Towards Natural Language Story Understanding with Rich Logical Schemas

Lane Lawley and Gene Louis Kim and Lenhart Schubert
Department of Computer Science, University of Rochester
{llawley,gkim21,schubert}@cs.rochester.edu

Abstract

Generating "commonsense" knowledge for intelligent understanding and reasoning is a difficult, long-standing problem, whose scale challenges the capacity of any approach driven primarily by human input. Furthermore, approaches based on mining statistically repetitive patterns fail to produce the rich representations humans acquire, and fall far short of human efficiency in inducing knowledge from text. The idea of our approach to this problem is to provide a learning system with a "head start" consisting of a semantic parser, some basic ontological knowledge, and most importantly, a small set of very general schemas about the kinds of patterns of events (often purposive, causal, or socially conventional) that even a one- or two-year-old could reasonably be presumed to possess. We match these initial schemas to simple children's stories, obtaining concrete instances, and combining and abstracting these into new candidate schemas. Both the initial and generated schemas are specified using a rich, expressive logical form. Unlike the slot-and-filler structures often used in knowledge harvesting, this logical form allows us to specify complex relations and constraints over the slots. Though formal, the representations are language-like, and as such readily relatable to NL text. The agents, objects, and other roles in the schemas are represented by typed variables, and the event variables can be related through partial temporal ordering and causal relations. To match natural language stories with existing schemas, we first parse the stories into an underspecified variant of the logical form used by the schemas, which is suitable for most concrete stories. We include a walkthrough of matching a children's story to these schemas and generating inferences from these matches.

1 Introduction

Artificial general intelligence research tends to fall into one of two broad categories: connectionism, which emphasizes the importance of neural architectures in the human brain, and computationalism, which models human intelligence at a more abstract level, making use of knowledge representations and reasoning procedures. One common assumption in connectionist approaches is that an AI system can be trained from scratch, as a *tabula rasa*, once a suitable architecture has been specified. No representational or inferential mechanisms are presupposed—perhaps inevitably, because of the "black box" character of neural net functioning.

We believe that the need for exposure to massive amounts of sensory data can be averted with a suitable "head start". The basic knowledge representations, reasoning procedures, language abilities, and world knowledge of a 1- to 2-year-old human child must be attained in any general intelligence architecture, and we believe that learning from text, implemented atop a suitably powerful symbolic framework, will be easier than doing so using data-fitting methods alone.

Our approach is to generate knowledge in the form of abstract logical "schemas". Our system's "head start" includes a semantic parser, a general inference system over a highly expressive logical form, and an initial set of simple schemas that a very young child could plausibly possess. Our system parses natural language stories into a logical form, matches the story to existing schemas, draws inferences, and, upon recognition of patterns, generalizes new schemas, whose variable roles take the place of the

individuals that vary from story to story.[1]

Schematic knowledge representations have a long history in artificial intelligence research; to underscore their usefulness, we will briefly outline that history and discuss some modern schema-oriented systems. We'll then describe our schema model in detail, differentiating it from past approaches, and describe its basic inference and generalization procedures, along with some examples of those procedures at work.

2 A Brief History of Schemas

The theoretical foundation of schemas has a history within cognitive science from even before AI existed as a field in order to explain the interactions of abstracted prior experience with language comprehension (Piaget, 1923; Bartlett, 1932; van Dijk and Kintsch, 1983). Cognitive science used schema-based theories to explain cognitive dissonance that occurs when faced with new information that cannot be accommodated by existing abstractions (Piaget and Inhelder, 1969), mistakes in memory surrounding schematic situations (Brewer and Treyens, 1981), and how we understand stories through an underlying grammar (Rumelhart, 1975). AI researchers with an eye for cognition computationalized these ideas into scripts and plans (Schank and Abelson, 1977) and frames (Minsky, 1975; Fillmore and Baker, 2010). Schank and Abelson's scripts successfully answered questions in the restaurant domain (among others) and Minsky's frames formed the basis for a number of AI systems for the remainder of the 20th century (Bobrow and Winograd, 1976; Fikes and Kehler, 1985; MacGregor and Burstein, 1991). However, these systems were limited to generating inferences from manually constructed frames.

Recent progress in learning schema-like knowledge has primarily been driven by applying statistical or neural network-based methods to large text corpora (Chambers and Jurafsky, 2011; Chambers, 2013; Pichotta and Mooney, 2016; Yuan et al., 2018). The learned schemas are quite limited in their capacity to enable inferences since they only describe high-level roles or temporal sequences. Perhaps this is all that can be expected from methods that are given minimal guidance and rely on many similar examples to find patterns. Wanzare et al. (2017) use crowdsourcing to help improve their clustering results and their scripts are made up of graph-ordered event clusters, but the clusters are groups of unstructured text segments. The goal of the schema framework we describe is to generate rich inferences which can be used to learn further schemas from a relatively limited number of examples.

