4 Learning from Entailment for Semantic Parsing

"The basic aim of semantics is to characterize the notion of a true sentence (under a given interpretation) and of entailment."

– Richard Montague, *Universal Grammar (1970)*

4.1 Modeling Entailment for Semantic Parsing

4.1.1 The Idea

Throughout this thesis, we have treated semantic parsing as primarily a translation problem, which we have studied independently of the other subtasks (i.e., knowledge representation and symbolic reasoning) associated with the general NLU program outlined in Chapter 1. It is worth bearing in mind, however, that the ultimate goal, as described in the quote above by Montague, is to generate formal meaning representations that capture facts about truth and entailment and facilitate deep symbolic reasoning. In this chapter, we examine the following question: do the formal representations being learned for semantic parsing actually help us to model entailment, and if not, how can we learn representations that do?

To illustrate this idea, Figure 4.1 shows a variant of the pipeline model introduced in Chapter 1 that includes the following sentence (in red) that is *logically entailed* by the first sentence (for details about entailment, see Appendix C):

Find some sample that contains a major element. (4.1)

A consequence of this logical entailment is that the denotation of the second sentence (i.e., the set of answers that make this sentence true) should always be a subset of the denotation of the first sentence, regardless of the target dataset or knowledge source being used. Linguists in the Montague tradition have long used

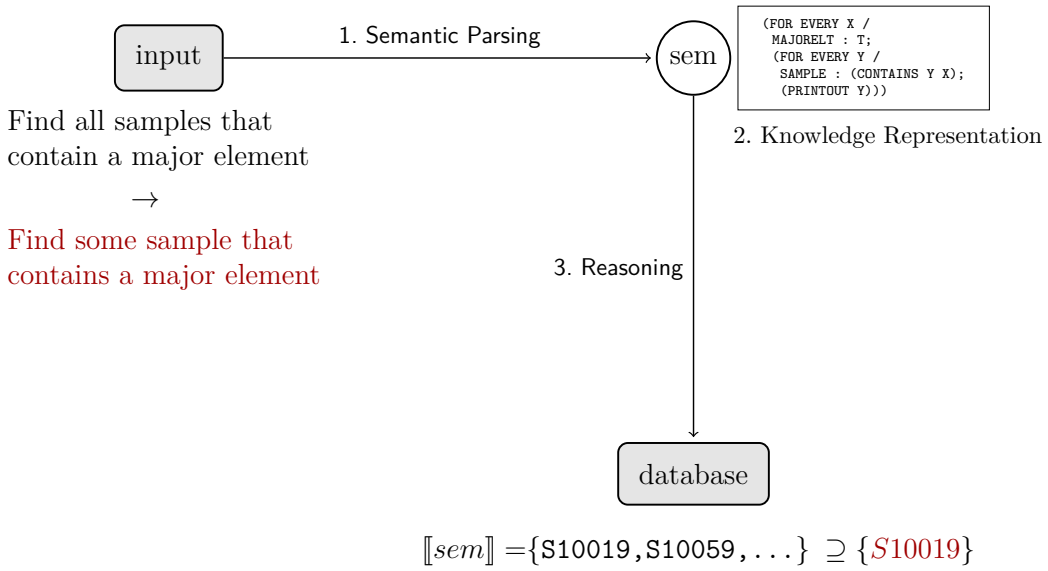


Figure 4.1: The global NLU picture with entailment.

judgements about entailment as the main tool for motivating and evaluating different theories of semantics. Using an analogy with programming and software, we can think about tests of entailment in semantics as a kind of *unit test* for system development, as described below:

- **Entailment as a Unit Test**: For a set of target sentences, check that our semantic model accounts for particular entailment patterns observed between pairs of sentences; modify our model when such tests fail.

The question investigated here is: what happens when we subject our semantic parsers to such a unit test? In doing this, we adopt the loose definition of entailment used in the recognizing textual entailment challenges (RTE), where entailment is defined in terms of the following task (Dagan et al., 2005): given a **text** t and **hypothesis** h, determine if h is entailed by t where say that t entails h if a human reading t would typically infer that h is most likely true. Figure 4.2 shows example sentence pairs and logical forms (or LFs, generated by a semantic parser) from the Sportscaster corpus (Chen and Mooney, 2008a) already encountered in

			Entailments	
	Text t and gold LF	**Hypothesis** h and gold LF	**Human**	**Naïve**
1.	Pink 3 quickly kicks to Pink 7 `pass(pink3,pink7)`	Pink 3 kicks over to Pink 7 `pass(pink3,pink7)`	t `(entail)` h h `(uncertain)` t	`entail`
2.	Purple 10 kicks the ball `kick(purple10)`	Purple 10 shoots for the goal `kick(purple10)`	t `(uncertain)` h h `(entail)` t	`entail`
3.	Pink 10 kicks the ball `kick(pick10)`	Pink 10 passes over to Pink 7 `pass(pink10,pink7)`	t `(uncertain)` h h `(entail)` t	`contr.`
4.	Pink 7 makes a long kick `kick(purple7)`	Purple team scores another goal `playmode(goal_l)`	t `(uncertain)` h h `(uncertain)` t	`contr.`

Figure 4.2: Example sentence pairs and entailments in the Sportscaster domain.

Chapter 3. Each example is marked with an entailment judgement provided by humans in both the $t \rightarrow h$ and $h \rightarrow t$ directions. For example, in Figure 4.2-1, we can paraphrase the entailment from $t \rightarrow h$ in the following way:

> In all scenarios (e.g., possible game events) in which 'pink 3 quickly kicks to pink 7' is true, it is always simultaneously true (or nearly always true) that 'pink 3 kicks over to pink 7'

Subjecting our semantic parsers to an RTE test involves seeing if the semantic LF representations being generated can be used to derive and identify such correct entailments (i.e., entailments that are consistent with human judgements).

4.1.2 Yet Another Resource Problem!

The problem with the corpus LFs in Figure 4.2, however, is that while they capture the general events being discussed, they often fail to capture other aspects of meaning. Here, the **naïve** judgement is the entailment generated by comparing the LFs associated with t and h (i.e., by assigning `entail` when the LFs match, and `contradict` when they mismatch), which captures the full inferential power of the target LFs. In several cases, the naïve inferences result in judgements that are inconsistent with the human judgements. Therefore, some of the semantic representations provided in the corpus fail to pass the test described above.

In considering these examples, we identify the following two issues:

1. **Imprecise Corpus Representations**: The corpus representations fail to account for certain aspects of meaning. For example, the first two sentences in Figure 4.2-1 map to the same formal meaning representation (i.e., `pass(pink3,pink7)`) despite having slightly different semantics and divergent entailment patterns. This shift in meaning is related to the adverbial modifier *quickly*, which is not explicitly analyzed in the target representation. The same is true for the modifier *long* in example 4, and for all other forms of modification. For a semantic parser or generator trained on this data, both sentences in 1 are treated as having an identical meaning.

   As shown in the example 2, other representations fail to capture important sense distinctions, such as the difference between the two senses of the `kick` relation. While *shooting for the goal* in general entails *kicking*, such an entailment does not hold in the reverse direction. Without making this distinction explicit at the representation level, such inferences and distinctions cannot be made.

2. **Missing Domain Knowledge** Since the logical representations are not based on an underlying logical theory or domain ontology, semantic relations between different symbols are not known. For example, computing the entailments in example 3 requires knowing that in general, a `pass` event entails or implies a `kick` event (i.e., the set of things *kicking* at a given moment includes the set of things *passing*). Other such factoids are involved in reasoning about the sentences in example 4: `purple7` is part of the `purple team`, and a `score` event usually entails a `kick` event (but not conversely).

The more general resource problem involved here can be described in the following way: while we have a sufficient amount of parallel data for training a semantic parser in a given domain (thus solving the initial resource problems discussed in Sections 2.1.2 and 3.1.2), the gold `LF`s provided in the corpus are deficient and not able to capture the full range of NLU phenomena. Recalling our setup from Chapter 1, as shown again in Figure 4.3, this issue also touches on the shortcomings of how we evaluate our semantic parsing models.

One common way to deal with such resource problems is to re-annotate the corpus representations and the relevant background knowledge (Toledo et al., 2013).

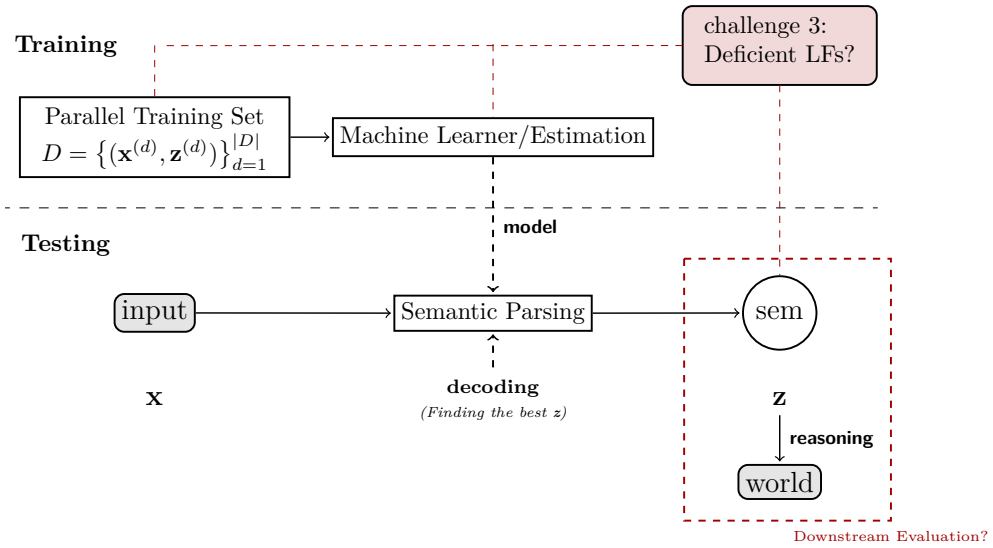


Figure 4.3: The standard semantic parsing setup and the third resource challenge (from Section 1.3).

We instead investigate whether this missing information can be learned, in particular by using example entailment judgements as a weak form of training supervision, which is a new learning framework that we call *learning from entailment*. Similar to the idea of *learning from denotation* in semantic parsing (Clarke et al., 2010; Liang et al., 2013; Berant et al., 2013), the intuition behind learning from entailment is that entailments give general information about denotations (i.e., the set of possible scenarios associated with entities and events), and that asymmetries in entailment judgements can be used for finding holes in the target representations and learning better representations. For example, given the mismatch in the entailments in Figure 4.2-1, one can infer that $\mathbf{t}$ has more specific information than $\mathbf{h}$ (or that its denotation is a subset of the denotations of h), which then requires learning a model that can identify this additional information and ultimately derive the semantics of this missing information.

