Computer Science Thesis Proposal

A Method for Learning Schemas for Story Understanding and Inference

Lane Lawley

Supervisor
Dr. Lenhart Schubert

April 2020

University of Rochester
Department of Computer Science

Abstract

Story understanding requires significant knowledge about the real world, and the ability to reason about that knowledge. This is because stories seldom include all relevant details, and rely on the reader's knowledge of stereotypical patterns of events to fill in the gaps; if a sleepy character in a story takes an empty mug into the kitchen, we can infer, from our own world knowledge, that they are likely going to make coffee.

We propose a method for accumulating a corpus of such event patterns—or "schemas"—using relatively small data sets composed of simple natural language stories. Our method has three key components: (1) a rich model of schemas based on Episodic Logic (EL), (2) a semantic parser to convert simple stories to EL, and (3) a "head start" of some very simple, general schemas that a young child might already know.

We describe a system wherein the "head start" schemas can be matched to specific details in stories, and then re-generalized into "learned" schemas. As learned schemas accumulate, they can be composed into more complex schemas, or matched to even more stories to learn even better generalizations. In this proposal, we present initial results supporting the efficacy of this approach and describe a plan to accumulate a larger corpus of schemas, use them for prediction and question answering, and evaluate the results with human judgment.

Contents

Chapter 1

Introduction

Truly understanding a story requires a large amount of knowledge about the world, and the ability to reason about that knowledge. Schemas have a long history in cognitive science and artificial intelligence, beginning with their history in psychological literature. In *Remembering* (Bartlett, 1932), Bartlett noted that humans fill in details in stories they don't fully remember based on general knowledge about similar events. Jean Piaget suggested that children start life with a small set of schemas to "seed" the lifelong schema generation process—it is this suggestion, in fact, that forms the philosophical basis of our learning approach.

Schemas have also been heavily explored in more computational musings. Marvin Minsky introduced "frames", in his 1974 article "A Framework for Representing Knowledge" (Minsky, 1974), as data structures that "package up" information about entity types (or specific entities). The FrameNet project (Baker et al., 1998) aims to provide a comprehensive corpus of semantic frames that describe events, objects, and abstract relationships in the real world, although this corpus is hand-generated and represented largely in natural language.

The basic ideas—storing generalized patterns of events, matching small pieces of them to observations, and "filling in" the matched values to the rest of the schema—persist throughout the literature.

Accumulating a sufficient corpus of schemas for story understanding is not an easy task; the breadth of even "common sense" knowledge is immense, and so such knowledge resists manual entry. Corpora like FrameNet (Baker et al., 1998) and PropBank (Palmer et al., 2005) attempt to tackle the problem with hand-generated schemas, but as they use human input to construct each schema, their representations are largely natural language-based, and schema construction is slow. Past approaches to automatic schema acquisition have generally applied statistical or neural network-based techniques to large text corpora, e.g. (Chambers and Jurafsky, 2011;

Pichotta and Mooney, 2016a; Yuan et al., 2018). Symbolic approaches like GENESIS (Mooney, 1990a) learned specializations of known schemas, but started with very complex schemas such as "police officer wanting to arrest criminals", and were not well equipped to deal with actions they had no prior knowledge of. Symbolic approaches have also been largely abandoned in the age of deep learning.

We propose a system that learns schemas in the manner a two-year-old child might: Starting with a handwritten corpus of a few dozen (to perhaps a few hundred) simple *protoschemas* like "X helps Y with action A", "X eats food F to alleviate hunger", and other general actions a small child would be likely to understand, we generate schema matches from simple children's stories, combine these matches into more complex schemas, and save those learned schemas to continue the learning process.

The descriptions of the participants and events in a schema are expressed as formulas in Episodic Logic (Hwang, 1992), a rich and expressive logical form that mirrors many features of natural language and supports efficient inference. We parse stories into the same representation, and match logical formulas from the story parse to logical formulas in our schemas, substituting constant terms for the variables they matched in the rest of the schema. We combine schemas based on heuristics like precondition/postcondition unification, goal/postcondition unification, and temporal ordering, and then generalize the constants of those learned schemas into typed variables for later use.

Using a corpus of several hundred children's stories and a semantic parser to convert them into EL, we have obtained some promising initial results, suggesting the efficacy of the approach: we've learned general schemas like "monkeys eat coconuts" and "animals leave their homes to find and eat food outside" from only one or two examples, and applied them to unseen stories to make predictions. We present a plan, in several broad steps, to scale the system up and demonstrate the effectiveness of this approach for large-scale schema learning:

1. **Construction of a reliable EL parser.** Our current parser is able to convert many simple sentences into Episodic Logic formulas, but it also has many failure cases. Because stories are the "fuel" for our schema learning, our first point of focus should be the systematic analysis of the parser's failure cases. Depending on the number and breadth of complexity of parse errors, we may choose either to switch to an entirely new parser architecture, or just to modify the parser we have and apply pre- and post-processing rules to repair problematic parses.

2. **Creation of more protoschemas.** We currently have 13 general protoschemas, but even a 1- or 2-year-old human child should not be underestimated;

they are aware of many, many sorts of actions in the world that occur in even the simplest stories. To give our system a proper "head start", we plan to hand-craft dozens more protoschemas. Although we believe that hand-crafting is not scalable for schemas in general, even these 13 protoschemas yielded hundreds of matches, as we'll show in Section 4. Because protoschema matches are composed into more complex, multi-step schemas, we believe that the number of schemas our system can learn from a fixed number of stories is a superlinear function of the number of initial protoschemas; some extra manual time investment will likely be well worth it.

3. **Accumulation of a schema corpus.** Once we have an accurate and robust EL parser, and algorithms by which more complex schemas can be pieced together from simpler matches, we will be able to automatically accumulate a schema corpus—after all, our whole purpose is to extract structured "common knowledge" from texts, and there is a lot of knowledge to extract.

4. **Evaluation of the extracted knowledge.** We need to verify that the schemas our system learns are useful and representative of actual common human knowledge. By formulating the learned EL schemas in English, we can present them to untrained human judges, who can rate them according to how logical and general they are. They can also be used to make predictions about novel stories, and those predictions can then be rated on how likely they are.

In this proposal, we will motivate this work by discussing past approaches to schema learning, highlighting their differences from this approach and describing their limitations. Then, we will describe our model of schemas, including the parsing and learning system we've developed so far, and present our initial results. Finally, we will provide more detail on the plan above to finish the system and evaluate the results.

Chapter 2

Background and Related Work

2.1 Symbolic Schema Learning

2.1.1 The Integrated Partial Parser (IPP)

One of the first computational applications of the theory of scripts in (Schank and Abelson, 1977)—as well as the theory of memory Schank presented in (Schank, 1982)—is the **Integrated Partial Parser** (**IPP**), first presented by Michael Lebowitz, a student of Roger Schank's, in his dissertation (Lebowitz, 1980). [1] IPP's main focus is on the integration of *episodic memory*—a term coined by Endel Tulving to describe the comprehensive memory of an event, as opposed to the explicit knowledge of declarative facts, which he termed *semantic memory* (Tulving et al., 1972)—into conceptual understanding and generalization processes. Schank argued, in (Schank and Abelson, 1977) as well as many future works, that episodic memory must be organized around "stereotyped" episodes—i.e., scripts—for the purposes of both efficient lookup *and* reasoning about new situations that seem to "follow" the scripts. IPP provides a concrete first computer implementation of this sort of general episodic memory, and demonstrates its potential for generalization.

Outline of the Approach

IPP organizes its episodic memory in terms of two fundamental types of unit. The first is called the **Simple Memory Organization Packet** (**S-MOP**), and is inspired by the Memory Organization Packets defined in (Schank, 1980). S-MOPs correspond to the general notion of schemas: they are packets of information rep-

[1] A contemporary application of (Schank, 1982) is the **CYRUS** fact retrieval system developed by Janet Kolodner (Kolodner, 1983), which I don't describe in this paper.

resenting generalized, stereotypical events (i.e., scripts). The other type of unit is the **Action Unit** (**AU**), which represents a concrete event, though not necessarily an instance of an S-MOP. AUs resemble frames in the Minsky sense: they are attribute-value structures.

N.B.: in all of the literature on IPP, the term "Action Unit" is used to describe both the specification of the AU's attributes and *any concrete instantiation of an AU specification. For the remainder of this section, I will use the terms* "AU template" *and* "AU instance" .

An example of two AU instances is shown in Figure 2.1.

S-MOPs comprise three core sections, each of which is hierarchical in that it may be composed of other S-MOPs, or of AUs. The AUs are "atomic", and thus somewhat of an analog to the active conceptualizations in Schank's theory, although they may represent much more complex actions or situations. The three core sections of an S-MOP are:

1. **Methods**: this section represents "the way the S-MOP is carried out", and is analogous to the *track* in Schankian scripts. For example, the S-MOP `S-EXTORT` might have the AU `$HIJACK` as its method. An S-MOP only has one method.

2. **Results**: this section represents the effects made true when the event described by an S-MOP is complete. Example results for `S-EXTORT` are the AUs `GS-RELEASE-HOSTAGES` and `GS-GET-RANSOM`.

3. **Scenes**: this section represents events that "often occur" in instances of the S-MOP; it is analogous to the scenes in Schankian scripts. Scenes are an optional component of an S-MOP: they are not required to understand the nature and effects of the S-MOP, but can provide additional information on it. Example scenes for `S-EXTORT` are the AUs `G$-NEGOTIATE` and `G$-SIEGE`.

Story Parsing

To understand stories, IPP must convert them into its memory representation. They analyze the story with a syntactic parser, which identifies AUs by keywords, and then makes use of more keywords, as well as syntactic features, to fill their roles. An example of a rule specification for filling the roles of the `$HIJACK` AU is shown in Figure 2.2. When processing the story `A JORDANIAN GUNMAN HIJACKED A KUWAITI JETLINER WITH 112 PASSENGERS ABOARD THURSDAY AND FORCED IT TO FLY TO BAHRAIN`, the word `HIJACKED` triggers an instantiation of the `$HIJACK` AU. The subject of the sentence is assigned to the `ACTOR` slot. "Jetliner" is recognized as an airplane, and is assigned to the "VEHICLE" slot. The means by which "Jetliner" is known to be

```
$HIJACK
    ACTOR       = *gunman*
    PLANE       = *Portuguese 727*
    PASSENGERS  = *83 passengers*

GS-GET-RANSOM
    ACTOR       = *gunman*
    AMOUNT      = *$10 million*
```

Figure 2.1: Two example instances of IPP's action units (AUs)

```
ACTOR       [terrorist]
PASSENGERS  [group of people]
VEHICLE     [airplane]
TO          [city or country] [prep "to"]
FROM        [city or country] [prop "from"]
```

Figure 2.2: The rules for filling the roles in the `$HIJACK` AU

an airplane are unclear, but alluded to in various sections of the dissertation: words can store pointers to other words as synonyms, and things like common names can be recognized to satisfy a "person predicate"; in any case, this lexical knowledge is hardcoded into the system.

S-MOPs may be instantiated by lexical "triggers" in similar ways, and their instantiation triggers the attempted instantiation of their components, which are either more S-MOPs or AUs. There are several complex heuristics for attaching the AUs to S-MOPs. I won't detail the specifics of these rules here; for our purposes, it suffices to note that these rules exploit the hierarchical nature of composed S-MOPs, and the shared role fillers between them and their children, to identify the children in a "top-down" manner, or to attach previous "orphan" instances to their relevant S-MOPs.

Generalization

Instantiated S-MOPs are generalized into something called **spec-MOPs** (specialized S-MOPs), which represent the general class shared by several concrete instances of an S-MOP. For example, one of their results was the `ITALY-KID-GEN` spec-MOP, derived from several `S-EXTORT` instances of stories about kidnappings in Italy. The spec-MOP generalization algorithm works on the basis of the storage system used to index S-MOPs, and their instances, by similarity.

S-MOP instances are stored in a structure called a *discrimination net* (Charniak et al., 1980), a branching graph of predicates that classifies items by successively applying the predicates to move them along the path to their class. It can be thought of as a generalization, to directed acyclic graphs and n-ary branch orders, of a decision tree. S-MOP instances are indexed by adding them to this structure as terminal leaves, with the predicate branches given by tests for their features. The logarithmic depth of a balanced discrimination net enables efficient retrieval of S-MOP instances, and thus of the S-MOPs they instantiate: few features need to be tested for retrieval.

To keep the number of leaves small and ensure efficient balancing, leaf S-MOP instances are generalized based on a similarity heuristic. The heuristic used in the original system was "share at least 4 features" when generalizing S-MOP instances into a spec-MOP, or "share at least 2 features" when generalizing spec-MOP instances into another, even more specific spec-MOP. However, the author suggests that better heuristics are admitted and encouraged. The new spec-MOP template is created by removing the shared features, constraining them to their shared value, as can be seen in the generalization of `S-EXTORT` instances to `ITALY-KID-GEN` based on, for example, the shared location of Italy.

