterance



Behavior

Tight coupling between machine learning and representation means there's always risk that some new semantic phenomenon arises and suddenly our model is useless

Evaluation Chart

Type	Meaning	Similarity Meaning	Grounded Meaning	Lexical Meaning
String	✗	✗	✗	✓
Vector				
Formal				

Evaluation Chart

Type	Meaning	Similarity Meaning	Grounded Meaning	Lexical Meaning
String	✗	✗	✗	✓
Vector	?	✓	?	✓
Formal				

Evaluation Chart

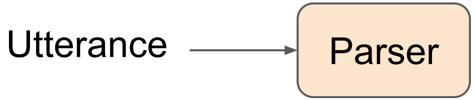
Type	Meaning	Similarity Meaning	Grounded Meaning	Lexical Meaning
String	✗	✗	✗	✓
Vector	?	✓	?	✓
Formal	✓	✓	✓	✓

Formal Semantics

- For a long time, such an explicit meaning representation has dominated the research of NLP community
  - Reasoning
  - Perception
  - Action

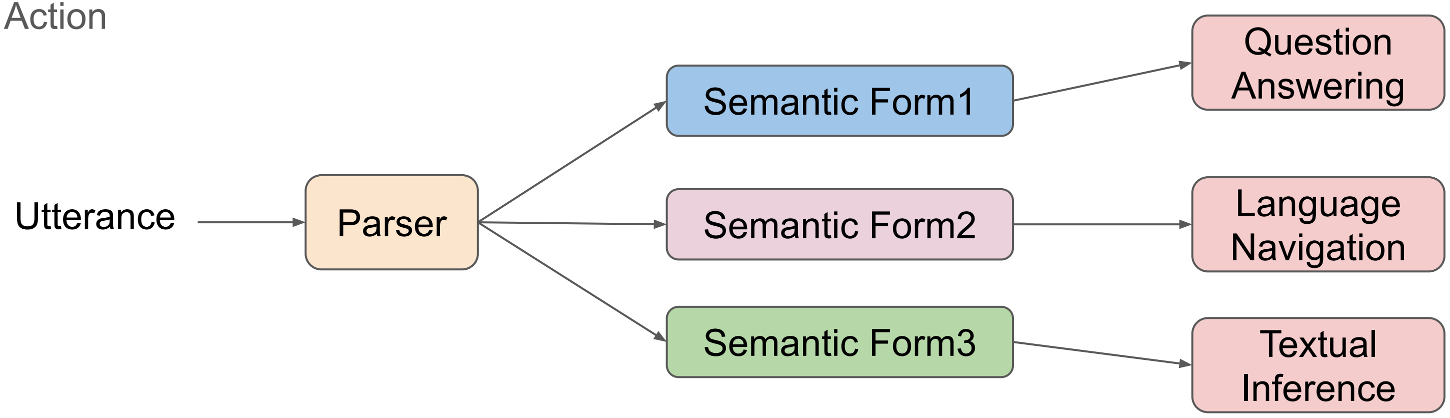
Formal Semantics

- For a long time, such an explicit meaning representation has dominated the research of NLP community
  - Reasoning
  - Perception
  - Action



Formal Semantics

- For a long time, such an explicit meaning representation has dominated the research of NLP community
  - Reasoning
  - Perception
  - Action



List of Semantic Forms

- Introduction of formal semantics
  - Lambda Calculus (Turing 1937)
  - Combinatory categorial grammar (CCG) (Steedman et al. 1996, 2000)
  - Weighted linear CCG (Clark & Currant et. al 2007)
  - Dependency-based compositional semantics (DCS) (Liang et al. 2011)
  - Lambda DCS (Liang et al. 2013)
  - Abstract Meaning Representation (Banarescu et al. 2013)
  - Domain Specific Languages (DSL)
    - Functional program semantics (Liang et al. 2017, Dawn et al. 2018)
    - SQL, etc (Zhong et al. 2017, Yu et al. 2018)

Semantic Forms

- Combinatory Categorial Grammar
  - Definition
  - Rules
  - Learning
- Dependency-based compositional semantics
  - Definition
  - Rules
  - Learning
- Neural Symbolic Machine (Functional-Program)
  - Definition
  - Learning

Yoav et. al, Learning Compact Lexicons for CCG Semantic Parsing, 2014
Yoav et. al, Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions, 2013
Luke et. al, Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars, 2005
Steedman et. al, Surface Structure and Interpretation. 1996
Steedman et. al, The Syntactic Process. 2000

Combinatory Categorial Grammar

- Categorial Formalism
  - Compositional
  - Puts information on the words

Combinatory Categorial Grammar

- Categorial Formalism
  - Compositional
  - Puts information on the words
- Transparent interface between syntax and semantics

Combinatory Categorial Grammar

- Categorial Formalism
  - Compositional
  - Puts information on the words
- Transparent interface between syntax and semantics
- Including Predicate-argument structure, quantification and information structure.

Combinatory Categorial Grammar

- Categorial Formalism
  - Compositional
  - Puts information on the words
- Transparent interface between syntax and semantics
- Including Predicate-argument structure, quantification and information structure.
- Same expressive power as lambda calculus.

CCG Categories

- Basic Building Block
- Capture Syntactic and Semantic Information Jointly

ADJ : $\lambda x. func(x)$

CCG Categories

- Basic Building Block
- Capture Syntactic and Semantic Information Jointly

Syntax ADJ : $\lambda x.\, func(x)$ Semantic

CCG Categories

- Primitive symbols: N, S, NP, ADJ and PP
- Syntactic combination operation (/,\)
- Slashes specify argument order

Syntax	ADJ	:	$\lambda x.\, func(x)$	Semantic
Syntax	(S\NP)/ADJ	:	$\lambda f.\, \lambda x.\, f(x)$	Semantic
Syntax	NP	:	S	Semantic

CCG Lexical Entries

- Pair words and phrases with meaning
- Meaning captured by a CCG category

fun	$\vdash$	ADJ: $\lambda x.\, fun(x)$
Natural Language		CCG Category

CCG Lexical Entries

- Pair words and phrases with meaning
- Meaning captured by a CCG category

fun	|-	ADJ: $\lambda x.\, fun(x)$
is	|-	(S\NP)/ADJ: $\lambda f.\, \lambda x.\, f(x)$
CCG	|-	NP: CCG

CCG Operations

- Equivalent to function application
- Two direction of application

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

CCG Operations

- Equivalent to function application

- Two direction of application

$$\overset{\text{Argument}}{\boxed{B : g}} \; \overset{\text{Function}}{\boxed{A\backslash B : f}} \Rightarrow \overset{\text{Result}}{\boxed{A : f(g)}} \quad (<)$$

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

CCG Parsing

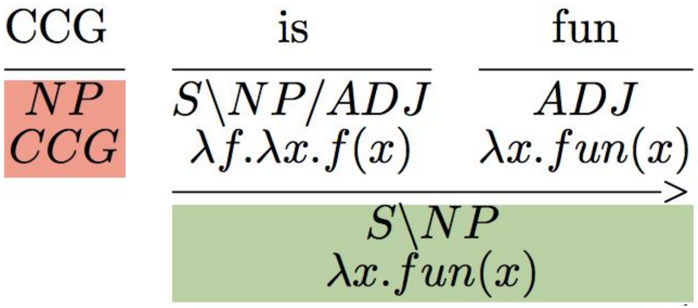
- Use Lexicon to match words and phrases with their CCG categories

- Combine categories using operators

$$\frac{\text{CCG}}{\begin{array}{c} NP \\ CCG \end{array}} \quad \frac{\text{is}}{\begin{array}{c} S\backslash NP/ADJ \\ \lambda f.\lambda x.f(x) \end{array}} \quad \frac{\text{fun}}{\begin{array}{c} ADJ \\ \lambda x.fun(x) \end{array}}$$

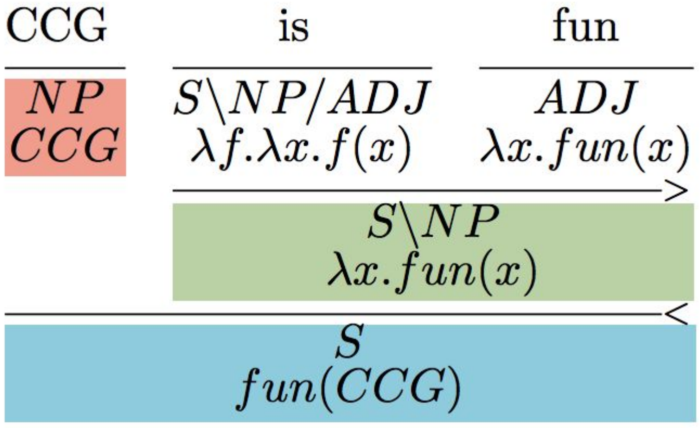
CCG Parsing

- Use Lexicon to match words and phrases with their CCG categories
- Combine categories using operators



CCG Parsing

- Use Lexicon to match words and phrases with their CCG categories
- Combine categories using operators



