Reasoning about Robocup-soccer Narratives

Hannaneh Hajishirzi[1], Julia Hockenmaier[1], Erik T. Mueller[2], and Eyal Amir[1]
[1]{hajishir, eyal, juliahmr}@illinois.edu, [2]etm@us.ibm.com
[1]University of Illinois at Urbana-Champaign, [2]IBM TJ Watson

Abstract

The ability to translate natural language into a semantic representation that is amenable to further inference is a hallmark of natural language understanding. Since the interpretations of individual sentences have to be combined into a coherent whole, it has long been known that a planning-like approach to natural language understanding which incorporates world knowledge in the form of preconditions and effects of events can be used e.g. in text generation systems. This paper argues that such domain knowledge can also play an important role in learning to understand text. We present a planning-based approach which learns to translate simple narratives into a coherent sequence of events without labeled training data. We apply our approach to the reconstruction of Robocup soccer games, and show that it outperforms state-of-the-art supervised learning systems in this domain.

1 Introduction

Natural language understanding requires the ability to translate individual sentences into a semantic representation of the underlying entities, their properties, relations, actions and the effect of their actions on the state of the world. It furthermore requires the ability to combine the semantic representations of individual sentences into a coherent whole, which in turn makes it possible to draw inferences that go beyond what is explicitly mentioned, and is therefore necessary for a 'deep' understanding of the text. For example, knowing who has possession of the ball at any point during a soccer game from a commentary of the game alone requires the ability to infer numerous events that are implied but may not be explicitly mentioned.

This paper argues that the assumption that text is coherent provides a strong bias that can be exploited when learning to understand language. One particularly simple form of coherence requires that events can only take place when their preconditions are met. For example, a soccer player cannot kick the ball unless he is currently in possession of the ball. We present an unsupervised approach which incorporates such domain knowledge in the form of soft constraints to learn to understand sports commentaries. We use human commentaries of four championship games in the Robocup simulation league(Chen, Kim, & Mooney 2010), and map each narrative to a sequence of events such as *pass, kick, steal, offside*. Soccer commentaries are simple narratives which differ from more complex narratives and other forms of text in that they report a linear sequence of events that unfold over time. In a simple narrative, each sentence leads therefore to an incremental update of the overall semantic representation, or discourse model (Webber 1978; Johnson-Laird 1983; Grosz & Sidner 1986), making it possible to reconstruct the original temporal sequence of events and draw further inferences.

In this paper we examine how prior domain knowledge of the preconditions and effects of events (specified in a STRIPS-like framework (Fikes & Nilsson 1971)) and a bias towards coherent discourse models can be exploited in learning how to map narratives to event descriptions. In contrast to other recently proposed approaches (e.g., (Zettlemoyer & Collins 2009; Chen, Kim, & Mooney 2010; Kate & Mooney 2007)), we do not require any labeled training data. Our system also does not require an agent which receives indirect supervision by interacting with a physical environment (e.g., (Branavan *et al.* 2009; Vogel & Jurafsky 2010)). It is often very hard, if not infeasible, to either create human-annotated data or to have access to an interactive environment that provides indirect supervision. In our experiments we show that knowledge about the preconditions and effects of events alleviates the need for labeled training data. In particular, our unsupervised approach outperforms the state-of-the-art supervised approach of (Chen, Kim, & Mooney 2010) on understanding Robocup soccer commentaries, even when we extend it to incorporate similar domain knowledge at inference time.

Similar to (Chen, Kim, & Mooney 2010), we formulate language understanding as a classification problem in which we have to predict events from individual sentences. In contrast to other approaches, our classifier also receives as input our guess of the current state of the world. The classifier assigns a score to each possible event. We interpret these scores as utilities, and use dynamic programming to find the sequence of events that has maximal utility. Events that violate domain constraints (because their preconditions are not met in what we assume to be the current state of the world) are penalized. An alternative approach (to be explored in future work) might treat this task as a complex sequence labeling problem where each element of the sequence corresponds to an individual sentence and the label to the predicted event. Our classifier is trained in an iterative fashion that is reminiscent of self-training or hard EM (Bishop

2006). Starting from an initial guess of a coherent event sequence we iteratively retrain it on the sequence of events that has maximal utility according to the current version of the classifier.

1.1 Related Work

There are several symbolic narrative understanding systems (e.g., (Hobbs *et al.* 1993)) which apply abductive reasoning to a very large knowledge base by considering every element of the text as a logical element, but do not model uncertainty which is essential for narrative understanding.

There are other approaches that map natural language text to meaning representations. Some (Zettlemoyer & Collins 2009) use annotated labeled data and are focused in texts describing facts rather than describing dynamics of a system. Most similar to our approach are (Branavan *et al.* 2009; Chen, Kim, & Mooney 2010) which map narratives to sequence of events with applications in understanding instructions or generating commentaries. (Branavan *et al.* 2009) use reinforcement learning and have access to a physical environment to provide supervision for assigning rewards to selecting events. (Chen, Kim, & Mooney 2010) have access to the actual events of soccer games and use a mapping between commentaries and real events of the game for training. In contrast, our approach uses prior knowledge about events, does not have access to real events happened in the soccer, does not interact with a real physical environment, and uses binary classifiers instead of reinforcement learning.