3 Episodic Logic and Its Underspecified Form

Before diving into the details of our schemas, we first must describe the logical formalism in which the schemas are encoded, called Episodic Logic (EL) (Hwang, 1992; Hwang and Schubert, 1993; Schubert and Hwang, 2000). EL is a logical semantic representation that closely matches the form and expressive capacity of natural languages by extending FOL with semantic types and operators common in all languages. EL uses a small number type-shifting operators to map between specifically designated types to support the expressive power of natural language while keeping the underlying theoretical mechanisms simple. For example, EL uses reification operators to map predicate and sentence intensions to individuals. This allows predicate arguments in EL to denote both concrete and abstract entities (*Avicenna, the activity of writing poems, the idea that the universe revolves around the Earth*). Quantification remains first-order, with noun phrases treated as place-holders for constrained, quantified variables, in contrast with semantic theories that treat noun phrases as higher-order types. Another distinctive feature of EL, accounting for its name, is the *characterizing* operator. This operator, written $**$, relates an arbitrary EL formula to an episode it characterizes. If we restrict ourselves to positive, atomic predicates, $**$ closely resembles the Davidsonian use of event variables in predicates (Davidson, 1967). However, $**$ also allows complex characterizations of episodes, such as *Dana going hiking every weekend*, or *No nuclear nation being willing to eliminate its arsenal* (Schubert, 2000). An ontology of types is defined over the

[1] The code for our system is available at `https://github.com/bitbanger/schemas`.

Sentence *"Spot ran in the park"*

Episodic Logic	Unscoped Logical Form
`(some.d e: [e before.p |Now1|]`	`(|Spot| ((past run.v)`
`  [[e in-loc.p |Park1|] ∧ [[|Spot| run1.v] ** e]]))`	`  (adv-e (in.p (the.d park.n)))))`

Figure 1: EL and ULF formulas for an example sentence.

domain of individuals and includes categories such as basic individuals (e.g., *John, the Blarney Stone*, or *the Earth's magnetic field*), episodes (events, situations, processes), sets, numbers, propositions, and kinds. Schubert and Hwang (2000) provide a complete description of the ontology. EL has been shown to be suitable for deductive inference, uncertain inference, and Natural-Logic-like inference and has been used successfully to represent inference-enabling verb axioms (Morbini and Schubert, 2009; Schubert and Hwang, 2000; Schubert, 2014; Purtee and Schubert, 2017; Kim and Schubert, 2016). Inferences can be generated using the EPILOG inference engine (Morbini and Schubert, 2009; Schaeffer et al., 1993).

Most lexical items in EL are represented in the form `[word][sense num].[lexical type]`, e.g., `run1.v`.[2] Lexical types in EL are closely related to POS tags (e.g. `.v`, `.p`, and `.d` for verbs, prepositions, and determiners, respectively) but are constrained in their use by the EL semantic type system; e.g., modal *can* becomes `can.aux-s` or `can.aux-v` depending on its function as a sentence-level possibility operator or a VP-level ability operator. Names (denoting basic individuals) are represented by `|[name]|`, e.g. `|John|`. Predefined EL operators lack dot-extensions, e.g., `**`, `k`, `adv-a`. EL uses prefixed operators at the subsentential level (e.g., `(touch_down.v (on.p-arg |Mars|))`), and infix form at the sentence level (e.g., `[|InSight| (touch_down.v (on.p-arg |Mars|))]`). The distinction is emphasized in written EL by reserving square brackets for sentential formulas. Connectives allow for multiple arguments, i.e., `[w1 conn w2 ... wn]`, where the `wi` are sentential formulas and `conn` is `and` ($\wedge$) or `or` ($\vee$). Quantification has a distinct syntax: `([determiner] [variable]: [restrictor] [nuclear scope])`. If omitted, the restrictor is implicitly True, thus allowing for FOL quantifiers $\forall$ and $\exists$.

Figure 1 shows an example of a complete EL formula and demonstrates how episode variables are used. The `**` operator characterizes an episode `e` (constrained by `[e before.p |Now1|]`) with the formula `[|Spot| run1.v]`, and the conjunct adds locative information about `e` derived from the `adv-e` modifier in the ULF version. Using the `**` operator, EL can characterize episodes with arbitrarily complex well-formed EL formulas.

Finally, here we introduce a few type shifting operators that appear in the schema examples. The kind forming operator, `k`, is a function from monadic nominal predicate intensions to kinds. The kind of action forming operator, `ka`, is a function from monadic verbal predicate intensions to kinds of actions.[3] The proposition forming operator, `that`, is a function from sentence intensions to propositions. Some examples (with tense suppressed):

"Gold is yellow" – `[(k gold.n) yellow.a]` *"Peter likes to run"* – `[|Peter| like.v (ka run.v)]`

"My dog believes that it is a wolf" – `(my.d x: [x dog.n] [x believe.v (that [x wolf.n])])`

3.1 Unscoped Logical Form

Unscoped (Episodic) Logical Form (ULF) is an underspecified variant of EL which models the formal types and predicate argument structure of EL while leaving anaphora, word sense, and operator scopes unresolved. It is the first step in the EL parsing process and captures the semantic and pragmatic signals that can be accessed from natural language syntax. Its proximity to syntax makes semantic parsing into ULF relatively simple while its commitment to EL types enables inferences that preserve semantic coherence within the logical formalism. It turns out that ULF provides enough semantic resolution for enabling schema inferences in most of the first-reader stories we have considered. For a small number of cases, the ambiguous components in ULFs are resolved on an as-needed basis. Here we describe ULF to the extent necessary to understand its application within the presented schema description and examples.