Implications for the Natural Emergence of Schemas

It is particularly interesting that IPP's generalization procedure can be seen as a "by-product" of a well-known procedure for building balanced decision trees. This illustrates how the capacity for story generalization—and, later, even analogical reasoning—can *emerge* from the somewhat more mundane, logistical-sounding task of optimizing the recall of specific episodes from a database. This property of IPP provides compelling computational evidence for this statement from (Schank and Abelson, 1977): *"As an economy measure in the storage of episodes, when enough of them are alike they are remembered in terms of a standard generalized episode which we will call a script."*.

IPP's storage-derived generalization procedure is not the only support for that statement: in a study of recollections made by former White House Counsel John Dean, Ulric Neisser compared Dean's statements with factual recordings of the events Dean claimed to be describing, and found that Dean's recollections shared many properties of the actual events, but were not specifically identical to any of them (Neisser, 1981). Neisser called this "repisodic" memory, and offhandedly referred to the case study as possible evidence for "theoretical ideas about 'story grammars' and 'schemata'".

Construing Neisser's study of John Dean as evidence for schemas-as-memory in the human brain also supports an interpretation of IPP's generalization procedure as evidence for the natural emergence of schemas from efficient episodic retrieval techniques. Evidence has been presented suggesting that potential neural circuitry for episodic memory evolved slowly, in pieces, across a long timeline of ancestors and species (Allen and Fortin, 2013). This suggests that each "step along the way" to our own episodic memory was locally beneficial to the survival of a species, and thus that schema-like representations in the human brain could be the result of compounded "micro-improvements" to our memory systems—much like IPP's generalized S-MOPs are a consequence of its efficient episode indexing system. Of course, I'm not an evolutionary psychologist, so I'll put this line of speculation to rest here.

Discussion

My excessive fawning *supra* over possible implications of IPP's generalization scheme is not to say that it is necessarily a scalable system, as-is, for the practical task of schema learning; in fact, I believe it has several practical shortcomings, mostly related to the amount of hand-engineered information needed to "bootstrap" it. Note that IPP can only generate *spec-MOPs*: *specifications* of known S-MOPs. This means that the initial set of S-MOPs must, in principle, conceptually "cover" all schemas one wishes to generate.

This scalability concern also holds for the Action Unit definitions—including the rules used to match tokens in a syntactic parse to AU roles. Converting a syntactic parse tree into a cognitive action representation requires a magnitude of semantic reasoning and knowledge about objects, and the ontologies in which those objects are embedded, that is intractable to embed by hand in the IPP's format.

Many of these bootstrapping issues have enjoyed the attention of reasonably successful contemporary research: **(1)** "fuzzy" semantic word representations have seen rapid improvement as distributional word embeddings, e.g. (Mikolov et al., 2013a), as well as in more comprehensive *language models* (distributions of possible sentences), e.g. ELMo (Peters et al., 2018); **(2)** the problem of textual semantic role labeling given a corpus of known frames needs no longer rely on syntactic heuristics alone, as it does in IPP (see Section **??** for examples of automatic SRL systems); and **(3)** "world knowledge" about objects, of the kinds that IPP requires for role filling, is beginning to accumulate, in both hyponymy ontologies (for determining "is-a" relationships), e.g. WordNet's hypernym graph (Miller, 1995), and in corpora of what Pustejovsky (1991) calls "telic" knowledge (for determining "what something is *good for*") (Yamada and Baldwin, 2004).

But IPP assumes one thing a mass schema generator cannot assume: the existence of a large number of initial schemas. We'll now look at later approaches to schema generation, and examine whether they are better suited to unsupervised schema learning.

2.1.2 GENESIS

Most approaches to schema acquisition have used statistical extraction models to cope with the vast amount of natural language text needed for automatic generalization. However, the GENESIS system, first presented in Ray Mooney's dissertation (Mooney, 1988), attacks the schema learning problem symbolically, generalizing causal, structured schemas without the need for a huge number of examples.

GENESIS generalizes schemas [2] from single narratives using *explanation-based learning* (Mitchell et al., 1986), a symbolic learning method which uses a set of symbolic rules to "explain" observed actions, thereby offering clues for what is generalizable without compromising the underlying teleology of the narrative. [3]

[2] I am hesitant, unlike these authors, to use the Greek pluralization rule for the word *schema*.

[3] As (Mooney, 1990b) notes, explanation-based learning can be viewed in a dichotomy with *similarity-based learning*, which depends on a large number of examples and a similarity metric to learn generalized structures—neural networks and other statistical schema learning approaches, to be detailed in further subsections of this chapter, are examples of this latter end of the dichotomy.

Outline of the Approach

At its core, GENESIS is a cyclic combination of classical planning and pre-existing schema matching techniques. A schema can only be used to interpret a teleological sequence of actions (i.e. a *plan*) if that schema is already known. Absent any relevant schemas, GENESIS resorts to classical situation planning techniques, manipulating an internal logical representation of a story to generate a plan, composed of known operations, that aligns with the character's actions in the story. If such a story-explaining plan is found, it is then analyzed, and novel action sequences necessary to achieve important goals are generalized and stored as schemas.

The technique of storing plans to enable "sub-sequence" analysis is not new: the STRIPS problem solver system stored generalized plans in just such a way using *triangle tables*: data structures that allowed efficient determination of the next applicable step of a multi-step plan given an arbitrary world state (Fikes et al., 1972). In fact, this similarity is the result of direct inspiration: GENESIS's internal logical representation and planning algorithm are modified versions of those used in STRIPS.

Knowledge Representation

GENESIS first converts stories into an internal knowledge representation, whose ontology comprises *object*, *attribute*, *state*, and *action* classes. *Objects* are atomic, symbolic representations of entities in the planning domain, e.g. *sunrise* or *Jacob*. Note that these are not necessarily *physical* entities, but rather, the entire "domain" of the logical system in which the planning inferences occur. *Attributes* are "rules" about objects that are unaffected by actions. *States* are conjunctions of temporal propositions representing the world at a given time. *Actions* are temporal predicates endowed with axioms whereby associated *preconditions* are implied by the action predicate's truth at some time, associated *postconditions* are implied at some time after the action, and associated *motivations* for the action are implied of relevant entities by the action's truth at some time.

The "frame problem" (McCarthy and Hayes, 1969) of axiomatizing the temporal persistence of these predicates is handled by the "STRIPS assumption" (Fikes et al., 1972): stated simply, propositions may be added to or deleted from the working knowledge base at some timestep [4] if and only if an action occurred with those postconditions at that timestep.

[4]Under the *closed-world assumption* (Reiter, 1987), we assume that any predicates not present in the knowledge base are false; the knowledge base is parameterized by a discrete time variable.

Arrest(?a,?b,?d(?b))			
Constraints	**Preconds**	**Motivations**	**Postconds**
Isa(?a,Character) Isa(?b,Character) ¬Equal(?a,?b) Illegal(?d(?b))	At(?b,?l) At(?a,?l)	Believe(?a,?d(?b)) Occupation(?a,cop)	Arrested(?b,?d(?b),?a)

Figure 2.3: A table from (Mooney, 1990b) illustrating an *Arrest* example action in the GENESIS representation. Constraints are atemporal and checked against state-independent attributes, whereas preconditions are temporal and state-dependent.

The Planning Process

Because GENESIS is a schema learning system, it deals primarily with *action*-type knowledge; an example action, taken from (Mooney, 1990b), is shown in Figure 2.3. If GENESIS encounters an action in a story that does not match an existing (or instantiable) schema, it will attempt to deduce a teleology, via planning, to explain the action with past actions. The unknown action's preconditions are taken as goals, and a backward-chaining inference process begins on the Horn clauses in the knowledge base at the action's time: the system must prove that the preconditions and motivations of the unknown action were made true by the postconditions of past actions.

The authors note that motivations are seldom made explicit, and so they manually encode prior *thematic goals* [5] into GENESIS. An example of a thematic goal, taken from their paper, is a police officer's desire to arrest lawbreakers: `Arrested(?b,?d,?a)` $\wedge$ `Occupation(?a, cop)` $\implies$ `ThemeGoalMet(?a, Arrested(?b, ?d, ?a))`.

Schema Generalization

Once the process of achieving a goal has been explained, GENESIS uses three criteria to determine whether to generalize the explanation into a schema, in order to keep the massive number of possible generalizations to a useful minimum. These criteria are designed to encourage the generation of novel "end-to-end" schemas that one character can undertake as composite actions in order to achieve an inherently useful and interesting goal. They are:

1. The goal achieved should be a thematic goal, rather than a composite goal.

[5] (defined by (Schank and Abelson, 1977) as atomic goals, derived from atomic needs and wants, which require no explanation)

2. The explanation should not just be an instantiation of an already learned schema.

3. The top-level actions of the explanation should be undertaken by the character whose goal we've explained.

If an explanation meets these criteria, GENESIS "prunes" the explanation, removing irrelevant and/or overly restrictive facts. For example, a top-level explanation of an arrest might have four steps: a police officer goes to a known drug producer's house, the police officer tricks the drug producer into letting them inside, the police officer observes the drug lab, and, finally, the police officer makes an arrest, satisfying a thematic goal. However, these actions are composite: perhaps the police officer tricks the drug producer into letting them inside by pretending to be their uncle. Pretending to have a false identity may be essential to satisfying a constraint that A only lets B in their house if A thinks B is trusted. However, the fact that uncles are trusted is not necessary in this causal graph, and may thus be pruned for generalization.

Finally, GENESIS generalizes the schema by removing unnecessary restrictions using a generalization algorithm (EGGS) given by (Mooney, 1988) [6]. Mooney gives the following example plans to help illustrate what restrictions are "necessary':

- `GoThru(Door1,Room1,Room2)`
- `PushThru(Box1,Door1,Room2,Room1)`

With the constant-to-variable substitutions `Door1` ← `?d`, `Room1` ← `?r1`, `Room2` ← `?r2`, and `Door1` ← `?b`, the resulting plan is *too specific*: the preconditions of the operators, as defined earlier in that paper, do not require that the robot push the box into the same room it started in. However, consider this plan:

- `GoThru(Door1,Room1,Room2)`
- `PushFromRoom2ToRoom1(Box1)`

The conditions for `PushFromRoom2ToRoom1` do, in fact, require that the robot start in room 2 and end in room 1. So, simply replacing the constants with variables is *too general*: the rooms *must* be Room1 and Room2. So, the algorithm must replace constants by variables whenever possible, subject to the constraint that the pre- and post-conditions of the relevant operators are respected.

[6](which is itself derived from the STRIPS generalization algorithm given by (Fikes et al., 1972))

```
Input: Alice was at a corner wearing tight blue jeans.
   Stan told Alice if she had sex with him then he would give her $75.
   Stan went to jail.

Ready for questions:

> Paraphrase.
Stan solicited Alice's sexual favors for $75. Stan was put in a jail.

> Why did Stan tell Alice if she had sex with him then he would give
   her money?
Because Stan believed that Alice was a prostitute and because Stan
   needed to have sex.

> Who arrested Stan?
Answer unknown.
```

Figure 2.4: An unproductive Q&A session with GENESIS after it had only seen the sparse test narrative.

It turns out that the algorithm for doing this involves unifying all possible pairs well-formed formulas (WFFs) in the logical representation using the unification algorithm given by (Robinson et al., 1965). The resulting unifications (which can be thought of as bindings, or substitution maps) are then applied to the entire explanation structure. Once this is done, GENESIS has produced a generalized schema for doing "the kind of thing" a character did in the story to achieve their goal.

Example Generalization

The authors present a "case study" of GENESIS wherein a "learning narrative", which described a schema instance in great detail, was presented to the system, followed by a "test narrative", which described another instance of the same schema but left out many important actions. The system was able to apply the generalized schema it obtained from the learning narrative to the test narrative, constructing an explanation from very sparse information.