Several approaches introduce reasoning, learning, or planning algorithms in a probabilistic logical framework to model events. Planning and reasoning approaches, unlike us, assume that the probability distribution over events are known. Exact reasoning approaches (e.g.,(Baral & Tuan 2002; Reiter 2001)) are not feasible in our problem as they usually consider all the possible paths of the probabilistic event sequence. Sampling possible deterministic events of the narrative (Hajishirzi & Amir 2008) is still too expensive. Probabilistic planning approaches (Majercik & Littman 1998) usually find the most likely plan given initial and goal states.Learning approaches (Deshpande *et al.* 2007; Zettlemoyer, Pasula, & Kaelbling 2005) compute the probability distribution over different events. Unlike us, these approaches use annotated labeled data and train their classifier for a single probabilistic event rather than accumulating information from selected events using an inference subroutine. corresponding to every transition is known.

Our approach is also related to research in plan and (more so) activity recognition. Some apprpaches (e.g., (Kautz 1987)) use logical elements and symbolic reasoning, but are not able to rank different consistent plans. Other approaches use probabilistic reasoning (e.g., Bayesian networks in (Charniak & Goldman 1993) or HMMs in (Bui 2003)). Other approaches (e.g., (Riley & Veloso 2004; Liao *et al.* 2007)) incorporate learning and reasoning in dynamic models such as HMMs or Markov Decision Processes. These approaches are usually augmented with annotated labeled data or do not use logics to model the domain. Most recently, (Sadilek & Kautz 2010) recognize activities by applying relational inference and learning (with noisy GPS information as training labels). They neither improves initial estimates of labels nor use consistency checking. Moreover, using hard and soft constraints, they augment their system with richer (more expensive) prior knowledge compared to our few event descriptions.

2 Problem Definition

The problem that we address here is to find the best sequence of events that interprets an input narrative. Events are described in terms of preconditions and effects.

2.1 Natural Language Narratives

A narrative describes the dynamics of a system as a sequence of sentences in natural language. Specifically, a narrative is a temporal sequence of length T of sentences $\langle w_1, w_2, \ldots, w_T \rangle$. Examples of such narratives are commentaries, stories, reports, and instructions. Through this paper we work with commentaries of four final games of Robocup soccer simulation league taken from (Chen, Kim, & Mooney 2010).

Sentence: Every sentence in the narrative is either an observation about the state of the system (e.g., "Offside has been called on the Pink team.") or a change that happens in the state of the system (e.g., "Pink9 tries to kick to Pink10 but was defended by Purple3"). There is uncertainty in understanding the associated meaning of a sentence as the sentences are in natural language. For example, the above sentence can be interpreted as *passing* between players, *bad-passing* between players, a player *kicking* the ball, or a player *defending* the other player.

State of the world: The state of the world depicts the underlying state of the system that is changing over time. For example, after sentence "Pink goalie kicks off to Pink2" the state of the world is "Pink2 has possession of the ball" which shows the actual state of the soccer game.

2.2 Meaning Representations

Our meaning representation framework consists of (1) logical elements for representing the domain and (2) prior knowledge about specifications of events.

Representation Language: We use a logical language to represent events, entities, and states. The language consists of a finite set of constants (e.g., teams $PurpleTeam$, $PinkTeam$ and players $Pink1$, $Purple1$, $Pink2$, $Purple2$), variables (e.g., $player1$, $team$), predicates (e.g., $Holding(player, ball)$, $atCorner()$, $atPenalty()$), and events (e.g., $pass(player1, player2)$, $kick(player1)$, $steal(player1)$).

Definition 1. The language $\mathcal{L}$ of our meaning representation framework is a tuple $\mathcal{L} = (C, V, F, E)$ consisting of

- C a finite set of constants representing objects in the domain
- V a finite set of variables
- F a finite set of predicates (called *fluents*) whose values change over time
- E a finite set of deterministic event names

We define a fluent literal as a formula of the form $f(x_1, \ldots, x_k)$ or $\neg f(x_1, \ldots, x_k)$ (also represented by $f(\vec{x})$)

where $x_1, ..., x_k \in V \cup C$ are either variables or constants. In this setting, the grounding of a fluent $f(\vec{x})$ is defined as replacing each variable in $\vec{x}$ with a constant $c \in C$.

A state s in this framework is defined as a full assignment of $\{true, false\}$ to all the groundings of all the fluents in F. However, it is generally the case that, at any particular time step, the values of many fluents are not known. Therefore, we define states of narratives as partial states which are conjunctions of fluent literals whose truth values are known. A partial state σ is a function $\sigma : GF \rightarrow \{true, false, unknown\}$ where GF is the set of ground fluents. We interchangeably represent a partial state σ as a conjunction of fluent literals where a fluent literal is in the form of either f (for $\sigma(f) = true$) or $\neg f$ (for $\sigma(f) = false$).

Event: Events are represented as event names together with a list of parameters and are generally specified with preconditions and effects as STRIPS actions. Every event either describes the state or deterministically maps a state to a new state. Event descriptions are available to our system as prior knowledge and are described in a relational form. This means that the parameters of the events are variables rather than constants.