[2] We use WordNet senses in this document (Miller, 1995), but EL is not strictly tied to WordNet.

[3] `ka` can be expressed in terms of `k`, forming a kind whose instances are agent-event pairs.

Kim and Schubert (2019) provides a more complete description of ULF and its uses outside of schemas.

Figure 1 shows the ULF for a sentence alongside the EL interpretation and demonstrates a few key differences between EL and ULF.

1. **ULF does not have episode variables.** ULF preserves type coherence in the face of implicit episodes and actions by introducing operators to form episode- and action-modifying adverbials from predicate intensions (`adv-e`, `adv-a`). Tense and aspectual operators are also implicitly episode-modifiers.
2. **Scope, anaphora, and word sense are unresolved.** Without scope or anaphora resolution, *"the park"* is simply `(the.d park.n)` in ULF compared to the referentially resolved `|Park1|` (or a Skolem constant if unresolvable) in EL. Without word senses, the lexical items are represented as `[word].[lexical tag]`.
3. **ULF only uses parentheses for bracketing.** The operator position is inferred from the semantic types of the bracketed elements.

4 Schema Form and Meaning

We will refer to the example schema in Figure 2—the generic schema for one agent giving an object to another agent for possession—as we explain the form and meaning of our schemas. This is one of our system's initial schemas. It is particularly simple, even in terms of the initial schemas we are assuming, in that it contains just a single "step". Such single-step schemas can specialize the type of action represented by the single step, or even just supply type information, preconditions, and effects, without specializing the step-concept.

```
(epi-schema
  ((?x give_obj_for_poss.v ?y ?o) ** ?e)
  (:Nonfluent-conds
    !r1 (?x agent.n)
    !r2 (?y agent.n)
    !r3 (?o object.n)
    !r4 (?l location.n) )

  (:Init-conds
    ?i1 (?x (can.aux-v (give_to.v ?y ?o)))
    ?i2 (?x (be.v (adv-e (at.p ?l))))
    ?i3 (?y (be.v (adv-e (at.p ?l)))) )

  (:Goals
    ?g1 (?x want.v
      (that (?y (have.v ?o)))) )

  (:Steps
    ?e1 (?x (give_to.v ?y ?o)) )

  (:Post-conds
    ?p1 (?y (possess.v ?o)) )

  (:Episode-relations
    !w1 (?e1 same-time ?e)
    !w2 (?e1 consec ?p1)
    !w3 (?e1 cause-of.n ?p1) ) )
```

Figure 2: An example schema.

4.1 Overall Structure

A schema comprises a schema type and header, shown here in the first line of the schema, followed by a list of sections. Sections are lists of logical formulas, indexed by a unique (across all sections) identifier, and fall into one of two types: *episode sections*, in which all identifiers begin with a question mark (*?*), and *nonfluent sections*, in which all identifiers begin with an exclamation mark (*!*). Each *?*-identifier in an episode section introduces an episode characterized by the formula that follows.

4.2 Episode vs. Nonfluent Sections

Fluent conditions, or episodic conditions, are "susceptible to change" over time. Nonfluent conditions hold true regardless of time. Within nonfluent sections, such as `:Nonfluent-conds` or `:Episode-relations`, we interpret any nonfluent condition identifier $!\mathtt{nf}_N$ as an alias (metavariable) that can be freely substituted for its associated formula Φ_N. Within episode sections, such as `:Steps` or `:Init-conds`, we interpret any episode condition identifier $?\mathtt{e}_N$ as an episode variable that is characterized by its associated formula Ψ_N.

4.3 Initial and Post Conditions

Initial conditions are a form of fluent predication—they must be true at the outset of the schema, but might not be true for the entire schema episode. They might be true before the start of the schema, or

they might become true at the exact start time of the schema. This temporal relationship is encoded, for each initial condition episode ?i_N, by the tacit assumption of the nonfluent episode-relational predication `((start-of.f ?e) during ?`i_N`)`. Postconditions may be true at any time, but must be true at least at the end point of the schema, or immediately afterward, so for each postcondition episode ?p_N, we tacitly assume the nonfluent predication `((end-of.f ?e) during ?`p_N`)`.

