CS11-747 Neural Networks for NLP

Neural Semantic Parsing

Graham Neubig

Carnegie Mellon University

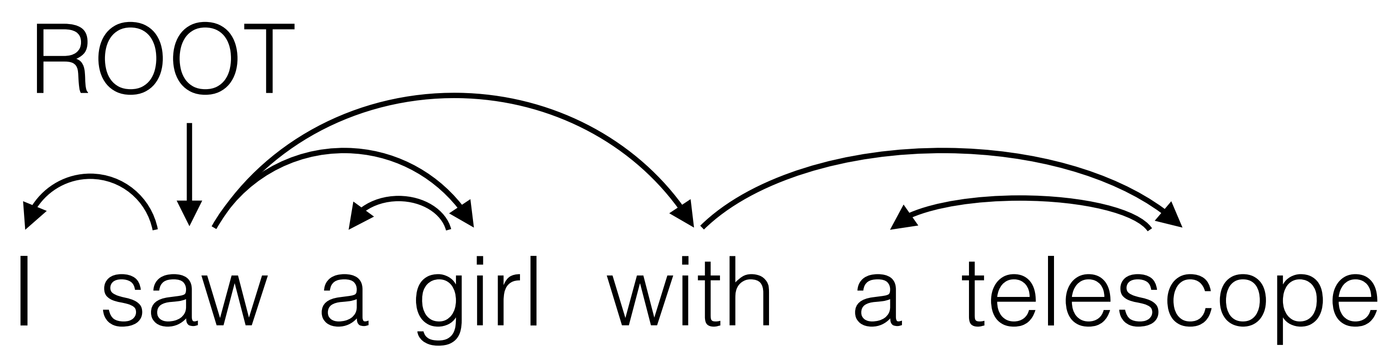
Language Technologies Institute

Site

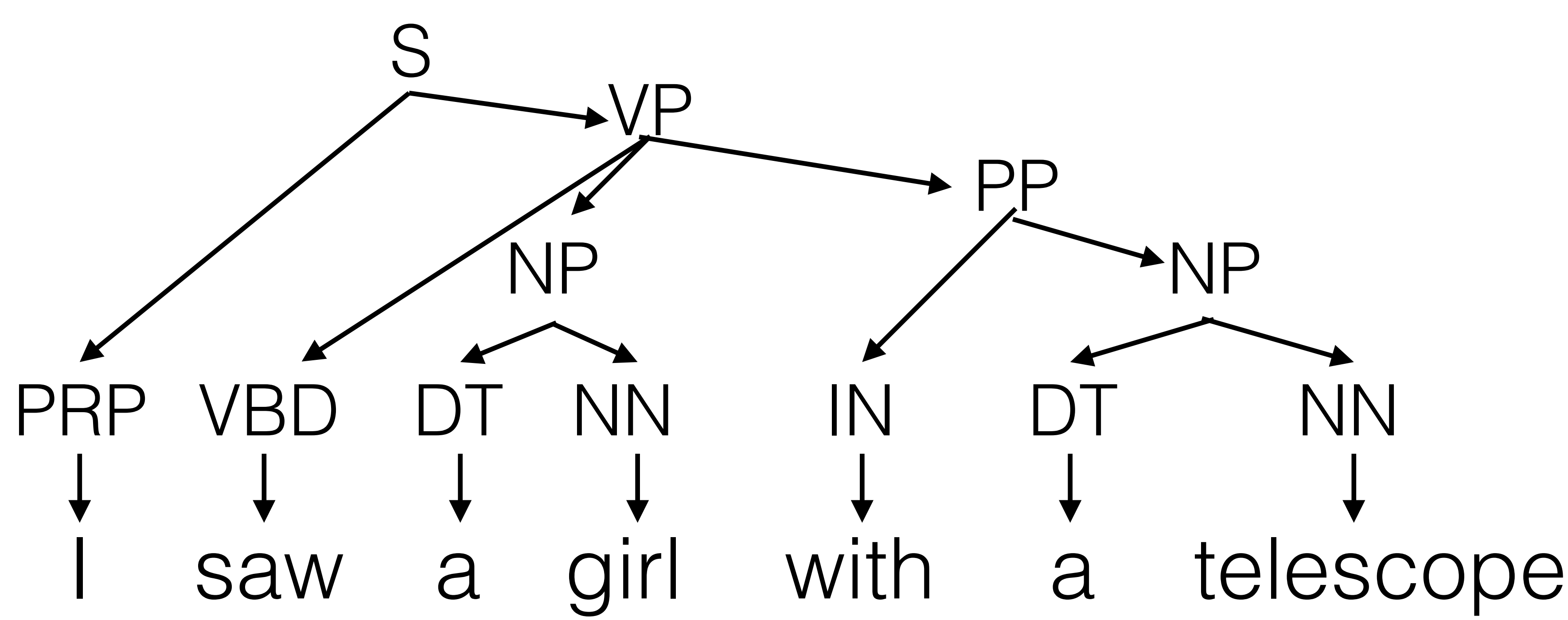
https://phontron.com/class/nn4nlp2017/

Tree Structures of Syntax

- **Dependency:** focus on relations between words



- **Phrase structure:** focus on the structure of the sentence



Representations of Semantics

- Syntax only gives us the sentence structure
- We would like to know what the **sentence really means**
- Specifically, in an **grounded and operationalizable way**, so a machine can
  - Answer questions
  - Follow commands
  - etc.