Definition 2. Let *e* be an event name and $\vec{x}$ be variables as event parameters. If *Precond*($\vec{x}$) and *Effect*($\vec{x}$) are conjunctions of fluent literals then the effect axiom for the event $e(\vec{x})$ is represented as:

- Preconditions: *Precond*($\vec{x}$)
- Effects: *Effect*($\vec{x}$)

Here we use the frame assumption that the truth value of a fluent stays the same unless it is changed by an event. For example, "$pass(player_1, player_2)$ with Preconditions: $holding(player_1)$, Effects: $holding(player_2)$" describes the event $pass$ that changes the possession of the ball from $player_1$ to $player_2$. Or the event "$kick(player_1)$ with Preconditions: $holding(player_1)$, Effects: $\neg holding(player_1)$" describes that the player1 is no longer holding the ball after he shoots.

Most events associated with Robocup commentaries involve actions with the ball such as kicking and passing. There are some other events that show game information such as whether the current state is penalty, offside, or corner. For example the event *corner* is described as "$corner()$ Preconditions: $true$, Effects: $atCorner()$".

The set of deterministic events includes a noise event called *Nothing* that has no preconditions and no effects. The reason for including this noise event is that some narratives include sentences that are not mapped to any actual events. For example, sentence "Today we have a nice match between pink and purple teams" does not map to any of the described events for the soccer game. In addition, the noise event helps to fix inconsistencies that exist in the narrative. For example, a soccer commentary is not always consistent if the commentator has missed commenting on some of the events of the game. Mapping sentences to the noise event allows some flexibility for mapping other sentences correctly.

2.3 Transition Model for Sentences

Each sentence in the narrative describes an uncertainty among different events. We assign a score to all the possible events associated with a sentence. This score also depends on the current state. The score corresponding to a sentence w and state s is represented as $P(e_i|w, s)$ for all the events e_i.

To map the narrative to sequence of events we first need to compute $P(e_i|w, s)$ for every sentence in the narrative. We model this score as a logistic function (Bishop 2006) and learn it in an iterative learning procedure (Section 3.1). Each iteration involves estimating the label for the training examples and modifying the model accordingly.

We assume that at most one of the domain events will be mapped to each sentence. For instance, for sentence "Pink6 tried to pass to Pink10 but was intercepted by Purple3" the goal is to map the sentence to final event $BadPass(Pink6, Purple3)$ rather than fine grained events like $kick(Pink6)$, then $pass(Pink6, Pink10)$, and then $BadPass(Pink6, Purple3)$.

3 Mapping Narratives to Event Sequences

Our approach, *ITerative Event Mapping* (ITEM), uses prior knowledge and is built upon two subroutines of inference and learning. Iterative learning subroutine (Section 3.1) learns scores of different events corresponding to every sentence. Inference subroutine (Section 3.2) finds the best event sequence using the learned scores. Our approach uses prior knowledge in initializing labels, building training examples by updating the current state, and in the inference subroutine.

3.1 Iterative Learning

Our iterative learning subroutine *IterTrain* is illustrated in Figure 2. The inputs to this subroutine are narratives and prior knowledge about event effect axioms. We train a binary classifier to separate the correct event from the other events for every sentence and state.

Our algorithm divides the set of input narratives to training and test narratives. It then generates training examples (e_i, w, s) from the training narratives using *Example Generator*. Afterwards, it computes features $\vec{\Phi}(e_i, w, s)$ for each training example using *Feature Extractor*. Since the correct labels of training examples are not known, the algorithm estimates initial training labels generated by *Initial Label Generator* that uses prior knowledge. Next, it uses *Classifier* to learn the model parameters $\vec{\Theta}$ for the current training examples and the estimated labels. It uses logistic regression (Bishop 2006) and computes the binomial probability $P(e_i|w, s)$ that event e_i is the correct interpretation of sentence w in state s.

In the next iterations, the algorithm uses the learned model parameters and finds new labels for the training examples. These steps will re-iterate until convergence i.e., $||\vec{\Theta}_{t+1} - \vec{\Theta}_t|| < \epsilon$.

For testing, we use *ComputeP* over the final learned model to compute scores of events associated with every sentence in the test narrative. In following we describe details of different subroutines used for training.

Training Example Generator: This module takes a training narrative as a sequence of sentences $\langle w_1 \dots w_T \rangle$. It returns training examples in the form of (sentence w_t, event

Algorithm 1. *ITEM*(Train $Tx_{1..3}$, Test *Tx*, *EA*)
- Input: Train narratives $Tx_{1..3}$, Test narrative *Tx*, *PAM*

1. $\vec{\Theta} \leftarrow$ *IterTrain*$(Tx_{1..3}, EA, K, N)$
2. *Inference*$(Tx, \vec{\Theta})$

Algorithm 2. *IterTrain*(Tx, PAM, N)
- Input: training narrative *Tx*, effect axioms *EA*, language $\mathcal{L}$

1. Repeat until $\vec{\Theta}_{t+1} - \vec{\Theta}_t < \epsilon$
   (a) $(w_i, e_i, s_i)_{i:1..S} \leftarrow$ *ExampleGenerator*(Tx, EA, N)
   (b) **for** i : 1 to S
      i. $\vec{F}_i \leftarrow$ *FeatureExtractor*(w_i, e_i, s_i)
      ii. $l_i \leftarrow$ *LabelGenerator*(w_i, e_i, s_i, EA)
   (c) $\vec{\Theta} \leftarrow$ *Classifier*$(\vec{F}_{i:1..S}, l_{i:1:S})$