To show that the test narrative was insufficient for question-answering purposes, the authors published the following interaction with GENESIS on the basis of their input (the test narrative). This is shown in Figure 2.4. After presenting it with the learning narrative (Figure 2.5), it generalized a schema, and applied that schema to the test narrative to fill in the story details (Figure 2.6).

```
Input: Jane is a policewoman.
   She dressed in a short red skirt and went to a corner.
   Bob approached the corner and told her if she had sex with
      him then he would give her $50.
   Jane arrested Bob for soliciting.
   Bob is Mary's husband and he told her that Jane entrapped him.

Thematic goal achieved: Jane is happy that Bob is under arrest for
solicting[7] Jane's sexual favors for the $50.
Explanation suitable for generalization.
Pruning...
Generalizing...
Packaging...
Creating New Schema: (DressSolicitArrest ?b9 ?c2 ?s2 ?s4 ?I4 ?a9 ?c3)
?b9 is not equal to ?a9.
?b9 is a person. ?c2 is an apparel. ?b9 has ?c2. ?b9 puts on ?c2.
?s2 is a location. ?I4 is a corner. ?b9 is at ?s2.
?b9 goes from ?s2 to ?14. ?a9 is a person. ?s4 is a location.
?a9 is at ?s4. ?s9 goes from ?s4 to ?14. ?c3 is money.
?a9 needs to have sex. ?c2 is sexy.
?a9 solicits ?'b9s sexual favors for ?c3. ?b9 is a police officer.
?b9 arrests ?a9 for soliciting ?'b9s sexual favors for ?c3.
Having: ?b9 put on ?c2. suggests DressSolicitArrest
Having: ?a9 solicited ?b9's sexual favors for ?c3. suggest[8]
   DressSolicitArrest
Having: ?b9 arrested ?a9 for soliciting ?'b9s sexual favors for ?c3,
   suggest
DressSolicitArrest
Unknown word `?b9 entrap ?'a9 refers to DressSolicitArrest
Ready for questions:
> Paraphrase.
Jane had a skirt. Jane put on a skirt. Jane was at some place.
Jane went to a corner. Bob was at some place. Bob went to the corner.
Bob needed to have sex. The skirt was sexy.
Bob solicited Jane's sexual favors for $50. Jane was a police officer.
Jane arrested Bob for soliciting 'Janes sexual favors for the $50.
> Why did Bob go to the corner?
So Bob could solicit 'Janes sexual favors for the $50.
> Why did Bob solicit 'Janes sexual favors?
Because Bob believed that Jane was a prostitute and because Bob
needed to have sex. .
> Why did Jane arrest Bob?
Because Jane believed that Bob solicited 'Janes sexual favors for the
$50 and because Jane was a police officer.
```

Figure 2.5: A schema derivation and Q&A session with GENESIS after it had seen the learning narrative.

```
Input: Alice was at a corner wearing tight blue jeans.
Stan told Alice if she had sex with him then he would give her $75.
Stan went to jail.
Thematic goal achieved: Alice is happy that Stan is under arrest for
   soliciting 'Alices sexual favors for the $75.
Ready for questions:
> Paraphrase.
Alice had a pair of jeans. Alice put on the pair of jeans.
Alice was at some place. Alice went to a corner.
Stan was at some place. Stan went to the corner.
Stan needed to have sex. The pair of jeans were sexy.
Stan solicited Alice's sexual favors for $75.
Alice was a police officer.
Alice arrested Stan for soliciting 'Alices sexual favors for the $75.
> Who arrested Stan?
Alice arrested Stan for soliciting 'Alices sexual favors for the $75.
> Why did Alice arrest Stan?
Because Alice was a police officer and because Alice believed that
Stan solicited 'Alices sexual favors for the $75.
```

Figure 2.6: A much more productive Q&A session regarding the test narrative with GENESIS after it had formed a schema from the learning narrative.

Discussion

The generalized schema shown in figure 2.5 seems quite comprehensive: in addition to "type" predicates on its entities, e.g. `?b9 is a person`, it also relates those entities via actions, e.g. `?s9 goes from ?s4 to ?l4`. Furthermore, the actions are not simply subject-verb-object triples: they specify pre- and post-conditions, as well as motivations, all of which contribute to the complexity and goal-orientedness of the final schema. However, I do want to note some important concerns related to the scalability of its schema learning.

The stories that GENESIS learned from were artificial, containing very little extraneous detail and really instantiating only one substantial schema. Real stories often have many simultaneously active schemas, as well as details and stylistic artifacts that complicate schema recognition. GENESIS's causal explanation-based generalization does seemingly deal with some of these issues, but its efficacy on real, "dirty" stories has not been demonstrated at scale.

The causal explanations that GENESIS relies on for schema construction also rely on a significant corpus of hand-generated actions, which specify preconditions, postconditions, and goals. The STRIPS-style, first-order predicate logic these actions are written in also lacks the expressive power of natural language. This all limits GENESIS rather severely: without a complete understanding of the world-meaning of every action verb it encounters, GENESIS is unable to generalize schemas from natural language. This sort of complete understanding would be very difficult to give a schema learning system *a priori*.

GENESIS (subsection 2.1.2) was a rather compelling attack on the task of schema learning, but I do think that these issues ultimately lead to poor scalability.

2.2 Statistical Schema Learning

Shortly after GENESIS, there was a cultural shift in approaches to schema learning: probabilistic models of event sequences dominated, and classical explanation-based learning fell out of vogue. The reasons for this seem to be twofold: (1) "connectionist" machine learning techniques like neural networks were becoming viable again due to the introduction of the backpropagation training algorithm by (Rumelhart et al., 1986); and, (2) as Karl Pichotta points out in his Ph.D. thesis (Pichotta, 2017), natural language is often ambiguous, and non-probabilistic, symbolic models

[7] [sic]

[8] Also [sic], and somewhat suspicious—why would this inconsistency with "suggests" on the previous line occur in what is ostensibly the literal output?

of events do not easily support the kind of probabilistic inferences we make when interpreting ambiguous texts.

This section will explore some examples of the statistical kind of schema learning that has come to dominate the literature since the mid-80s, then briefly discuss the advantages and disadvantages of this sort of approach.

2.2.1 Unsupervised Learning of Narrative Event Chains

One of the earlier influential papers on statistical script learning is *Unsupervised Learning of Narrative Event Chains* (Chambers and Jurafsky, 2008). This paper defined, and sought to learn, *narrative event chains*: a set of events with a mutual "protagonist" and a partial temporal order. The events in the chains were modeled as 2-tuples of verbs and entities.

Event Model

The authors defined events as 2-tuples of of the form *(event, dependency)*, where *event* is a verb, and *dependency* is an entity—the protagonist—as either the *subject* or *object* of the verb. This latter element of the tuple is called a *typed dependency*.

The Learning Method

The authors use a three-stage approach to learning event chains:

1. First, they perform **narrative event induction** on the syntactically parsed text corpus: events are extracted from the parsed clauses. Then, they calculate, using approximate pointwise mutual information (PMI), a pairwise relationship metric[9] for the set of events, based on the co-occurrence of shared grammatical arguments—the hypothesis here is that shared arguments imply co-occurrence of the events in a narrative chain. The pairwise event co-occurrence scores can then be combined to find the next most likely event in a chain given all the events in the chain so far by maximizing $\max_{j:0<j<m} \sum_{i=0}^{n} pmi(e_i, f_j)$[10], where e_i is the i-th event in a chain of n events, candidate event f_j is the j-th event

[9] More specifically: the pointwise mutual information of the outcomes of two random variables—here, event representations drawn from a distribution of observed ones—is a quantification of the effect of statistical dependence in the co-incidence of both outcomes.

[10] N.B.: maximizing this quantity is equivalent to picking the event f_j from the training corpus with the most occurrences of shared co-referring entity arguments with e_i, as can be inferred from the definitions to follow.

in the training corpus of m events, and $pmi(e_i, f_j)$ is an approximation of the pointwise mutual information of e_i and f_j.[11]

The approximate pointwise mutual information of the two events is defined by

$$pmi(e_i, f_j) = \log \frac{P(e_i, f_j)}{P(e_i)P(f_j)}$$

The marginal event probabilities in the denominator of the PMI approximation are given simply by the observed frequencies of those events in the training corpus. The approximated pairwise joint event distribution in the numerator is defined by

$$P(e_i, f_j) = \frac{C(e_i, f_j)}{\sum_{e1} \sum_{e2} C(e1, e2)}$$

where, for any pair of events $e1$ and $e2$ having respective words $w1$ and $w2$ and respective dependencies $d1$ and $d2$, $C(e1, e2)$ is the number of times the dependencies $d1$ and $d2$ were filled by co-referring entities. Put more simply: this PMI approximation assumes that any statistical dependence in the joint distribution is due to shared argument entities, and thus quantifies the effect of entity co-reference in event co-occurrence. Put *even more* simply: an event is more likely to come next if it shares arguments with the last event.[12]

2. After the narrative event chain distribution is induced, they compute **partial temporal ordering** by using support vector machines (SVMs) to predict the presence of a *before* relation of two events using syntactic information, including verb tense and grammatical aspect. The training data for the SVM was taken from the Timebank corpus of event relations (Pustejovsky et al., 2003), whose relations were simplified down into the *before* relation used by this paper, and an *other* relation.

3. Finally, they use an agglomerative clustering algorithm—in which events begin with their own clusters, which are then merged based on a similarity metric—to **identify the event chains**. The clustering algorithm uses the pointwise mutual information from the first step as a similarity score. Once the chains are constructed, their temporal ordering is induced by the SVM model in the second step.

[11] N.B.: they only need to calculate pointwise mutual information here, rather than the full distributional mutual information, because of their "Markov assumption" that an event's likelihood is predicted only by the last event.

[12] Put *comedically* simply: "yeah, that makes sense, I recognize that guy from before".

Experimental Setup

The authors used the Gigaword corpus (Parker et al., 2011) as a document source for their experiments. The protagonist for each document is the entity with the highest number of mentions—they used the OpenNLP co-reference engine (Baldridge, 2005) for entity recognition. Evaluation was performed using a metric the authors introduced in this paper: the **narrative cloze** task, an extension to narratives of the cloze test defined by (Taylor, 1953), in which humans were tasked with filling in missing words in texts. In the narrative cloze test, the system fills missing event tuples into narrative chains.

Results

They defined the baseline for their evaluation as a version of their model that did not use the protagonist, or its relation to each event as a subject or object, in making predictions. As the baseline could then only produce verb event chains, rather than 2-tuple event chains, they then modified their experimental model to use the protagonist co-reference information during training, but to omit it in the output chains. It is unclear to me, from reading the paper, where the "golden" event data for each document they used to compute accuracy came from. They reported a nearly **37%** accuracy improvement relative to the baseline, demonstrating the efficacy of the typed protagonist information. Removing the OpenNLP co-reference resolution step caused accuracy to drop by **5.7%**.

Discussion

This was a hugely influential paper in statistical script learning, and while the authors themselves note that the event sequences are a far cry from the richness of Schank & Abelson's scripts, as defined in (Schank and Abelson, 1977), they did pave the way for more expressive event models in future approaches, such the 5-tuples in (Pichotta and Mooney, 2016b), which I describe in Section 2.2.2.

2.2.2 Learning statistical scripts with LSTM recurrent neural networks

Karl Pichotta and Raymond Mooney's paper *Learning statistical scripts with LSTM recurrent neural networks* (Pichotta and Mooney, 2016b) was the first application of long short-term memory (LSTM) neural networks to the task of script learning. LSTM units, introduced by (Hochreiter and Schmidhuber, 1997), can be thought of as memory cells that learn when to store, and when to forget, input items. Recurrent

neural networks—that is, neural networks whose neuron connection graphs contain a cycle–enhanced with LSTM units are extremely powerful sequence learners, and have been applied to such diverse tasks as handwriting recognition (Graves and Schmidhuber, 2009), machine translation (Sutskever et al., 2014), and music generation (Eck and Schmidhuber, 2002). Hoping that the memory cells would help learn long-range narrative structure in event sequences, Pichotta and Mooney applied them here to script learning.

Event Model

Pichotta and Mooney extend the 2-tuple event representation from (Chambers and Jurafsky, 2008) into a 5-tuple representation of the form (v, e_s, e_o, e_p, p), where v is a lemmatized verb (i.e. the naked dictionary-form verb); three nouns e_s, e_o, and e_p are related to the event as the subject, object, and prepositional attachment of v, respectively; and p is the preposition that attaches e_p to v. Each element of the tuple is known as an *event component*, and, excepting v, may be *null* (i.e. unspecified). (Note that e_p is unspecified if and only if p is unspecified; they are coupled event components.)

The authors define two types of script model from these event tuples. **Noun models** predict the verb, noun, and preposition lemmas of events in a sequence, as a generative model. **Entity models** predict almost the same sequences, but substitute *entity IDs* for nouns. Entity IDs are unique integers representing entities with multiple mentions in the text, and are derived from a co-reference analysis. They note that prior script learning systems, e.g. (Chambers and Jurafsky, 2008), had all been entity models.

Given these script model definitions, they define four separate architectures, each trained independently:

1. **LSTM-noun-noun**: predicts argument nouns given argument nouns.
2. **LSTM-ent-ent**: predicts entity IDs given entity IDs.
3. **LSTM-both-noun**: predicts argument nouns given both argument nouns and entity IDs.
4. **LSTM-both-ent**: predicts entity IDs given both argument nouns and entity IDs.

The Learning Method

Because LSTMs make their sequential predictions "one step at a time", each tuple must take five timesteps to predict. Each timestep's input comprises several one-

hot vectors (i.e. vectors of all 0s, except for one 1), which are assigned continuous distributional representations, a.k.a. *embeddings* (similarly to the one-hot vocabulary vectors in (Mikolov et al., 2013a)). The first one-hot vector gives the current tuple index, ranging from 0 to 5; this modular sequence is easy for the LSTM to learn, and ensures that the timesteps output valid groups of 5 tuple elements. The second one-hot vector represents the vocabulary word at that index. Finally, the third one-hot vector—present in all architectures except for **LSTM-noun-noun**—represents the entity ID at the current timestep.