4.4 Header

The schema in Figure 2 is an *epi-schema*, that is, an *episodic* schema. It describes an episode in the episodic logic sense. The episode it describes, here `?e`, is called the "head episode" of the schema. The header provides a characterizing formula of the head episode. Derivation of the header characterization from a story sentence, here `give_obj_for_poss.v`, certainly implies the rest of the schema, but that is not necessarily true in the reverse direction (see Section 5.1 for a note on "confirming" schemas). An epi-schema can be viewed as defining an episode type (at least partially), where all *?*-variables in the header and body of a schema, except the head episode, are Skolem functions (equivalently, role functions) of that episode. As such, they can be equated to externally supplied constants. For example, the location variable `?l` in Figure 2 is interpreted as a Skolem function of `?e`, so that if `?e` receives a particular value, say `|EP1|`, then `(?l |EP1|)` in effect denotes the location of the participants; this might become equated to an external constant such as `|Home23|`.

4.5 Steps and Goals

The steps section enumerates the episodes that occur in, and constitute, the head episode, and their order of enumeration here sets the default event ordering. Goals are also episodes within the schema, but underlining their teleological contribution is necessary for action understanding, planning, and meta-reasoning—people do things for reasons.

4.6 Episode Relations

The episode relations section specifies temporal and causal constraints on the subepisodes belonging to the schema episode, including overriding the default relations for `:Init-conds` and `:Steps`. In Figure 2, episode `?e1` is characterized by some agent `?x` giving some object `?o` to some agent `?y`. A postcondition episode, `?p1`, is characterized by `?y` having that object. The episode relation section says that `?e1` and `?p1` are consecutive, and further that `?e1` is the direct cause of `?p1`. Temporal constraints in this section—which can relate the start times and end times of episodes, or provide uncertainty about start or end times—can induce a directed acyclic graph (DAG) of start and end times, a full interval graph, or a causality graph for action planning. Even the simplest induced graph, the DAG of start times, helps us to rule out schemas whose events are temporally inconsistent with a story. In the example schema, the constraint `!w1` says that step episode `?e1` shares a start and end time (a.k.a. `same-time`) with the schema's head episode `?e`. We intend `give_to.v` to generalize `give_obj_for_poss.v`. Under certain assumptions about event individuation (Schubert, 2000), this generalization relation together with the `same-time` predication actually implies that `(?e1 = ?e)`; so finding a story assertion matching `(?x give_to.v ?y ?o)` would suggest that the instantiation attempt for the schema is "on the right track".

5 Schema Matching and Inferences

Using schemas to make sense of a story requires casting the story in terms of schemas: We must find "matches" between sentences and individuals in the story, and formulas and roles in the schema. We have designed, implemented, and experimented with a basic algorithm which takes a ULF parse of a story, performs some preprocessing to get a basic EL form, matches the formulas in the story to new schemas as well as schema instances stored in "working memory", generates inferences from those schemas after filling in their variables, and uses those inferences to fill in yet more schemas. In this section, we detail

the algorithm, and walk through an example from one of our experiments on a children's first reader story from *The New McGuffey First Reader* (McGuffey, 1901).

5.1 Matching Algorithm

Algorithm 1 Algorithm for matching story formulas to schemas

INPUT: set of story episodes ST
MODIFIES: knowledge base of facts KB
$KB \leftarrow \varnothing$
for story episode E in ST **do**
  for conjunctive sub-formula F in E **do**
    $KB \leftarrow KB \cup \{F\}$
  while $\exists$ unprocessed formula $F_{ST} \in KB$ **do**
    for episode E_{SCH} in SCH episodes (linearized timeline) **do**
      for formula $F_{SCH} \in E_{SCH}$ **do**
        $MGU \leftarrow$ most general unifier of F_{SCH} and F_{ST}
        if MGU does not exist **then**
          continue
        for variable binding $B = V_{SCH} \rightarrow T_{ST}$ **do**
          if V_{SCH} is not already bound to something else **then**
            substitute T_{ST} for all occurrences of V_{SCH} in the schema instance
        if schema instance is confirmed **then**
          for fully-bound formula F_B in the instance **do**
            $KB \leftarrow KB \cup \{F_B\}$
    mark F_{ST} as processed

The matching algorithm takes a (potentially partially-filled) schema instance SCH, which is a schema and a map of its bound variable names to their values, and a story ST, which is a list of episodes, with their characterizing formulas, along with type predications, etc. Each step to match a story WFF is based on a global "knowledge base" of episodes and formulas we know to be true. The knowledge base is initialized with the story WFF, and the schema instantiation process then begins. We can create a new schema instance, or update an existing one, when a WFF in the knowledge base "matches" a WFF in the schema, i.e., is successfully unified. We obtain the "most general unifier" (MGU) using the algorithm by Robinson (1965). The MGU acts as a variable-to-term mapping, which we then use to replace variables throughout the schema instance. If a schema instance is "confirmed", we "infer" formulas within the schema whose variables have all been made concrete, and we add them to the knowledge base. As our initial schemas are quite simple, our current confirmation heuristic is simply whether all episodes in the `:Steps` section have been matched. As we continue to develop with more complex stories, and as schemas grow more complex, a more nuanced approach—perhaps involving certainties and numbers of variables mapped—is called for. Once the knowledge base has been updated, we return back to the schema matching step; we stop when we can no longer instantiate new schemas, or update existing instances, and move on to the next story WFF.