Algorithm 3. *ComputeP*(event e, sentence *w*, state *s*, $\vec{\Theta}$)
1. for e_i in *PAM.DA*:
   (a) $\vec{\Phi}_i \leftarrow$ *FeatureExtractor*(w, e_i, s)
   (b) $score_i \leftarrow$ *LogisticFunction*$(\vec{\Phi}_i, \vec{\Theta})$
2. $P(e|s, st) \leftarrow$ normalize $score_i$

Algorithm 4. *Inference*(*Tx*,$\vec{\Theta}$, *E*, *EA*)
- Input: Test narrative $Tx = \langle w_1 \dots w_T \rangle, \vec{\Theta}, E$
- Output: sequence of events $\langle e_1, \dots, e_T \rangle$

1. if $t = 1$ initialize $V_{1,e_i}, S_{1,e_i}, Seq_{1,e_i}$ from Eq. 1
2. for $t = 2 \dots T$
   (a) for e_i in *E*:
      i. $e \leftarrow \arg\max_{e_i \in E}(V_{t-1,e_i})$
      ii. $s_{t-1} \leftarrow S_{t-1,ev}$
      iii. $V_{t,e_i} =$ *ComputeP*$(e_i, w_t, s_{t-1}, \vec{\Theta}) + V_{t-1,e}$
      iv. if *Precond*$(e_i) \not\models s_{t-1}$: $V_{t,e_i} \leftarrow V_{t,e_i} - 1$
      v. $Seq_{e_i,t} \leftarrow Seq_{e,t-1} + [e_i]$
      vi. $S_{t,e_i} \leftarrow$ *Progress*(s_{t-1}, e_i)
3. $e_T \leftarrow \arg\max_{e_i \in E}(V_{T,e_i}), e_{1..T-1} \leftarrow Seq_{e_t,t}$

Figure 1: The ITerative Event Mapping (ITEM) algorithm to find the best event sequence corresponding to a narrative, with subroutines *IterTrain* to find the model parameters $\vec{\Theta}$ by training on train narratives, *ComputeP* to compute the normalized score over different events associated with every sentence, and *Inference* to find the best event sequence given the scores and the event descriptions.

e_i, state s). Recall that our learning algorithm computes the probability that e_i is the interpretation of w_t in the s. This module first samples events from the space of events and generates pairs of (sentence, event). It then updates the state of the narrative for each pair of event and sampled event. This provides the final triplets.

To build pairs of (w_t, e_i) for each sentence w_t, we first sample N events e_i uniformly from the set of events E. For example, we sample events $pass$, $steal$, and $kick$ for the sentence "P7 passes the ball to P9" in Figure 2. Next, we extract the words in the sentence that correspond to a player name. For that, we assume that we know the list of players for the soccer game. For example, arguments for the above sentence are $P7$ and $P9$. We then ground the sampled events using these extracted arguments from the sentence. To ground the event we replace the variables in the event $pass(player_1, player_2)$ with $P7$ and $P9$ that are constants in our logical language. For the above sentence the grounded event is $pass(P7, P9)$.

We then compute the state of the narrative given the sampled events. So far we have N sequences of sampled events $evs_i = \langle e_1, \dots e_T \rangle$ corresponding to consec-

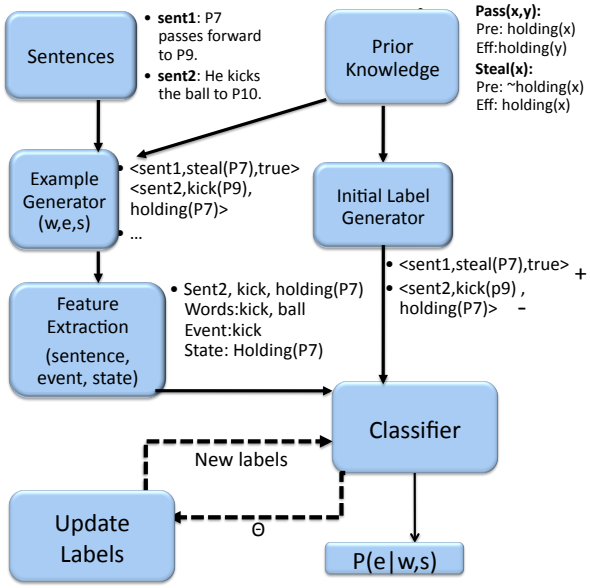

Figure 2: The architecture of our iterative learning approach *IterTrain* (Algorithm 2) to compute the model parameters $\vec{\Theta}$ and returning the normalized score of every event given the sentence and the current state.

utive sentences. We build a new sequence in the form of $\langle s_0, e_1, s_1, e_2, s_2, \dots, e_T, s_T \rangle$. Every s_t is the state of the narrative at time t and is computed using prior knowledge of event axioms. We initialize the state of the world by *true* i.e., $s_0 = true$. We update each state s_{t-1} at time t given event e_t using a *progress* subroutine. We form training examples as triplets (w_t, e_t, s_{t-1}) from the new sequence.