Lexicon Problem

- Key component of CCG
- Same words often paired with many different categories
- Difficult to learn from limited data

Lexicon Problem

- Key component of CCG
- Same words often paired with many different categories
- Difficult to learn from limited data

The house dog	house	|-	ADJ:	$\lambda x.\, house(x)$
The dog of the house	house	|-	N:	$\lambda x.\, of(x, ly.\, house(y))$

Factored Lexicons

house |- N: $\lambda x. of(x, ly. house(y))$

house |- ADJ: $\lambda x. house(x)$

garden |- N: $\lambda x. of(x, ly. garder(y))$

Similar

Factored Lexicons

house |- N: $\lambda x.\, of(x, ly.\, house(y))$

house |- ADJ: $\lambda x.\, house(x)$

garden |- N: $\lambda x.\, of(x, ly.\, garder(y))$

Lexemes

(garden, {*garden*})
(house, {*house*})

Templates

$\lambda(\omega, v_i).$
$[\omega \to ADJ : \lambda x.\, of(x, ly.\, v_1(y))]$
$\lambda(\omega, v_i).$
$[\omega \to N : \lambda x.\, v_1(x)]$

Factored Lexicons

Original Lexicon

flight $\vdash S|NP : \lambda x.flight(x)$
flight $\vdash S|NP/(S|NP) : \lambda f.\lambda x.flight(x) \land f(x)$
flight $\vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.flight(x) \land f(x)$
ground transport $\vdash S|NP : \lambda x.trans(x)$
ground transport $\vdash S|NP/(S|NP) : \lambda f.\lambda x.trans(x) \land f(x)$
ground transport $\vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.trans(x) \land f(x)$

Compress

Factored Lexicon

(flight, $\{flight\}$)
(ground transport, $\{trans\}$)

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP : \lambda x.v_1(x)]$
$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP/(S|NP) : \lambda f.\lambda x.v_1(x) \land f(x)]$
$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.v_1(x) \land f(x)]$

Factored Lexicons

- Capture systematic variations in word usage
- Each variation can then be applied to compact units in lexical matching
- Abstracts the compositional nature of the word

Learning (Data)

- (Natural Language, Lambda Form)

Show me flights to Boston

$\lambda x.flight(x) \wedge to(x, BOSTON)$

I need a flight from baltimore to seattle

$\lambda x.flight(x) \wedge from(x, BALTIMORE) \wedge to(x, SEATTLE)$

what ground transportation is available in san francisco

$\lambda x.ground\_transport(x) \wedge to\_city(x, SF)$

Learning (Structure Prediction)

show me	flights	to	Boston
S/N	N	PP/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda x.to(x,y)$	$BOSTON$

$$\xrightarrow{\quad>\quad} \quad PP \quad \lambda x.to(x, BOSTON)$$

$$N\backslash N \quad \lambda f.\lambda x.f(x) \wedge to(x, BOSTON)$$

$$\xrightarrow{\quad<\quad} \quad N \quad \lambda x.flight(x) \wedge to(x, BOSTON)$$

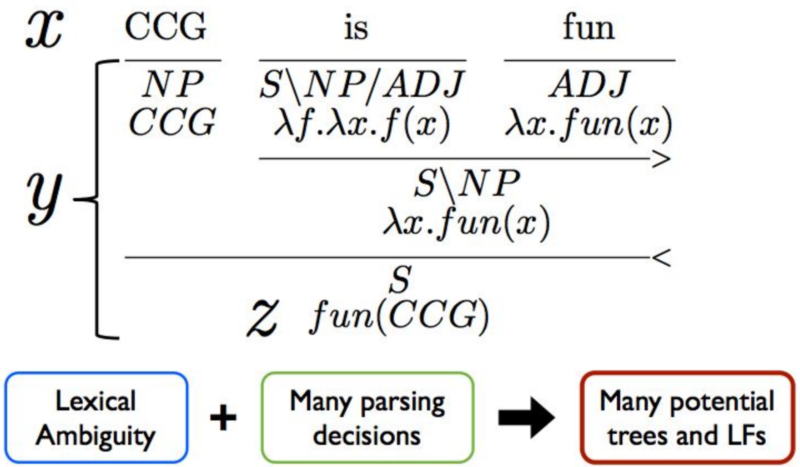
$$\xrightarrow{\quad>\quad} \quad S \quad \lambda x.flight(x) \wedge to(x, BOSTON)$$

Lexicon

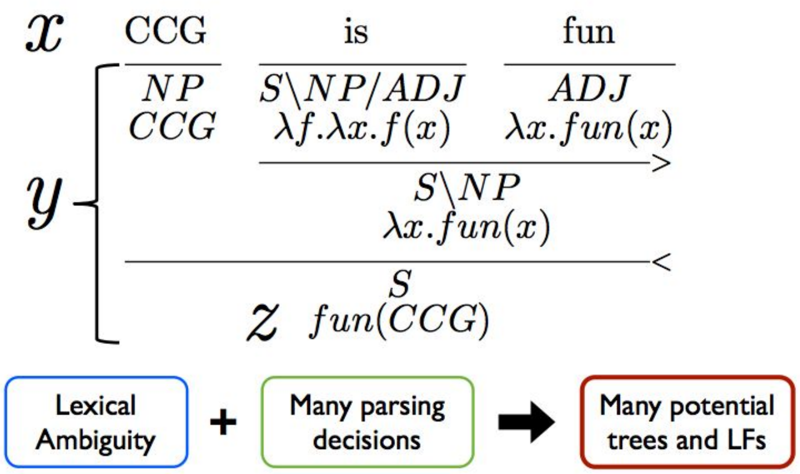
Combinators
Predefined

Log-Linear
$\mathcal{W}$

Learning (Log-Linear Scorer)



Learning (Log-Linear Scorer)



- Given a weighted linear model:
  - CCG lexicon Λ
  - Feature function $f : X \times Y \to \mathbb{R}^m$
  - Weights $w \in \mathbb{R}^m$
- The best parse is:

$$y^* = \arg\max_y w \cdot f(x, y)$$

- We consider all possible parses y for sentence x given the lexicon Λ

GENLEX

show me	flights	to	Boston

Latent

$$\frac{}{\substack{S \\ \lambda x.flight(x) \wedge to(x, BOSTON)}}>$$

$$GENLEX(x, \mathcal{V}; \Lambda, \theta)$$

show me	flights	to	Boston
S/N	N	PP/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda x.to(x, y)$	$BOSTON$

GENLEX

Lexical Generation Procedure

$$GENLEX(x, \mathcal{V}; \Lambda, \theta)$$

Given:

Sentence x
Validation Function $\mathcal{V}$
Log-Linear Model θ
Lexicon Λ_0

GENLEX

Lexical Generation Procedure

$$GENLEX(x, \mathcal{V}; \Lambda, \theta)$$

Given:

- Sentence x
- Validation Function $\mathcal{V}$
- Log-Linear Model θ
- Lexicon Λ_0

Lexemes

(garden, {*garden*})
(house, {*house*})

Templates

$\lambda(\omega, v_i).$
$[\omega \rightarrow ADJ : \lambda x. of(x, ly. v_1(y))]$

Overly Generate

Lexicon

Λ_0

GENLEX

Lexical Generation Procedure

$$GENLEX(x, \mathcal{V}; \Lambda, \theta)$$

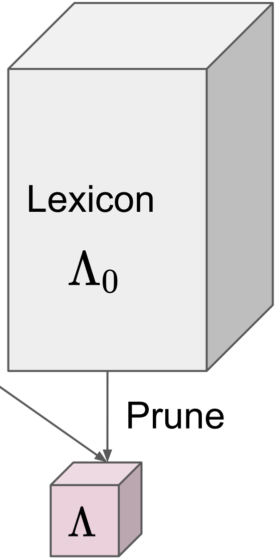
Given:

Sentence x

Validation Function $\mathcal{V}$

Log-Linear Model θ

Lexicon Λ_0



Unified Learning Algorithm

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

- Online
- Input:

  $\{(x_i, \mathcal{V}_i) : i = 1 \ldots n\}$
- 2 steps:
  - Lexical generation
  - Parameter update

Unified Learning Algorithm

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Iterate over data

Lexical generation

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

Lexical generation

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n :$

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Get top parses

Lexical generation

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Get lexical entries from
highest scoring
valid parses

Lexical generation

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Update model's lexicon

Update parse scorer

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:

$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$\quad s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$\quad s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:

$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$\quad - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Re-parse parsers and divide them into “good” and “bad” ones

Update parse scorer

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$\quad s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$\quad s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$\quad - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Pick the "good" and "bad" parsers violating the margin

Update parse scorer

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Update towards 'good' parses and against 'bad' parses

Update parse scorer

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Output Lexicon & Log-Linear Model

Semantic Forms

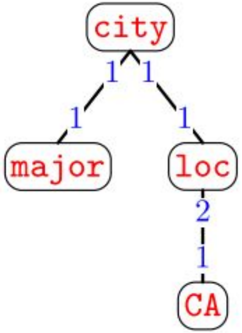
- Combinatory Categorial Grammar
  - Definition
  - Rules
  - Learning
- Dependency-based compositional semantics
  - Definition
  - Rules
  - Learning
- Neural Symbolic Machine (Functional-Program)
  - Definition
  - Learning

Liang et. al, Learning dependency-based compositional semantics, 2011
Liang et. al, Lambda dependency-based compositional semantics, 2013
Jonathan et. al, Semantic parsing on Freebase from question-answer pairs, 2013
Jonathan et. al, Semantic parsing via paraphrasing, 2014

Dependency-based Compositional Semantics

- Weakness of CCG Parse
  - Too much hand-coded rules for lexicon construction.
  - Learning algorithm is complicated.
  - The grammar rules are too strict.
  - Learning requires annotating logic form.