5.2 Planning and Meta-Reasoning

As the reasoning procedures and initial schemas are hypothesized to be quite general, we can make inferences about the beliefs and desires of agents within the story by "simulating" their reasoning with our own algorithms. A simple example of such an inference is that, if a mother wants her daughter to have a cat, and we know, via our general schemas, that owning a cat is pleasurable, we infer that the mother knew that her daughter having a cat would cause her daughter to experience pleasure, and we infer that the mother wanted her daughter to experience pleasure. We can draw many parallels to the

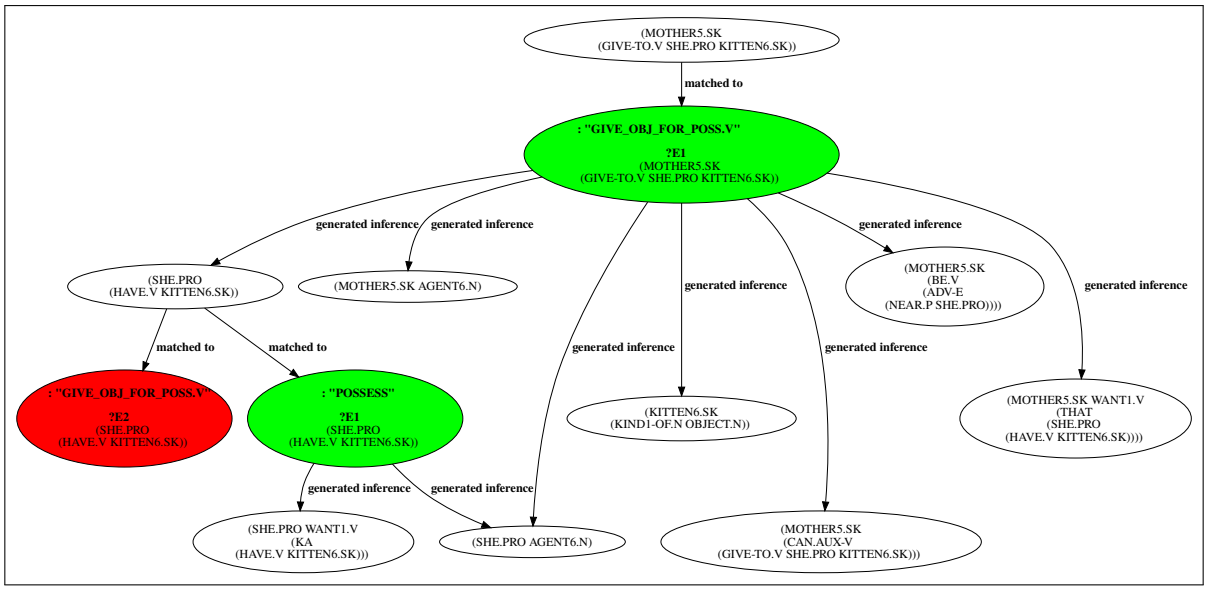

Figure 3: A graph of inferences made while processing a single story sentence. White vertices are inferred and story WFFs, green bubbles are "confirmed" schemas (see Section 5.1 for definition), and red bubbles are unconfirmed schemas.

planning domain: As agents have desires and beliefs, and schemas have goals, preconditions, and side effects, one could use existing schemas to solve planning problems, and, in turn, use planning algorithms to hypothesize new schemas for accomplishing certain goals.

5.3 Matching Example

Starting with the story sentence "Her mother gave the kitten to her" parsed into its EL form `(MOTHER5.SK (GIVE_TO.V SHE.PRO KITTEN6.SK))`, we'll walk through the matching algorithm, whose generated inferences are shown in Figure 3.

5.3.1 ULF Processing

We first process the ULF verb predication `((past give.v) (the.d kitten.n) (to.p-arg her.pro))` in Figure 4 to attach the argument marking preposition to the verb and float its argument to the front of the list. We then Skolemize the `the.d` determiners, demoting the relational noun predicate `(mother-of.n she.pro)` to the bare noun predicate `mother.n` to derive the Skolem name `MOTHER5.SK` (the 5 is a cumulative counter for Skolem constants; it is arbitrary, and an artifact of our Skolemization process), and using the noun predicate `kitten.n` to derive the Skolem name `KITTEN6.SK`. We finally obtain the EL sentence `(MOTHER5.SK (GIVE_TO.V SHE.PRO KITTEN6.SK))`, which is ready for matching.