For example, a sequence of events sampled for the narrative in Figure 2 is $\langle steal(P7), kick(P9) \rangle$. We update the state given the sampled event sequence and derive $\langle true, steal(P7), holding(P7), kick(P9), \neg holding(P9) \rangle$. The reason is that s_1 is $holding(P7)$ if event $steal(P7)$ happens in state $s_0 = true$. From this sequence we extract two training examples $(sent1, steal(P7), true)$ and $(sent2, kick(P9), holding(P7))$.

Progress subroutine, *Progress*(s_{t-1}, e_t), takes as input an event e_t and the current state s_{t-1}. It then returns the updated state s_t if the preconditions of the event e_t is consistent with s_{t-1}. The current state s_t is updated by applying the effect axioms of the event e_t.

Feature Extractor: This module takes an example (sentence w, event e, state s) as input and returns the corresponding feature vector $\vec{\Phi} = (1, \vec{\phi}_w, \vec{\phi}_e, \vec{\phi}_s)$ where 1 is a bias. $\vec{\phi}_w$ is a binary vector representing the sentence w where each element in the vector shows the presence of the corresponding words of the vocabulary in the sentence. $\vec{\phi}_e$ is a binary vector to represent the event e where each element in the vector represents the presence of the corresponding event name in the training example. $\vec{\phi}_s$ is a binary vector representing the state s where each element in the vector represents the truth value of the corresponding fluent name in the state s.

Label Generator: This module takes as input an example (sentence w, event e, state s) and automatically generates a

boolean label as an estimate of the actual label. This module distinguishes us from supervised learning approaches as here we automatically generate labels. The intuition behind a positive label is that event e is a correct meaning of the sentence w given that the state s.

Initial Label Generator: In the first iteration, this module uses prior knowledge about events (event effect axioms) and automatically assigns labels to the training examples. More specifically, an example will be assigned a positive label if the preconditions of event e are consistent with the current state s and the name of event e has low edit distance to a word in the sentence w. For instance, training example $(sent1, steal(P7), true)$ has been assigned a positive label since $steal(P7)$ is feasible in $s_0 = true$. However, the example $(sent2, kick(P9), holding(P7))$ has been assigned a negative label as $kick(P9)$ is not feasible if $s = holding(P7)$.

Update Labels: In next iterations, labels are generated by applying the current classifier to training examples. This module uses the weights learned by the current classifier and assigns a score to training examples. An example (w, e, s) would be assigned a positive label if its score is higher than the score of replacing the event e in the example with other events in the dataset.

Classifier: At each iteration, the *classifier* takes training examples (e_i, w, s) together with generated labels for that iteration and returns a weight vector. This module first removes negative examples randomly to balance training examples and make the number of positive and negative labels comparable. It then trains a linear classifier (*logistic regression* (Bishop 2006)). We use logistic regression to compute binomial probability $P(e_i|w, s)$ to assign a score to the events given the sentence w and state s. We model this score as a logistic function i.e., $P(e_i|w, s) = 1/(1 + \exp(-\vec{\Theta}^t \times \vec{\Phi}(e_i, w, s))$ where $\vec{\Phi}$ is the feature vector associated with training examples and $\vec{\Theta}$ is a vector of model parameters which we want to learn. The output of the classifier is model parameters $\vec{\Theta} = (\theta_1, \vec{\theta}_w, \vec{\theta}_e, \vec{\theta}_s)$ which is used to compute the score of events.

To test, *ComputeP* computes the score of events e_i corresponding to the test sentence w in the state s. *ComputeP* first uses the arguments of the sentence and grounds all the possible events e_i associated with the sentence. It computes the score of each test example using logistic function with learned parameters. It then normalizes the scores of examples (w,e_i,s).

Then, the inference subroutine (next section) uses the computed scores to find the best event sequence corresponding to the test narrative.

3.2 Inference

The *inference* subroutine is a Viterbi-like (Rabiner 1989) dynamic programming approach that finds the best event sequence corresponding to the input narrative and our model. *Inference* (Algorithm 4) takes as input the narrative $\langle w_1, \ldots, w_T \rangle$, event effect axioms *EA*, and the normalized scores of different events computed for the narrative sentences.

Intuitively, the inference subroutine selects the event that is most likely and feasible in the state of the narrative. To model this, we assign a value V_{t,e_i} to selecting event e_i at time t. Notice that V represents the utility of selecting the corresponding event and is not a probability function. This utility is computed by summing the accumulated utility up to time t and the immediate normalized score of selecting event e_i. The utility is penalized if the event e_i is not feasible in the current state. The following recursive relations show how we compute the utility function.

$$\begin{aligned} &V_{1,e_i} = \textit{P}(e_i|w_1, s_0),\ S_{1,e_i} = \textit{Progress}(s_0, e_i) \\ &V_{t,e_i} = \textit{P}(e_i|w_t, s_{t-1}) + V_{t-1,e} + r_{s_{t-1},e_i} \\ &S_{t,e_i} = \textit{Progress}(s_{t-1}, e_i) \end{aligned} \quad (1)$$

where $s_0 = \textit{true}$, $e = \arg\max_{e_i \in E}(V_{t-1,e_i})$, $s_{t-1} = S_{t-1,e}$, and $r_{s,e_i} = \begin{cases} -1 & \textit{Precond}(e_i) \not\models s \\ 0 & \text{otherwise} \end{cases}$ is a function for penalizing infeasible events. Here V_{t,e_i} shows the value of selecting event e_i in the time step t. This value is initialized with the normalized score of events for the first sentence and the initial state. These scores are derived using the *ComputeP* procedure: $\textit{P}(e_i|w_1, s_0) = \textit{computeP}(e_i, w, s, \vec{\Theta})$. If the preconditions of e_i is not consistent with the current state s_{t-1} we penalize the value of choosing this event using a penalty function r_{s_{t-1},e_i} which is a real number between 0 and -1. In our experiments, we set this penalty function as -1 to penalize the events with higher scores more than the events with lower scores. The current state S_{t,e_i} is derived by *Progressing* state s_{t-1} with event e_i.