The embeddings of these vectors are given as input to the LSTM network, which produces an output vector of the same length as the vocabulary. The vector is normalized into a probability distribution by a softmax function. The network is trained by back-propagating an error value calculated with the *cross-entropy method* (Rubinstein and Kroese, 2013), in which the cross-entropy—roughly, the average number of bits needed to encode an outcome from some distribution Q, weighted by the probability of that outcome in another distribution P—between the output distribution and the target distribution is iteratively minimized for each training sample. Entity models also produce a probability distribution over the next entity ID in the same way. Likely tuples must be constructed from the most likely sequences of samples from five consecutive word and entity distributions; finding the most likely such sequence is intractable, as there are $|V|$ entries in each probability distribution (where V is the vocabulary set), for a total search time bounded by $O(5^{|V|})$. So, for a more tractable approximation, likely sequences of 5 observations are generated using a five-step beam search.

Narrative Cloze Experiment

The first evaluation was performed on the narrative cloze model introduced by (Chambers and Jurafsky, 2008) (and described here in Section 2.2.1 on page 21). Because they defined a new event representation, the authors had to define their baseline against which to evaluate their architectures. They actually defined four baselines:

1. The **unigram model** ignores the document and samples events by their frequency in the training set. They note that (Pichotta and Mooney, 2014) showed this baseline to be "surprisingly competitive". They created a noun unigram model and an entity unigram model to cover all of their architectures.

2. The **all-bigram model** constructs statistics of event "skip" bigrams, where up to two events were allowed to "intervene" in between the events in the bigram. This is called a 2-skip bigram model, and is an instance of the more general

skip n-gram model developed for script learning systems by (Jans et al., 2012). The 2-skip bigram model models the probability of an event in the sequence given the previous event in the sequence, and can be thought of as a Markov approximation of the full joint distribution. To generate an inference for some position t in the sequence, the authors maximize the objective function

$$S(a) = \sum_{i=0}^{t-1} \log P(a|s_i)$$

where s_i represents the i-th event of the sequence, and $P(b|a)$ is the probability of event b given the previous event a, given by the 2-skip bigram model.

3. The **rewritten all-bigram model** is a version of the all-bigram model with different behavior for co-occurring events, that is, events with co-occurring entities. For all co-occurring events a and b in the training process, the training data is augmented with all possible event pairs equivalent to a and b except for some subset of the shared entity IDs being re-written as other entity IDs. The intuition here, the authors say, is that co-occurrence relationships between events are likely to be *generalizable* to other entities. Inferences are generated by maximizing $S(a)$, as they were in the all-bigram model.

4. The **2D rewritten all-bigram model** is like the rewritten all-bigram model, except it generates inferences by maximizing the objective function

$$S_2(a) = \sum_{i=0}^{t-1} \log P(a|s_i) + \sum_{j=t+1}^{l} \log P(s_j|a)$$

where l is the sequence length. (Note that the first term of $S_2(a)$ is the definition of $S(a)$ from the all-bigram model.) The idea here is to incorporate not just the probability of an event following some event s_i, as in $S(a)$, but also the probability of that event *preceding* some event s_j.

The authors found **LSTM-both-noun** to be the best noun model, and **LSTM-both-end** to be the best entity model, indicating that receiving noun and entity information together gives the strongest performance. The full table of their results is not reproduced here, as it is not particularly useful for our discussion; it can be found in (Pichotta and Mooney, 2016b).

Experimental Setup for Mechanical Turk

Citing the inherent difficulty of the narrative cloze task, the authors also evaluated the likelihoods of their event inferences using annotators on Amazon Mechanical

Turk. They presented the annotators with English story snippets and five examples of event inferences, four of which were generated by the model being evaluated, and one of which was always generated by a randomly chosen event from the set of 10,000 most frequent events. The annotators rated the likelihoods of each of the five inferences on a scale from 0 ("Nonsense") to 5 ("Very Likely"). They collected three annotator ratings for each event inference. Inferences were generated for each of four systems: **LSTM-both-noun**, **All-bigram-noun**, **LSTM-both-ent**, and **All-bigram-ent**. They calculated a "filtered" overall score for each model by discarding ratings from annotators whose average score for the randomly chosen event was greater than 1 ("Very Unlikely/Irrelevant").

The authors found that both LSTM models outperformed both bigram models, with filtered average ratings of 3.67 for **LSTM-both-noun** and 3.08 for **LSTM-both-ent**. The average filtered scores for the baseline models were 2.21 for **All-bigram-noun** and 2.87 for **All-bigram-ent**. The filtered score for the random events was 0.87.

Discussion

Pichotta and Mooney made two notable contributions here: (1) they presented a richer model than (Chambers and Jurafsky, 2008) by both extending the event representation and applying script learning techniques suggested by (Jans et al., 2012); and, (2) they demonstrated the effectiveness of LSTMs in learning script sequences. I've represented some examples of learned sequences in Figure 2.7 on the following page.

Though the script sequences in that figure appear to tell stories, it's important to note that important semantic information has been introduced by the authors into the grammar of the interpretations. For example, the event tuple bigram `(capture, squad, villager, ., .) (give, inhabitant, group, ., .)` was interpreted as a continuous sentence, but "give" is a multi-argument verb: an equally valid interpretation of the latter event tuple could be "The inhabitants gave a group", with "group" filling in for the object-given argument rather than the recipient argument. Of course, one could imagine solving such ambiguities using semantic role disambiguation tools, making use of semantic corpora such as FrameNet (Section **??** on page ??), but the fundamental issue of the restrictive event representation would remain.

However, despite the limited event representation, the authors have shown that LSTMs can learn interesting event structures from large text corpora, and one can imagine several uses and extensions. Perhaps using a conceptual hierarchy to generalize entities at the word-level could help create more general scripts from lone story examples? Or maybe the generated event sequences could be used as suggestions for

Generated event tuples	Authors' interpretations
(pass, route, creek, north, in)	The route passes the creek in the North
(traverse, it, river, south, to)	It traverses the river to the South
(issue, ., recommendation, government, from)	A recommendation was issued from the government
(guarantee, ., regulation, ., .)	Regulations were guaranteed
(administer, agency, program, ., .)	The Agency administered the program
(post, ., correction, website, through)	A correction was posted through a website
(ensure, standard, ., ., .)	Standards were ensured
(assess, ., transparency, ., .)	Transparency was assessed
((establish, ., ., citizen, by)	Established by citizens, ...
(end, ., liberation, ., .)	... the liberation was ended
(kill, ., man, ., .)	A man was killed
(rebuild, ., camp, initiative, on)	The camp was rebuilt on an initiative
(capture, squad, villager, ., .)	A squad captured a villager ...
(give, inhabitant, group, ., .)	... [which] the inhabitants had given the group

Figure 2.7: Examples of probabilistically generated scripts from the LSTM model given by (Pichotta and Mooney, 2016b), also showing those authors' English interpretations of the event tuples.

an explanation-based learning system with a richer event representation to elaborate on. This system invites many interesting extensions.

Chapter 3

Our Schema System

In this chapter, we describe the schema representation we've developed, the idea of the "protoschema approach" to schema learning, and the system we've implemented to perform story parsing and schema learning.

3.1 Schema Representation

A knowledge base used for story understanding should be computationally manipulable and facilitate powerful inference procedures. Past approaches, like those described in Chapter 2, have made use of either simplified tuple event representations or FOL-based logical representations with poor expressiveness. Our EL-based schema system maintains a form similar in expressive capacity to natural language, while still offering representational features not found in existing schema learning systems, such as inter-related entity slots, complex temporal relationships between events, schemas embedded as steps within other schemas, and probabilistic certainty scores for constraints.

In this section, we will describe our schema model, examples of which can be seen in Figures 3.5, 3.6, and 3.7. Our schemas are formulated in terms of Episodic Logic (EL) formulas, organized into sections. We describe the sections below.

3.1.1 Overall Structure

A schema is represented by its *header*, seen in line 1 of Figure 3.7. A schema's header is an EL proposition and an episode characterized by the proposition, here `?E`. The header episode summarizes the entire schema. This can be used to "embed" a schema as a step in another schema—in fact, each step in Figure 3.7 is a header

for an embedded schema—or it can be matched directly to a proposition in a story to automatically invoke the schema.

The rest of the schema is laid out in two types of sections: *fluent* and *nonfluent* sections. Nonfluent sections such as `Roles` and `Episode-relations`, whose formula IDs all begin with exclamation marks, contain formulas that hold true regardless of time. Fluent sections, such as `Steps` and `Preconds`, contain formulas identified by variables starting with question marks; these formulas are susceptible to change over time. We will now examine these sections, and what they're used for, in more detail.

3.1.2 Roles

The **Roles** section of a schema is a *nonfluent* section meant for putting "eternal" type constraints on the participating entities in the schema. All of the schema's variables, starting with question marks, can be viewed as Skolem functions of the schema's header episode, and they can be constrained in this section. In addition to type constraints, e.g. `(?X DOG.N)`, relational constraints between entities can also be specified in this section, e.g. `(?X PERTAINS_TO.N ?Y)`.

In Figure 3.5, the protoschema for movement from location to location, there are three main participating entities: `?X`, the agent doing the movement, and `?L1` and `?L2`, the source and destination locations. Those "types" are EL predicates, and are enforced by the predications in the protoschemas **Roles** section.

When formulas are unified with a schema, these formulas are used as constraints for evaluating whether the individuals bound to the variables satisfy the schema's semantics. They are evaluated in a knowledge base populated by facts from the story, and potentially by formulas predicted from other confirmed schema matches. Partial matches, or matches with partial constraint satisfaction, can still be valuable: "Breaking the mold" of known schemas is part of the learning process. More detail on constraint evaluation and scoring will be given in Section 3.2.3.

3.1.3 Preconditions, Postconditions, and Goals

Actions undertaken in the real world are frequently goal-driven, and a representation of how actions change the world and why characters might perform them is necessary to understand the story. The episodes characterized by the formulas in the precondition section are tacitly assumed to start before the schema's header episode (adjoining or overlapping it), and those characterized in the postcondition section extend beyond the header episode (post-adjoining or overlapping it).

Schema matches can be "chained together" into composite, multi-step schemas by unifying their pre- and post-conditions, or their goals and post-conditions. An

example of a learned "chained" schema is given in Figure 3.7.

3.1.4 Temporal Relations

Schemas characterize EL episodes that encompass their temporal duration. They also contain many other episodes, such as steps and preconditions, with their own temporal bounds. These episodes can all be complexly inter-related using constraints from the Allen Interval Algebra (Allen, 1983). Pre- and post-conditions are implicitly constrained to be true at the start and end of the schema's header episode, respectively, and steps, by default, are ordered sequentially as listed in the schema, but additional constraints can be specified in the **Episode-relations** section of each schema. To evaluate these interval constraint propositions, we implemented a time graph specialist module (Gerevini and Schubert, 1993).

3.2 Schema Learning

In this section, we describe how our system learns new schemas from natural language stories. We describe our parsing process for natural language stories, the process of matching parsed stories to schemas, how schema matches can be generalized to create new schemas, and how partial schema matches can be used to predict events in similar stories with missing details.

3.2.1 The Protoschema Approach

As noted, generate new schemas from stories by starting with an initial set of *protoschemas* that we would expect a 1- or 2-year-old child to have; these encode very general knowledge about physical and communicative actions, with their preconditions and effects. Examples of protoschemas we've already written include movement of an agent from one location to another (Figure 3.5), consumption of food, and possession and transfer of possession. These protoschemas are then invoked by actions in stories—for example, the "travel" protoschema in Figure 3.5 matched a "climb" action in the story in Figure 3.3 to yield a "monkey climbs a tree" schema, shown in Figure 3.6. [1]

3.2.2 Story Parsing

Formulas in schemas are represented in Episodic Logic, and can only be matched to other EL formulas. We have developed a stage-based parser to convert stories into

[1] "travel" was invoked by "climb" by way of the WordNet hypernym hierarchy.

EL formulas ready to be matched with schemas. In the first stage, the raw story text is run through the AllenNLP co-reference analyzer (Gardner et al., 2017), and co-referring word spans are tagged with entity numbers. The text is then parsed by the BLLIP parser (Charniak, 2000), the output of which is transduced into *Underspecified Logical Form* (ULF) (Kim and Schubert, 2019), an underspecified variant of EL without scoped quantifiers or temporally de-indexed tenses. A series of transduction rules then converts the ULF to full EL, splitting complex formulas into conjunctions of simpler ones to facilitate "piecewise" matching with the short formulas in schemas.

3.2.3 Matching

Matching formulas in semantically parsed stories to formulas in schemas underlies both learning and prediction. The formulas comprising a schema are intended to be relatively simple—with complex conjunctions split into separate formulas—and unifiable with formulas parsed from real stories. Unification of a story formula with a schema formula binds individual constants from the former to variables in the latter. These bindings are then substituted in the rest of the schema instance, thereby "filling in" some of the missing information. This information is likely to be correct if one or more story formulas matched to the same schema are apt to indicate the occurrence of the kind of pattern of events the schema captures. We refer to any schema instance with one or more bound variables as a *match*.

Using EL formula unification as a primitive, we implement schema matching roughly according to Algorithm 1. In its simplest form, the algorithm iterates through the formulas in an EL parse of a story, matching each formula to any schema formula it can and applying the bindings to the schema. When the story has been fully examined, the match is complete.