Meaning Representations

- **Special-purpose representations:** designed for a specific task
- **General-purpose representations:** designed to be useful for just about anything
- **Shallow representations:** designed to only capture part of the meaning (for expediency)

Parsing to Special-purpose Meaning Representations

Example Special-purpose Representations

- A database query language for sentence understanding
- A robot command and control language
- Source code in a language such as Python (?)

Example Query Tasks

- **Geoquery:** Parsing to Prolog queries over small database (Zelle and Mooney 1996)

x: "*what is the population of iowa ?*"
y: `_answer ( NV , ( _population ( NV , V1 ) , _const ( V0 , _stateid ( iowa ) ) ) )`

- **Free917:** Parsing to Freebase query language (Cai and Yates 2013)

1. What are the neighborhoods in New York City?
   λx . neighborhoods(new_york, x)
2. How many countries use the rupee?
   count(x) . countries_used(rupee, x)

- Many others: WebQuestions, WikiTables, etc.

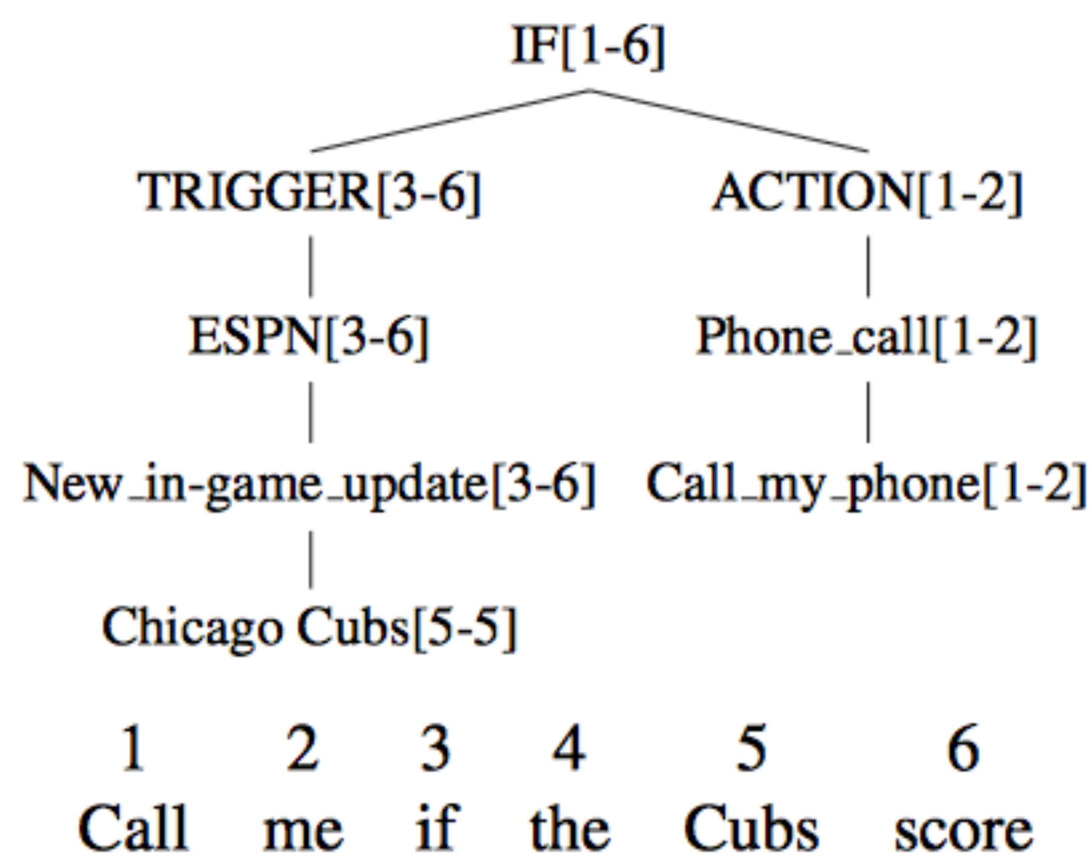
Example Command and Control Tasks

- **Robocup**: Robot command and control (Wong and Mooney 2006)

```
((bowner our {4})
 (do our {6} (pos (left (half our)))))
```

If our player 4 has the ball, then our player 6 should stay in the left side of our half.

- **If this then that:** Commands to smartphone interfaces (Quirk et al. 2015)



Example Code Generation Tasks

- Hearthstone cards (Ling et al. 2015)


```
class DivineFavor(SpellCard):
    def __init__(self):
        super().__init__("Divine Favor", 3,
    CHARACTER_CLASS.PALADIN, CARD_RARITY.RARE)

    def use(self, player, game):
        super().use(player, game)
        difference = len(game.other_player.hand)
       - len(player.hand)
        for i in range(0, difference):
            player.draw()
```