To update the best sequence of events, we use the following recursive formulas. $\textit{Seq}_{t,e_i}$ shows the best sequence of events if event e_i is selected at time t. The following recursive equations show how we model *Seq*.

$$\begin{aligned} &\textit{Seq}_{1,e_i} = [e_i] \\ &\textit{Seq}_{t,e_i} = \textit{Seq}_{t-1,e} + [e_i] \end{aligned} \quad (2)$$

where $e = \arg\max_{e_i \in E}(V_{t-1,e_i})$. The event sequence $\textit{Seq}_{t,e_i}$ is updated by keeping a pointer to the previously best selected event in the recursive step. Finally, the best sequence of events is derived as selecting the event at time T that has the highest value and backtrack recursively i.e., $e_T = \arg\max_{e_i \in E}(V_{T,e_i})$ and $e_{1..T-1} = \textit{Seq}_{e_T,T}$.

4 Experiments

In this section we evaluate our algorithm, ITEM, to map narratives to a sequence of events. We work with Robocup soccer commentaries. The task is to compute the accuracy of the mapped event sequence with respect to a gold standard event sequence. We compare the accuracy of our approach with baseline algorithms and state-of-the-art approach that uses annotated labeled data. Through our experiments we show that prior knowledge about event effect axioms alleviates the need for labeled data.

4.1 Robocup Soccer Commentaries

We use the Robocup soccer commentaries dataset (Chen, Kim, & Mooney 2010). The data is based on commentaries

of four championship games of Robocup simulation league in years 2001 to 2004. Each game is associated with a sequence of comments in English. There are in total of 1872 comments where 2001, 2002, 2003, and 2004 games have 672, 459, 398, and 343 comments, respectively. Figure 2 shows a sample sequence of sentences in the commentary of 2001 game.

The meaning representations corresponding to the commentaries are from the Robocup dataset. We add a *holding* fluent and a *Nothing* event to the list of events in (Chen, Kim, & Mooney 2010). In addition, we manually describe effect axioms for events. Events for the soccer commentary include actions with the ball or other game information. In total there are 16 event names among which 3 with two arguments, 4 with one argument, and 10 with zero arguments. The number of fluent names in the domain is 10. We assign constants as team names and player names. In total there are 24 constants.

In addition to the English comments about the game, the original dataset includes real events (represented in meaning representation language) that happen in the original game tagged with time. Our ITEM algorithm does not use this event log of the game. This makes us different from the approach in (Chen, Kim, & Mooney 2010). They use annotated labeled data in the form of a a mapping between natural language comment and the real events that occurred within 5 time steps of when the comment was recorded.

4.2 Mapping Sentences to Events

For evaluation purposes only, we use gold-standard labels in the dataset where each sentence is manually matched to the correct event. We evaluate the accuracy of the output sequence of events by computing the proportion of the events that have been correctly assigned to the sentences. We report the results for every game. We also report the micro-average accuracy over all the examples. For computing the micro-average accuracy we compute the weighted sum of the accuracies for each game given the number of sentences in the game. We compare the accuracy of our approach with (Chen, Kim, & Mooney 2010) and several baselines.

Our approach (ITEM) With respect to our learning algorithm, we train on three Robocup games (e.g., 2001, 2002, and 2003) and test on the last game (e.g., 2004). We run *IterTrain* with parameters N=10 (number of samples for each sentence) and compute the weight vector. The average number of training examples per iteration is about 700. To generate training examples, we use *Example Generator* module that samples events for sentences, ground events based on the arguments of the sentence (player names), and update the states based on the events. There are cases that we miss the arguments as for example the player name is mentioned in the form of "Pink Goalie" rather than "Pink1". There are cases that the sampled event requires more arguments than the player arguments extracted for the sentence. For example, sentence "He kicks the ball to P10" has one player argument $P10$. In these cases we use the last argument selected for the previous sentence as an argument of the event. Also, there are cases that the sentence has more arguments than the event. In this case, we consider different pairs of argument players as arguments of the events.

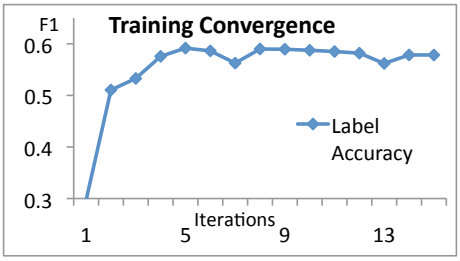

Figure 3: Convergence of the accuracy of estimated labels vs. number of iterations of training according to F_1 measure.