To experiment with this idea, we introduce a new semantic parsing model that learns jointly using structured meaning representations (as done in previous ap-

proaches) and raw textual inference judgements between random pairs of sentences. In order to learn and model entailment phenomena, our model integrates natural logic (symbolic) reasoning (MacCartney and Manning, 2009) directly into our semantic learner. We perform experiments on the Sportscaster corpus (Chen and Mooney, 2008a), which we extend by annotating pairs of training sentences in the original dataset with inference judgements. On a new RTE-style inference task based on this extended dataset, we achieve an accuracy of 73%, which is an improvement of 13 percentage points over a strong baselines. As a separate result, part of our approach outperforms previously published results (from around 89% accuracy to 96%) on the original Sportscaster semantic parsing task.

4.2 Related Work

As reviewed in the last several chapters, work in semantic parsing has focused on learning semantic parsers from parallel data, often in the form of raw collections of text-meaning pairs. The earliest attempts (Kate et al., 2005; Wong and Mooney, 2006; Zettlemoyer and Collins, 2009) focused on learning to map natural language questions to simple database queries for database retrieval using collections of target questions and formal queries (e.g., in the GeoQuery domain studied in the last chapter). A more recent focus has been on learning representations using weaker forms of supervision that require minimal amounts of manual annotation effort (Clarke et al., 2010; Liang et al., 2011; Krishnamurthy and Mitchell, 2012; Artzi and Zettlemoyer, 2013; Berant et al., 2013), which includes work on learning from denotation (see Liang and Potts (2015); Liang (2016)).

Most work done on learning from denotation, and indeed in semantic parser induction more generally, has centered around question-answering (QA) applications. For example, Liang et al. (2011); Berant et al. (2013) train semantic parsers in QA domains using the denotations (or answers, represented as discrete symbolic entities) of each question as the primary supervision. One can regard this approach as the simplest form of learning from entailment; given a fixed database (or a model of all known scenarios) and symbolic representations of all answers, the aim is to learn a semantic parser given information that that each question is entailed by its associated answer. Under this scenario, however, entailment is lim-

ited to entailments between questions and simple answers (often existential values of some kind), and does not involve entailments that involve abstract relations between generic events and predicates, as we consider in this work. In general, entailment and symbolic reasoning has played a marginal role in existing work in semantic parsing, perhaps largely due to the primary focus on simple QA.

One inherent difficulty in modeling entailment (especially in RTE settings) and learning more complex semantic parsing representations is the need for considerable amounts of background knowledge (LoBue and Yates, 2011; Clark, 2018). Attempts to integrate more general knowledge into semantic parsing pipelines have often involved additional hand-engineering or external lexical resources (Wang et al., 2014; Tian et al., 2014; Beltagy et al., 2014). As discussed above, our approach looks at learning background knowledge indirectly from scratch by optimizing our models to predict the correct entailments, which to our knowledge has not been done before in semantic parsing work.

4.3 Problem Description and Approach

In this section, we given a high-level description of the original Sportscaster semantic parsing task and our approach to learning from entailment. While we define each task separately, we train our semantic parsing models jointly and in an end-to-end fashion using a grammar-based approach. A key technical innovation in our approach is the integration of formal symbolic reasoning into our semantic parsing model, which we describe in the next section and sketch out in Section 4.3.2.

4.3.1 The Sportscaster Task

Figure 4.4 shows a training example from the original Sportscaster corpus, consisting of a text about a sports event $\mathbf{x}$ paired with a set of formal meaning representations $\mathbf{Z}$. In this case, each text was collected by having human participants watch a 2-d simulation of several Robocup soccer league games (Kitano et al., 1997) and comment on events in the game. Rather than hand annotating the associated logical forms, sentences were paired with symbolic renderings of the underlying simulator actions that occurred around the time of each comment.

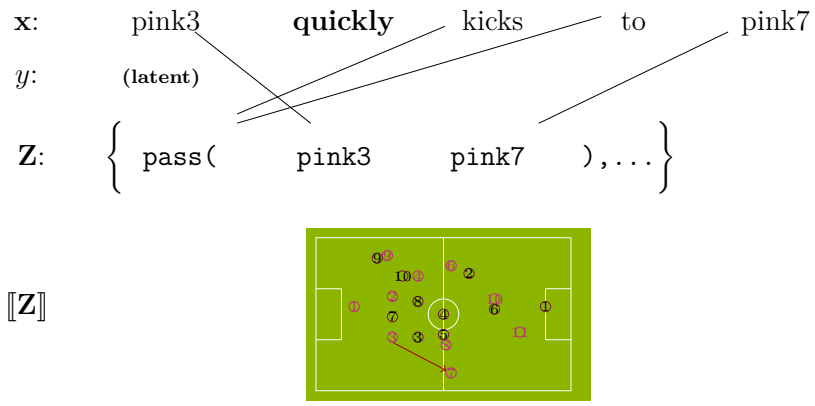


Figure 4.4: The original Sportscaster training setup.

These representations therefore serve as a proxy for the denotation of the event context and the individual events (shown as $[\![\mathbf{Z}]\!]$).

The goal is to learn a semantic parser `sp` given a training set D consisting of example sports descriptions and `LF`s, $D = \{(\mathbf{x}^{(d)}, \mathbf{Z}^{(d)})\}_{d=1}^{|D|}$, that can translate unseen descriptions to the correct `LF`s, as expressed below:

$$\texttt{sp} : \text{description} \rightarrow \texttt{LF}\ (\mathbf{z}) \tag{4.2}$$

In contrast to other work on learning from logical forms (e.g., in Chapter 2), the learning problem in this case is harder since the training data contains sets of possible `LF`s, as opposed to only gold `LF`s, which requires learning from *ambiguous supervision* (Mooney, 2008). The underlying idea is that these ambiguous contexts simulate the broader perceptual context associated with each comment, and hence provide a more realistic learning scenario.

The provided `LF`s (see examples in Figure 4.2) are expressed as atomic formulas in predicate logic defined over a small set of domain-specific predicates (e.g., `kick`, `pass`, `block`) and terms (e.g., `pink_team`, `pink1`, `purple11`). While our primary semantic parsing model generates exactly the representations provided in the original corpus, we reinterpret the semantics of these formulas in a way that it

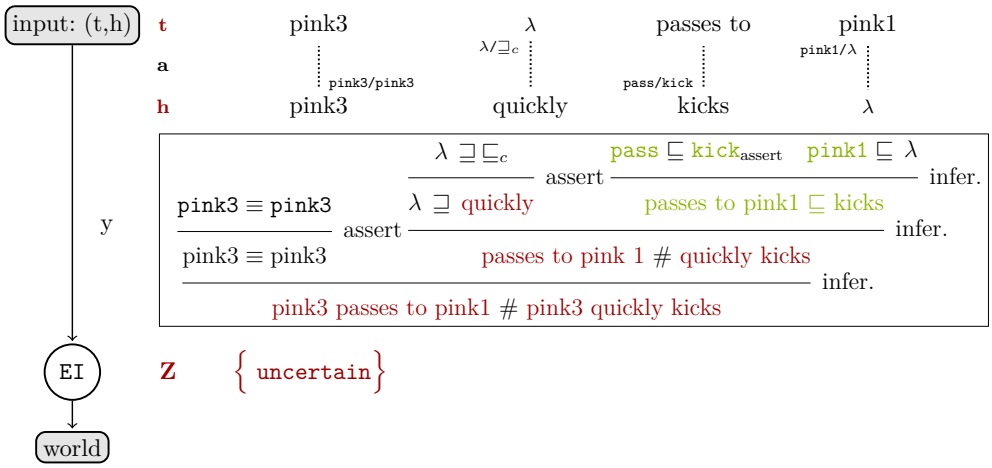


Figure 4.5: An example of learning from entailment.

makes it easier to model entailment. Specifically, terms are interpreted as separate predicates and the original event predicates are interpreted in a Neo-Davidsonian fashion (Parsons, 1990), as in the following example:

$$[\![\texttt{pass(pink3,pink7)}]\!] = \exists e.\exists x.\exists y.\texttt{pass}(e) \wedge \texttt{pink3}(x) \wedge \texttt{pink7}(y) \\ \wedge \texttt{arg1}(x,e) \wedge \texttt{arg2}(y,e)$$

where the terms `pink3` and `pink7` are treated as separate predicates (which makes it easier to model abstract relationships such as $[\![\texttt{pink1}]\!] \subset [\![\texttt{pink\_team}]\!]$) and event predicates and predicate argument information apply over event variables e (in the first case, making it easier to model relationships such as $[\![\texttt{pass}]\!] \subset [\![\texttt{kick}]\!]$).

4.3.2 Learning from Entailment

The problem with the approach described above, as discussed in Section 4.1.2, is that the representations being learned do not always capture the types of information needed for modeling entailment. The general idea of learning from entailment is to extend a given semantic parsing dataset D with pairs of training sentences

annotated with inference judgements (as shown in Figure 4.2). While training an ordinary semantic parser `sp`, we then use such pairs to train a model `infer` that can generate certain types of entailments from example pairs of descriptions, as shown below:

$$\texttt{infer} : \Big(\text{description}_1 = \mathbf{t}, \text{description}_2 = \mathbf{h}\Big) \rightarrow \texttt{entailment}\ (\mathbf{z}) \tag{4.3}$$

where entailments can be of the following three types (Cooper et al., 1996; Bentivogli et al., 2011): {`entail`, `contradict`, `uncertain/compatible`}. The approach pursued here involves integrating a logical reasoning system into our semantic parser that can reason about the target symbols being learned and prove theorems about the target entailments. The key idea is that the resulting proofs reveal distinctions not captured in the original representations, and can be used to improve the semantic parser's internal representations and acquire knowledge.

An illustration of this is shown in Figure 4.5, where the input consists of a text **t** and hypothesis **h**, and and a set of entailment judgements **Z** (in this case, a single `uncertain` judgement, which we represent using the variable `z`, as with LFs). *y* shows an example proof, or explanation, of how the model arrives at an `uncertain` inference based on a set of local inferences about relationships between aligned (via *a*) parts of **t** and **h**. For example, the model reasons that *pass to pink1* entails *kicks* (based on some *assertion* or axiom between `pass` and `kick`), whereas uncertainty is introduced with the modifier *quickly* in the hypothesis and *passes to pink1* and *quickly kicks*; this uncertainty then propagates up the proof using generic inference rules *infer* defined in the model (to be described in Section 4.4.2). Since example proofs are not provided at training time, the learning problem is to find the correct proofs within a large latent space of possible proofs.