However, the order of matched formulas matters: for example, if there are two agent slots in an "X feeds Y" schema, we may match the first suitable agent we see in the story to X, and that binding would persist for the remainder of the matching process, even if Y may have been a better option. To address this, we "shuffle" each story's EL formulas some number of times, and repeat the matching algorithm, stochastically pursuing a better match. With efficient caching of lower level unification operations, this turns out to be a fairly efficient way to explore the match space of a story and a schema.

Partial Matches and Scoring

When a schema is matched to a story, some constraints may be broken; this is a natural part of the learning process. A schema for a cow eating grass matching to a story about a dog eating grass violates the cow constraint on a participating entity,

but is a valuable source of knowledge if properly generalized. On the other hand, too many broken constraints are indicative of a poor match between a schema candidate and a story.

Schema matches are essentially scored by counting satisfied constraints, with some added complexity. For example, nonfluent Role constraints are worth half as many points as fluent events in the Steps section that have been matched to the story—intuitively, a full event is a stronger signal than a property of an entity. Confirming the schema's header formula is worth twice the points of any other event.

For inexact matches—e.g. `(?X COW.N)` and `(ROVER.NAME DOG.N)`—the score of the binding is further weighted by the approximate semantic similarity of the two words. If one subsumes the other on a hypernym hierarchy, the strength is scaled by the distance of the two in that hierarchy. If neither subsumes the other, but they share a common ancestor hypernym, the strength is half their average distance to that ancestor.

The hypernym score accounts for half of the overall weight of an inexact match; the other half is provided by their semantic similarity according to a pre-trained word embedding model. [2]

3.2.4 Generalizing Matches

To generalize a match into a new, "learned" schema, we need to incorporate incidental information about the matched value. For example, the variables of the `travel.v` protoschema in Figure 3.5 can be bound by the constants in the formula `((MONKEY27.SK (CLIMB.V TREE28.SK)) ** E34.SK)` in the story in Figure 3.4, but re-generalizing the constants `MONKEY27.SK` and `TREE28.SK` into variables would remove all the information we learned. However, if we incorporate formulas about the types of those objects into our new schema—like the formulas `(MONKEY27.SK MONKEY.N)` and `(TREE28.SK TREE.N)`—we can then generalize the constants but maintain knowledge of their types.

Re-Matching Learned Schemas

Once a protoschema has been matched to a story and generalized into a learned schema, it may contain extraneous details or overly specific constraints. However, the space of possible generalizations is large: if there are N constraints with even only one possible generalization each, like `DOG.N` and `ANIMAL.N`, the possible generalizations of the entire schema number 2^N.

[2] *GoogleNews-vectors-negative300.bin*, Mikolov et al. (2013b)

To learn reasonable generalizations of learned schemas, we can simply continue our matching process by adding them to the set of initially known schemas, alongside the protoschemas. If they are retrieved as candidates and receive high match scores for another story, shared details of each match can more certainly be incorporated into a general schema.

For example, Figure 3.1 is a "first-order" schema derived from a match of an eating protoschema with the sentence "the cow ate the grass". When matched to the similar sentence "the dog ate the grass", we derive the "second-order" schema in Figure 3.2. Note that `DOG.N` and `COW.N` have been automatically generalized, via the WordNet hypernym hierarchy, to `PLACENTAL.N`. The former two constraints remain in the schema, albeit at a lower *certainty* score of 1/2 each. The latter general constraint has a higher certainty of 2/2. These correspond to the frequency of observance in all stories that have contributed to the learned schema.

3.2.5 Learning Composite Schemas

If schemas could only be learned by progressively refining existing protoschemas, schemas would mostly be single-step actions. Some of those, like "monkey eats banana", would still be valuable world knowledge, but many situations would not be learnable. Our schemas can have multiple steps, however, and even nest other schemas within them as steps; we would like to be able to learn schemas that make use of that complexity.

"First-order" schemas—that is, refinements of our starting protoschemas—can be strung together into composite schemas in multiple ways. If a schema B has a pre-condition that unifies with one of schema A's post-conditions, and schema A's episode is temporally before schema B's, the two can be "chained" together; these chains can be further extended by other simple schemas. This chaining process was how the system learned the schema in Figure 3.7. Goals may also be unified with post-conditions to establish chains.

3.2.6 Prediction

Prediction is relatively straightforward: Given a story, such as the one in Figure 3.8, we try to identify a similar schema, such as the learned schema in Figure 3.7, and match as many formulas as we can to it. After we've substituted story entities for variables, we may fill in other formulas in the schema. Schema formulas whose variables have all been filled in, but are not present in the story, are predictions: we guess that the schema underlies the observed events, and infer the rest of the situation from what we know.

```
(EPI-SCHEMA ((?X_D EAT.379.V ?X_C)
                 ** ?X_E)
  :ROLES
    !R1 (?X_D AGENT.N)
    !R2 (?X_C FOOD.N)
    !R3 (?X_C GRASS.N)
    !R4 (?X_D COW.N)
  :GOALS
    ?G1 (?X_D (WANT.V (THAT (NOT
        (?X_D HUNGRY.A)))))
  :PRECONDS
    ?I1 (?X_D HAVE.V ?X_C)
    ?I2 (?X_D HUNGRY.A)
  :POSTCONDS
    ?P1 (NOT (?X_D (HAVE.V ?X_C)))
    ?P2 (NOT (?X_D HUNGRY.A))
  :EPISODE-RELATIONS
    !W1 (?P1 AFTER ?X_E)
    !W2 (?I1 BEFORE ?X_E)
  :NECESSITIES
    !N1 (!R1 NECESSARY-TO-DEGREE 1.0)
)
)
```

Figure 3.1: A schema learned by applying the eating protoschema to the sentence "the cow ate the grass".

Algorithm 1 Basic algorithm for matching a story to a schema

INPUT: set of story EL formulas $STORY$, a candidate schema SCH, number of shuffles $SHUF$
OUTPUT: best schema *match*
$match \leftarrow null$
for i from 0 to $SHUF$ **do**
  $STORY \leftarrow shuffle(STORY)$
  for ϕ in $STORY$ **do**
    for ψ in SCH **do**
      if ϕ and ψ unify with variable bindings B **then**
        $SCH \leftarrow SCH$ with all bindings in B applied
  if $score(SCH) > score(match)$ **then**
    $match \leftarrow SCH$

```
(EPI-SCHEMA ((?X_D EAT.7.V ?X_C)**?X_E)
  :ROLES
    !R1 (?X_D AGENT.N)
    !R2 (?X_C FOOD.N)
    !R3 (?X_D COW.N)
    !R6 (?X_C GRASS.N)
    !R7 (?X_D DOG.N)
    !R8 (?X_D PLACENTAL.N)
  :GOALS
    ?G1 (?X_D (WANT.V (THAT (NOT
             (?X_D HUNGRY.A)))))
  :PRECONDS
    ?I1 (?X_D HAVE.V ?X_C)
    ?I2 (?X_D HUNGRY.A)
  :POSTCONDS
    ?P1 (NOT (?X_D (HAVE.V ?X_C)))
    ?P2 (NOT (?X_D HUNGRY.A))
  :EPISODE-RELATIONS
    !W1 (?P1 AFTER ?X_E)
    !W2 (?I1 BEFORE ?X_E)
  :CERTAINTIES
    !C1 (!R8 CERTAIN-TO-DEGREE (/ 2 2))
    !C2 (!R3 CERTAIN-TO-DEGREE (/ 1 2))
    !C3 (!R7 CERTAIN-TO-DEGREE (/ 1 2))
  :NECESSITIES
    !N1 (!R1 NECESSARY-TO-DEGREE 1.0)
)
)
```

Figure 3.2: A more general schema learned by applying the learned schema in Figure 3.1 to the sentence "the dog ate the grass".

```
The monkey can climb a tree.
He climbs the tree and gets a cocoanut.
He drops the cocoanut to the ground.
He comes down and eats it.
```

Figure 3.3: An example of a children's story used as input to our system for schema learning.

```
(TREE28.SK TREE.N)
(MONKEY27.SK MONKEY.N)
((MONKEY27.SK((CAN.MD CLIMB.V)
    TREE28.SK))**E26.SK)
((MONKEY27.SK (CLIMB.V TREE28.SK))
    ** E34.SK)
((MONKEY27.SK (GET.V COCOANUT32.SK))
    ** E33.SK)
(COCOANUT32.SK COCOANUT.N)
(TREE28.SK TREE.N)
(GROUND39.SK GROUND.N)
((MONKEY27.SK (DROP.V COCOANUT32.SK))
    ** E40.SK)
(COCOANUT32.SK COCOANUT.N)
((MONKEY27.SK (EAT.V COCOANUT32.SK))
    ** E44.SK)
```

Figure 3.4: The story in Figure 3.3 parsed into Episodic Logic

```
(EPI-SCHEMA((?X TRAVEL.V
    (FROM.P-ARG ?L1) ?L2)**?E)
  :ROLES
    !R1 (?X AGENT.N)
    !R2 (?L1 LOCATION.N)
    !R3 (?L2 LOCATION.N)
  :GOALS
    ?G1 (?X (WANT.V
                (KA ((ADV-A (AT.P ?L2))
                 BE.V))))
  :PRECONDS
    ?I1 (?X (AT.P ?L1))
    ?I2 (NOT (?X (AT.P ?L2)))
  :POSTCONDS
    ?P1 (NOT (?X (AT.P ?L1)))
    ?P2 (?X (AT.P ?L2))
  :NECESSITIES
    !N1 (!R1 NECESSARY-TO-DEGREE 1.0)
)
```

Figure 3.5: An example of a *protoschema*: a general schema used to "bootstrap" the system and generate more specific schemas.

```
(EPI-SCHEMA ((?X_B CLIMB.481.V
                 (FROM.P-ARG ?L1) ?X_C)
                     ** ?X_E)
  :ROLES
    !R1 (?X_B AGENT.N)
    !R2 (?L1 LOCATION.N)
    !R3 (?X_C LOCATION.N)
    !R4 (?X_C TREE.N)
    !R5 (?X_B MONKEY.N)
  :GOALS
    ?G1 (?X_B (WANT.V
                 (KA ((ADV-A
                 (AT.P ?X_C)) BE.V))))
  :PRECONDS
    ?I1 (?X_B (AT.P ?L1))
    ?I2 (NOT (?X_B (AT.P ?X_C)))
  :POSTCONDS
    ?P1 (NOT (?X_B (AT.P ?L1)))
    ?P2 (?X_B (AT.P ?X_C))
  :NECESSITIES
    !N1 (!R1 NECESSARY-TO-DEGREE 1.0)
)
```

Figure 3.6: A schema match of the story in Figure 3.3 to the protoschema in Figure 3.5

```
(EPI-SCHEMA ((?X_B CLIMB_GET_EAT.PR
                ?X_A ?X_C) ** ?E)
  :ROLES
    !R1 (?X_A TREE.N)
    !R2 (?X_C INANIMATE_OBJECT.N)
    !R3 (?X_B MONKEY.N)
    !R4 (?X_C FOOD.N)
    !R5 (?X_C COCOANUT.N)
  :STEPS
    ?E1 (?X_B CLIMB.481.V
            (FROM.P-ARG ?L1) ?X_A)
    ?E2 (?X_B GET.511.V ?X_C
            (AT.P-ARG ?X_A))
    ?E3 (?X_B EAT.541.V ?X_C)
  :EPISODE-RELATIONS
    !W1 (?E1 BEFORE ?E2)
    !W2 (?E2 BEFORE ?E3)
    !W3 (?E1 DURING ?E)
    !W4 (?E2 DURING ?E)
    !W5 (?E3 DURING ?E)
)
```

Figure 3.7: An example of a multi-step schema learned by our system from protoschema matches to the story in Figure 3.3

```
The cocoanut tree is tall.
It is very pretty.
Many cocoanuts grow on the tree.
Simeon can climb the tree.
prediction:
   (SIMEON.NAME MONKEY.N)
prediction:
   (TREE1604.SK ((NN COCOANUT.N) TREE.N))
prediction:
    (SIMEON.NAME CLIMB.481.V TREE1604.SK)
He gets the cocoanuts for his mother.
prediction:
    (SIMEON.NAME EAT.541.V COCOANUT1626.SK)
His mother likes cocoanuts.
She likes to play with Simeon.
```

Figure 3.8: Another children's story, interspersed with predictions made by matching its parse to the learned schema in Figure 3.7

Chapter 4

Initial Results

This chapter describes an experiment we ran with our current implementation of the schema learning system and highlights its results.

4.1 Experimental Setup

Using the parser and matching algorithms described in Chapter 3, we set up an experiment with 561 simple stories taken from a children's first reader McGuffey, 1901 and the ROCstories corpus Mostafazadeh et al., 2016. The stories were parsed into EL and run through the schema matcher. The matcher had a "head start" of the following 13 protoschemas:

1. `avoid_action_to_avoid_displeasure.v`
2. `give.v`
3. `receiving_verb.?` [1]
4. `travel.v`
5. `eat.v`
6. `feed.v`
7. `play.v`

[1] The `?` predicate suffix denotes this as a "metapredicate": it is a predicate over predicates, allowing matching predicates to unify with it during the matching stage. In the case of `receiving_verb.?`, verbs like `take.v` and `get.v` were allowed to match the metapredicate.