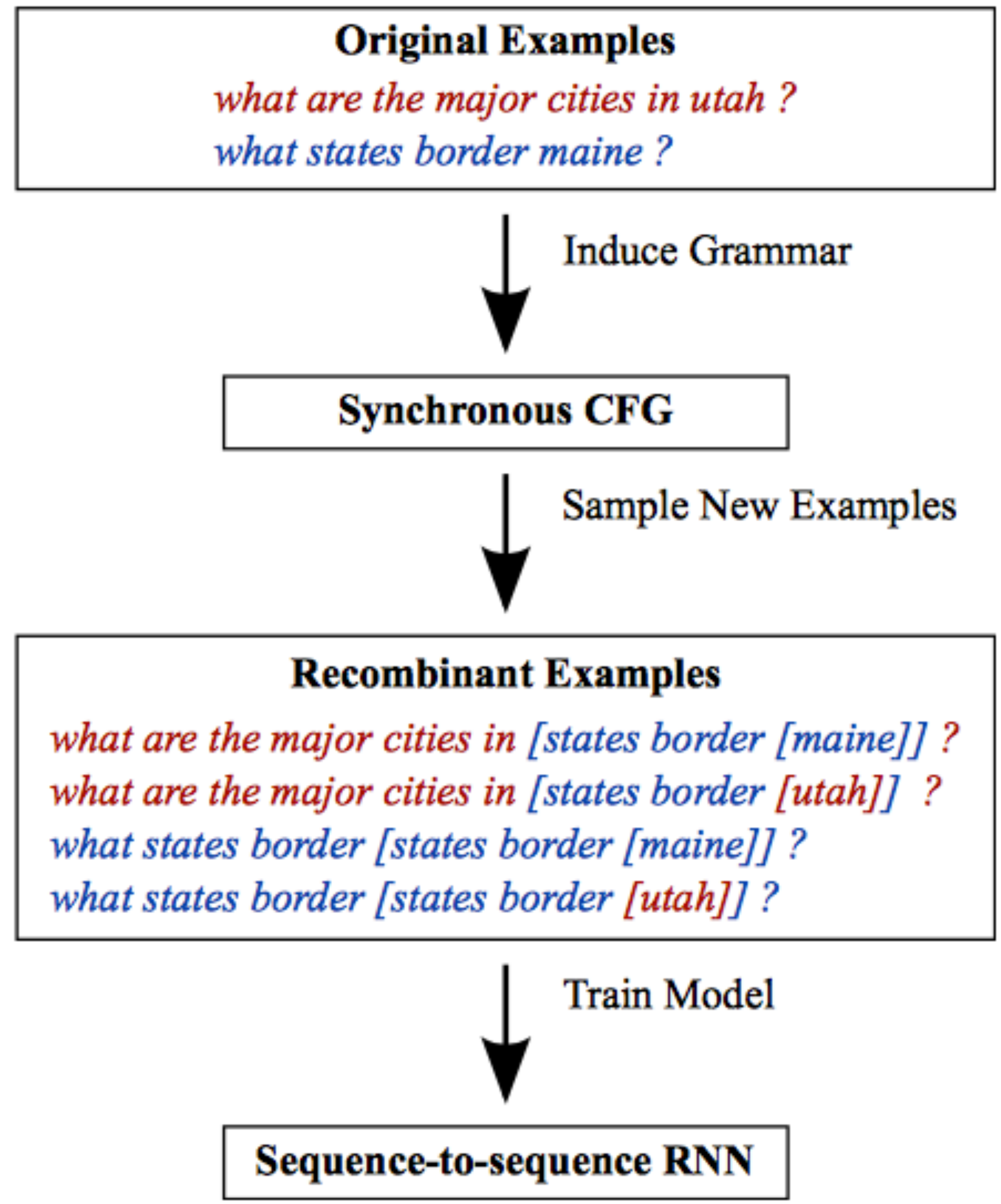
- Django commands (Oda et al. 2015)

convert cull_frequency into an integer and substitute it for self._cull_frequency.

```
self._cull_frequency = int(cull_frequency)
```