This particular proof gives rise to several new assertions or facts: the `pass` symbol is found to forward entail or imply (shown using the set inclusion symbol $\sqsubseteq$) the `kick` symbol. The adverbial modifier, which is previously unanalyzed, is treated as an entailing modifier $\sqsubseteq_c$, which results in a reverse entailment or implication (shown using the symbol $\sqsupseteq$) when inserted (or substituted for the empty symbol λ) on the hypothesis side. The first fact can be used for building a domain theory, and the second for assigning more precise labels to modifiers in the semantic parser.

In the latter case, we might assign the following *improved* representation to the input *pink3 quickly kicks* (using the semantics described in the previous section):

$$\exists e.\exists x.\texttt{kick}(e) \wedge \texttt{quickly}(e) \wedge \texttt{pink3}(x) \wedge \texttt{arg1}(x, e)$$

in which we have a new predicate `quickly` derived from its use as a forward entailing modifier in the example proof.

Computing entailments in our approach is specifically driven by learning the correct semantic assertions between primitive domain symbols, as well as the semantic effect of deleting/inserting symbols. We focus on learning the following very broad types of linguistic inferences (Fyodorov et al., 2003):

- **construction-based**: inferences generated from specific (syntactic) constructions or lexical items in the language
- **lexical-based**: inferences generated between words or primitive concepts due to their inherent lexical meaning

Construction-based inferences are inferences related to modifier constructions: *quickly*`(pass)` $\sqsubseteq$ `pass`, `goal` $\sqsupseteq$ *nice*`(goal)`, *gets_a*`(free kick)` $\equiv$ (equivalence) `free kick`, where the entailments relate to default properties of particular modifiers when they are added or dropped. Lexical-based inferences relate to general inferences and implications between primitive semantic symbols or concepts: `kick` $\sqsupseteq$ `score`, `pass` $\sqsubseteq$ `kick`, and `pink1` $\sqsubseteq$ `pink team`.

4.4 Grammar-based Semantic Parsing

To model `sp` and `infer`, we use a grammar-based approach based on probabilistic context-free grammars (PCFG). In both cases, the target model assigns to each input a tree structured representation corresponding either to an `LF` representation or an entailment `z`. Grammar models build such structures using a finite set of (probabilistic) rewrite rules, which are created via a rule extraction process defined over the target parallel data described in the previous sections (as illustrated in Figure 4.6). In this section, we discuss the general PCFG formalism and explain its use in semantic parsing, then describe our rule extraction procedure (Sec-

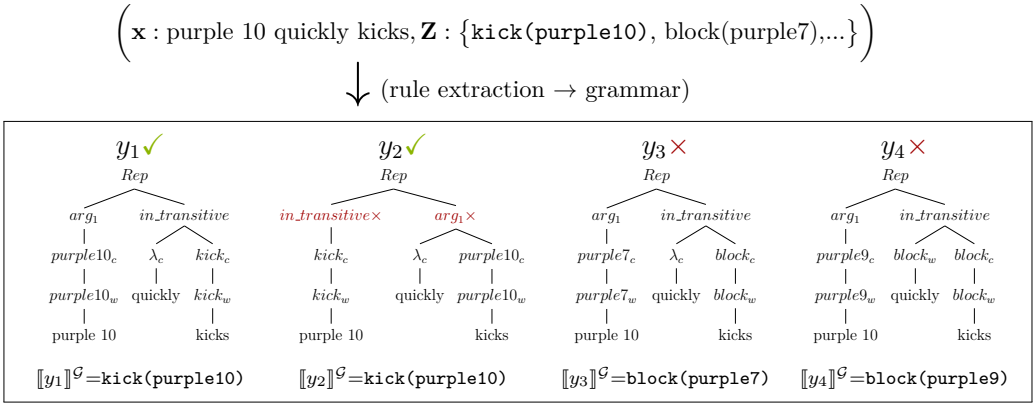


Figure 4.6: Semantic grammar rule extraction and example derivations.

tion 4.4.1) and the integration of logical reasoning into this model (Section 4.4.2). In Section 4.4.3 we finish by describing how we estimate our models from parallel data using a simple EM bootstrapping approach.

Modeling Preliminaries: Translating with PCFGs

Formally, a PCFG defines a 5-tuple $\mathcal{G}_\theta = (\Sigma, N, S, R, \theta)$ consisting of a set of terminal (i.e., source language) symbols Σ, a set of non-terminal grammar symbols N, a start symbol $S \in N$, a set of rewrite rules $R = \{N \rightarrow \beta \mid \beta \in (N \cup \Sigma)^*\}$ and a parameter vector $\theta \in \mathbb{R}^{|R|}$ (without θ, this defines a CFG). Using θ, each rule $N \rightarrow \beta$ (consisting of a left hand side (lhs) N and a right hand side (rhs) β) is assigned a score $\theta_{N\rightarrow\alpha}$ subject to the following constraints (where R_N is used to denote the set of rules from R that share the same lhs N):

$$\forall N \rightarrow \beta \quad 0 \leq \theta_{N\rightarrow\beta} \leq 1$$
$$\forall R_N \sum_{(N\rightarrow\alpha)\in R_N} \theta_{N\rightarrow\alpha} = 1$$

A derivation y over an input $\mathbf{x}$ is any application of rules that results in a tree rooted by S such that yield of the tree (i.e., the sequence of terminal nodes in the tree) is equal to $\mathbf{x}$. For example, Figure 4.6 shows an example derivation y for the

sentence *purple 10 quickly kicks* (shown below in a standard Lisp format):

```
(Rep
 (arg1 (purple10c (purple10w purple 10)))
 (in_transitive (λc quickly)
                (kickc
                 (kickw kicks))))
```

where rules include $\{\texttt{Rep} \rightarrow \texttt{arg}_1\texttt{in\_transitive},\ \texttt{arg}_1 \rightarrow \texttt{purple}_c, ...\} \subseteq R$ with the start node `Rep`, and the yield of the derivation is the left-to-right sequence of terminal symbols *purple 10 quickly kicks* (i.e., the input sentence). Imagining that probabilities are associated with rules, the score of this derivation y is computed as a product over the individual rule probabilities $N_j \rightarrow \beta_j$ in that derivation:

$$p_\theta(y) = \prod_i^{|y|} \theta_{N_i \rightarrow \beta_i} \tag{4.4}$$

As a generative model, PCFGs can be used to model the joint distribution $p(\mathbf{x}, y)$, which allows us to compute the probability of a given input $\mathbf{x}$ by marginalizing over all derivations over $\mathbf{x}$, or $\mathcal{Y}_\mathbf{x}$ (where computing each joint probability reduces to computing the probability of each derivation):

$$p(\mathbf{x}) = \sum_{y \in \mathcal{Y}_\mathbf{x}} p(\mathbf{x}, y) \tag{4.5}$$

$$= \sum_{y \in \mathcal{Y}_\mathbf{x}} p_\theta(y) \qquad \text{via Equation 4.4} \tag{4.6}$$

Under this formulation, one natural application of PCFGs is language modeling, or assigning scores (in this case, probabilities) to input sentences (Jurafsky et al., 1995). For most NLP applications, however, it is not the probability of the string that is of interest but rather the best derivation (or set of derivations) associated with input, since the particular grammar rules in each derivation often contain important details about linguistic structure.

The trick involved with using PCFGs for semantic parsing is that we associate each derivation with a unique `LF`, as shown in Figure 4.6 (on the bottom of each derivation). For example, in the derivation considered above, the *interpretation* of this derivation, which we express as $[\![y]\!]^\mathcal{G}$ (see Section 4.4.1 for more details about

Algorithm 8 CKY Recognition Algorithm

Input: CFG $\mathcal{G}$ in Chomsky Normal Form, input $\mathbf{x} = (x_1, .., x_{|\mathbf{x}|})$, start symbol S
Output: TRUE if $\mathbf{x}$ is accepted, FALSE otherwise

$\mathcal{T} \leftarrow \emptyset$ ▷ Initialize chart data structure
for j from 1 up to $|\ \mathbf{x}\ |$ **do**
  for all terminal rules $A \rightarrow \alpha \in \mathcal{G}_R$ **do** ▷ Search for terminal rule matches
    if rule is $A \rightarrow x_j$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} + [j-1, A, j]$
  for i from $j-2$ down to 0 **do** ▷ Search for binary rule matches
    for k from $i+1$ to $j-1$ **do**
      for all binary rules $A \rightarrow BC \in \mathcal{G}_R$ **do**
        if $[i, B, k]$ and $[k, C, j] \in \mathcal{T}$ **then**
          $\mathcal{T} \leftarrow \mathcal{T} + [i, A, j]$
return $[0, S, |\mathbf{x}|] \in \mathcal{T}$

how this interpretation is computed), is the following `LF`:

$$\mathbf{z} = \texttt{kick(purple10)}$$

In doing this, we can then define a conditional distribution over `LF` outputs $\mathbf{z}$ (or entailments when modeling entailment) given inputs $\mathbf{x}$, as in the following:

$$p_\theta(\mathbf{z} \mid \mathbf{x}) \propto \sum_{y \in \mathcal{Y}_\mathbf{x} \mid [\![y]\!]^\mathcal{G} = \mathbf{z}} p_\theta(y) \tag{4.7}$$

which allows us in effect to use the PCFG as a special kind of constrained translation model (with which we can model weighted relations between natural languages and semantic languages as first discussed in Section 1.2).

One inherent difficulty with PCFGs and the computations described above is that the space of derivations $\mathcal{Y}_\mathbf{x}$ can be exponential over the size of each input $\mathbf{x}$ (this is similar to the issue of computing all alignments in the translation models from Section 2.3.1). Often these issues can be overcome by applying standard dynamic programming techniques (as described in the next section), however not all such techniques can be applied when using our model in the manner described above for semantic parsing, as we discuss next.