Dependency-Based Compositional Semantics (DCS)



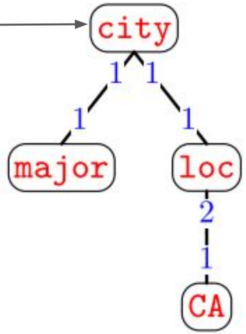
DCS

$$\lambda c\, \exists m\, \exists \ell\, \exists s\, .\ \mathtt{city}(c) \wedge \mathtt{major}(m) \wedge \mathtt{loc}(\ell) \wedge \mathtt{CA}(s) \wedge c_1 = m_1 \wedge c_1 = \ell_1 \wedge \ell_2 = s_1$$

major cities in CA

Dependency-Based Compositional Semantics (DCS)

Predicate



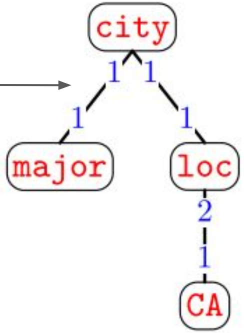
DCS

$$\lambda c\, \exists m\, \exists \ell\, \exists s\, .$$
$$\texttt{city}(c) \wedge \texttt{major}(m) \wedge$$
$$\texttt{loc}(\ell) \wedge \texttt{CA}(s) \wedge$$
$$c_1 = m_1 \wedge c_1 = \ell_1 \wedge \ell_2 = s_1$$

major cities in CA

Dependency-Based Compositional Semantics (DCS)

1st elem of
“city” equals
1st elem of
“major”

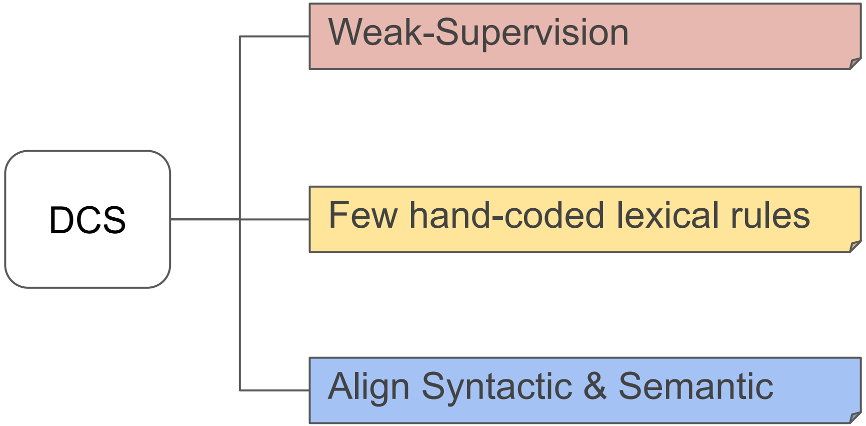


DCS

$$\lambda c \, \exists m \, \exists \ell \, \exists s \, . \\ \texttt{city}(c) \wedge \texttt{major}(m) \wedge \\ \texttt{loc}(\ell) \wedge \texttt{CA}(s) \wedge \\ c_1 = m_1 \wedge c_1 = \ell_1 \wedge \ell_2 = s_1$$

major cities in CA

Dependency-Based Compositional Semantics (DCS)



Strong vs. Weak Supervision

Detailed Supervision (current)
- doesn't scale up
- representation-dependent

What is the largest city in California?

$\Big\downarrow$ **expert**

$\mathtt{argmax}(\{c : \mathtt{city}(c) \wedge \mathtt{loc}(c, \mathtt{CA})\}, \mathtt{population})$

Strong vs. Weak Supervision

Detailed Supervision (current)
- doesn't scale up
- representation-dependent

What is the largest city in California?

$\Big\downarrow$ **expert**

$\texttt{argmax}(\{c : \texttt{city}(c) \wedge \texttt{loc}(c, \texttt{CA})\}, \texttt{population})$

Natural Supervision (new)
- scales up
- representation-independent

What is the largest city in California?

$\Big\downarrow$ **non-expert**

Los Angeles

Lexicon vs. Triggers

CCG

$$\frac{\text{is}}{\begin{array}{c} S\backslash NP/ADJ \\ \lambda f.\lambda x.f(x) \end{array}}$$

requires strict grammar rule during parsing and harsh type constraint

Lexicon vs. Triggers

CCG

$$\frac{\text{is}}{S\backslash NP/ADJ \atop \lambda f.\lambda x.f(x)}$$

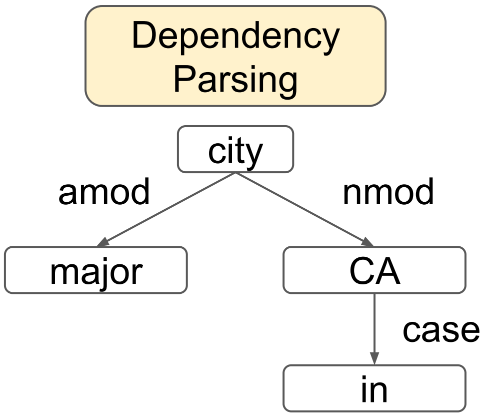
requires strict grammar rule during parsing and harsh type constraint

DCS

{(city, city),(city, state),(city, river), . . . }

requires only trigger words for predicates without grammar rule

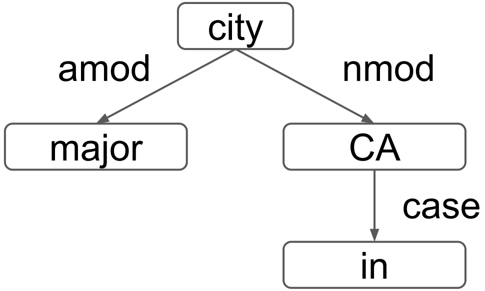
Syntactic & Semantic Alignment



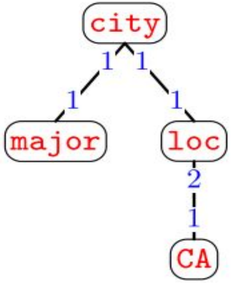
Utterance: major cities in CA

Syntactic & Semantic Alignment

Dependency Parsing



Dependency-based Compositional Semantics



Utterance: major cities in CA

Parsing: Lexical Triggers

				`city`	`city`			
				`state`	`state`			
				`river`	`river`			
			`argmax`	`population`	`population`		`CA`	
What	*is*	*the*	*most*	*populous*	*city*	*in*	*CA*	*?*

Parsing: Lexical Triggers

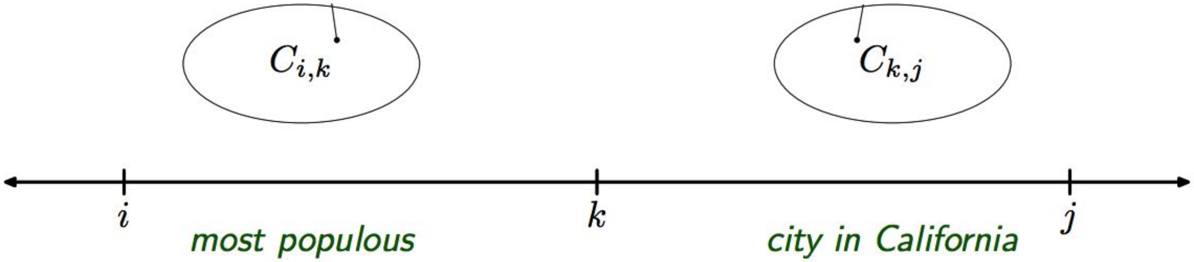
				city	city			
				state	state			
				river	river			
			argmax	population	population		CA	
What	*is*	*the*	*most*	*populous*	*city*	*in*	*CA*	*?*

Lexical Triggers:

1. String match — *CA* ⇒ `CA`
2. Function words (20 words) — *most* ⇒ `argmax`
3. Nouns/adjectives — *city* ⇒ `city state river population`