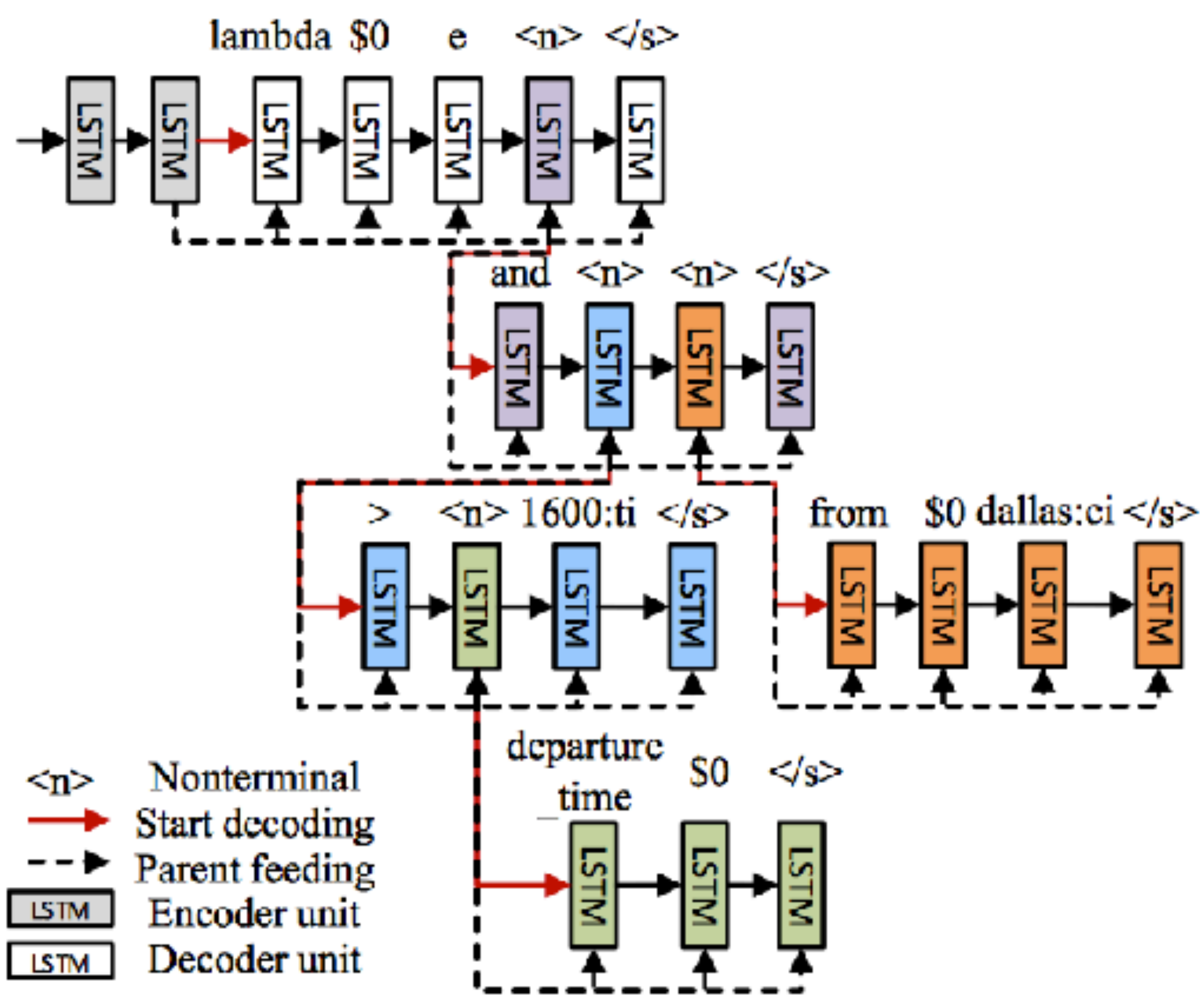
A First Attempt: Sequence-to-sequence Models (Jia and Liang 2016)

- Simple string-based sequence-to-sequence model
- Doesn't work well as-is, so generate extra synthetic data from a CFG



A Better Attempt: Tree-based Parsing Models

- Generate from top-down using hierarchical sequence-to-sequence model (Dong and Lapata 2016)



Query/Command Parsing: Learning from Weak Feedback

- Sometimes we don't have annotated logical forms

- Treat logical forms as a latent variable, give a boost when we get the answer correct (Clarke et al 2010)

x: What is the largest state that borders Texas?

y:

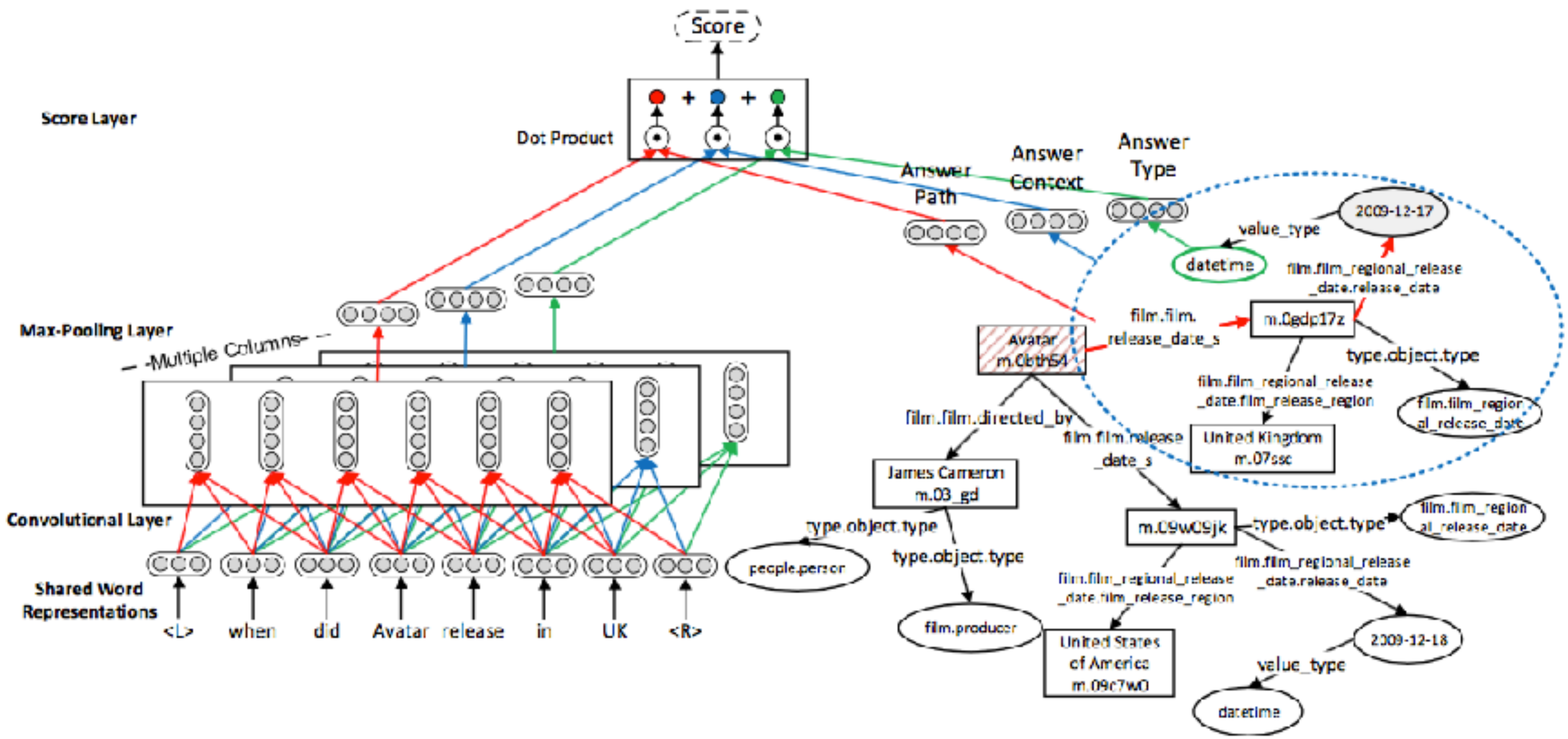
z: `largest(state(next_to(const(texas))))` **Latent**