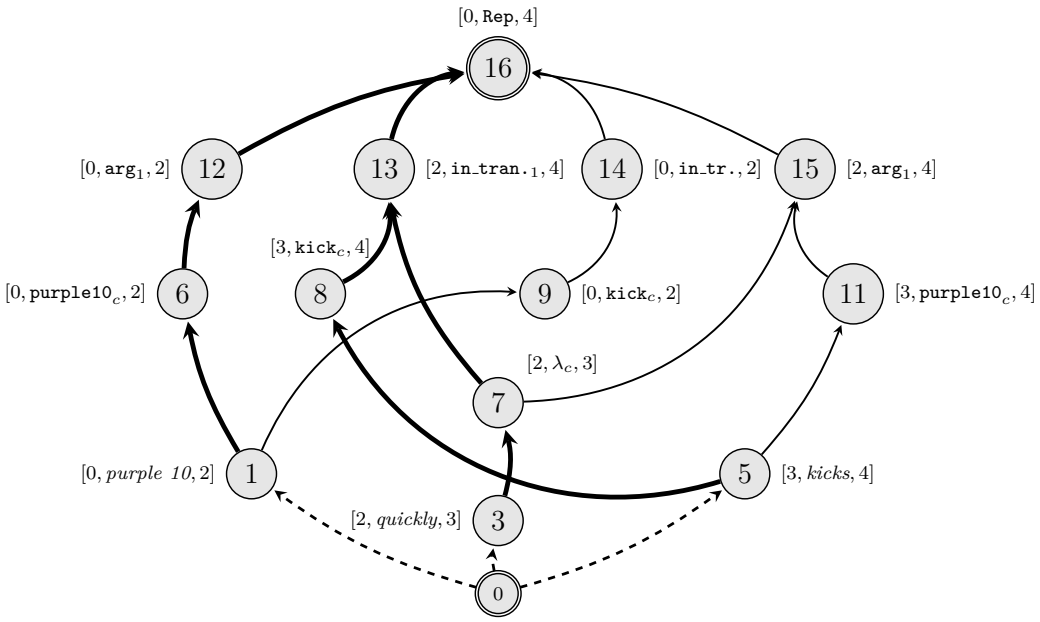


Figure 4.7: An acyclic hypergraph representation of the first two derivations in Figure 4.6, with the shortest path (or tree) shown in bold.

Recognition and Decoding Given a generic PCFG $\mathcal{G}_\theta$ and an input $\mathbf{x}$, the decoding problem involves finding the most probable derivation y^* associated with the input, as expressed below:

$$y^* = \arg\max_{y \in \mathcal{Y}_\mathbf{x}} \Big\{ p(y \mid \mathbf{x}) \Big\} \tag{4.8}$$

which can be reformulated in the following way via the *decoding rule* (Smith, 2011), which is based on the fact that $p(\mathbf{x})$ remains fixed for each candidate y:

$$\begin{aligned}
y^* &= \arg\max_{y \in \mathcal{Y}_\mathbf{x}} \Big\{ p(y \mid \mathbf{x}) \Big\} && \text{Eq. 4.8} \\
&= \arg\max_{y \in \mathcal{Y}_\mathbf{x}} \Big\{ \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} \Big\} && \text{Definition} \\
&= \arg\max_{y \in \mathcal{Y}_\mathbf{x}} \Big\{ p(\mathbf{x}, y) \Big\} && \text{constant } p(\mathbf{x})
\end{aligned}$$

Algorithm 9 Directed Acyclic Hypergraph (DAH) Shortest-Path Search

Input: DAH $\mathcal{H}$, edge labels parameters θ (probabilities, e.g., grammar rule parameters)
Output: Shortest hyperpath

1: $d[V[\mathcal{H}]] \leftarrow \infty$ ▷ Standard initialization (i.e., for DAG SSSP)
2: $\pi[V[\mathcal{H}]] \leftarrow Nil$
3: $d[0] \leftarrow 0$
4: **for** each node $v \in V[\mathcal{H}]$ in sorted order **do**
5: **for** each hyperedge $e = \big(\{u_1, u_2, ..., u_{|e|}\}, v, l\big) \in \texttt{BS}(v)$ **do**
6: $\texttt{score} \leftarrow -\log(\theta_l) + \sum_i^{|e|} d[u_i]$ ▷ hyperedge score computation via θ
7: **if** $d[v] > \texttt{score}$ **then** ▷ Standard relaxation step
8: $d[v] \leftarrow \texttt{score}$
9: $\pi[v] \leftarrow e$
10: **return** FindPath$(\pi, \mid V \mid, 0)$ ▷ Backtrace to find shortest hyperpath

Again, the difficulty here involves efficiently computing all the derivations in $\mathcal{Y}_{\mathbf{x}}$. One way do this is to use a variant of the CKY algorithm shown in Algorithm 8 (Kasami, 1965; Nederhof and Satta, 2010). As presented, the CKY solves the more fundamental problem of *recognition*, or determining if an input string $\mathbf{x}$ is in the language defined by a CFG, which similarly involves searching through all possible derivations. This is done efficiently by using a chart data structure $\mathcal{T}$ and dynamic programming to efficiently search and store all applications of rules in intermediate derivations (lines 4 and 9, see Manning and Schütze (1999) for more details).

The chart data structure $\mathcal{T}$ that results from the CKY search can be interpreted as a special type of directed graph called a directed hypergraph (Gallo et al., 1993), which extends ordinary directed graphs by allowing edges to connect to multiple nodes. In the parsing case, nodes are associated with particular rule applications (i.e., each $[i, R, j]$ from lines 5 and 10 in Algorithm 8) and edges are associated with production rules, as shown in Figure 4.7. With this graph, one can then do decoding by extending the shortest path algorithms for directed graphs (Section 3.4) to hypergraphs (Knuth, 1977; Klein and Manning, 2004; Huang, 2008).

Formally, a directed hypergraph $\mathcal{H} = (V, E)$ consists of a set of nodes V and directed hyperedges E, where each hyperedge e takes the following form (see Huang

(2008) for a more general overview and notation):

$$e = \Big(\{u_1, u_2, ..., u_{|e|}\}, v, l\Big)$$

and consists of a set of *tail nodes* $t(e) = \{u_1, .., u_{|e|}\} \subseteq V$, a *head node* $h(e) = v \in V$ and (for convenience) a label l. In the case of the CKY algorithm, the hypergraph that is generated is an acyclic directed hypergraph, which has the property (as with DAGs) that nodes can be sorted into numerical (topological) order. The associated shortest path algorithm, therefore, is nearly identical to the one for DAGs (see Algorithm 4), and is shown in Algorithm 9 (where $\texttt{BS}(v) = \{e \in E \mid h(e) = v\}$[1] and in the parsing case, labels are used to identify grammar rules R associated with each e). The shortest path (or best derivation tree) can then be constructed by moving backwards (via the FindPath routine) from the final node (or the start node S) to the source node using the *predecessor* π.

Returning to the use of our PCFG as a semantic parser, the decoding problem (i.e., finding the best `LF` or entailment $\mathbf{z}^*$ given $\mathbf{x}$) can be described in the following

$$\mathbf{z}^* = \arg\max_{\mathbf{z}} \Big\{p_\theta(\mathbf{z} \mid \mathbf{x})\Big\} \tag{4.9}$$

and is at first glance more difficult than Equation 4.8 given that computing this requires finding all *valid derivations* $\{y \mid \mathbf{z} = [\![y]\!]^{\mathcal{G}}\}$ as per Equation 4.7. The problem is that computing each valid derivation often requires a non-local combination of rules in the target tree, which cannot be accomplished using dynamic programming. For example, computing the `LF kick(purple10)` from the first derivation tree in Figure 4.6 requires combining information from the two subtrees rooted by $\texttt{purple10}_c$ and $\texttt{kick}_c$, which are not adjacent in $\mathcal{T}$. Therefore, computing valid trees requires enumerating an intractable number of trees and interpreting them.

One way to get around this is to approximate this search by taking $\mathbf{z}^*$ to be the

[1]The *backwards star* $\texttt{BS}(v)$, which in ordinary directed graphs denotes the set of incoming edges to v, has a *forward* variant $\texttt{FS} = \{e \mid v \in t(e)\}$ that is analogous to `Adj` in Algorithm 4. We note that for DAG SSSP search, either type of traversal order can be used, whereas `BS` traversal is more straightforward for hypergraphs (see discussion in Huang (2008)).

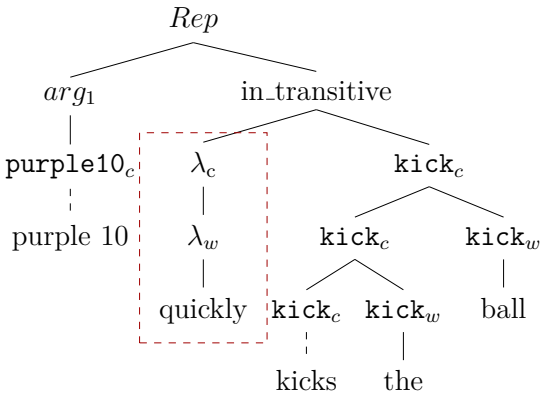


Figure 4.8: An example derivation (simplified) in the base semantic grammar with a gap rule λ_c applied over the modifier *quickly*.

interpretation of the most probable derivation:

$$\mathbf{z}^* \approx [\![y]\!]^{\mathcal{G}} = \arg\max_{y \in \mathcal{Y}_{\mathbf{x}}} \left\{ p_\theta(y) \right\} \tag{4.10}$$

which is what we do when evaluating our models. While this works well for decoding at test time, it is still a problem when estimating our models (i.e., finding the expected counts of rules in valid derivations during the training phase). We discuss this more in Section 4.4.3, and propose a simple EM bootstrapping method that similarly involves sampling the best derivations via k-shortest path decoding (using variations of the DAG k-SSSP algorithms used in Section 3.4).

4.4.1 Rule Extraction and $[\![\cdot]\!]^{\mathcal{G}}$

As already discussed, rule extraction is the process of constructing the grammar rules R needed for generating $\mathbf{z}$'s from input $\mathbf{x}$. We start by describing rule construction for grammars that generate LFs (or what we call *base semantic grammars*) and return to how rule extraction works for modeling inference in Section 4.4.2 (the *inference grammars*). In this first case, such rules are constructed automatically using a small set of rule templates defined over the target set of LFs, as done

in Börschinger et al. (2011) (BB). The basic idea in BB is to break down all LFs in the target corpus of the form $\texttt{R}(x, y)$ into the following production rules:

$$\texttt{S} \rightarrow \texttt{R}(\texttt{x}, \texttt{y})$$
$$\texttt{R}(\texttt{x}, \texttt{y}) \rightarrow \left\{\texttt{R}_c \ \texttt{x}_c \ \texttt{y}_c\right\}$$

where the lhs of the second rule is a representation of the full LF, and the rhs consists of all orderings (as indicated by $\{\cdot\}$) of the constituent parts of the LF expressed as grammar symbols (i.e., the relation name R and the arguments, all marked here as $\texttt{X}_c$). Each non-terminal $\texttt{X}_c$ is then associated with a word rule $\texttt{X}_w$, that rewrites to all unigrams in the target corpus via a left-recursive rule that models a unigram Markov-process (Johnson and Goldwater, 2009) (e.g., the rule sequence for *kicks the ball* in Figure 4.8):

$$X_c \rightarrow X_w$$
$$X_c \rightarrow X_c X_w$$
$$X_w \rightarrow w \mid w \in \text{Corpus}$$

Using this basic idea, additional structure and information can be added into the grammar as needed. For example, BB use word order rules that make explicit the different orderings of constituent rules in $\{\cdot\}$, as well as more complex word rules that allow for modeling empty (or skip) words λ_w. We adopt both of these ideas, and pad all nodes $\texttt{X}_c$ with a *gap rule* λ_c that allows the model to learn larger spans of unanalyzed text. For example, in Figure 4.8 the model learned that *quickly* is not analyzed in the target LF, which is information that can be used by our inference model (described in Section 4.4.2) to reason about the semantics of these gaps. Rather than representing full LFs in the grammar as atomic symbols, we also assign more abstract role types to the concepts X_c (e.g., $\text{arg}_1, \text{in\_transitive}$ in Figure 4.8), to make the rules more generalizable (see Appendix C for a full description of the rule templates we use in our experiments).