8. `make.v`
9. `sit.v`
10. `search.v`
11. `find.v`
12. `drink.v`
13. `put_in_container.v`

By attempting to match these 13 protoschemas to formulas from each story, we obtained 296 specific schema matches. Each match was scored according to the metric described in Section 3.2.3, with a mean score of 1.5, a median score of 2.4, a minimum score of 1.405, and a maximum score of 4.5.

We then ran a second stage of matching, where the initial set of 13 protoschemas was augmented with the 296 learned schemas, all of which could be matched to the same set of stories. We modified the matcher so that no story could be matched with a schema learned from itself. If a story individual was mapped to a schema slot with a type that had a common ancestor, the slot type was generalized to that common ancestor with a high certainty, although two other slot constraints with the more specific types were retained at a lower certainty.

4.2 Results

The second round of matching yielded 665 schemas, with a mean score of -0.899, a median score of 0.292, a minimum score of -19.304, and a maximum score of 4.5. The full set of single-story protoschema matches can be found here. The full set of two-story matches can be found here.

4.2.1 Examples of Positive-Scoring Schemas

We consider the top-scoring 50 percent of schema matches generalized from at least two distinct stories matched to one underlying protoschema. The lower 50 percent had a large number of nonsensical, unsupported constraints—these were the result of "smashing" two stories together into one schema without many shared, or mutually generalizable, constraints. As such, we did not consider them for presentation in this section.

Figure 4.1 is a "sitting in a chair" schema learned from two different stories. The sentences that contributed to the `sit.v` protoschema were "Ethan sat in a chair"

and “Ada is sitting in May’s chair”. Role constraint `!R5`—that the chair belongs to someone else—was learned from only the latter story example, and thus was assigned a certainty of 0.5 in the learned schema.

Figure 4.2 is a “going somewhere to get something” schema. The sentences from two stories that contributed to it were “So he went and got one” and “She will get the ball”. This is also an example of a “chain match”, where schema matches for going somewhere and getting something were chained together into two-step schemas by their pre- and post-conditions. Note that the role conditions that the agent is female and the object being gotten is a ball are each rated at one-half certainty.

Figure 4.3 is a “monkey eats a cocoanut” schema learned from a story where a monkey climbs a tree, gets a cocoanut, and eats it, and another story where a monkey simply eats a cocoanut. The respective sentences are “He comes down and eats it” and “He will eat it”, but the sentences “The monkey can climb a tree”, “He climbs the tree and gets a cocoanut”, and “The monkey has a cocoanut” all provide enough information for the AllenNLP co-reference resolver to automatically resolve the pronouns.

4.2.2 A Three-Step Schema and Prediction

The story in Figure 4.4 matched to protoschemas `travel.v`, `find.v`, and `eat.v`, and, via pre- and post-condition chaining, the system automatically generated the three-step schema shown in Figure 4.6.

During the secondary matching phase, the story in Figure 4.5 strongly invoked that learned schema, due to the presence of words like “field”, “cows”, and “grass” in both stories. The schema’s second step—“cows find a spot of grass”—matched to the sentence “They find sweet nectar in the clover flowers”. The field individual from the sentence “It grows in the fields with red clover and yellow buttercups” matched to the field slot in the schema, and the individual representing the bees’ home matched to the initial location of the “go” step in the schema. These caused the match to be rated as strong even though two of the three main steps were unobserved.

The two unobserved steps, with their slots filled in with values bound from the other matches, state that the bees first went from their home to the field, and that, after they found the nectar in the fields, they ate something (what they ate is unknown, and unmatched to anything in the story). This demonstrates, like the predictions in Figure 3.8, that the schemas this system learns can be used to predict likely, but unobserved, events and relationships in stories.

4.3 Discussion

The schemas our system has learned so far are very promising, and have led to several unexpected predictions of events in other, similar stories. The generality and composability of the protoschemas is promising, and with more protoschemas, a more reliable parser, and more ways of combining multiple schemas into more complex ones, it seems very likely that the protoschema approach will continue to generate general, useful schemas—like the "animals go to a field to find food" example in Figure 4.7—that can be used for story understanding.

```
(EPI-SCHEMA ((?X_B ((ADV-A (ON.P ?X_C)) ((ADV-A (IN.P ?X_C))
        SIT.36.V))) ** ?X_D)
  (:ROLES
    (!R1 (?X_B AGENT.N))
    (!R2 (?X_C INANIMATE_OBJECT.N))
    (!R3 (?X_C FURNITURE.N))
    (!R4 (?L LOCATION.N))
    (!R5 (?X_C (PERTAIN-TO ?X_A)))
    (!R6 (?X_C CHAIR.N))
  )
  (:GOALS
    (?G1 (?X_B (WANT.V (KA REST.V))))
  )
  (:PRECONDS
    (?I1 (?X_B (AT.P ?L)))
    (?I2 (?X_C (AT.P ?L)))
  )
  (:STEPS
  )
  (:EPISODE-RELATIONS
    (!W1 (?X_D (AT-ABOUT ?X_E)))
    (!W2 (?X_D (BEFORE ?X_F)))
  )
  (:NECESSITIES
    (!N1 (!R1 NECESSARY-TO-DEGREE 1.0))
    (!N2 (!R2 NECESSARY-TO-DEGREE 1.0))
  )
  (:CERTAINTIES
    (!C1 (!R5 CERTAIN-TO-DEGREE (/ 1 2)))
  )
)
```

Figure 4.1: A "person sits in chair" schema learned from two different stories.

```
(EPI-SCHEMA ((?X_A GO_GET.3268.PR ?L ?X_B) ** ?E)
  (:ROLES
    (!R1 (?X_B INANIMATE_OBJECT.N))
    (!R2 (?X_B PRED?.N))
    (!R3 (NOT (?X_A = ?X_B)))
    (!R4 (?X_A FEMALE.A))
    (!R5 (?X_A AGENT.N))
    (!R6 (?X_B BALL.N))
  )
  (:PRECONDS
    (?I1 (?X_A (AT.P ?L1_2)))
    (?I2 (NOT (?X_A (AT.P ?L))))
  )
  (:STEPS
    (?E1 (?X_A ((ADV-A (FROM.P ?L1)) ((ADV-A (TO.P ?L)) GO.673.V)) ?L))
    (?E2 (?X_A GET.672.V ?X_B (AT.P-ARG ?L2)))
  )
  (:POSTCONDS
    (?P1 (?X_A HAVE.V ?X_B))
  )
  (:EPISODE-RELATIONS
    (!W1 (?E1 BEFORE ?E2))
    (!W2 (?E2 (AFTER ?X_D)))
    (!W3 (?X_D (AT-ABOUT ?X_E)))
  )
  (:CERTAINTIES
    (!C1 (!R2 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C2 (!R4 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C3 (!R5 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C1 (!R6 CERTAIN-TO-DEGREE (/ 1 2)))
  )
)
```

Figure 4.2: A "person goes to get something" schema learned from two different stories.

```
(EPI-SCHEMA ((?X_D EAT.461.V ?X_C) ** ?X_I)
  (:ROLES
    (!R1 (?X_C FOOD.N))
    (!R2 (?X_D MONKEY.N))
    (!R3 (?X_C COCOANUT.N))
    (!R4 (?X_A GROUND.N))
    (!R5 (?X_C (TO.P ?X_A)))
    (!R6 (?X_D MALE.A))
    (!R7 (?X_D AGENT.N))
    (!R8 (?X_B (PERTAIN-TO ?X_D)))
    (!R9 (?X_B (PLUR EYE.N)))
  )
  (:GOALS
    (?G1 (?X_D (WANT.V (THAT (NOT (?X_D HUNGRY.A))))))
  )
  (:PRECONDS
    (?I1 (?X_D HAVE.V ?X_C))
    (?I2 (?X_D HUNGRY.A))
  )
  (:STEPS
  )
  (:POSTCONDS
    (?P1 (NOT (?X_D (HAVE.V ?X_C))))
    (?P2 (NOT (?X_D HUNGRY.A)))
  )
  (:EPISODE-RELATIONS
    (!W1 (?P1 AFTER ?X_I))
    (!W2 (?I1 BEFORE ?X_I))
    (!W3 (?X_I CAUSE-OF ?P1))
    (!W4 (?X_E (CONSEC ?X_I)))
    (!W5 (?X_I (DURING ?X_G)))
    (!W6 (?X_E (DURING ?X_G)))
    (!W7 (?X_F (AT-ABOUT ?X_H)))
    (!W8 (?X_G (AT-ABOUT ?X_H)))
    (!W9 (?X_I (AFTER ?X_J)))
    (!W10 (?X_J (AT-ABOUT ?X_K)))
  )
  (:NECESSITIES
    (!N1 (!R1 NECESSARY-TO-DEGREE 1.0))
  )
  (:CERTAINTIES
    (!C1 (!R4 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C2 (!R5 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C3 (!R6 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C4 (!R8 CERTAIN-TO-DEGREE (/ 1 2)))
    (!C5 (!R9 CERTAIN-TO-DEGREE (/ 1 2)))
  )
)
```

Figure 4.3: A "monkey eats a cocoanut" schema learned from two different stories.

```
; "The cow left the barn."
; "It went out to the field."
; "The other cows were out in the field."
; "The cow found a spot of grass."
; "The cow ate the grass."
```

Figure 4.4: A story about cows.

```
; "This is red clover."
; "The bees like it."
; "They find sweet nectar in the clover flowers."
; "They take the nectar home to make honey."
; "Here is white clover."
; "It is sweet."
; "It has nectar, and bees like it, too."
; "It grows in the fields with red clover and yellow buttercups."
; "Horses and cows eat clover."
```

Figure 4.5: A story about bees.

```
(EPI-SCHEMA ((?X_C GO_FIND_EAT.566.PR ?X_A ?X_B ?X_D) ** ?E)
    (:ROLES
        (!R1 (?X_A FIELD.N))
        (!R2 (?X_B (OF.P (K GRASS.N))))
        (!R3 (?X_B SPOT.N))
        (!R4 (?X_C COW.N))
        (!R5 (?X_D FOOD.N))
        (!R6 (?X_D GRASS.N))
    )
    (:PRECONDS
        (?I1 (?X_C (AT.P ?L1_2)))
        (?I2 (NOT (?X_C (AT.P ?X_A))))
    )
    (:STEPS
        (?E1 (?X_C (OUT.ADV ((ADV-A (TO.P ?X_A)) ((ADV-A (FROM.P ?L1)) GO.563.V))) ?X
        (?E2 (?X_C FIND.562.V ?X_B))
        (?E3 (?X_C EAT.564.V ?X_D))
    )
    (:POSTCONDS
        (?P1 (NOT (?X_C (HAVE.V ?X_D))))
        (?P2 (NOT (?X_C HUNGRY.A)))
    )
    (:EPISODE-RELATIONS
        (!W1 (?E1 BEFORE ?E2))
        (!W2 (?E2 BEFORE ?E3))
    )
)
```

Figure 4.6: A three-step chained schema learned from the story in Figure 4.4

```
(EPI-SCHEMA ((BEES715.SK GO_FIND_EAT.861.PR FIELD738.SK
              NECTAR1.SK
              ?X_D)
             ** ?E)
    (:ROLES
        (!R1 (FIELD738.SK FIELD.N))
    (!R2_1 (NECTAR1.SK NECTAR.N))
    (!R2_2 (NECTAR1.SK SWEET.A))
    (!R2_3 (NECTAR1.SK (OF.P (K GRASS.N))))
    (!R2_4 (NECTAR1.SK SPOT.N))
        (!R4 (BEES715.SK COW.N))
        (!R5 (?X_D FOOD.N))
        (!R6 (?X_D GRASS.N))
        (!R7 (BEES715.SK (PLUR BEE.N)))
        (!R8 (FLOWERS721.SK ((NN CLOVER.N) (PLUR FLOWER.N))))
        (!R9 (FIELD738.SK (PLUR FIELD.N)))
    )
    (:PRECONDS
        (?I1 (BEES715.SK (AT.P ?L1_2)))
        (?I2 (NOT (BEES715.SK (AT.P FIELD738.SK))))
    )
    (:STEPS
        (?E1
   (BEES715.SK
    (OUT.ADV
     ((ADV-A (TO.P FIELD738.SK)) ((ADV-A (FROM.P HOME724.SK)) GO.860.V)))
    FIELD738.SK))
        (E722.SK
   (BEES715.SK ((ADV-A (IN.P FLOWERS721.SK)) FIND.562.V) ?X_N))
        (?E3 (BEES715.SK EAT.564.V ?X_D))
    )
    (:POSTCONDS
        (?P1 (NOT (BEES715.SK (HAVE.V ?X_D))))
        (?P2 (NOT (BEES715.SK HUNGRY.A)))
    )
    (:EPISODE-RELATIONS
        (!W1 (?E1 BEFORE E722.SK))
        (!W2 (E722.SK BEFORE ?E3))
        (!W3 (E722.SK (AT-ABOUT NOW178)))
    )
    (:SUBORDINATE-CONSTRAINTS
        (!S1 ((?X_A<- ?E1) = HOME724.SK))
        (!S2 ((?X_B<- ?E1) = BEES715.SK))
        (!S3 ((?X_C<- ?E1) = FIELD738.SK))
        (!S4 ((?E1<- ?E1) = ?E1))
    )
)
```

Figure 4.7: The schema in Figure 4.6 applied to the story in Figure 4.5, showing the predictive ability of inexact schema matches

Chapter 5

Research Plan

This chapter outlines our central research questions, and sketches a plan for the remainder of the work on resolving them, including how we will evaluate our results.