r: **New Mexico**

- Can be framed as a reinforcement learning problem (more in a couple weeks)
- Problems: spurious logical forms that get the correct answer but are not right (Guu et al. 2017), unstable training

Large-scale Query Parsing: Interfacing w/ Knowledge Bases

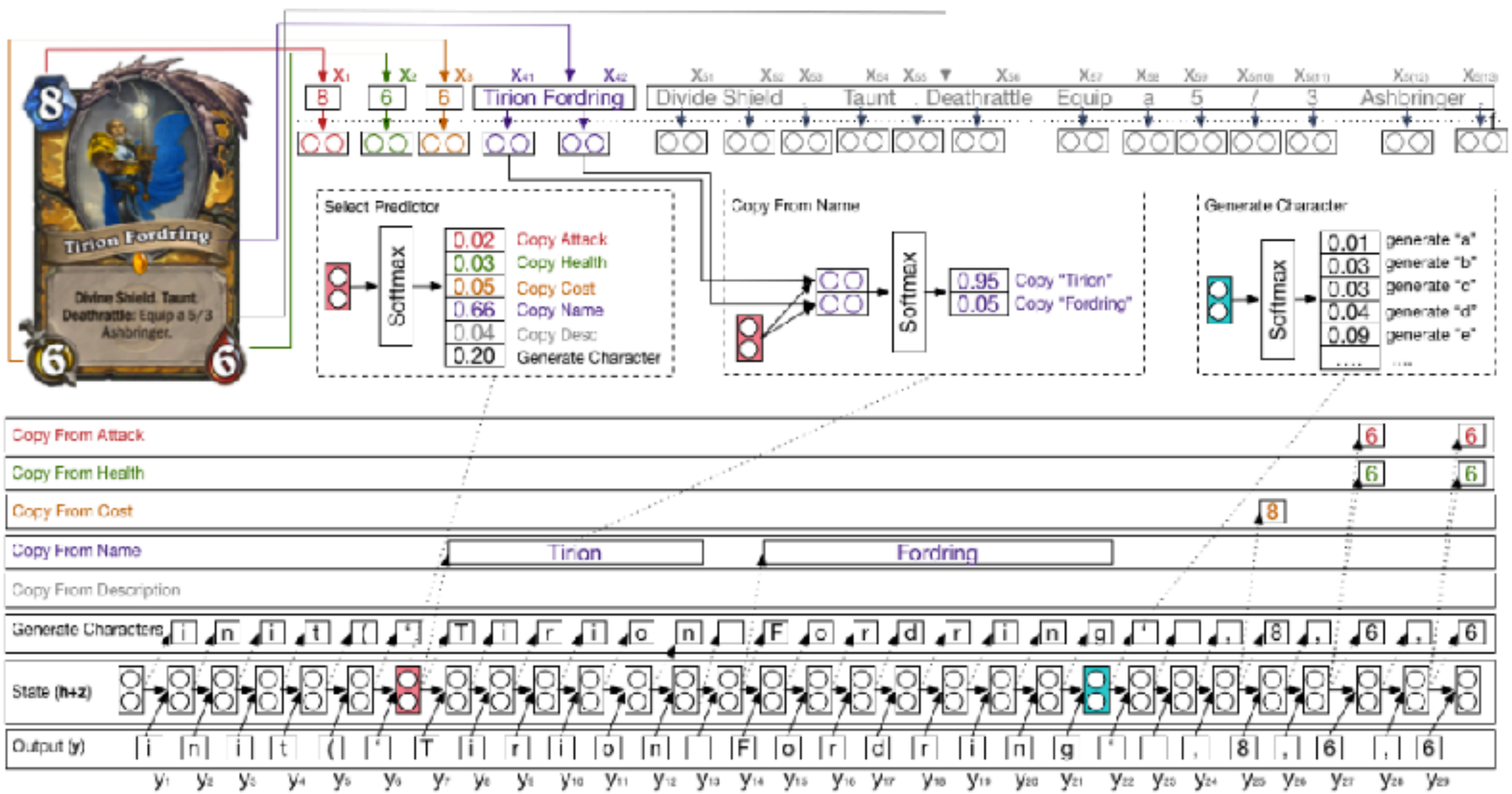
- Encode features of the knowledge base using CNN and match against current query (Dong et al. 2015)