As discussed above, each derivation tree y can be interpreted to a unique LF via

an interpretation function $[\![\cdot]\!]^{\mathcal{G}}$:

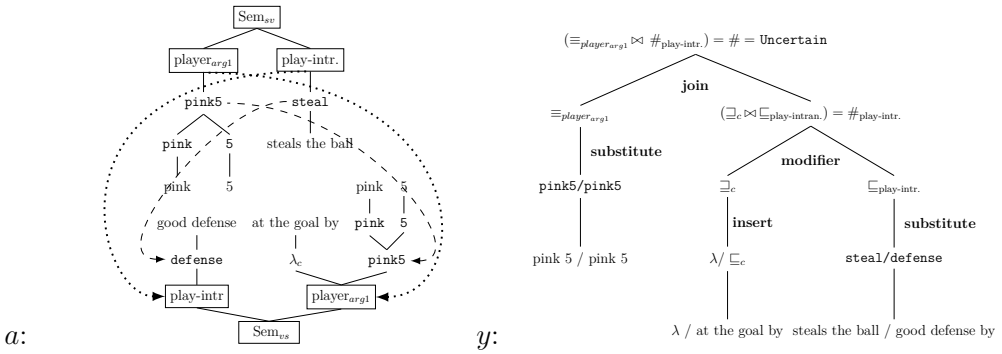
$$[\![\cdot]\!]^{\mathcal{G}} : y \ (\textit{derivation}) \rightarrow \mathbf{z} \tag{4.11}$$

In our case, this function works by deterministically mapping each grammar symbol $\texttt{X}_c$ to an atomic logical symbol, and combining these symbols in a way that is consistent with the assigned roles. For example, $\texttt{kick}_c$ in Figure 4.8 is mapped to the symbol `kick` and assigned to the main predicate slot given the *in_transitive* role, and $\texttt{purple10}_c$ is mapped to `purple10` as assigned as the first argument slot given the arg_1, which results in the `LF` representation `kick(purple10)`.

An important feature of the resulting grammars is that they overgenerate (as shown in Figure 4.8); given a text input, the grammar will generate a large space of possible derivations, many of which interpret to incorrect `LF`s. By assigning weights to these rules and formalizing the model as a PCFG, the learning problem reduces to a grammatical inference problem, or finding a grammar $\mathcal{G}_\theta$ with parameters θ that is able to distinguish correct derivations (i.e., derivations that have the correct interpretations) from incorrect derivations. Under a hypergraph approach, we can equivalently describe the learning problem as finding a model that is able to identify the correct paths through graphs such as the one in Figure 4.7 (see Section 4.4.3 for more details about learning).

4.4.2 Natural Logic and Inference Grammars

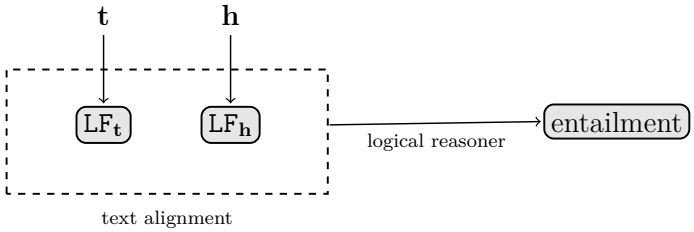
Given the *base semantic grammars* described in the previous section, we can build a semantic parser that (standardly) translates text to output `LF`s, however the resulting derivation trees still have gaps (i.e., unanalyzed spans of text as shown in Figure 4.8) and our model continues to lack the background knowledge needed for reasoning about entailment. As already proposed, we aim to learn this missing information by extending our training corpus to include pairs $\mathbf{x} = (\mathbf{t}, \mathbf{h})$ annotated with entailment information. With this information, our approach works in the following way: align the related spans of text in $\mathbf{t}$ and $\mathbf{h}$ and apply logical reasoning over these spans to construct a proof of an entailment, as sketched below:



a: y:

$$\Big((\mathbf{t} = \textit{pink 5 steals the ball}, \mathbf{h} = \textit{good defense at the goal by pink 5}), \mathbf{z} = \texttt{Uncertain} \Big)$$

Figure 4.9: An end-to-end example produced by our inference grammar model.



This idea is further illustrated in Figure 4.9, where an alignment a between $\mathbf{t}$ and $\mathbf{h}$ is computed by heuristically matching related roles in the semantic parse for each sentence (generated using the base semantic grammars described above). The associated spans of aligned text are then provided to a logical reasoner that remaps the aligned spans to logical symbols, then generates a structured proof y over these symbols that corresponds to a unique entailment judgement. Importantly, gap rules in $\mathbf{t}$ or $\mathbf{h}$ (e.g. *at the goal by* in $\mathbf{h}$, marked as λ_c) and unaligned arguments are matched to the empty string λ so that the logical model can reason about the semantics of *inserting* or *deleting* expressions in $\mathbf{h}$ and $\mathbf{t}$.

Relation	Symbol	Set Definition	First-order Logic	RTE label
forward entail	$R \sqsubseteq S$	$R \subset R$	$\forall x. \left[R(x) \rightarrow S(x)\right]$	`entail`
reverse entail	$R \sqsupseteq S$	$R \supset S$	$\forall x. \left[S(x) \rightarrow R(x)\right]$	`uncertain`
equivalence	$R \equiv S$	$R = S$	$\forall x. \left[R(x) \leftrightarrow S(x)\right]$	`entail`
alternation (negation)	$R \mid S$	$R \cap S = \emptyset \wedge R \cup S \neq D$	$\forall x. \neg \left[R(x) \wedge S(x)\right]$	`contradict`
independence	$R \# S$	(all other cases)	–	`uncertain`

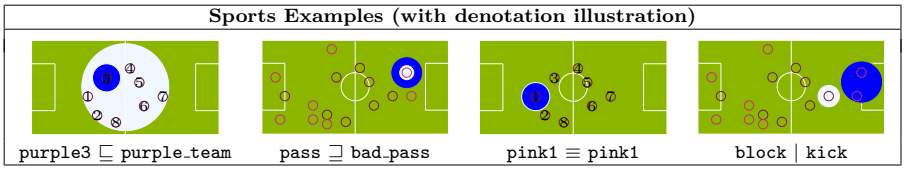


Figure 4.10: A description of relations used from the natural logic calculus (top) with examples (bottom) from Sportscaster.

Natural Logic Calculus

In order to do modeling of this kind, we need a logical calculus that can perform reasoning over spans of text. Since our main goal is to learn proofs and the background knowledge that drives the proofs, such a model should also support uncertainty. Given these constraints, we use a fragment of the natural logic calculus defined in MacCartney and Manning (2009, 2008). In our simplified version, the model has two components: 1) a set of relations that define abstract semantic relationships between concepts (i.e., primitive symbols in our target `LF` representations) and 2) a set of inference rules that can compose relations. The full set of semantic relations are shown in Figure 4.10, along with a definition of their meaning in set theory and first-order logic (or FOL, following Pavlick et al. (2015)). For example, the following relation between `pass` and `kick`:

$$\texttt{pass} \sqsubseteq \texttt{kick}$$

can be read in FOL as a general implication (van Benthem, 1986): for all events e involving passing, e also involves kicking. Assuming that we are given two pairs of relations, $\texttt{pink5} \sqsubseteq \texttt{pink\_team}$ and $\texttt{pass} \sqsubseteq \texttt{kick}$, the join rule $\bowtie$ defined in

Figure 4.11 then composes these two relations to derive a new relation:

$$\Big(\texttt{pink5} \sqsubseteq \texttt{pink\_team}\Big) \bowtie \Big(\texttt{pass} \sqsubseteq \texttt{kick}\Big) = \sqsubseteq$$

which in this case allows us to conclude that *pink5 passed* (forward) entails that *the pink team kicked* (on the assumption that `pink5` forward entails `pink_team` and `pass` forward entails `kick`). Joins can be applied an arbitrary number of times; for example, we might continue by adding $\lambda \sqsupseteq$ `quickly` to model in the RTE context the insertion of a modifier *quickly* on the **h** side:

$$\Big(\texttt{pink5 pass} \sqsubseteq \texttt{pink\_team kick}\Big) \bowtie \Big(\lambda \sqsupseteq \texttt{quickly}\Big) = \#$$

which results in a new relation #. This process can continue further until all target relations have been consumed, which will result in a final semantic relation and entailment (see Figure 4.10 to see the mapping from relations to RTE labels).

We note that our simplified model uses only a subset of the seven relations from MacCartney and Manning (2009), since these additional relations were not needed to model the types of inferences we encountered in the Sportscaster domain. As a cautionary note, we also point out that our model fails to capture various complex inferences. For example, assuming `every` $\equiv$ `every` and `company` $\sqsupseteq$ `small_company`, our model cannot generate the following entailment using the join inference rule:

$$\texttt{every company} \sqsubseteq \texttt{every small\_company}$$

since this particular inference is related to special properties of `every`, which convert $\sqsupseteq$ inferences to $\sqsubseteq$ when doing composition in this context. To handle this, the full natural logic calculus has a *projectivity* mechanism that defines how certain constructions alter the inferences of arguments in such contexts. In our simple mode and domain, *projectivity* is limited to a single rule that always projects negations | up the proof tree in order to capture the following inferences:

$$\texttt{pink5 kick} \mid \texttt{purple\_team pass}$$

where under the assumption that `pink5` | `purple_team` and `kick` $\sqsupseteq$ `pass`, our

$\bowtie$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\mid$	#
$\equiv$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\mid$	#
$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	#	$\mid$	#
$\sqsupseteq$	$\sqsupseteq$	#	$\sqsupseteq$	#	#
$\mid$	$\mid$	#	$\mid$	#	#
#	#	#	#	#	#

Figure 4.11: The join inference rule $\bowtie$ table for the set of relations in Figure 4.10.

join rule would incorrectly assign # (or an `uncertain` entailment). This is again motivated by the types of predicates we model in the Sportscaster domain, which all tend to have the projectivity properties of *functional relations* (Russell (1995), see MacCartney (2009)[Chapter 6.2.5] for more discussion).