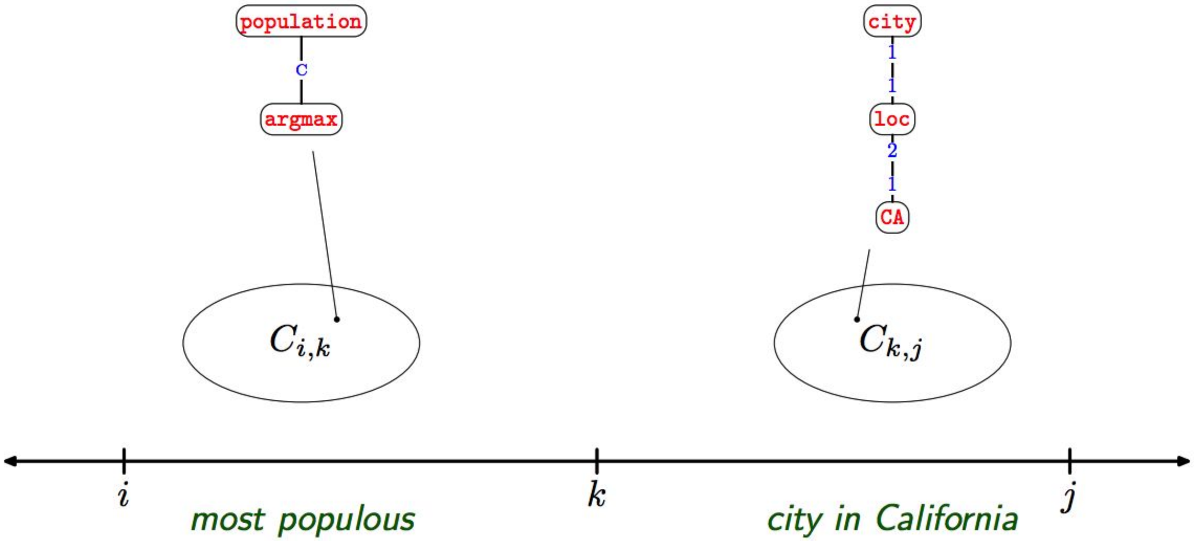
Parsing: Predicates to DCS Trees (DP)

$C_{i,j}$ = set of DCS trees for span $[i, j]$



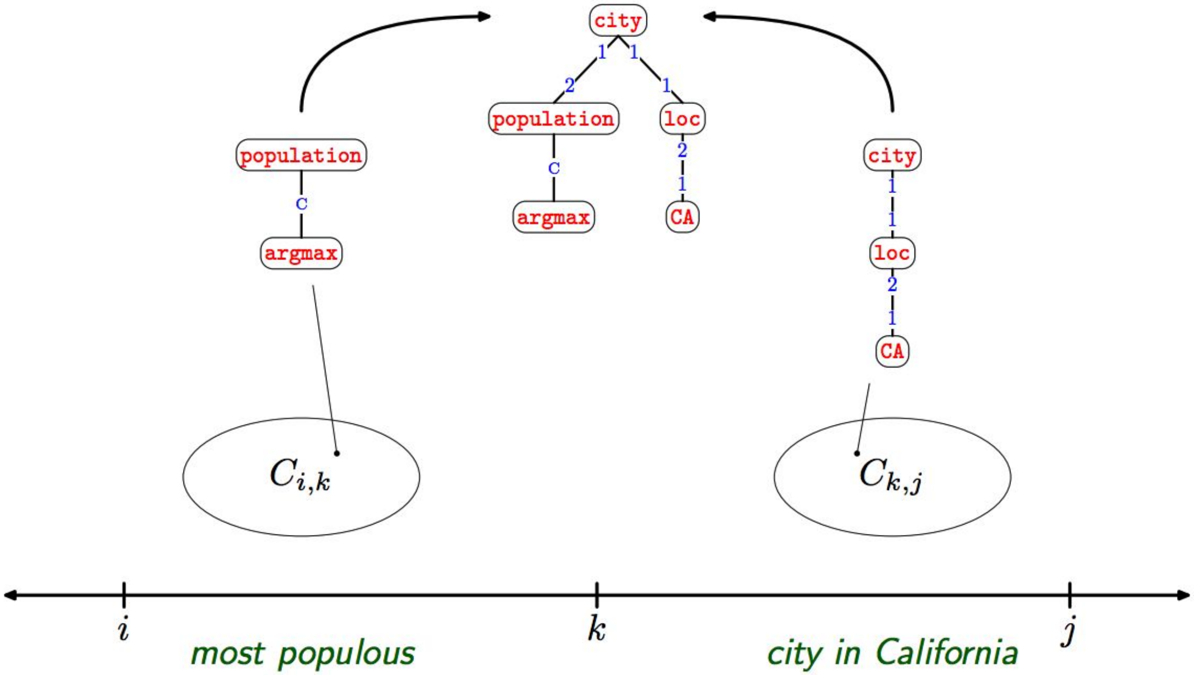
Parsing: Predicates to DCS Trees (DP)

$C_{i,j}$ = set of DCS trees for span $[i, j]$



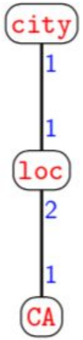
Parsing: Predicates to DCS Trees (DP)

$C_{i,j}$ = set of DCS trees for span $[i, j]$



Denotation

DCS tree



Database

city

San Francisco
Chicago
Boston
...

loc

Mount Shasta	California
San Francisco	California
Boston	Massachusetts
...	...

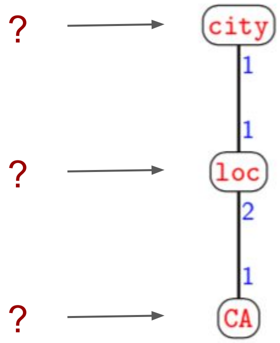
CA

California

Assuming we have our world knowledge containing 1) unary predicates 2) binary predicates in a database.

Denotation

DCS tree



Database

city

San Francisco
Chicago
Boston
...

loc

Mount Shasta	California
San Francisco	California
Boston	Massachusetts
...	...

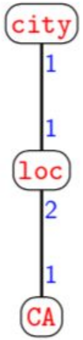
CA

California

How can we assign values to different nodes?

Denotation

DCS tree



Constraints

$c \in \texttt{city}$

Database

city

San Francisco
Chicago
Boston
...

loc

Mount Shasta	California
San Francisco	California
Boston	Massachusetts
...	...

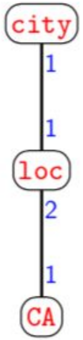
CA

California

Constraint 1: the value in the each node should come from its corresponding predicate table.

Denotation

DCS tree



Constraints

$c \in \texttt{city}$

$c_1 = \ell_1$

$\ell \in \texttt{loc}$

Database

city

San Francisco
Chicago
Boston
...

loc

Mount Shasta	California
San Francisco	California
Boston	Massachusetts
...	...

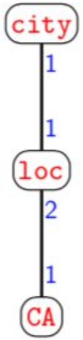
CA

California

Constraint 2: the assignment of adjacent nodes need to meet the requirement in the edge.

Constraint Satisfaction Problem

DCS tree



Constraints

$c \in \texttt{city}$

$c_1 = \ell_1$

$\ell \in \texttt{loc}$

$\ell_2 = s_1$

$s \in \texttt{CA}$

Database

city

San Francisco
Chicago
Boston
...

loc

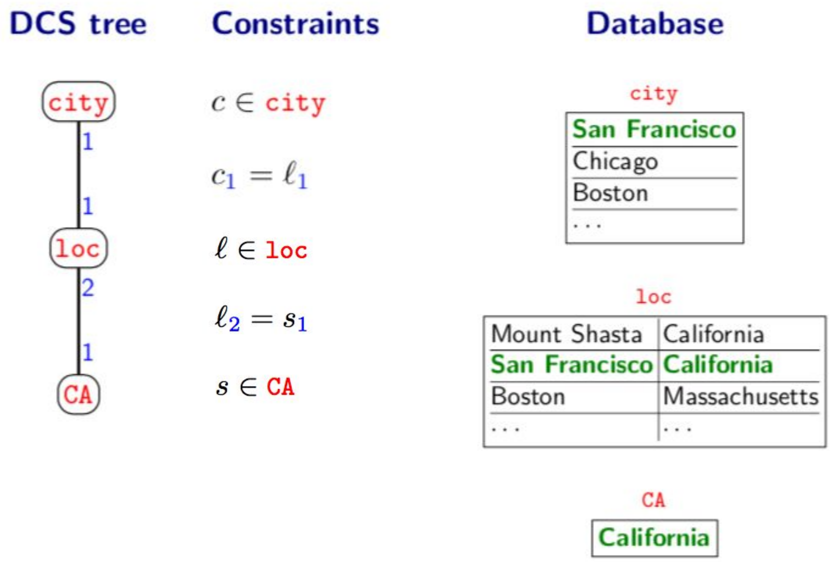
Mount Shasta	California
San Francisco	California
Boston	Massachusetts
...	...