5.1 Central Research Questions

5.1.1 Q0. Does the protoschema approach work?

This thesis project is meant to investigate whether the protoschema approach to learning can generate high-quality schemas from few story examples. Schema accumulation, however, is not our *ultimate* goal; as Van Dijk and Kintsch (1983) and Schank and Abelson (1977) have shown, schema-like reasoning, regardless of the representation, is necessary for genuine story understanding. As such, we will seek to demonstrate whether the results of our schema learning approach facilitate genuine story understanding. Using the evaluation methods described in Section 5.2, we seek to show that human judges find the schemas learned by our system to be useful world knowledge, and the inferences generated by our schemas to be feasible and novel.

5.1.2 Q1. Can we infer schemas for unknown words?

The initial set of protoschemas will target the knowledge of a very young human child, and so the meaning of many words, like "buy" or "vacation", will not be encoded in any protoschemas. Although basic ontological knowledge can be provided to the system by e.g. WordNet's hypernym hierarchy (Miller, 1995) or the TRIPS ontology (Ferguson, Allen, et al., 1998), knowledge of their inherent motivations, preconditions, and postconditions will be limited. However, if "X goes to the store, buys Y, and takes Y home" is a recurring pattern of events, we can start to form

a partial "schema" for buying: we learn, for instance, that it is often done to an object at a store, and that a frequent postcondition is the possession of that object. If the objects being bought have properties in common across several examples, we can incorporate that information into type constraints.

We believe that good candidates for unknown verbal predicates with inferrable meanings are those that occur temporally between two other schema matches; the schema matches temporally before and after the unknown word provide pre- and post-condition candidates, which are essential to defining the semantics of the unknown action. As we learn more schemas and match them to texts, we will investigate unmatched, unknown actions in the stories, incorporate situational details into candidate schemas for those actions, and evaluate the quality of those schemas to determine how many and what kind of actions can be learned without protoschemas.

5.1.3 Q2. What number and kind of protoschemas are needed?

As we noted in Chapter 4, only 13 initial protoschemas yielded hundreds of learned schemas, all of which were candidates for combination into multi-step schemas. The protoschemas form a strong foundation for an understanding of the most fundamental events in a text, upon which many more complex schemas can be built. The more actions we understand, the more we can compose. The number of composable schemas is likely a superlinear function of the number of initial protoschemas, making them a valuable use of valuable human time.

Our 13 initial protoschemas are a good start, but human children [1] understand many more basic action patterns than that by the age of two. Protoschemas needn't just correspond to individual verbs or actions, either; very young human children understand certain multi-step action patterns like "calling someone for help after failing to do something" and a large number of "object schemas" (see Section 5.1.5) that provide knowledge about common entities in the world, including their physical properties and ways they can be used. We are targeting a few dozen to a few hundred protoschemas for the final system, covering as much of the knowledge of a two-year-old human child as is feasible to encode.

A central research question, during the development of more protoschemas, will be how the number and variety of protoschemas affect the quality of schema matching—we suspect that more protoschemas will improve the variety and accuracy of schemas, up to a point, but may plateau as additional protoschemas necessarily grow more and more specific. We can run the system with various subsets of protoschemas, and evaluate the learned schemas using the methods in Section 5.2, to learn which

[1] Several of my proofreaders have objected to the phrase "human children", but I would like to exclude sentient non-human children, e.g. the system developed by Data and Picard (2366).

protoschemas are most important for good schema matches.

5.1.4 Q3. Can "downstream" schema information help repair semantic parses?

Coreference

Most semantic parsers, including ours, make mistakes. The AllenNLP coreference resolver we use occasionally fails to correlate two spans of text that refer to the same entity, or spuriously correlates two spans that don't. Pichotta (2017) found that event co-occurrence models improved the performance of noun coreference resolvers. We believe that our rich event model will improve the accuracy of coreference resolution.

We believe we can reduce the incidence of these errors by treating coreference in the parsing stage as tentative; the EL form can refer to the entities as different, but maintain uncertain relational propositions in the knowledge base asserting that the *might* be equal. If their equivalence breaks too many logical constraints in the story, and fewer constraints would be broken by resolving the co-reference with another entity, the coreference can be re-resolved accordingly. We will compile the list of all corrections made by the schema system to the parser's coreference clusters and evaluate them to judge the effect schema matching has on coreference performance.

Idioms and Paraphrases

A good semantic parser should ideally abstract multiple ways of phrasing a sentence into a common representation. Synonyms are easily abstracted via existing lexical resources, but more complex phrasing patterns, like "he told her" and "he let her know", or "he realized" and "it dawned upon him", receive extremely different EL representations. As a result, some story phrases are unable to match to protoschemas. Children's stories are often simple enough that many idioms are missing, but a surprising number persist, and so some automated paraphrase resolution would likely save some information from being lost. We plan to explore resources like PPDB (Ganitkevitch et al., 2013) to help resolve this problem at the parser level.

We can also address some degree of confusing paraphrases by allowing learned schemas to predict the meaning when the verbs are unclear. For example, if the schema engine had learned a schema for someone buying a car for some amount of money, the phrase "he scored a new car for a thousand dollars", which does not use the word "buy" or a common synonym, might match "new car" and "thousand dollars" against the known schema, allowing the meaning of the unknown word to be inferred. This is a good opportunity to explore how inferences from partial schema

matches can compensate for an overly literal semantic parse, and even, in the case of a parameter-based parser, allow corrections to the parsing model.

5.1.5 Q4. Can we extract useful object schemas from event schemas?

Once our event schemas have been extracted, we will also attempt to transduce "object schemas" from their entities. These schemas will provide information about certain types of objects, including common properties and what Pustejovsky calls "telic" information about them, i.e., "what they're good for" (Pustejovsky, 1991). If a general event schema is observed many times for a person going to a store to buy something, then an object schema for a store can include the learned information that it is a kind of location, and that people go to them to buy things. If a "person holds a fruit" event schema is learned, then a fruit object schema could include the detail that it is holdable by a person, thus learning important information about size. Object schemas can lead to predictions in a story where no relevant events occur, but a unique object is observed: if someone walks to a plaza and sees a store, the system could predict that they may go in and buy something. And if a story has a box with fruit in it, the fruit's object schema could allow the system to infer that the box is likely also holdable by a person.

Many event schemas can yield object schemas if "looked at from a different angle", and many different event schemas can contribute to the same object schema. We plan to extract these from any learned event schemas and evaluate them alongside the event schemas.

5.2 Evaluation

The goal of schema acquisition is to acquire knowledge that is useful for understanding real stories, and we design our result evaluation metric with this in mind.

Yuan et al. (2018) evaluate their system by asking human experts to manually formulate schemas, and then comparing their automatically learned schemas slot-for-slot with the expert ones. This approach does not scale well: since we are, by definition, aiming to learn more types of schemas than can be feasibly entered manually, we cannot rely on annotators to pre-construct schemas for all of the events the system will learn about; we must judge the results after they are generated.

We plan to evaluate our system in three ways: human schema evaluation, human evaluation of story understanding, and automated narrative cloze evaluation.

5.2.1 Schema Evaluation

To evaluate our schemas, we will use untrained human judges. We will render the schema's EL formulas into English, building on the ULF-to-English work by Kim et al. (2019), and present the readable schema steps, constraints, preconditions, postconditions, and goals to human judges. We will ask the judges to independently rate the schema's accuracy and generality. An *accurate* schema corresponds to events that could actually happen in the world; "a man goes to the store for cabbage" is an accurate schema, whereas "a dog goes outside to eat a refrigerator" is inaccurate. A *general* schema has abstracted entities such that they are not overly specific; "a man in a blue shirt goes to the store for cabbage" is less general than "a person goes to the store for vegetables". Because the generalization space is large, and many possible generalizations might be acceptable, we will ask the judges to rate the generality of individual constraints in the story; "man" instead of "person" might be a more acceptable over-specification than the totally unnecessary detail "the person is wearing a blue shirt".

5.2.2 Understanding Evaluation

We can use the learned schemas to make predictions on held-out stories, like in Figure 3.8, and render those predictions in English as well. Schema predictions can take several forms:

- **Temporal predictions** predict a prior or subsequent event from an observed event, like those in Figure 3.8.
- **Property predictions** predict attributes or details not mentioned in the story. Figure 3.8 also has an example of this: Simeon was predicted to be a monkey, though this detail was not given in the story.
- **Motivational predictions** predict the reason an agent did something. If a story mentioned a cow leaving its barn to go to a field, the learned schema in Figure 4.6 would predict that the cow ultimately did this because it was hungry.

The story formulas inferred by the schemas should ideally constitute the system's "understanding" of the story, and we will test the depth and the quality of that understanding with human judges. After rendering the inferred formulas into English—"X did this because they wanted to..." for goals, "X was a ..." for roles, "X went on to..." for temporal predictions, etc.—the English predictions can be presented to untrained human judges, who will rate their likelihood.

5.2.3 Automatic Narrative Cloze Evaluation

Chambers and Jurafsky (2008) introduced the *Narrative Cloze* metric for schema evaluation, where a corpus of text is parsed into sequences of event representations, some events are "blanked out", and the learned schemas are evaluated on their ability to predict the "blanked out" events in the test sequences. Pichotta and Mooney (2016b) use the Narrative Cloze task to evaluate their schemas, as well. Because the "gold standard" events for the blanks to be filled are generated in the same representation the evaluators use for their learner, this method cannot be used to demonstrate that the representation is rich enough to facilitate human-level understanding of a story, or that humans find the predictions it can generate useful. However, it can demonstrate that the learning method is capable of learning common event sequences within that representation.

We can adapt the Narrative Cloze test to automatically evaluate our own system by parsing stories into EL and selectively withholding not only sentences or temporal events, but also atemporal "type" and relational constraints. These reduced stories can then be matched to learned schemas, and the proportion of withheld formulas inferred by the schema system can be used as a heuristic for the system's learning ability. This could be useful in the development phase, when we are more concerned with iterative improvement of schema formation than with demonstrating the intuitiveness of our schemas to human judges.

5.3 Timeline

In order to address our ultimate research question, Q0, we will need to make significant changes to our parsing pipeline, run our matcher to accumulate a large number of candidate schemas, and then evaluate the schemas.

5.3.1 Fall 2020: Parser Development

In Fall 2021, we'll need to improve our parsing pipeline to output higher quality Episodic Logic formulas. Currently, the pipeline has a number of issues, enumerated below, which we'll need to investigate. Although research question Q3 primarily deals with the parser, that work cannot be done until we have a sufficient schema corpus to begin correcting some parser errors; to accumulate this corpus, we must first improve the "non-integrated" parser as much as possible.

Adding Temporal Information

EL episodes are temporally related. The sentences in stories parsed by the EL parser each have "speech times", arranged one after the other, representing the time at which the sentence was "spoken" by the narrator. The events within the sentence are related to the speech time by grammatical tense, but the relationships between the actual event times aren't inferred by the parser. We'll need to explore methods of annotating parses with temporal information. Chambers and Jurafsky (2008) trained an SVM model on the TimeBank corpus to predict pairwise temporal ordering, but our temporal model includes intervals and nonlinear event ordering. Solutions to the TempEval task (Verhagen et al., 2010), which extracts pairwise event relations and allows for partial interval overlap, may prove more useful. [2] Like with the parser-stage coreference annotations provided by AllenNLP, we can treat these temporal annotations as tentative, and allow the "upstream" schema matcher to override them if too many temporal contradictions are encountered when fitting events to known schemas.

General Mis-Parses

The sentence "the dog and the hen ran all the way home" is parsed into `(((SET-OF DOG2.SK HEN3.SK) (RUN.V HOME6.SK)) ** U1.SK)` and `(HOME6.SK ((NN WAY.N) HOME.N))`. This interprets the home as a composite noun phrase "way home", and discards the "all the". In reality, of course, "all the way" is a modifier. This is the result of either a syntactic parse failure in the BLLIP parser, or a failure of the parse-to-ULF conversion process. The exact point of failure for mis-parses like this one is still under investigation, but as semantic parsing is already a very challenging task, it seems likely that many sentences will be mis-interpreted in this way.

The use of a cache transition parser architecture for direct English-to-ULF conversion is being investigated by Kim (2019), and may improve our parsing accuracy.