5.3.2 The Initial Match

`GIVE_TO.V` immediately matches to the schema shown in Figure 2, as that schema has the same name. We unify the story formula `(MOTHER5.SK (GIVE_TO.V SHE.PRO KITTEN6.SK))` with the schema header, producing the unifier `(?x ← MOTHER5.SK, ?y ← SHE.PRO, ?o ← KITTEN6.SK)`, and we then make that substitution throughout the entire schema. Notably, every role in the schema is now bound except for the location `?l`. Note that, although the question of whether a schema is "confirmed" to have happened at any point in the matching process is difficult to answer in general, it is easy here: The actual verb in the schema's single step has been observed, so in the absence of type conflicts it is quite likely that the rest of the schema happened. All of the filled-in WFFs in the `Events`, `Goals`, `Init-conds`, and `Nonfluent-conds` sections, except those including the still-unbound variable `?l`, are added to our knowledge base as inferences from a confirmed schema.

```
; Here is May with her kitten.
(|May| ((pres be.v) here.a
        (adv-a (with.p (the.d (λx ((x kitten.n) and.cc (x (poss-by her.pro))))))))
; Her mother gave the kitten to her.
((the.d (mother-of.n her.pro)) ((past give.v) (the.d kitten.n) (to.p-arg she.pro)))
; She is kind to the pretty kitten.
(she.pro ((pres be.v) kind.a
          (adv-a (to.p (the.d (pretty.a kitten.n))))))
; She likes to see it jump and play.
(she.pro ((pres like.v) (ka (see it.v (jump.v and.cc play.v)))))
; See it run with May's ball!
(({you}.pro ((pres see.v) it.pro
    (run.v (adv-a (with.d (the.d (λ x ((x ball.n) and.cc (x (poss-by |May|))))))))) !)
; It does not run far with it.
(it.pro ((pres do.aux-s) not
         (run.v far.adv-a (adv-a (with.p it.pro)))))
; If May can get the ball she will not take it.
((if.ps (|May| ((pres can.aux-v) (get.v (the.d ball.n)))))
 (she.pro ((pres will.aux-s) not (take.v it.pro))))
; She will give it to the kitten to play with.
(she.pro ((pres will.aux-s)
          (give.v (to.p-arg (the.d kitten.n)) it.pro
           (adv-a ({for}.p (ka (play-with.v {it}.pro)))))))
```

Figure 4: A children's story, in partially post-processed ULF form. These were manually annotated but there are promising preliminary results on parsing ULF (Kim, 2019)

5.3.3 Further Matching

While the story sentence has now been matched, it has generated inferences, and those inferences might match other schemas. So, we are not done; we must attempt to match each of those before moving on to the next story sentence. Indeed, in this case, the inference `(SHE.PRO (HAVE.V KITTEN6.SK))` matches to two schemas: `GIVE_TO.v` and `POSSESS.V`; the former is unconfirmed (but, upon close inspection, would appear to be an incomplete copy of the instance that generated its matched WFF), and the latter was triggered (and confirmed) by the `HAVE.V` verb predicate in the inference. The `POSSESS.V` schema generates two more inferences, one of which was also generated by the original `GIVE_TO.V` schema, and the matching process concludes for this sentence, as all generated inferences have been matched to all possible schemas.

6 Conclusion & Future Work

Our approach to schemas comprises a rich logical language, initial schemas providing "low-level" inferences about the effects and motivations behind simple, general actions, and a way of doing "fuzzy matching" of inexact, but similar, formulas (see Section 6.2). All of these components work together to provide inferences that enable a comprehensive, structured, and human-oriented "fleshing out" of events in stories. The inferences we generate can then be combined, generalized, and reasoned about, thus creating new schemas from relatively few examples. There is much work remaining to further develop our parsing, matching, and especially our generalization processes. We proceed here to outline some of that future work—our initial results from matching a small handful of initial schemas to a short children's story are promising.

6.1 Immediate Future: Generalization

Currently, we are focusing our efforts on the task of schema generalization: Given two (or more) sequences of predications, which may themselves instantiate nested schemas, can we create a schema that describes both stories with a more general pattern? We are currently experimenting on four short children's stories about fishing, and each story includes similar words used in similar contexts: "These men

fish in the sea", "We will take the long rod, and the hook and line", "Here is Tom with his rod and line", "Sometimes they sit on the bank of the river", and many more thematically parallel sentences in the stories strongly suggest a schema where people are near water, have a rod, and catch fish with a rod or a net. After extracting a set of recurring events from the stories—like going to water, having a rod, catching fish, and putting fish in a basket—we plan to experiment with using narrative models like tense trees (Hwang and Schubert, 1992) to find subsequences of the events that could be interpreted as steps in a schema.

After extracting subsequences of similar events that occur in two or more stories, we can use knowledge of basic motivations—e.g., people often want to possess things—to infer a teleology of those events. If some unknown action "catch `?x`" always occurs before "possess `?x`", we might hypothesize that the catching something has an effect of possessing that thing. If that sequence is only ever seen with fish and crabs as `?x`, our confidence in the "catch to possess" schema might be lower when the object is not a marine animal, implying a "catch marine animal to possess" schema. Teleological inferences will likely prove to be very useful in sifting out reasonable new schema generalizations from a large collection of partly nonsensical ones.