CA

California

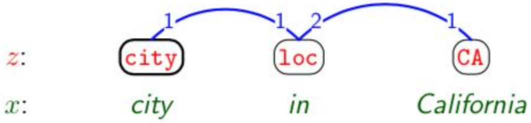
A DCS tree assignment becomes a constraint satisfaction problem (CSP)

Constraint Satisfaction Problem



A DCS tree assignment becomes a constraint satisfaction problem (CSP)
Dynamic Programming => time complexity = O(#node)

Learning



$$\text{features}(x, z) = \begin{pmatrix} \textit{in} \cdots\cdots \texttt{loc} & : 1 \\ \boxed{\texttt{city}}\text{-}1\text{—}1\text{-}\boxed{\texttt{loc}} & : 1 \\ \cdots & \end{pmatrix} \in \mathbb{R}^d$$

$$\text{score}(x, z) = \text{features}(x, z) \cdot \theta$$

$$p(z \mid x, \theta) = \frac{e^{\text{score}(x, z)}}{\sum_{z' \in \mathcal{Z}(x)} e^{\text{score}(x, z')}}$$

Learning

Objective Function:

$$\max_{\theta} \sum_{z} \underbrace{p(y \mid z, w)}_{\textbf{Interpretation}} \underbrace{p(z \mid x, \theta)}_{\textbf{Semantic parsing}}$$

Learning

Objective Function:

$$\max_{\theta} \sum_{z} \underbrace{p(y \mid z, w)}_{\textbf{Interpretation}} \; \underbrace{p(z \mid x, \theta)}_{\textbf{Semantic parsing}}$$

EM-like Algorithm:

parameters θ

$$(0.2, -1.3, \ldots, 0.7) \;\; \underset{\text{numerical optimization (L-BFGS)}}{\overset{\text{enumerate/score DCS trees}}{\rightleftarrows}}$$

k-best list

`tree3` ✓
`tree8` ✓
`tree6` ✗
`tree2` ✗
`tree4` ✗

Semantic Forms

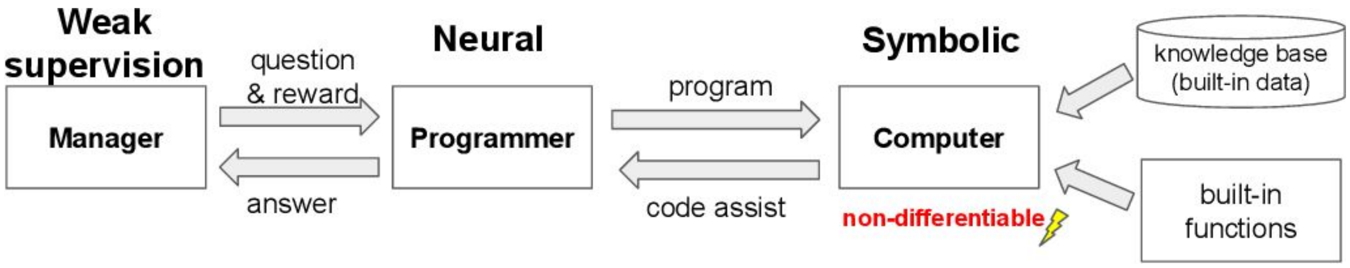
- Combinatory Categorial Grammar
    - Definition
    - Rules
    - Learning
- Dependency-based compositional semantics
    - Definition
    - Rules
    - Learning
- Neural Symbolic Machine (Functional-Program)
    - Definition
    - Learning

Liang, Chen et al. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. 2017
Liang, Chen et al. Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing. 2018
Agarwal et al. Learning to Generalize from Sparse and Underspecified Rewards. 2019

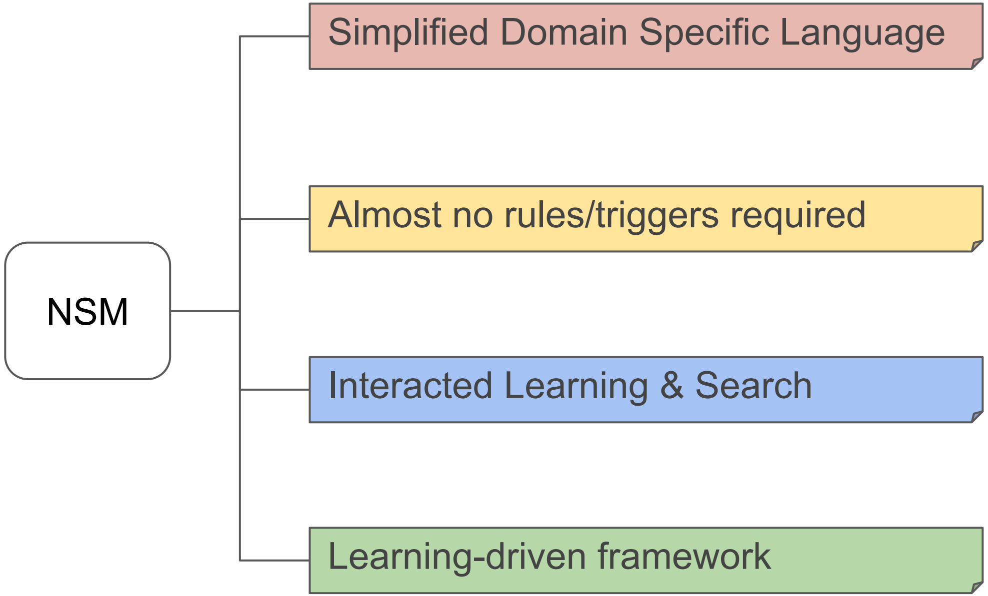
Neural Symbolic Machine

- Weakness of traditional Semantic Parsing
  - The previous methods heavily rely on the predicate mapping, the mapping procedure is based on rules.
  - The coverage is problematic when dealing with more complex utterance.
  - The previous frameworks are dominated by the composition rules enforced as prior, only few parameters to learn.
  - The symbolic executor and the ranking model are separated without interaction.

Neural Symbolic Machine



Neural Symbolic Machine



Domain Specific Language

`GO (Hop M1 ParentOf)` **`(Argmin M2 BornIn)`** `RETURN`

`(Argmin M2 BornIn)`

$$= \{e_1 \mid e_1 \in \texttt{M2}, \exists e_2\colon\ (e_1, \texttt{BornIn}, e_2) \in \mathcal{K}, \forall e \in \texttt{M2}, e_3\colon\ (e, \texttt{BornIn}, e_3) \in \mathcal{K} \Rightarrow e_2 \leq e_3\}$$

$= \{\texttt{m.Malia}\}$

MEMORY	
M1	{m.Obama}
M2	{m.Malia,m.Sasha}
M3	
…	…

$\mathcal{K}$: BornIn	
…	…
m.Obama	1961
m.Malia	1998
m.Sasha	2001
…	…

Domain Specific Language

Evolution

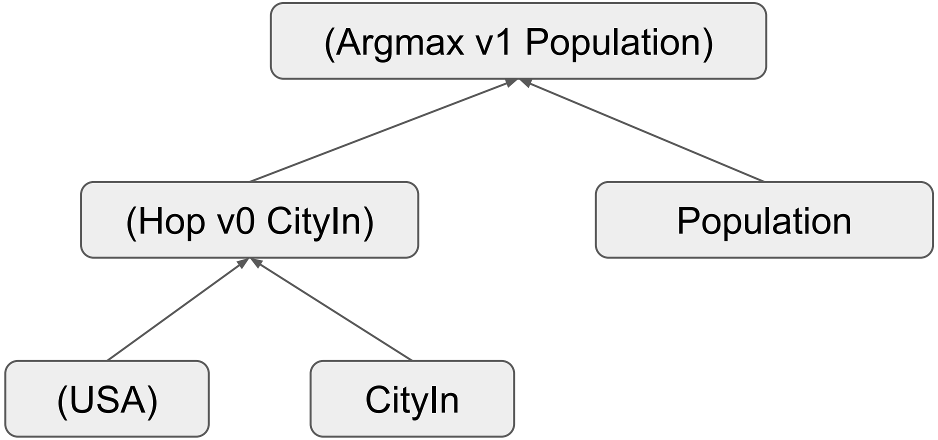
1. CCG/DCS are both general-purpose semantic form.
2. Evolve gradually towards more domain-specific languages.
3. Sacrifice some generalization, but greatly simplify the induction procedure.