- (More on knowledge bases in a month or so)

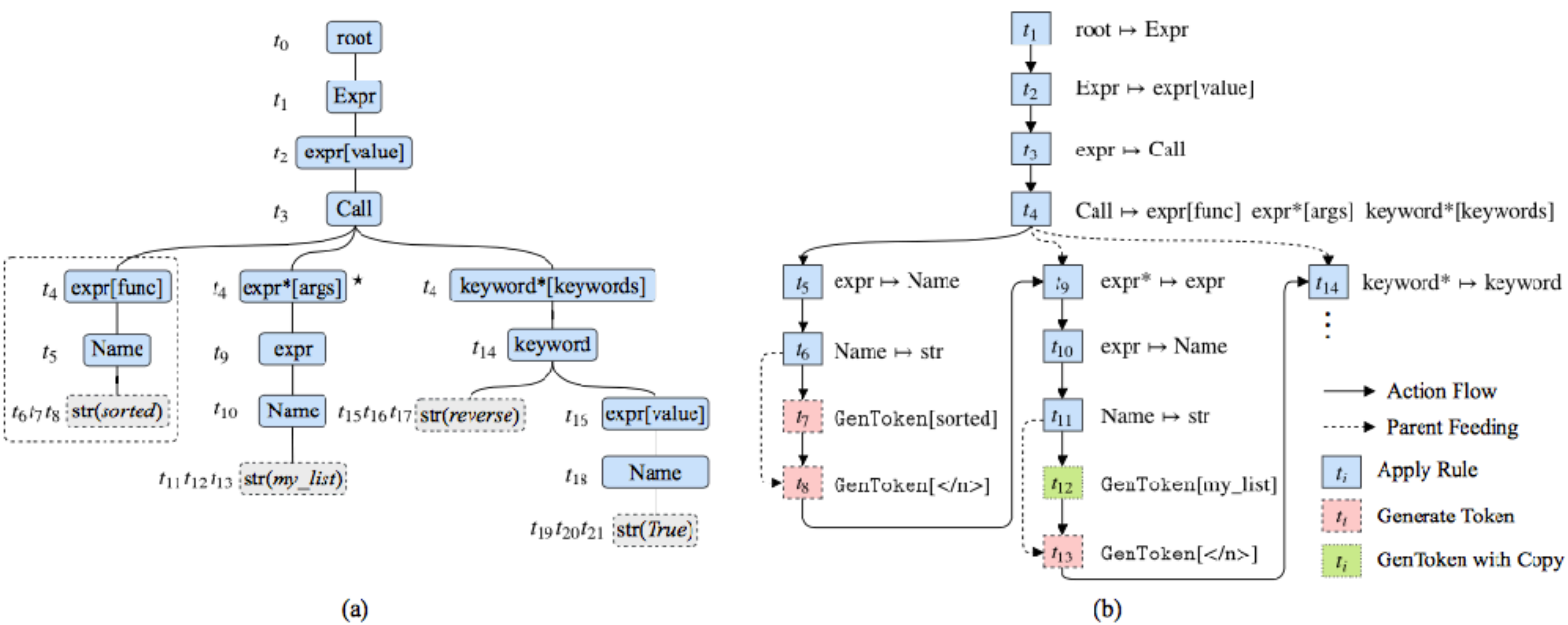
Code Generation: Character-based Generation+Copy

- In source code (or other semantic parsing tasks) there is a significant amount of copying
- **Solution:** character-based generation+copy, w/ clever independence assumptions to make training easy (Ling et al. 2016)



Code Generation: Handling Syntax

- Code also has syntax, e.g. in form of Abstract Syntax Trees (ASTs)
- Tree-based model that generates AST obeying code structure and using to modulate information flow (Yin and Neubig 2017)



Input: `sort my_list in descending order`

Code: `sorted(my_list, reverse=True)`

General-purpose Meaning Representation

Meaning Representation Desiderata (Jurafsky and Martin 17.1)

- **Verifiability:** ability to ground w/ a knowledge base, etc.
- **Unambiguity:** one representation should have one meaning
- **Canonical form:** one meaning should have one representation
- **Inference ability:** should be able to draw conclusions
- **Expressiveness:** should be able to handle a wide variety of subject matter

First-order Logic

- Logical symbols, connective, variables, constants, etc.
  - There is a restaurant that serves Mexican food near ICSI.
    `∃xRestaurant(x)∧ Serves(x,MexicanFood)∧`
    `Near((LocationOf(x),LocationOf(ICSI))`
  - All vegetarian restaurants serve vegetarian food.
    `∀xVegetarianRestaurant(x) ⇒`
    `Serves(x,VegetarianFood)`
- Lambda calculus allows for expression of functions
  `λx.λy.Near(x,y)(Bacaro)`
  `λy.Near(Bacaro,y)`

Abstract Meaning Representation

(Banarescu et al. 2013)