Generalization can also take place from *one* example: after matching certain individual constants from a story to slots in a schema, we can consult a generalization hierarchy for those individuals' types (e.g. dog -> canine -> animal -> object), and select appropriate generalizations for the slots. These selections do not necessarily need to be made right away: the entire generalization hierarchy can be stored, and usage frequencies can be accumulated when more examples are found. However, they also do allow immediate generalization: a "reasonable guess" could be provided by the generalized word that's used the most frequently in some text corpus, for example. Further, if conditions are imposed on the entity by other formulas in the schema, the most general word that still satisfies those conditions could be selected, similar to the schema generalization in the GENESIS system (Mooney, 1990). For example, "border collie" could be replaced with "dog" if there were a later assertion that "the border collie competed in a dog show", but replacing it with "animal" would be too general.

6.2 "Fuzzy Matching"

So far, we have mostly discussed exact matching of WFFs using the MGU algorithm. However, in many cases, we will want to match something like a "Segway" to a schema predication about a "vehicle", just as readily as we would match "car" to "vehicle". Additionally, synonyms and intuitively similar words, like "run" and "jog", should be able to match as well, perhaps with some certainty score. Hypernym hierarchies, semantic word embeddings, and logical world knowledge of object properties all affect whether we want to make inexact matches. We are actively researching how best to gather and use this information in the matching process—as well as how to index schemas for quick retrieval, even when matches are inexact.

6.3 A Note on Condition Strictness

It is currently unclear exactly how and when condition violations are acceptable. Certainly, to generalize new schemas, we must allow some "slack" in condition violations: a schema that fits, say, a "fishing" schema perfectly, except the variable constrained to be a fish is actually a crab, should cause us to infer that "catching seafood" might be a good generalization to store. However, different conditions intuitively seem violable to different degrees, and in different contexts, depending on what other conditions are met or violated. Many factors seem to affect whether we match or abandon a schema, or create a new schema, when trying to make sense of a story. These factors will be the subject of many future experiments.

6.4 Scaling the Matching Algorithm

Our algorithm is in an early stage, and several scaling problems must be solved to bring it to bear on an unsupervised learning task. We are currently experimenting on an assortment of children's first reader

stories to determine rules and heuristics to guide us as we develop the matching algorithm to cope with large numbers of possible schema matches.

6.4.1 Role Assignment Restriction

As the number of schemas scales, it will be combinatorially infeasible to maintain every possible schema instance a story WFF might prompt; if there are three humans in a schema, and ten humans mentioned in a story, there are $\binom{10}{3}$ ways to instantiate the human roles alone in that schema. In many cases, it may suffice to "let the actions speak for the agents", that is, prefer to use verb predications to identify individuals for schema roles, rather than considering role assignments directly. The schema relates arguments to its verb predications to their role definitions and "type" predications, so if we can infer that `?x` is `|Mary|` from an action that she did, we can then simply confirm or refute the additional constraint `(?x human.n)`. However, there could be examples where there is still suitable ambiguity, or where "action-first" role assignment isn't the most efficient; we hope to discover and address these examples in our ongoing story experiments.

6.4.2 Instance Abandonment

What might seem like a "promising" schema match at first could diverge suitably from the story and become useless; we would want to abandon it to free up memory, and so we don't continue fruitless comparisons of story WFFs to it. However, it is difficult to know when a schema instance is unlikely to continue to be matched to WFFs; sentences read, or inferences generated, much later on in a story could provide the missing piece to a schema instance matched many sentences ago. We plan to use our ongoing story experiments to formulate abandonment heuristics as well, or even identify potential applications of machine learning to the problem.

6.4.3 Schema Retrieval

As the number of schemas grows, and as we allow for inexact matching, whether of hypernyms, synonyms, or "functionally equivalent" terms (based on world knowledge), the task of identifying reasonable schemas as match candidates grows more difficult in turn. As a first step, indexing schemas by a handful of specific predications—specific verbs and nouns, chosen to maximally prune the search space—could allow for quick identification of relevant schemas. From there, indexing by sequences, or subsequences, of events should help us quickly identify highly specific sequential schemas within a large corpus; the human brain seems to be able to recall schemas very quickly with very terse sequences, as Winograd (1973) demonstrates with the sequence "skid, crash, hospital". Finally, any indexing we've done will need to be augmented to perform well even with inexact matching as described above. Schema retrieval optimization will be heavily informed by our experiments with the matching space on real stories, and especially as our number of schemas grows.

7 Acknowledgements

We would like to thank Benjamin Kane, Benjamin Kuehnert, Sophie Sackstein, and Georgiy Platonov for their development assistance and participation in schema-related discussions. This work was supported by the NSF NRT Fellowship DGE-1449828, and DARPA CwC subcontract W911NF-15-1-0542.

References

Bartlett, F. C. (1932). *Remembering: A Study in Experimental and Social Psychology*. Cambridge University Press.

Bobrow, D. G. and T. Winograd (1976). On overview of KRL, a knowledge representation language. *Cognitive Science 1*, 3–46.

Brewer, W. F. and J. C. Treyens (1981). Role of schemata in memory for places. *Cognitive Psychology 13*(2), 207–230.

Chambers, N. (2013). Event schema induction with a probabilistic entity-driven model. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807. Association for Computational Linguistics.