Inference Grammars and Alignment

Given the tree-like nature of the natural logic proofs described above, our idea is to represent the inference steps as CFG rewrite rules, as shown in Figure 4.12[2]. Under this approach, relations between pairs of concepts are rules where the rhs contains the pair of ordered concepts (delimited by `/`) and the lhs contains their resulting relation. The same idea applies to our inference rule $\bowtie$: the rhs consists of two relations and the lhs contains the result of joining these relations. As before, additional structure can be added to the grammar as needed, such as information about the types of concepts being compared and composed (see Appendix C for a complete list of the rules we use, as well as Figure 4.15).

In particular, we use an additional set of *gap rules*, as shown in Figure 4.12, that model whether certain types of insertions/deletions are forward-entailing (represented using $\sqsubseteq_c$) or non-entailing ($\equiv_c$). In the first case, this includes adverbial modifiers such as *quickly* in *quickly kicked* which modify entailment, whereas *the*

[2]This particular grammar formulation can be regarded as a type of inversion transduction grammar (Wu, 1997), or a *simple transduction grammar* (Lewis II and Stearns, 1968), where each terminal rule is marked with an input and output symbol and non-terminals are the same as in ordinary CFGs. Recognition and decoding with these models is equivalent to ordinary CFG parsing as described above (Melamed, 2004; Lopez, 2008).

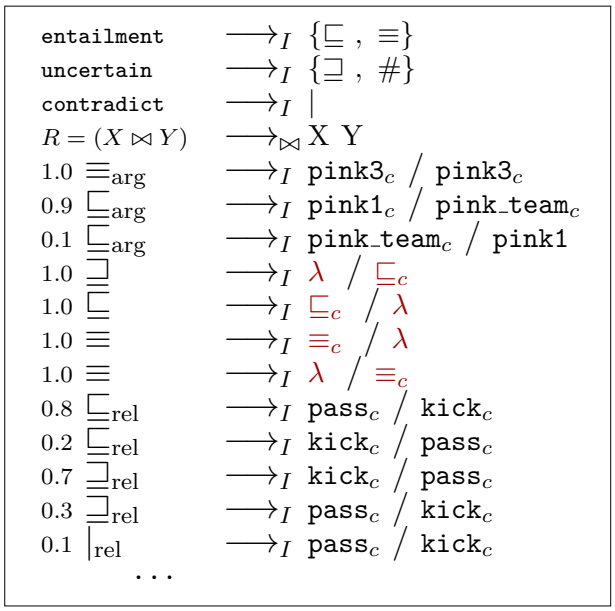


Figure 4.12: An example inference grammar for the Sportscaster domain with gap rules shown in red.

ball in *kick the ball* does not appear to effect entailment. To model the subtle differences between different concept senses, we also mark symbols with latent sense labels. For example, in the following rule:

$$\sqsubseteq_{rel} \rightarrow \texttt{kick}_{c1} \ / \ \texttt{kick}_{c2}$$

we have two senses for `kick`, which we can use to model entailments between *kick the ball* and *score a goal* (which are both annotated as `kick` in Sportscaster).

As discussed at the onset, given a pair $\mathbf{x} = (\mathbf{t}, \mathbf{h})$, we can generate trees in this grammar by heuristically aligning related spans in $\mathbf{t}$ and $\mathbf{h}$ (see again Figure 4.9, and Appendix C for more details), then by labeling each part of the the spans with concept labels and applying the grammar rules. The interpretation of a given derivation (proof tree) $[\![y]\!]^{\mathcal{G}}$ is the entailment provided at the top node of each tree. As shown in the grammar in Figure 4.12, the concept labels $\texttt{X}_c$ are the same

as in our LF semantic parser, which allows us to create a single grammar by combining the inference rules with the base semantic grammar (and therefore use the same learned concept mapping rules as in our main semantic grammar). In our experiments, the decision to jointly train the sem and infer models using a single grammar is based on the following modeling assumption:

- **Joint entailment modeling**: When learning a semantic parser, improvements on learning the correct entailments should help improve (and are tied to) learning translations to LFs, and vice versa.

As with the base semantic grammars, an important feature of the inference grammars described above is that they overgenerate; given an input, the grammar will generate a large space of possible proofs, many of which interpret to the wrong entailment. This is largely due to the fact that we do not know the correct relations between the underlying concepts and modifiers and start by assuming all possibilities. For example, since we do not know the relation between pass and kick, we start with the following three rules:

$$\sqsubseteq_{rel} \rightarrow \texttt{pass}_c \texttt{ / kick}_c$$
$$\sqsupseteq_{rel} \rightarrow \texttt{pass}_c \texttt{ / kick}_c$$
$$|_{rel} \rightarrow \texttt{pass}_c \texttt{ / kick}_c$$

The key idea is that by interpreting these grammars as PCFGs, we can then associate weights with individual rules of this type and learn the correct relations by training our grammar on example entailments. In this case, the goal is to learn that the first rule should have a higher weight than the other two rules since pass forward entails kick as inferred from its appearance in example proofs. Given that particular orderings of join inferences can effect the resulting entailments (MacCartney, 2009), the PCFG approach also allows for learning optimal inference combinations.

Under the PCFG formulation, our model can therefore handle probabilistic inference, which distinguishes it from most other formulations of natural logic (for a similar idea, see Angeli and Manning (2014)). While our particular rule templates might seem arbitrary at first glance, we note that the probabilistic logic that re-

sults from this formulation seems to have a sensible semantics. In addition, our use of grammar representations allows us to apply efficient search strategies from parsing to the problem of entailment search. For example, the probability of an `entail` using Equation 4.7 is given by the following:

$$p_\theta(\mathbf{z} = \texttt{entail} \mid \mathbf{x} = (\mathbf{t}, \mathbf{h})) = \sum_{y \in \mathcal{Y}_\mathbf{x} | [\![y]\!]^\mathcal{G} = \texttt{entail}} p(y \mid \mathbf{x})$$

and is interpreted as all proofs between $\mathbf{t}$ and $\mathbf{h}$ that evaluate to *entailment* (within the space of all possible proofs and across all individual semantic interpretations of $\mathbf{t}$ and $\mathbf{h}$). Since our approach involves a heuristic alignment between $\mathbf{t}$ and $\mathbf{h}$ (and hence is not burdened by having to search all possible alignments), the basic proof search is therefore bound to the complexity of ordinary recognition (e.g., using the CKY algorithm), or $O(|\mathbf{x}|^3 \cdot |\mathcal{G}_R|)$. Since the interpretation in these grammars only requires reading a single node, computing the above equation can be done exactly with the same complexity using the inside algorithm (Lari and Young, 1990).

4.4.3 Learning

As discussed in the previous section, we model `sem` and `infer` using a joint PCFG model that uses the rules described above. To learn this model, we perform maximum likelihood estimation (MLE) over our parallel dataset $D = \{(\mathbf{x}^{(d)}, \mathbf{Z}^{(d)})\}_{d=1}^{|D|}$, consisting both of parallel semantic parsing data and parallel inference data. Formally, the objective is to find grammar parameters θ^* that maximize the following (where we use $\mathcal{C}_{(d)}$ to denote the set of *valid (interpretable) derivations* relative to each training annotation $\mathbf{Z}^{(d)}$ and input $\mathbf{x}^{(d)}$: $\{y \mid y \in \mathcal{Y}_{\mathbf{x}^{(d)}} \wedge [\![y]\!]^\mathcal{G} \in \mathbf{Z}^{(d)}\}$):

$$\theta^* = \max_\theta \log \prod_{d=1}^{|D|} \left[p_\theta(\mathbf{z}^{(d)} \mid \mathbf{x}^{(d)}) \right] \tag{4.12}$$

$$= \max_\theta \sum_{d=1}^{|D|} \log \left[\sum_{y \in \mathcal{C}_{(d)}} p_\theta(y) \right] \qquad \text{via Eq. 4.7} \tag{4.13}$$

To optimize this objective, we use a variant of the EM algorithm (for a review of EM, refer back to Section 2.3.1). As in normal EM for PCFGs (Lari and Young,

Algorithm 10 EM Grammar Boostrapping

Input: Grammar $\mathcal{G}$ with parameters θ, dataset D, interp. function $[\![\,]\!]^{\mathcal{G}}$, KBEST function with k
Output: Learned parameters θ

$\theta^0 \leftarrow$uniform initialization
$t \leftarrow 0$
repeat
  $c(N \to \beta) \leftarrow 0, \forall N \to \beta \in \mathcal{G}_R$ ▷ Initialize counters to collect rule counts
  $b(N) \leftarrow 0, \forall N \to \beta \in \mathcal{G}_R$
  for $(\mathbf{x}^{(d)}, \mathbf{Z}^{(d)})$ from $d = 1$ up to $\mid D \mid$ **do** ▷ E-Step: evaluate $p_\theta(\mathbf{Z} \mid \mathbf{x}) \sim$ KBEST
    $v \leftarrow [\,], n \leftarrow 0$
    for $(y, p) \in \text{KBEST}_{(d)}(\mathbf{x}^{(d)}, \mathcal{G}, \theta^t, k)$ **do** ▷ Find candidate derivation y, $p = p_\theta(y \mid \mathbf{x})$
      if $[\![y]\!]^{\mathcal{G}} \in \mathbf{Z}^{(d)}$ **then**
        $n \leftarrow n + p$ ▷ Add valid derivations with scores
        $v \leftarrow v + (y, p)$
    for $(y, p) \in v$ **do** ▷ Count rules in valid derivations
      for $N_i \to \beta_i$ from $i = 1$ up to $|y|$ **do**
        $c(N_i \to \beta_i) \leftarrow c(N_i \to \beta_i) + \frac{p}{n}$ ▷ Normalize using n to create prob. distr.
        $b(N_i) \leftarrow b(N_i) + \frac{p}{n}$
  for $N \to \beta \in \mathcal{G}_R$ **do** ▷ M-step: perform MLE updates
    $\theta^{t+1}_{N \to \beta} \leftarrow \frac{c(N \to \beta)}{b(N)}$
  $t \leftarrow t + 1$
until converged
  return θ^t

1990; Lafferty, 2000), the E-step involves finding the expected counts of individual production rules R in all latent derivations (or in our case, all valid derivations $\mathcal{C}_{(d)}$) given D and some posterior distribution $p_{\theta^t}(y \mid \mathbf{x}^{(j)})$:

$$c(R; D) = \sum_{d=1}^{|D|} \left[\sum_{y \in \mathcal{C}_{(d)}} p_{\theta^t}(y \mid \mathbf{x}^{(d)}) \sum_{i}^{|y|} \delta(r_i, R) \right] \tag{4.14}$$