Total Parse Failures

Some sentences fail to produce usable EL output when parsed; the initial ULF to EL conversion completes successfully, but outputs invalid, complex, and uninterpretable EL formulas that cannot be broken down or matched to schemas. These sentences must be thrown out, reducing the information content of the story and crippling the learning process. We estimate that between 30 and 50 percent of sentences in our

[2] Of special interest may be the work by UzZaman and Allen (2010) on using the TRIPS system for this task, as we already plan to use the TRIPS ontology for predicate generalization.

story corpus fail to parse entirely, for reasons we don't yet understand. The probability of a failed parse increases with sentence length and the presence of embedded quotations, but the incidence and role of other semantic features in parse failures is still unclear.

We suspect that many of these are caused by an inability of the parse-to-ULF conversion rules to cope with certain kinds complex parse tree; ideally, a direct English-to-ULF parser would bypass this issue. We will need to dedicate time to understanding and repairing the process by which BLLIP parse trees are converted to ULF, though, to make proper use of the stories we've collected.

5.3.2 Spring 2021: Protoschema and Matcher Development

During this stage, we'll investigate research question Q2 by creating a large number of protoschemas to cover as much general knowledge as possible. We'll likely need to train research assistants to create schemas in our representation to create a sufficient number—ideally dozens to hundreds—in this timeframe. We'll partition our stories into development and test sets, and use the development set stories to investigate which kinds of protoschemas are most likely to be useful. As we develop them, we can estimate which protoschemas are most useful by counting the number of formulas they match to in the development story corpus, and manually verifying that those matches are semantically reasonable. During this stage, we will also attempt to build tentative schemas for unknown words to investigate research question Q1.

5.3.3 Fall 2021: Schema Accumulation

By Fall 2021, we hope to have a more robust semantic parser and a more comprehensive set of protoschemas. Then, we can begin matching the protoschemas to stories. We'll begin with a smaller set of stories and manually assess the quality and variety of the schemas that come out. When we are satisfied that the parser is working well and the protoschemas provide suitable conceptual coverage, we'll move on to the evaluation stage. At that time, we can also investigate research question Q3, assessing the ability of the schemas we've learned to help perform coreference resolution and paraphrase interpretation.

5.3.4 Spring 2022 and beyond: Evaluation

Once a large schema corpus has been accumulated, we'll begin work on the EL-to-English code necessary to present EL schemas to untrained human judges. We'll also need to generate predictions to be judged by running some held out stories through

the schema matcher with the full learned schema corpus available, and render those predictions into English as well. Finally, we'll investigate research question Q4 by extracting object schemas for common entity types in the event schema corpus, and converting them into English representations for judgment as well. We'll present the schemas and predictions to a large number of untrained human judges, who will rate them on accuracy and generality, as described in Section 5.2.

5.4 Conclusion

In this proposal, I have motivated schemas as a means for story understanding, and discussed past approaches to schema learning and their limitations. In Chapter 3, I presented a novel model of schemas and the description of an approach to learn them, and in Chapter 4 I demonstrated some early experimental results that suggest that our approach is effective. I described the changes that should be made to the system—especially its parser and multi-step learning methods—to enable a greater breadth and complexity of learned schemas, and outlined an evaluation strategy to demonstrate the usefulness of the final learned schemas.

Bibliography

[1] James F Allen. "Maintaining knowledge about temporal intervals." In: *Communications of the ACM* 26.11 (1983), pp. 832–843.

[2] Timothy A Allen and Norbert J Fortin. "The evolution of episodic memory." In: *Proceedings of the National Academy of Sciences* 110.Supplement 2 (2013), pp. 10379–10386.

[3] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. "The Berkeley FrameNet Project." In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*. ACL '98/COLING '98. Montreal, Quebec, Canada: Association for Computational Linguistics, 1998, pp. 86–90. DOI: 10.3115/980845.980860. URL: https://doi.org/10.3115/980845.980860.

[4] Jason Baldridge. "The opennlp project." In: *URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012)* (2005), p. 1.

[5] Frederic C Bartlett. *Remembering: A study in experimental and social psychology*. Vol. 14. Cambridge University Press, 1932.

[6] Nathanael Chambers and Dan Jurafsky. "Template-Based Information Extraction without the Templates." In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 976–986. URL: http://aclweb.org/anthology/P11-1098.

[7] Nathanael Chambers and Dan Jurafsky. "Unsupervised learning of narrative event chains." In: *Proceedings of ACL-08: HLT* (2008), pp. 789–797.

[8] Eugene Charniak. "A maximum-entropy-inspired parser." In: *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics. 2000, pp. 132–139.

[9] Eugene Charniak et al., eds. *Artificial Intelligence Programming*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1980. ISBN: 0898596092.

[10] Lt. Cmdr. Data and Jean-Luc Picard. “The Offspring.” In: *Star Trek: The Next Generation*. Season 3, Episode 16. U.S.S. Enterprise, 2366.

[11] Douglas Eck and Juergen Schmidhuber. “Finding temporal structure in music: Blues improvisation with LSTM recurrent networks.” In: *Proceedings of the 12th IEEE workshop on neural networks for signal processing*. IEEE. 2002, pp. 747–756.

[12] George Ferguson, James F Allen, et al. “TRIPS: An integrated intelligent problem-solving assistant.” In: *Aaai/Iaai*. 1998, pp. 567–572.

[13] Richard E Fikes, Peter E Hart, and Nils J Nilsson. “Learning and executing generalized robot plans.” In: *Artificial intelligence* 3 (1972), pp. 251–288.

[14] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. “PPDB: The paraphrase database.” In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2013, pp. 758–764.

[15] Matt Gardner et al. “AllenNLP: A Deep Semantic Natural Language Processing Platform.” In: 2017. eprint: arXiv:1803.07640.

[16] Alfonso Gerevini and Lenhart Schubert. “Efficient temporal reasoning through timegraphs.” In: *IJCAI*. 1993, pp. 648–654.

[17] Alex Graves and Jürgen Schmidhuber. “Offline handwriting recognition with multidimensional recurrent neural networks.” In: *Advances in neural information processing systems*. 2009, pp. 545–552.

[18] Sepp Hochreiter and JÃŒrgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[19] Chung Hee Hwang. “A logical approach to narrative understanding.” PhD thesis. University of Alberta, 1992.

[20] Bram Jans et al. “Skip n-grams and ranking functions for predicting script events.” In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2012, pp. 336–344.

[21] Gene Kim and Lenhart Schubert. “A Type-coherent, Expressive Representation as an Initial Step to Language Understanding.” In: *Proceedings of the 13th International Conference on Computational Semantics*. Gothenburg, Sweden: Association for Computational Linguistics, 2019.

[22] Gene Kim et al. "Generating Discourse Inferences from Unscoped Episodic Logical Formulas." In: *Proceedings of the First International Workshop on Designing Meaning Representations*. 2019, pp. 56–65.

[23] Gene Louis Kim. "Towards Parsing Unscoped Episodic Logical Forms with a Cache Transition Parser." In: *the Poster Abstracts of the Proceedings of the 32nd International Conference of the Florida Artificial Intelligence Research Society*. Sarasota, Florida, USA, 2019.

[24] Janet L Kolodner. "Maintaining organization in a dynamic long-term memory." In: *Cognitive science* 7.4 (1983), pp. 243–280.

[25] Michael Lebowitz. "Generalization and Memory in an Integrated Understanding System." AAI8109800. PhD thesis. New Haven, CT, USA, 1980.

[26] John McCarthy and Patrick J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence." In: *Machine Intelligence 4*. Ed. by B. Meltzer and D. Michie. reprinted in McC90. Edinburgh University Press, 1969, pp. 463–502.

[27] William Holmes McGuffey. *The New McGuffey First Reader*. American Book Company, 1901.

[28] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality." In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.

[29] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality." In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[30] George A Miller. "WordNet: a lexical database for English." In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[31] Marvin Minsky. *A Framework for Representing Knowledge*. Tech. rep. Cambridge, MA, USA, 1974.

[32] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. "Explanation-Based Generalization: A Unifying View." In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 47–80. ISSN: 0885-6125. DOI: 10.1023/A:1022691120807. URL: http://dx.doi.org/10.1023/A:1022691120807.

[33] Raymond J. Mooney. "A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding." PhD thesis. Department of Computer Science, University of Illinois at Urbana-Champaign, 1988. URL: http://www.cs.utexas.edu/users/ai-lab/?mooney:phd88.

[34] Raymond J Mooney. *A general explanation-based learning mechanism and its application to narrative understanding*. Morgan Kaufmann, 1990.

[35] Raymond J. Mooney. "Learning Plan Schemata From Observation: Explanation-Based Learning for Plan Recognition." In: *Cognitive Science* 14.4 (1990), pp. 483–509. URL: http://www.cs.utexas.edu/users/ai-lab/?mooney:cogsci90.

[36] Nasrin Mostafazadeh et al. "A corpus and evaluation framework for deeper understanding of commonsense stories." In: *arXiv preprint arXiv:1604.01696* (2016).

[37] Ulric Neisser. "John Dean's memory: A case study." In: *Cognition* 9 (Mar. 1981), pp. 1–22. DOI: 10.1016/0010-0277(81)90011-1.

[38] Martha Palmer, Daniel Gildea, and Paul Kingsbury. "The proposition bank: An annotated corpus of semantic roles." In: *Computational linguistics* 31.1 (2005), pp. 71–106.

[39] Robert Parker et al. "English gigaword fifth edition, june." In: *Linguistic Data Consortium, LDC2011T07* 12 (2011).

[40] Matthew E Peters et al. "Deep contextualized word representations." In: *arXiv preprint arXiv:1802.05365* (2018).

[41] Karl Pichotta. "Advances in Statistical Script Learning." PhD thesis. Department of Computer Science, The University of Texas at Austin, 2017. URL: http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127649.

[42] Karl Pichotta and Raymond Mooney. "Statistical Script Learning with Multi-Argument Events." In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, Apr. 2014, pp. 220–229. DOI: 10.3115/v1/E14-1024. URL: https://www.aclweb.org/anthology/E14-1024.

[43] Karl Pichotta and Raymond Mooney. "Statistical Script Learning with Recurrent Neural Networks." In: *Proceedings of the Workshop on Uphill Battles in Language Processing: Scaling Early Achievements to Robust Methods*. Austin, TX: Association for Computational Linguistics, 2016, pp. 11–16. DOI: 10.18653/v1/W16-6003. URL: http://aclweb.org/anthology/W16-6003.

[44] Karl Pichotta and Raymond J Mooney. "Learning statistical scripts with LSTM recurrent neural networks." In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[45] James Pustejovsky. "The generative lexicon." In: *Computational linguistics* 17.4 (1991), pp. 409–441.

[46] James Pustejovsky et al. "The timebank corpus." In: *Corpus linguistics*. Vol. 2003. Lancaster, UK. 2003, p. 40.

[47] R. Reiter. "Readings in Nonmonotonic Reasoning." In: ed. by Matthew L. Ginsberg. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. Chap. On Closed World Data Bases, pp. 300–310. ISBN: 0-934613-45-1. URL: http://dl.acm.org/citation.cfm?id=42641.42663.

[48] John Alan Robinson et al. "A machine-oriented logic based on the resolution principle." In: *Journal of the ACM* 12.1 (1965), pp. 23–41.

[49] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

[51] R.C. Schank and R.P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry Into Human Knowledge Structures*. The Artificial Intelligence Series. Lawrence Erlbaum Associates, 1977. ISBN: 9780470990339. URL: https://books.google.com/books?id=YZ99AAAAMAAJ.

[52] Roger C Schank. *Dynamic memory: A theory of reminding and learning in computers and people*. Vol. 240. Cambridge University Press Cambridge, 1982.

[53] Roger C Schank. "Language and memory." In: *Cognitive science* 4.3 (1980), pp. 243–284.

[54] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[55] Wilson L Taylor. ""Cloze procedure": A new tool for measuring readability." In: *Journalism Bulletin* 30.4 (1953), pp. 415–433.

[56] Endel Tulving et al. “Episodic and semantic memory.” In: *Organization of memory* 1 (1972), pp. 381–403.

[57] Naushad UzZaman and James F. Allen. “TRIPS and TRIOS System for TempEval-2: Extracting Temporal Information from Text.” In: *Proceedings of the 5th International Workshop on Semantic Evaluation*. SemEval ’10. Los Angeles, California: Association for Computational Linguistics, 2010, pp. 276–283.

[58] Teun Adrianus Van Dijk and Walter Kintsch. *Strategies of discourse comprehension*. Academic press New York, 1983.

[59] Marc Verhagen et al. “SemEval-2010 Task 13: TempEval-2.” In: *Proceedings of the 5th International Workshop on Semantic Evaluation*. Uppsala, Sweden: Association for Computational Linguistics, July 2010, pp. 57–62. URL: https://www.aclweb.org/anthology/S10-1010.

[60] Ichiro Yamada and Timothy Baldwin. “Automatic discovery of telic and agentive roles from corpus data.” In: *Proceedings of the 18th Pacific Asia Conference on Language, Information and Computation*. 2004, pp. 115–126.

[61] Quan Yuan et al. “Open-Schema Event Profiling for Massive News Corpora.” In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM ’18. Torino, Italy: ACM, 2018, pp. 587–596. ISBN: 978-1-4503-6014-2. DOI: 10.1145/3269206.3271674. URL: http://doi.acm.org/10.1145/3269206.3271674.