- Designed to be simpler and easier for humans to read

- Graph format, with arguments that mean the same thing linked together

- Large annotated sembank available

LOGIC format:

$\exists$ w, b, g:
instance(w, want-01) $\wedge$ instance(g, go-01) $\wedge$
instance(b, boy) $\wedge$ arg0(w, b) $\wedge$
arg1(w, g) $\wedge$ arg0(g, b)

AMR format (based on PENMAN):

```
(w / want-01
  :arg0 (b / boy)
  :arg1 (g / go-01
          :arg0 b))
```

GRAPH format:

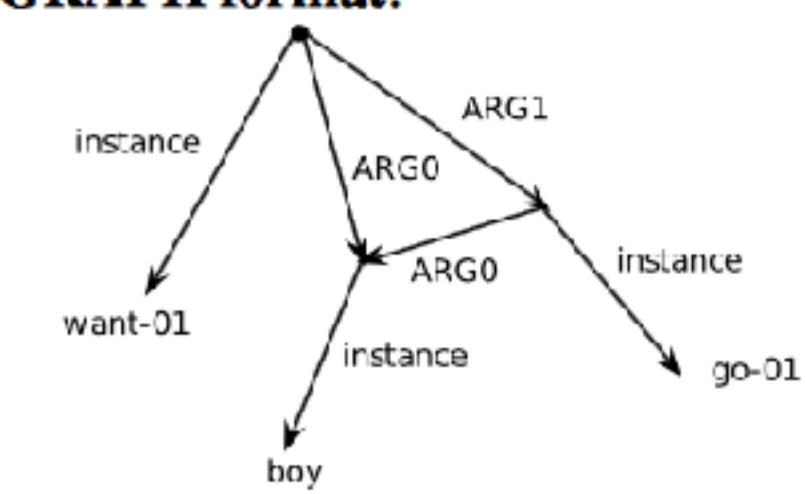
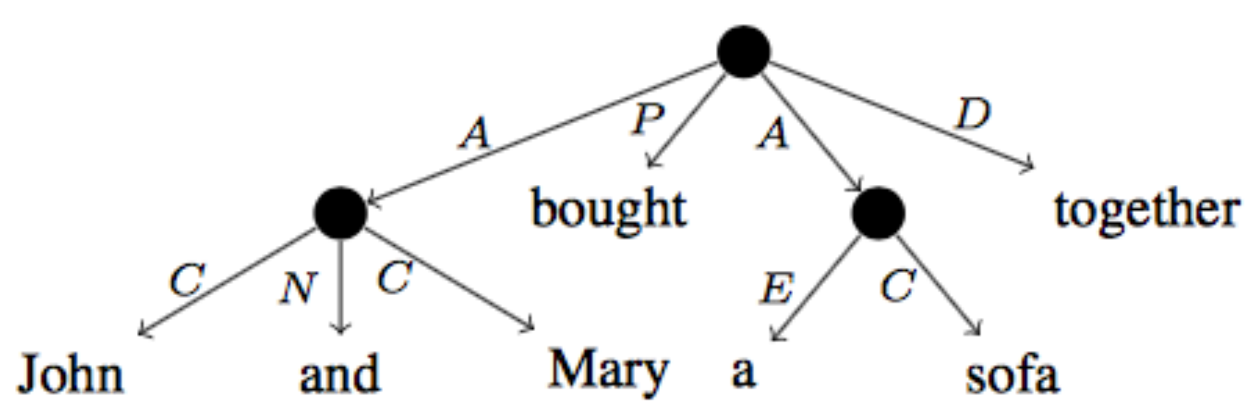


Figure 1: Equivalent formats for representating the meaning of "The boy wants to go".

Other Formalisms

- **Minimal recursion semantics** (Copestake et al. 2005): variety of first-order logic that strives to be as flat as possible to preserve ambiguity

- **Universal conceptual cognitive annotation** (Abend and Rappoport 2013): Extremely course-grained annotation aiming to be universal and valid across languages

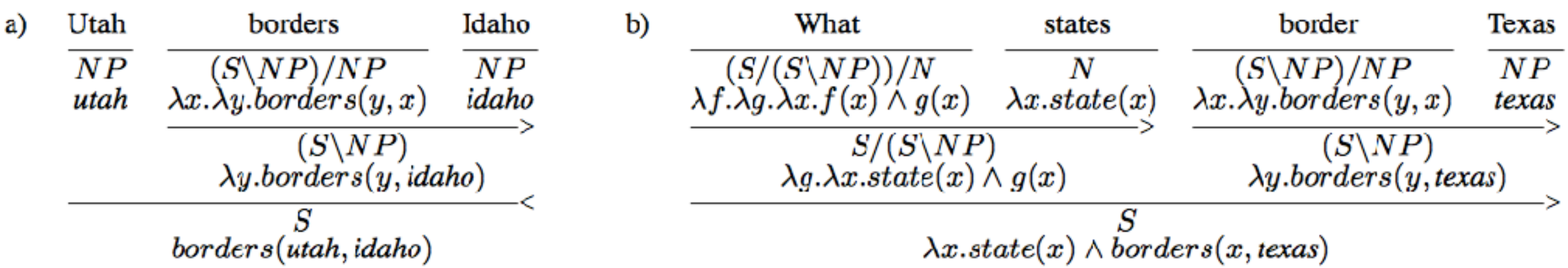


Syntax-driven Semantic Parsing

- Parse into syntax, then convert into meaning
- CFG → first order logic (e.g. Jurafsky and Martin 18.2)
- Dependency → first order logic (e.g. Reddy et al. 2017)
- Combinatory categorial grammar (CCG) → first order logic (e.g. Zettlemoyer and Collins 2012)