Using these counts, the M-Step then involves performing ordinary MLE updates, and the process then repeats until a convergence point:

$$\theta^{t+1}_{N \to \beta} = \frac{c(N \to \beta; D)}{\sum_{\beta'} c(N \to \beta'; D)} \tag{4.15}$$

As before, the main problem involves efficiently computing the set of valid derivations $\mathcal{C}_{(d)}$, since our interpretation function $[\![\cdot]\!]^{\mathcal{G}}$ involves non-local combinations

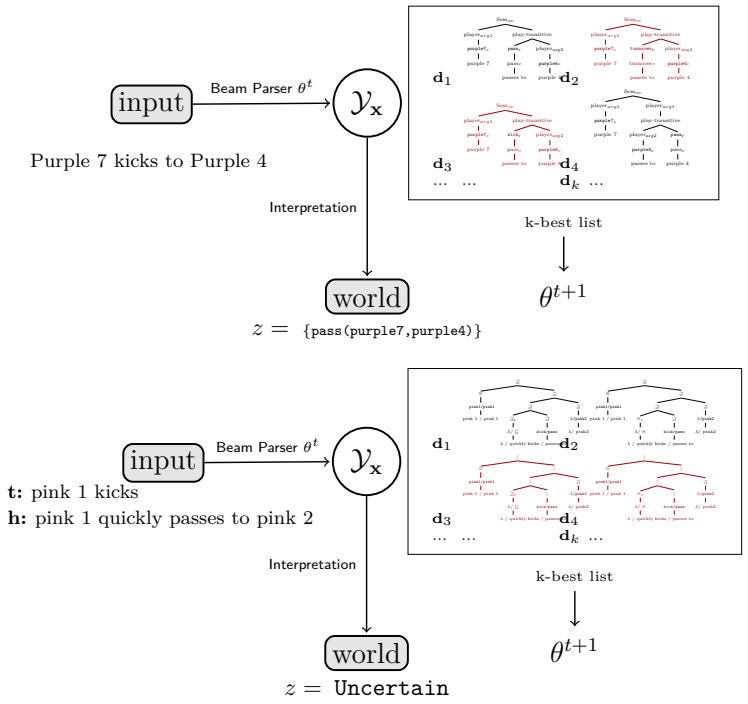


Figure 4.13: An illustration of EM bootstrapping for semantic parsing and learning from entailment, where green shows the valid derivations.

of derivation rules. We get around this by approximating each $\mathcal{Y}_\mathbf{x}$ in $\mathcal{C}_{(d)}$ with a k-best list of derivations $\text{KBEST}_{(d)} \approx \mathcal{Y}_\mathbf{x}$ (Angeli et al., 2012). Under the hypergraph approach outlined in Section 4.4, sampling the k-best derivations can be achieved by extending the branching $k-$SSSP method used in Algorithm 7 and Section 3.4.3 for DAGs to hypergraphs, as done in Nielsen et al. (2005). Given special features of parsing, however, more efficient methods based on hypergraphs have been developed, notably the lazy k-best algorithm from Huang and Chiang (2005), which is what we use in our experiments.

The full training algorithm is shown in Algorithm 10, along with an illustration in Figure 4.13 of detecting the valid derivations (line 9) and computing new parameters θ^{t+1} based on collected rule counts (lines 16-17). As discussed in Liang et al.

(2011), the idea is that learning starts in an unguided manner and improves over time by *bootstrapping* off of the easy examples. When training with the entailment pairs, the distinctions being made for modeling inference (e.g. sense distinctions, modifier types) and the word rules being used inform and reinforce the learning of the base semantic grammar. As the quality of the semantic parser improves, so should the quality of the background knowledge (i.e. semantic relations) used to generate the natural logic proofs.

4.5 Experimental Setup

In this section, we provide more details about the Sportscaster dataset and a new Sportscaster inference corpus that we created for modeling and evaluating entailment. We also describe our main experimental setup, which consists of two tasks: 1) the standard Sportscaster semantic parsing task, and 2) a new RTE-style entailment recognition task. We end the section by providing additional implementation and model details (for more information, see also Appendix C).

4.5.1 Datasets

Sportscaster The Sportscaster corpus (Chen and Mooney, 2008a) consists of 4 simulated Robocup soccer games annotated with human commentary. The English portion includes 1,872 sentences paired with sets **Z** of logical meaning representations. On average, each training instance is paired with 2.3 meaning representations. The representations have 46 different types of concepts, consisting of 22 entity types and 24 event (and event-like) predicate types (see Figure 3.9 for a description and implementation of the Sportscaster language).

While the domain has a relatively small set of concepts and limited scope, reasoning in this domain still requires a large set of semantic relations and background knowledge. From this small set of concepts, the inference grammar described in Section 4.4.2 encodes around 3,000 inference rules. Since soccer is a topic that most people are familiar with, it is also easy to get non-experts to provide judgements about entailment.

Task 1: Semantic Parsing	Match F_1(%)
LexDecoder (Chapter 3)	40.3
Kim and Mooney (2010)	74.2
Chen et al. (2010)	80.1
Best Seq2Seq model (Chapter 3)	83.4
Börschinger et al. (2011)	86.0
Gaspers and Cimiano (2014)	88.7
base semantic grammar (BSG) only	**95.7**
BSG + inference grammar (IG)	95.8
BSG + IG + More Data	**96.3**

Task 2: Inference Task	Accuracy (%)
Majority Baseline	33.1
RTE classifier	52.4
Naïve Inference	59.6
SVM Flat Classifier	64.3
inference grammar (Lex. Inference Only)	72.0
inference grammar (Full)	**73.4**
inference grammar + More Data	72.3

Table 4.1: Results on the semantic parsing (top) and inference (bottom) cross validation experiments (averaged over all folds)

Extended Inference Corpus The extended corpus consists of 461 unaligned pairs of texts from the original Sportscaster corpus annotated with sentence-level entailment judgements (as first shown in Figure 4.2). We annotated 356 pairs using local human judges an average of 2.5 times using a version of the elicitation instructions for RTE from Snow et al. (2008). Following (Dagan et al., 2005), we discarded pairs without a majority agreement, which resulted in 306 pairs (or 85% of the initial set). We also annotated an additional 155 pairs using Amazon Mechanical Turk, which were mitigated by a local annotator.

In addition to this core set of 461 entailment pairs, we separately experimented with adding unlabeled data (i.e., pairs without inference judgements) and ambiguously labelled data (i.e., pairs with multiple inference judgements) to train our inference grammars (shown in the results as *More Data* in Table 4.1) and test the flexibility of our model. This included 250 unlabeled pairs taken from the original dataset, as well as 592 (ambiguous) pairs created by deriving new conclusions from the annotated set. This last group was constructed by exploiting the transitive nature of various inference relations and mapping pairs with matching labels in training to {`Entail`,`Unknown`}.

4.5.2 Evaluation

We perform two types of experiments: first, a semantic parsing experiment (Task 1 in Table 4.1) to test our approach on the original task of generating Sportscaster `LF` representations. In addition, we introduce a new inference experiment (Task 2) to test our approach on the problem of detecting entailments between unobserved sentence pairs using our inference grammars.

For the semantic parsing experiment, we follow exactly the setup of Chen and Mooney (2008a): 4-fold cross validation is employed by training on all variations of 3 games and evaluating on a left out game. Each representation produced in the evaluation phrase is considered correct if it matches exactly a gold representation and (standardly) F_1 score is reported[3].

The second experiment imitates an RTE-style evaluation and tests the quality of the background knowledge being learned using our inference grammars. Like in the semantic parsing task, we perform cross-validation on the games using both the original data and sentence pairs to jointly train our models, and evaluate on left-out sets of inference pairs. Each proof generated in the evaluation phrase is considered correct if the resulting inference label matches a gold inference. We report on the accuracy of predicting the correct entailment label (within the set { `entail`, `contradict`, `unknown/compatible`}).

4.5.3 Implementation and Model Details

As already discussed, we implemented the learning algorithm shown in Algorithm 10 using the k-best algorithm of Huang and Chiang (2005) (i.e., for the KBest computation in line 8) with a uniform beam size k of 1,000. Following Angeli et al. (2012), we also smoothed rule counts (line 10) by using an additive prior α set to 0.05 for lexical word rules and 0.3 for non-lexical rules. To get good initial estimates of word and concept mapping rules, we pre-trained the joint `LF` and inference grammars by first training the base semantic grammars on the original semantic parsing data for 3 iterations. Lexical rule probabilities were also initialized using co-occurrence statistics estimated using an IBM Model1 word aligner

[3] As with Börschinger et al. (2011), since our grammar model parses every sentence, precision and recall are identical, making F_1 identical to accuracy.

(uniform initialization otherwise).

In the inference grammars, 5 additional senses were added to the most frequent event predicates. In terms of other added background knowledge, we made the default assumption that player terms (i.e., `purple1`,`pink1`,...) have a negation relation | with other player terms that they do not match, and assumed all possible semantic relations between all other types (see Appendix C for more details).

4.6 Experimental Results and Discussion

In this section, we detail the main results featured in Table 4.1 for both tasks, and provide some qualitative analysis on the resulting models.

Task 1: Semantic Parsing

We compare the results of our base semantic parser model with previously published semantic parsing results (including some of the experiments from Section 3.5). While our grammar model simplifies how some of the knowledge is represented in grammar derivations (e.g., in comparison to Börschinger et al. (2011)), the set of output representations or interpretations is restricted to the original Sportscaster formal representations making our results fully comparable. As shown, our base grammar (shown as *base semantic grammar* (only) in Table 4.1) strongly outperforms all previously published results even without the additional inference data and rules. Since our approach is similar to Börschinger et al. (2011), one takeaway is that better rule extraction seems to go a long way in improving accuracy, and might help to improve models such as the Seq2Seq model from the last chapter.

We also show the performance of our inference grammars on the semantic parsing task after being trained with additional inference sentence pairs. This was done under two conditions: when the inference grammar was trained using fully labeled inference data and unlabeled/ambiguously labeled data (*more data*). While not fully comparable to previous results, both cases achieve nearly the same results as the base grammar, indicating that our additional training setup does not lead to an improvement on the original task (but nonetheless has minimal effect of the resulting accuracy).