After generating examples, we extract features for each example. The length of feature vector is 250 including 16 events, 195 words, and 38 ground fluents. Initial labels are computed using prior knowledge about events and computing edit distance of the event name and words in the sentence. An example (e, w, s) gets positive label if e is feasible in state s and its name has low edit distance (≤ 3) to at least a word in the sentence. The labels are later changed in the next iterations using the learned classifier. Using the learned weight vector, we use the *ComputeP* subroutine to compute the score of every event associated with a sentence in the test narrative. Finally, we apply our inference subroutine and find the best event sequence for the test narrative.

We first show that our iterative learning approach converges and improves the accuracy of labels generated for the training examples. Graph 3 shows the convergence of our iterative method. It shows that the iterative step improves initial training labels. We report the expected accuracy of labels at each iteration in terms of F_1 measure over all four training scenarios.

Table 2 shows some examples of our right and wrong predicated labels. It shows that our approach can distinguish the meaning of sentence "Purple10 kicks to Purple11" by mapping it correctly to $pass$ rather than $kick$. This shows that our approach does not disambiguate based on the similarity of the event name and the verb name. Our wrong predicted labels show that distinguishing between $turnover$ and $badPass$ events is hard. $badPass$ refers to the event that the agent is trying to pass but the pass mistakenly goes to the next team's player. $turnover$ refers to the event that the player accidentally loses the ball. In addition, Table 1 shows the accuracy of the derived mapping by our approach.

Comparison to Baselines We compare our approach with different baselines. For each sentence, these baselines select events with specific properties that can be candidate interpretations of the sentences.

Uniform: The first baseline *Baseline-0* selects an event per sentence from a uniform distribution over events. This baseline shows how difficult the Robocup soccer commentary is, and as one can see from Table 1 this random selection performs poorly on the dataset.

Uniform+heuristics: The second group of baselines show how performing some heuristics for event selection helps. *Baseline-1a* selects uniformly among events whose

Correct Predicted Events

Sentence	event
Purple10 makes a quick pass to Purple11 on the side.	$pass(purple10, purple11)$
Purple11 tries to pass back but was picked off by Pink5.	$badPass(purple11, pink5)$
Pink5 made a bad pass that was intercepted by Purple10.	$badPass(pink5, purple10)$
Purple10 kicks to Purple11.	$pass(purple10, purple11)$
Purple11 threads a nice pass to Purple10 near the penalty area.	$pass(purple11, purple10)$

Wrong Predicted Events

sentence	correct event	our event
Pink6 steals the ball from Purple6.	$steal(pink6)$	$badPass(pink6, purple6)$
Pink6 tries to dribble toward the goal but turns the ball over to Purple3.	$turnover(pink6, purple3)$	$badPass(pink6, purple3)$
Purple10 tries to kick back to Purple11 but was intercepted by Pink2.	$badPass(purple10, pink2)$	

Table 2: *top* Some sentences in 2001 game and our responses that predicted the correct event. *bottom* Some examples in the dataset that was wrongly predicted by our approach.

Approach	2001	2002	2003	2004	Avg.
uniform+heuristics					
Baseline-1a	.688	.464	.628	.437	.574
Baseline-1b	.625	.570	.718	.720	.648
prior knowledge+heuristics					
Baseline-2a	.693	.455	.640	.454	.579
Baseline-2b	.629	.575	.826	.737	.677
Baseline-3	.687	.474	.658	.478	.590
prior knowledge (no labeled data) + our method					
Our ITEM	**.799**	.681	**.867**	.769	**.779**
annotated labeled data + WGIM					
WGIM	.721	.664	.683	.746	.703
prior knowledge+ annotated labeled data+WGIM					
WGIM-Inference	.767	**.721**	.638	**.798**	.734

Table 1: (*top*) Accuracy of different approaches for Robocup narratives and the micro-average accuracy. Our approach *ITEM* with no annotated data shows higher accuracy compared to other algorithms. *Baseline-0* uses uniform distribution for selecting events and returns accuracy of 0.062. *Baseline-1a* and *Baseline-1b* use heuristics for event selection. Heuristics include selecting similar events or equal arity events. *Baseline-2a* and *Baseline-2b* uses prior knowledge on top of the heuristics. *Baseline-3* combines all the possible heuristics and prior knowledge. *WGIM*[6] uses annotated data to select events.*WGIM-Inference* augments WGIM with prior knowledge about events. The results suggest that knowledge about event axioms together with iterative learning replaces the need of annotated labeled data and that prior knowledge improves learning with annotated data.

names have a small edit distance ($\leq$ 3) to at least one word in the sentence. We call these events *similar* events. *Baseline-1b* selects uniformly among events whose arity is equal to the number of arguments in the sentence. We call these events *same-arity* events. Table 1 show the results of these baselines. If no event has these properties, we randomly select among all the possible events. While the accuracy of these baselines is significantly lower than our ITEM algorithm, they show significant improvement over the random selection of *Baseline-0*.

Prior knowledge + Heuristics: The next baseline examines the effect of prior knowledge without learning. *Baseline-2a* uses prior knowledge about events over *similar* events. This baseline applies the *inference* subroutine (Algorithm 4) with a uniform distribution over *similar* events instead of using *ComputeP*. *Baseline-2b* applies the *inference* subroutine, but with a uniform distribution over *same-arity* events to the sentence. We apply prior knowledge during the *inference* subroutine over two previous heuristics. Our results suggest that prior knowledge helps, but it is important how to incorporate prior knowledge for event selection. Applying prior knowledge over uniform selection of arity events improves the accuracy up to 3%, but adding prior knowledge over uniform selection of *similar* events only improves the accuracy by 0.3%. Notice that still our ITEM approach has significantly higher accuracy compared to these baselines as it uses prior knowledge together with iterative learning.