Function definitions

- (Hop v p): entities reached from v using relation p
- (ArgHop v1 v2 p): entities in v1 from which v2 can be reached using relation p
- (ArgMin v n): entities in v which have the lowest number in field n
- (ArgMax v n): entities in v which have the highest number in field n
- …
- (Count v): number of entities in v

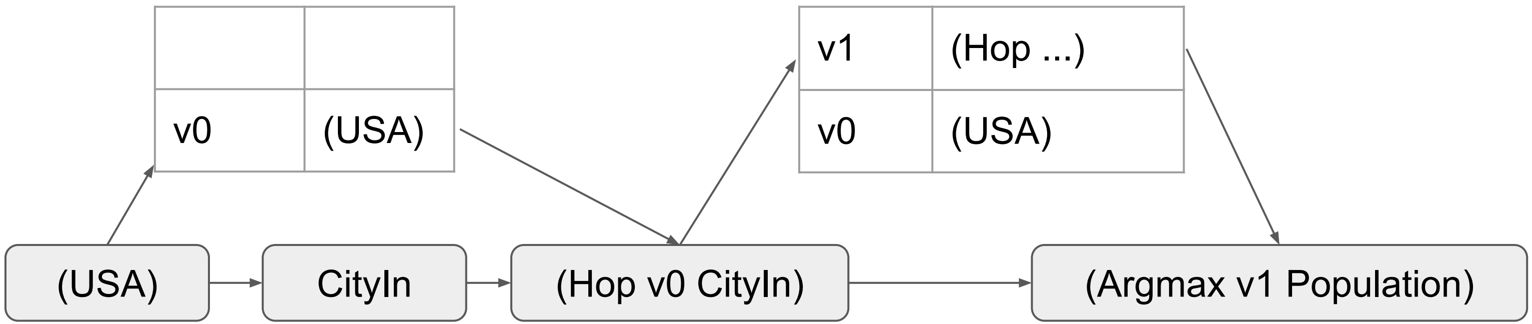
Compositionality

x: Largest city in the US => y: NYC



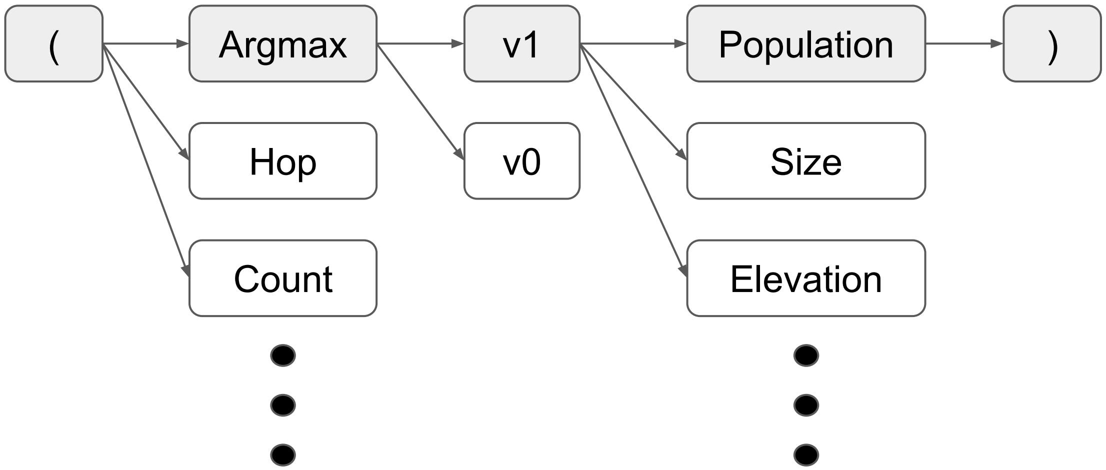
Memory-aided Sequence

x: Largest city in the US => y: NYC

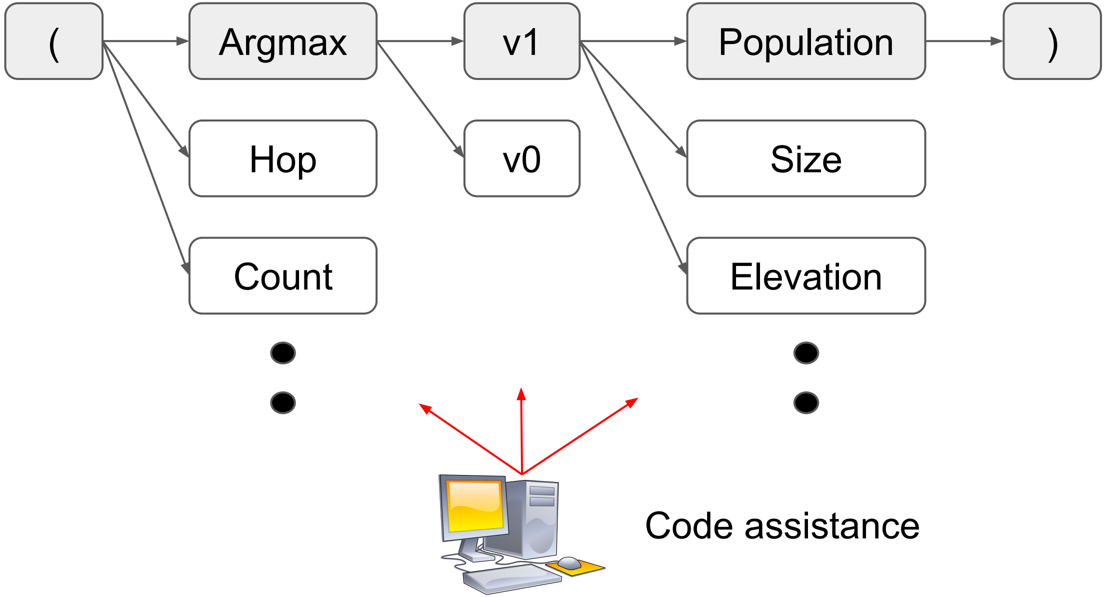


The graph structure is flattened as sequence

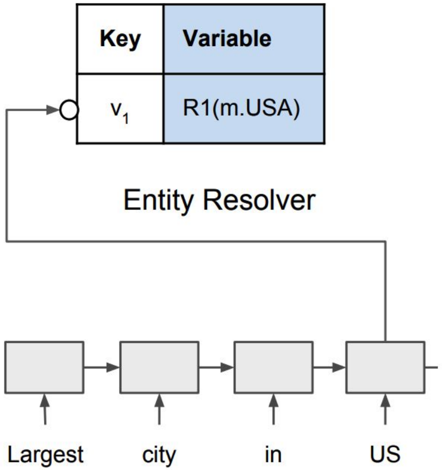
Large Search Space



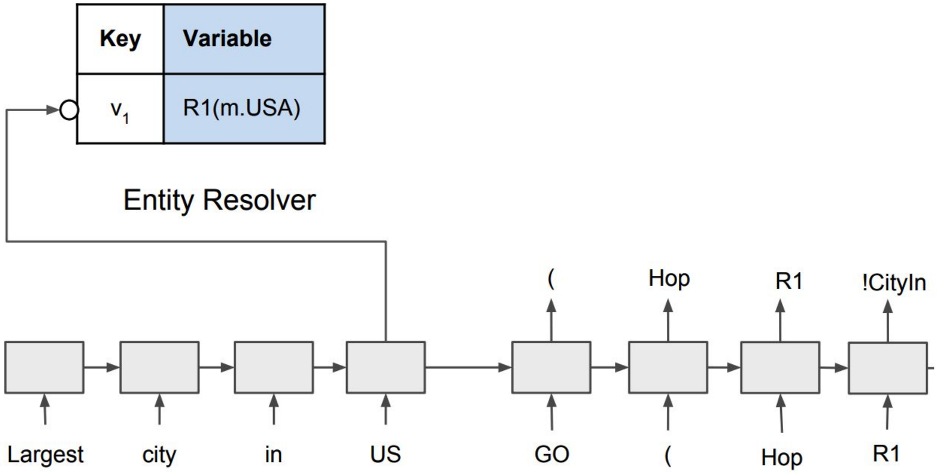
Compiler-aided prune



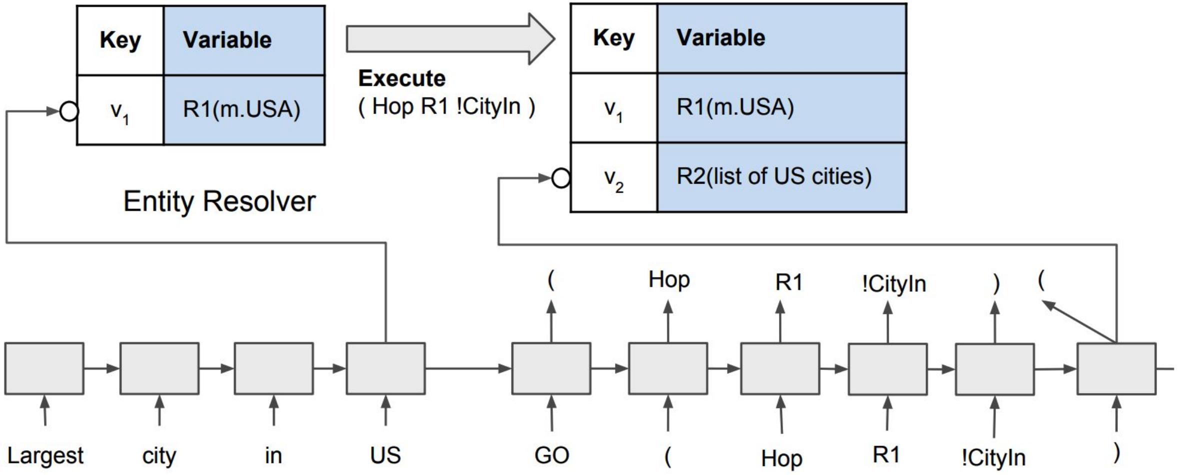
Semantic Parsing as Seq-to-Seq



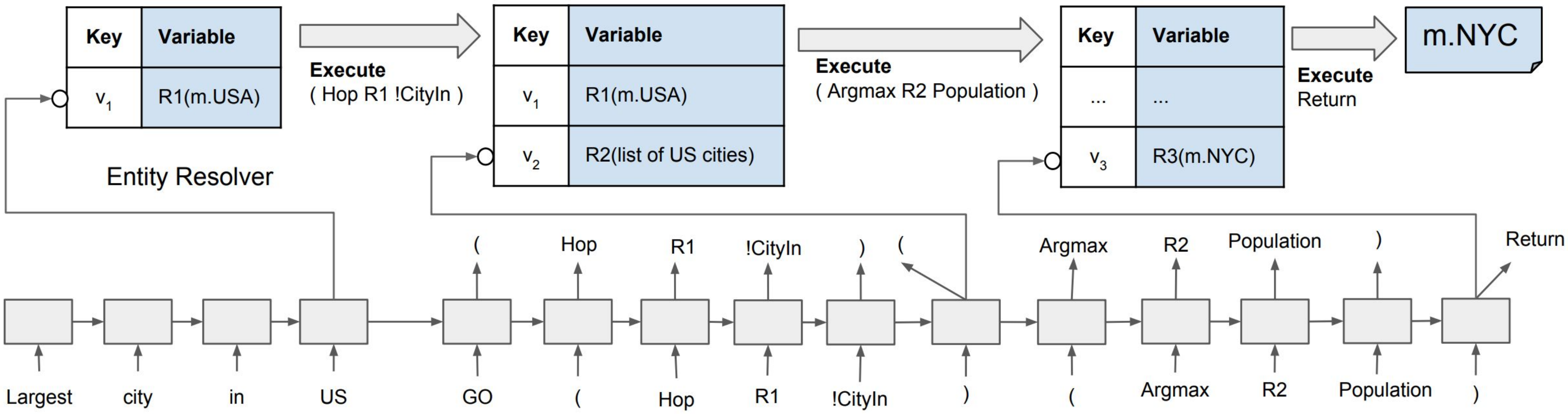
Semantic Parsing as Seq-to-Seq



Semantic Parsing as Seq-to-Seq



Semantic Parsing as Seq-to-Seq



Memory Enc-Dec Model

- Overall model is conditional distribution over all token sequences.

$$P_\theta(\boldsymbol{z}|\boldsymbol{x}) = P_\theta(z_1, \dots, z_T | x_1, \dots, x_N) = P_\theta(z_{1:T} | x_{1:N})$$