Chambers, N. and D. Jurafsky (2011). Template-based information extraction without the templates. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 976–986. Association for Computational Linguistics.

Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of Decision and Action*. University of Pittsburgh Press.

Fikes, R. and T. Kehler (1985, September). The role of frame-based representation in reasoning. *Commun. ACM 28*(9), 904–920.

Fillmore, C. J. and C. Baker (2010). A frames approach to semantic analysis. In B. Heine and H. Narrog (Eds.), *The Oxford handbook of linguistic analysis*, pp. 313–340. Oxford University Press.

Hwang, C. and L. Schubert (1993). Episodic Logic: A situational logic for natural language processing. In P. Aczel, D. Israel, Y. Katagiri, and S. Peters (Eds.), *Situation Theory and its Applications 3 (STA-3)*, pp. 307–452. CSLI.

Hwang, C. H. (1992). *A logical approach to narrative understanding*. Ph. D. thesis, University of Alberta.

Hwang, C. H. and L. K. Schubert (1992). Tense trees as the "fine structure" of discourse. In *Proceedings of the 30th Annual Meeting on Association for Computational Linguistics*, ACL '92, Stroudsburg, PA, USA, pp. 232–240. Association for Computational Linguistics.

Kim, G. and L. Schubert (2016, August). High-fidelity lexical axiom construction from verb glosses. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*, Berlin, Germany, pp. 34–44. Association for Computational Linguistics.

Kim, G. and L. Schubert (2019, May). 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.

Kim, G. L. (2019). 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*.

MacGregor, R. M. and M. H. Burstein (1991). Using a description classifier to enhance knowledge representation. *IEEE Expert 6*, 41–46.

McGuffey, W. H. (1901). *The New McGuffey First Reader*. American Book Company.

Miller, G. A. (1995, November). WordNet: A lexical database for english. *Communications of the ACM 38*(11), 39–41.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The psychology of computer vision*. Cambridge, Massachusetts: McGraw-Hill.

Mooney, R. J. (1990). *A general explanation-based learning mechanism and its application to narrative understanding*. Morgan Kaufmann.

Morbini, F. and L. Schubert (2009, June). Evaluation of Epilog: A reasoner for Episodic Logic. In *Proceedings of the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning*, Toronto, Canada.

Piaget, J. (1923). *Langage et pensée chez l'enfant (The Language and Thought of the Child)*. Neuchâtel: Delachaux et Niestlé.

Piaget, J. and B. Inhelder (1969). *The Psychology of the Child*. New York: Basic Books.

Pichotta, K. and R. Mooney (2016). Statistical script learning with recurrent neural networks. In *Proceedings of the Workshop on Uphill Battles in Language Processing: Scaling Early Achievements to Robust Methods*, pp. 11–16. Association for Computational Linguistics.

Purtee, A. and L. Schubert (2017). Simple rules for probabilistic commonsense reasoning. In *Proceedings of the 5th Ann. Conf. on Advances in Cognitive Systems (ACS 2017)*. Cognitive Systems Foundation.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*(1), 23–41.

Rumelhart, D. E. (1975). Notes on a schema for stories. In D. G. BOBROW and A. COLLINS (Eds.), *Representation and Understanding*, pp. 211–236. San Diego: Morgan Kaufmann.

Schaeffer, S., C. Hwang, J. de Haan, and L. Schubert (1993). EPILOG, the computational system for episodic logic: User's guide. Technical report, Department of Computer Science, University of Alberta.

Schank, R. C. and R. P. Abelson (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Oxford, England: Lawrence Erlbaum.

Schubert, L. (2014, June). From treebank parses to Episodic Logic and commonsense inference. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, Baltimore, MD, pp. 55–60. Association for Computational Linguistics.

Schubert, L. K. (2000). The situations we talk about. In J. Minker (Ed.), *Logic-based Artificial Intelligence*, pp. 407–439. Norwell, MA, USA: Kluwer Academic Publishers.

Schubert, L. K. and C. H. Hwang (2000). Episodic Logic meets Little Red Riding Hood: A comprehensive natural representation for language understanding. In L. M. Iwańska and S. C. Shapiro (Eds.), *Natural Language Processing and Knowledge Representation*, pp. 111–174. Cambridge, MA, USA: MIT Press.

van Dijk, T. A. and W. Kintsch (1983). *Strategies of Discourse Comprehension*. New York: Academic Press.

Wanzare, L., A. Zarcone, S. Thater, and M. Pinkal (2017). Inducing script structure from crowdsourced event descriptions via semi-supervised clustering. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pp. 1–11. Association for Computational Linguistics.

Winograd, T. (1973). A procedural model of language understanding. In R. Schank and K. Colby (Eds.), *Computer Models of Thought and Language*, pp. 152–186. New York, NY: WH Freeman.

Yuan, Q., X. Ren, W. He, C. Zhang, X. Geng, L. Huang, H. Ji, C.-Y. Lin, and J. Han (2018). Open-schema event profiling for massive news corpora. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, New York, NY, USA, pp. 587–596. ACM.