CCG and CCG Parsing

- CCG a simple syntactic formalism with strong connections to logical form
- Syntactic tags are combinations of elementary expressions (S, N, NP, etc)



- Strong syntactic constraints on which tags can be combined
- Much weaker constraints than CFG on what tags can be assigned to a particular word

Supertagging

- Basically, tagging with a very big tag set (e.g. CCG)
- If we have a strong super-tagger, we can greatly reduce CCG ambiguity to the point it is deterministic
- Standard LSTM taggers w/ a few tricks perform quite well, and improve parsing (Vaswani et al. 2017)
    - Modeling the compositionality of tags
    - Scheduled sampling to prevent error propagation

Parsing to Graph Structures

- In many semantic representations, would like to parse to directed acyclic graph
- Modify the transition system to add special actions that allow for DAGs
  - “Right arc” doesn’t reduce for AMR (Damonte et al. 2017)
  - Add “remote”, “node”, and “swap” transitions for UCCA (Hershcovich et al. 2017)
- Perform linearization and insert pseudo-tokens for re-entry actions (Buys and Blunsom 2017)

Shallow Semantics

Semantic Role Labeling
(Gildea and Jurafsky 2002)

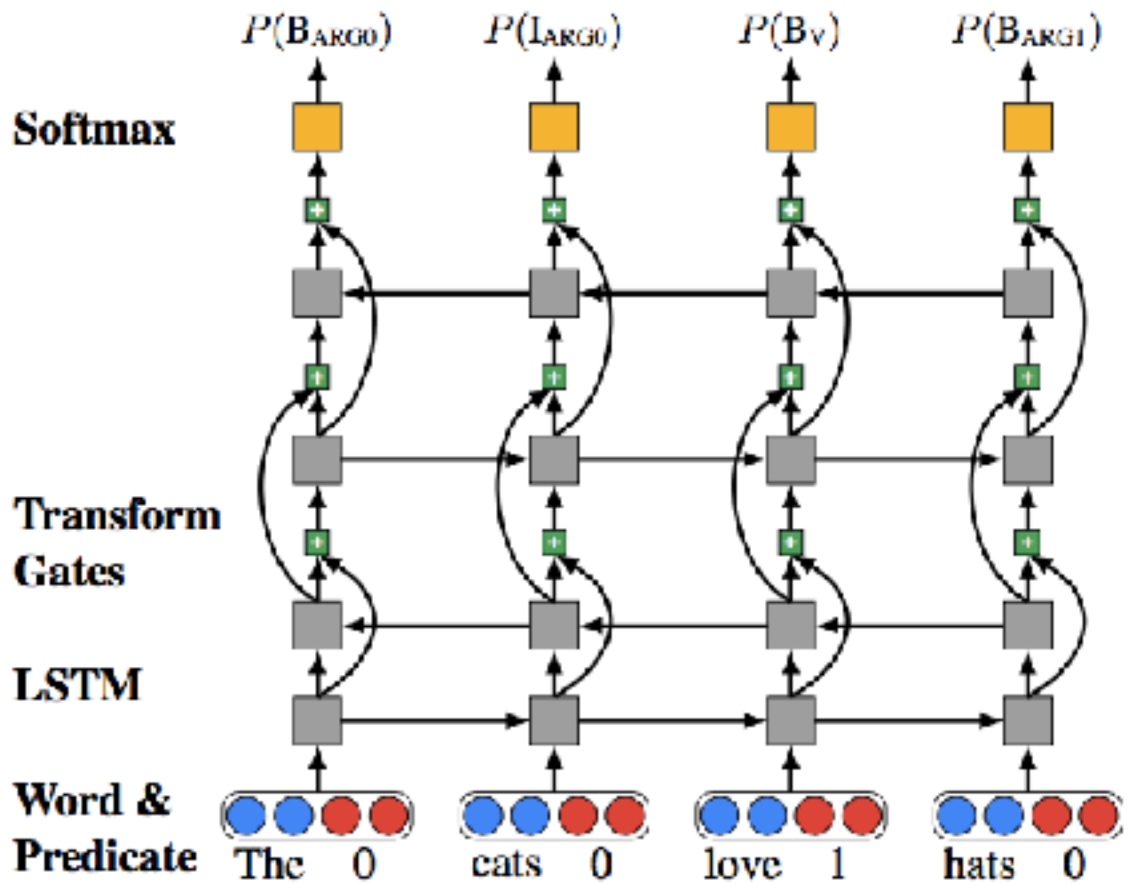
- Label “who did what to whom” on a span-level basis

(1) [$_{Judge}$ She] **blames** [$_{Evaluee}$ the Government] [$_{Reason}$ for failing to do enough to help] .

(2) [$_{Message}$ “I’ll knock on your door at quarter to six”] [$_{Speaker}$ Susan] **said**.

Neural Models for Semantic Role Labeling

- Simple model w/ deep highway LSTM tagger works well (Le et al. 2017)



- Error analysis showing the remaining challenges

Questions?