$$= P_\theta(z_1 | x_{1:N}) \cdot P_\theta(z_2 | x_{1:N}, z_1) \cdot P_\theta(z_3 | x_{1:N}, z_1, z_2) \cdot \dots \cdot P_\theta(z_T | x_{1:N}, z_{1:T-1})$$

Memory Enc-Dec Model

- Overall model is conditional distribution over all token sequences.

$$P_\theta(\boldsymbol{z}|\boldsymbol{x}) = P_\theta(z_1, \dots, z_T|x_1, \dots, x_N) = P_\theta(z_{1:T}|x_{1:N})$$

$$= P_\theta(z_1|x_{1:N}) \cdot P_\theta(z_2|x_{1:N}, z_1) \cdot P_\theta(z_3|x_{1:N}, z_1, z_2) \cdot \dots \cdot P_\theta(z_T|x_{1:N}, z_{1:T-1})$$

- NSM models the probability at each time step $P_\theta(z_2|x_{1:N}, z_1)$ conditioned on previous output tokens and utterance

Memory Enc-Dec Model

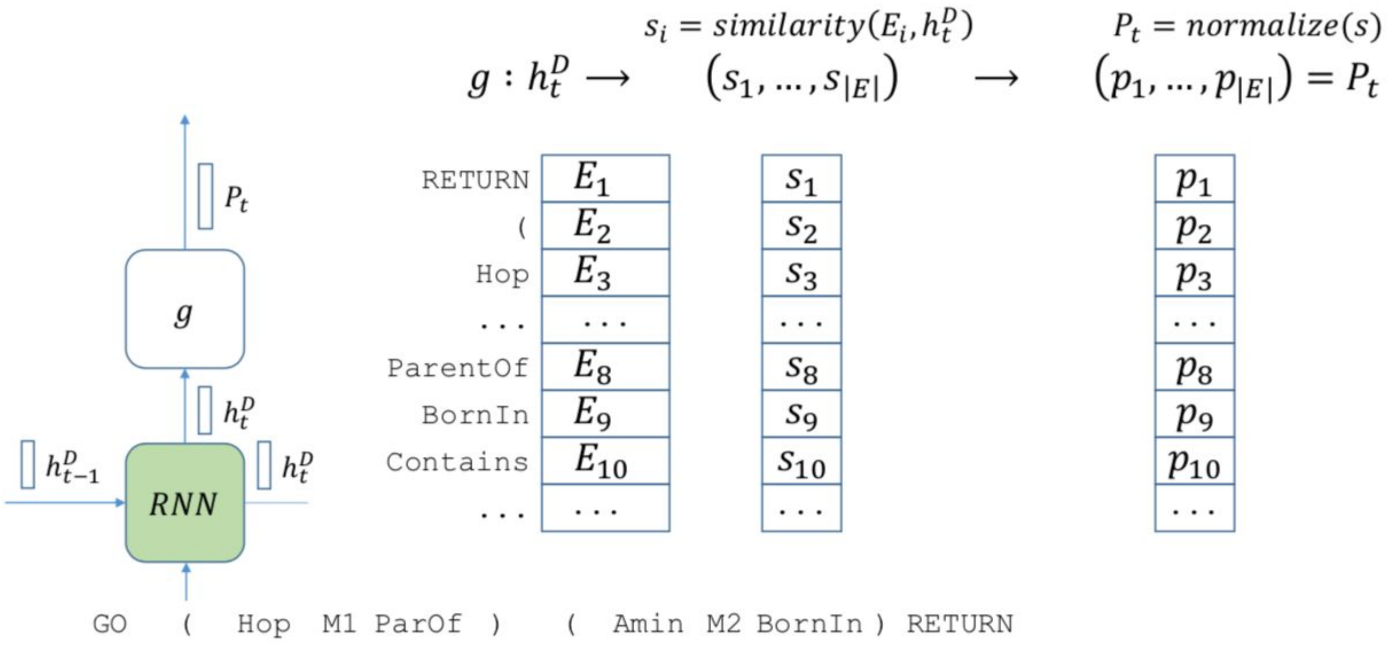
- Overall model is conditional distribution over all token sequences.

$$P_\theta(\boldsymbol{z}|\boldsymbol{x}) = P_\theta(z_1, \ldots, z_T | x_1, \ldots, x_N) = P_\theta(z_{1:T} | x_{1:N})$$

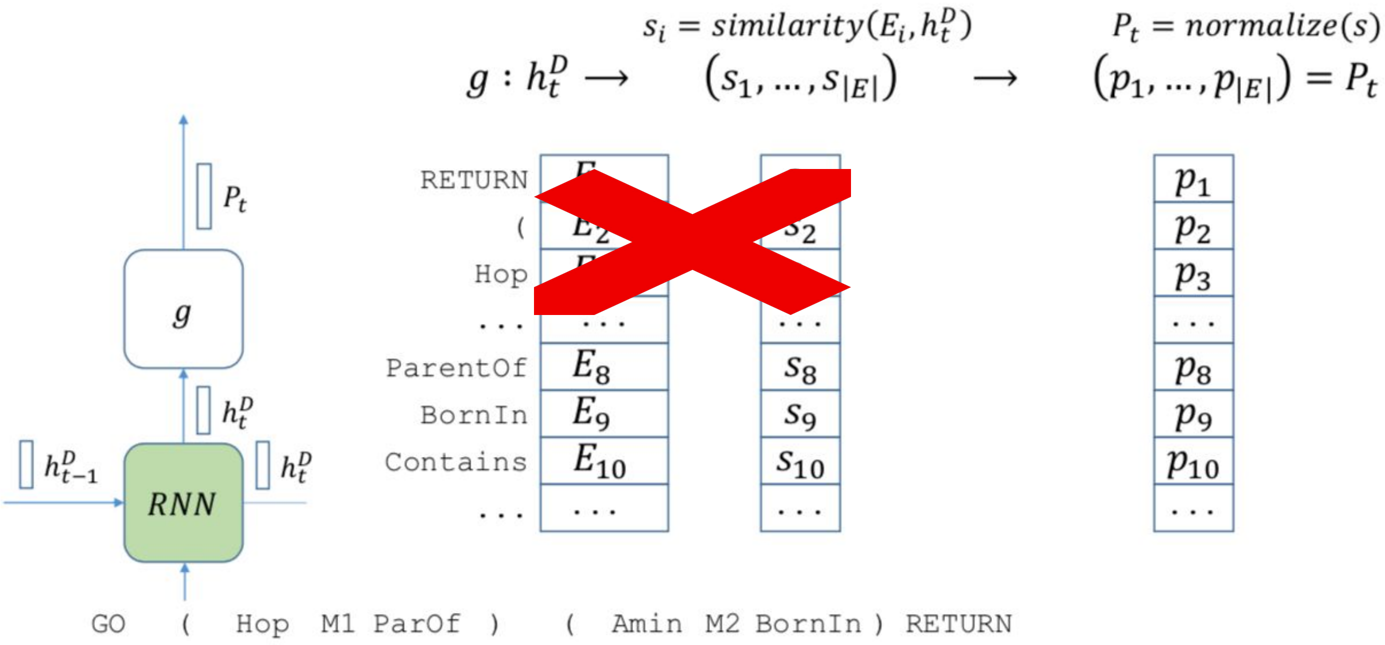
$$= P_\theta(z_1 | x_{1:N}) \cdot P_\theta(z_2 | x_{1:N}, z_1) \cdot P_\theta(z_3 | x_{1:N}, z_1, z_2) \cdot \ldots \cdot P_\theta(z_T | x_{1:N}, z_{1:T-1})$$