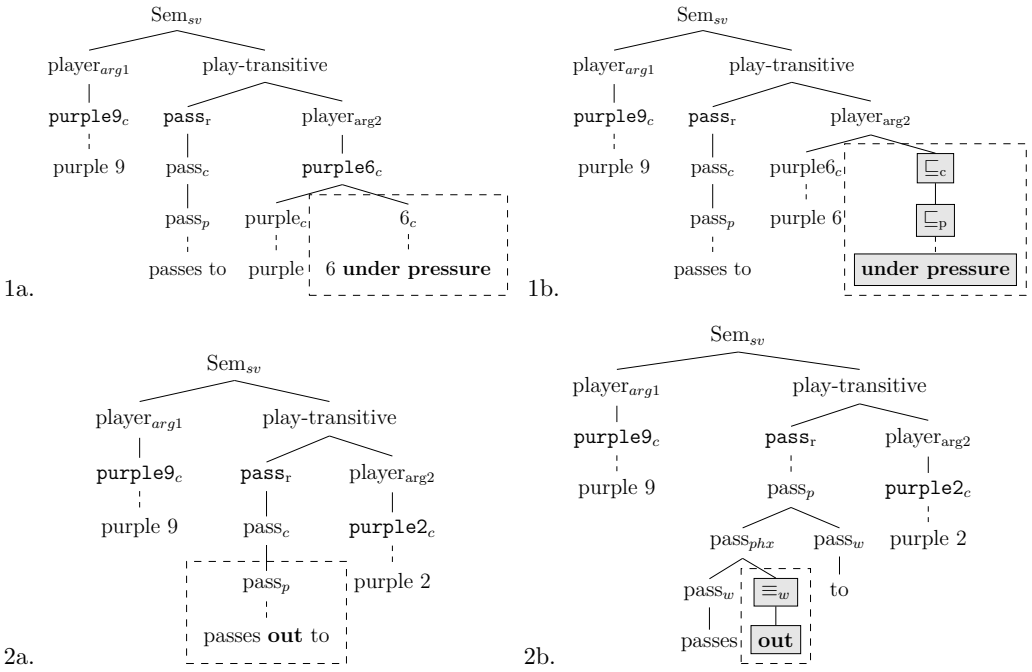


Figure 4.14: Example parse trees (1,2) before (a) and after (b) training on the extended inference corpus (new inferences shown in gray boxes).

Task 2: Inference Task

The main result of this chapter is the performance of our inference grammars on the inference task. For comparison, we developed several baselines, including a *Majority Baseline* (i.e., guess the most frequent inference label from training). We also use an *RTE (max-entropy) classifier* that is trained on the raw text inference pairs to make predictions. This classifier uses a standard set of RTE features (e.g., word overlap, word entity co-occurrence/mismatch). Both of these approaches are strongly outperformed by our main inference grammar (or *inference grammar (Full)*).

The *Naïve Inference* baseline compares the full Sportscaster representations generated by our semantic parser for each sentence in a pair and assigns an `entail` for representations that match and a `contradict` otherwise (as first discussed in

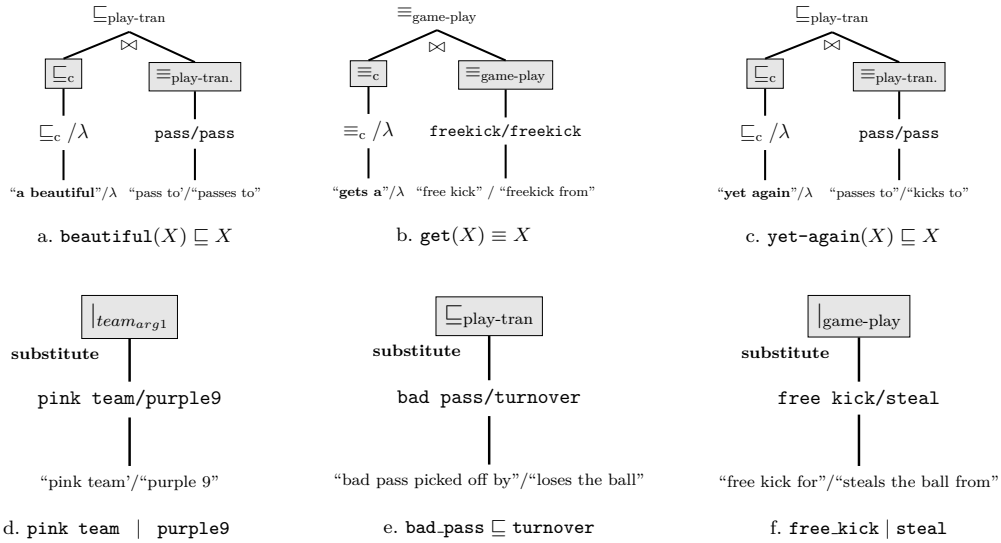


Figure 4.15: Example proof trees involving construction-based (top) and lexical-based (bottom) inferences generated by our model.

Section 4.3.2). This baseline compares the inferential power of the original representations (without background knowledge and more precise labels) to the inferential power of the inference grammars. The strong increase in performance suggests that important distinctions that are not captured in the original representations are indeed being captured in the inference grammars.

We tested another classification approach using a *Flat Classifier*, which is a multi-class SVM classifier (Joachims, 2002) that makes predictions using features from the input and part of the inference model. Such input includes both sentences in a pair, their parse trees and predicted semantic labels, and the alignment between the sentences. In Figure 4.9, for example, this includes all of the information excluding the proof tree in y. This baseline aims to test the effect of using hierarchical, natural logic inference rules as opposed to a *flat* or linear representation of the input, and to see whether our model learns more than the just the presence of important words that are not modeled in the original representations. Features include the particular words/phrases aligned or inserted/deleted, the category of

sense/context error:

1. **t**: *pink9 shoots*
   h: *pink9 shoots for the goal*
   z: `entail` (model prediction: `uncertain`)

semantic parser and alignment error:

2. **t**: *purple8 steals the ball back*
   h: *purple8 steals the ball from pink6*
   z: `uncertain` (model prediction: `contradict`)

alignment and modifier error:

3. **t**: *A goal for the purple team*
   h: *And the purple team scored another goal*
   z: `uncertain` (model prediction: `entail`)

Figure 4.16: Example cases where our inference grammars fail.

these words/phrases in the parse trees, the rules in both parse trees and between the trees, the types of predicates/arguments in the predicted representations and various combinations of these features. This is also strongly outperformed by our main model, suggesting that the natural logic system is learning more general inference patterns.

Finally, we also experimented with removing insertions and deletions of modifiers from alignment inputs to test the effect of only using lexical knowledge to solve the entailment problems (*Lexical Inference Only*). In Figure 4.9 this involves removing "at the goal" from the alignment input and relying only on the grammars knowledge about how `steal` (or "steals the ball") relates to `defense` (or *good defense by*) to make an entailment decision. This only slightly reduced the accuracy, which suggests that the real strength of the grammar lies in its knowledge of lexical relationships or lexical-based inferences.

Qualitative Analysis and Discussion

One benefit of our grammar-based approach is that we can inspect how the system reasons by looking at example derivation trees generated by the model. While our experiments show that the inference grammar does not increase accuracy on the original semantic parsing task, a manual inspection indicates that the model is

nonetheless learning improved representations, as we shown in Figure 4.14. In example 1, for example, the parser learns after being trained on the inference data that the modifier *under pressure* should be treated as a separate constituent in the parse tree. The particular analysis also captures the correct semantics by treating this phrase as forward-entailing, which allows us to predict how the entailment changes if we insert or delete this constituent. Similarly, the parser learns a more fine-grained analysis for the phrase *passes out* to by treating *out* as a type of modifier that does not affect entailment.

Figure 4.15 shows the types of knowledge learned by our system and used in proofs. The top row shows example *construction-based* inferences, or modifier constructions. For example, the first example treats the word *beautiful* in *a beautiful pass* as a type of modifier that changes the entailment or implication when it is inserted (forward-entails) or deleted (reverse-entails). In set-theoretic terms, this rule says that the set of *beautiful passes* is a subset of the set of all *passes*. The bottom row show types of *lexical-based* inferences, or relations between specific symbols. For example, the model learns that the `pink team` is disjoint from a particular player from the purple team, `purple9`, and that a `bad pass` implies a `turnover` event.

Figure 4.16 shows three common cases where our system fails. The first error (1) involves a sense error, where the system treats *shoots* as having a distinct sense from *shoots for the goal*. This can be explained by observing that *shoots* is used ambiguously throughout the corpus to refer to both shooting for the goal and ordinary kicking. The second example (2) shows how errors in the semantic parser (which is used to generate an alignment) propagate up the processing pipeline. In this case, the semantic parser erroneously predicted that *pink6* is the first argument of the `steal` relation (a common type of word-order error), and subsequently aligned *purple 8* to *pink6*. Similarly, the semantic parse tree for the hypothesis in the last (3) failed to predict *another* as a modifier, which would generate an alignment with the empty string λ. The last two cases show the limitation of our approach to generating alignments, and suggests that allowing the model to reason about different possible alignments might help avoid these errors.

Looking ahead, we note that while we use a simplified version of the full natural logic calculus (owing to the simplicity of the Sportscaster domain), our general

approach is amendable to more complex logical systems. For example, we could implement complex projection rules for quantifiers and other linguistic operators by simply introducing new symbols in our grammar with specially designed join rules. In introducing more complex rules that go beyond simple joins, however, we would be greatly expanding the space of possible proofs; whether learning in such a large space (with only minimal background assumptions) is feasible is an empirical question. It also remains to be seen whether using entailment as a learning signal might help to learn other types of complex linguistic structure, which is a question that we leave for future work.

4.7 Conclusions

In this chapter, we considered the problem of training semantic parsers in domains where the target logical forms are underspecified and fail to capture basic facts about entailment and inference. As a general solution to this problem, we introduced a new learning framework called *learning from entailment* that involves adding pairs of sentences annotated with entailment information to the semantic parser's training data. With this added data, we then force the semantic parser to generate explanations of the provided entailments, which forces the model to learn more about the target domain and find holes in the provided annotations.

To experiment with this idea, we performed experiments on the benchmark Sportscaster corpus from Chen and Mooney (2008a), which we expanded to include a corpus of sentence pairs annotated with entailments. As a way of operationalizing this idea of forcing the semantic parser to generate explanations, we created a novel grammar-based semantic parsing architecture and learning strategy that includes a probabilistic reasoning component based on the natural logic calculus (MacCartney and Manning, 2009). With this added machinery, the resulting model is able to jointly learn to generate logic forms, as well as perform symbolic reasoning over symbols in the target learning and solve entailment tasks.

Using our general approach, we achieved state-of-the-art results on the original Sportscaster semantic parsing task. To demonstrate the effectiveness of our model on modeling entailment, we also introduced a new RTE-style evaluation task based on the extended Sportscaster corpus, on which our main inference model strongly

outperformed several strong baselines (with around 73% accuracy). In conclusion, we found that learning from entailment can be an effective technique for improving the representations being learned for semantic parsing, and for making the learning of semantic parsers more robust.