- NSM models the probability at each time step $P_\theta(z_2 | x_{1:N}, z_1)$ conditioned on previous output tokens and utterance
- NSM uses interpreter to aid the decoder by removing impossible tokens at each step to dramatically shrink search space.

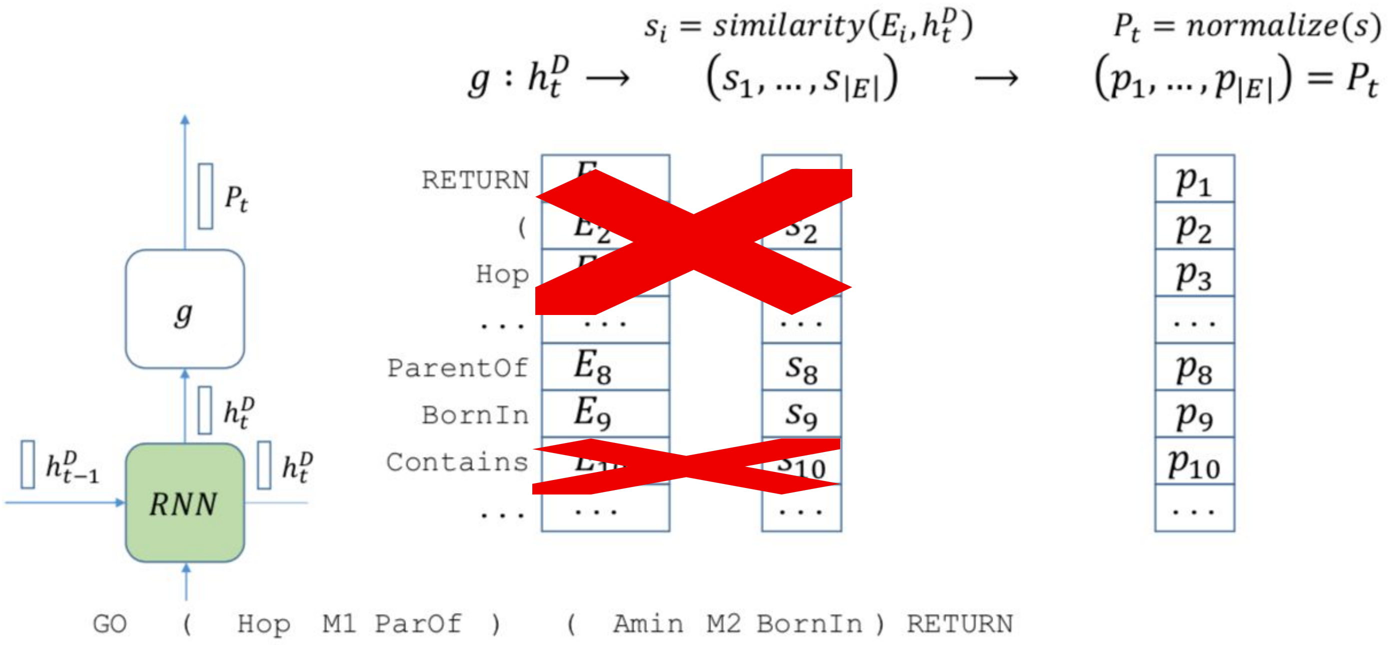
Memory Enc-Dec Model



Syntactic Constraint



Semantic Constraint



Augmented REINFORCE

- REINFORCE (explore)
- Iterative Maximum Likelihood (exploit)

$$\nabla_\theta \mathcal{J}_{\boldsymbol{x}}^{RL}(\theta) = \mathbb{E}_{\boldsymbol{z} \sim P_\theta(\cdot|\boldsymbol{x})}[\nabla_\theta \log P_\theta(\boldsymbol{z}|\boldsymbol{x}) \cdot F_1(y^{\boldsymbol{z}}, y^*)]$$

$$\approx 0.9 \cdot \frac{1}{\lambda} \sum_{n=1}^{k} P_\theta(\boldsymbol{z}_n|\boldsymbol{x}) \cdot \nabla_\theta \log P_\theta(\boldsymbol{z}_n|\boldsymbol{x}) \cdot F_1(y^{\boldsymbol{z}_n}, y^*) \quad +$$

$$0.1 \cdot \quad P_\theta(\boldsymbol{z}_{best}|\boldsymbol{x}) \cdot \nabla_\theta \log P_\theta(\boldsymbol{z}_{best}|\boldsymbol{x}) \cdot F_1(y^{\boldsymbol{z}_{best}}, y^*)$$

$$\{\boldsymbol{z}_n\} = k.\operatorname{argmax}_{\boldsymbol{z}} P_\theta(\boldsymbol{z}|\boldsymbol{x}), \quad \lambda = \Sigma_i P_\theta(\boldsymbol{z}_n|\boldsymbol{x})$$

Wrap up

- Formal Semantics does have many advantages like strong compositionality, explainability.
- However, the annotation effort or rigid structure makes it hard to scale up to large-scale domains or more realistic datasets.
- The vectorized representation in Deep Learning is still the favorable by the community.
- How to neuralize the formal semantics and apply it to modern architecture remains an open problem.

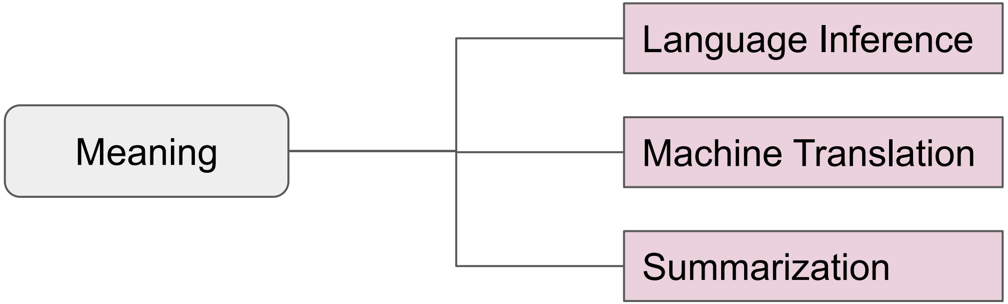
My Work

1. Natural Language Understanding on Structured Data
    a. Relation Question Answering on Knowledge Graph (Chen et al. NAACL 18)
    b. Table-based Fact Verification on semi-structured table (Chen et al. Arxiv)
    c. Multi-model Question Answering on structured scene graph (Chen et al. Arxiv)

2. Natural Language Generation on Structured Data
    a. Semantically conditioned dialog generation on structured semantic form (Chen et al. ACL 19)

Open Question

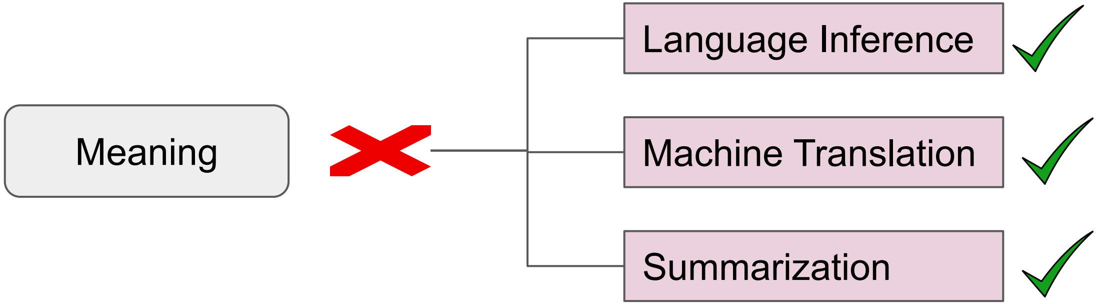
- Philosophical problem



Meaning serves for the success of downstream tasks

Open Question

- Philosophical problem



If we can already achieve high accuracy, why do we still need to care about meaning?