Last baseline *Baseline-3* uses prior knowledge together with both heuristics. It selects *similar* and *same-arity* events for a sentence. If no event is selected it considers all the possible events. It then applies the *inference* subroutine with a uniform distribution over selected events. Surprisingly, the results of table 1 show that the accuracy of combining both heuristics together with prior knowledge is lower than the accuracy of arity selection with prior knowledge. This shows that the naive way of incorporating prior knowledge together with heuristics does not help.

Annotated Labeled data: We also compare our algorithm with the state-of-the-art approach, WGIM (Chen, Kim, & Mooney 2010). These results show that our iterative learning method alleviates the need of annotated labeled data collection.

In the next experiment, we augment *WGIM* approach with knowledge about events (*WGIM-Inference*). We apply *inference* where the transition model for every sentence is derived from *WGIM* approach. At each step, we select the event that has highest score according to *WGIM* and is feasible in the current state. The current state is updated according to the prior selected events in the sequence. Accuracy of WGIM-Inference is still lower than our ITEM approach. Because original WGIM does not learn the event scores given the current state. However, results show that adding prior knowledge to WGIM improves WGIM's accuracy.

5 Discussion and Future Work

In this paper we have introduced an approach to map Robocup-soccer narratives to sequences of events without any annotated labeled data. In this approach we show that

knowledge about event models together with a careful design of representation, inference, and iterative learning alleviates the need of annotated label data. We show that this iterative learning approach achieves superior accuracy compared to heuristics and state-of-the-art approach that use labeled data.

We show that by collecting prior knowledge about events we do not need to annotate every sentence in the domain. It is usually very hard to scale labeled data since we need to generate more labels to be able to understand larger texts. However, if we collect prior knowledge for a specific context, we can understand large texts in that context. Moreover, using prior knowledge we can represent semantics (useful, for example, for question answering). We plan to extend our approach to understand other narratives and answering questions about them. Specifically, we like to work with stories in Remedia corpus or Weblog stories in (M. Manshadi & Gordon 2008). In this setting, we plan to use VerbNet (Schuler 2005). VerbNet is a comprehensive verb lexicon that contains semantic information such as preconditions, effects, and arguments of about 5000 verbs. We plan to use this information and a noise event to cover all the remaining verbs in the domain. In this setting, we also plan to extend our bag-of-words model with syntactic parsing of the sentences.

One shortcoming of our approach is that we assume the input narrative is in a sequential form. It is possible that some events have been missed or the sentences are represented in partial order. We think of using probabilistic and partial order planning approaches to infer the event sequences.

We also plan to use an alternative approach of multi-class classification instead of binary classification. We cast the problem to sequence labeling where each element of the sequence corresponds to an individual sentence and the label to the predicted event. Then the goal is to learn a multinomial probability distribution over different events corresponding to a sentence and the state.

References

Baral, C., and Tuan, L. 2002. Reasoning about actions in a probabilistic setting. In *AAAI*.

Bishop, C. 2006. *Pattern Recognition and Machine Learning*. Springer, 1st edition.

Branavan, S.; Chen, H.; Zettlemoyer, L.; and Barzilay, R. 2009. Reinforcement learning for mapping instructions to actions. In *ACL-IJCNLP*, 82–90.

Bui, H. H. 2003. A general model for online probabilistic plan recognition. In *IJCAI*, 1309–1318.

Charniak, E., and Goldman, R. P. 1993. A bayesian model of plan recognition. *Artif. Intell.* 64(1):53–79.

Chen, D.; Kim, J.; and Mooney, R. 2010. Training a multilingual sportscaster: Using perceptual context to learn language. *JAIR* 37:397–435.

Deshpande, A.; Milch, B.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning probabilistic relational dynamics for multiple tasks. In *UAI*, 83–92.

Fikes, R., and Nilsson, N. 1971. Strips: a new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2:189–208.

Grosz, B., and Sidner, C. 1986. Attention, intention and the structure of discourse. *Journal of Computational Linguistics* 12.

Hajishirzi, H., and Amir, E. 2008. Sampling first order logical particles. In *UAI*.

Hobbs, J. R.; Stickel, M. E.; Appelt, D. E.; and Martin, P. 1993. Interpretation as abduction. *Artificial Intelligence* 63:69–142.

Johnson-Laird, P. 1983. *Mental Models*. Cambridge: Cambridge University Press.

Kate, R. J., and Mooney, R. J. 2007. Learning language semantics from ambiguous supervision. In *AAAI*, 895–900.

Kautz, H. 1987. *A formal theory of plan recognition*. Ph.D. Dissertation, Univ. of Rochester.

Liao, L.; Patterson, D. J.; Fox, D.; and Kautz, H. A. 2007. Learning and inferring transportation routines. *Artif. Intell.* 171(5-6):311–331.

M. Manshadi, R. S., and Gordon, A. 2008. Learning a probabilistic model of event sequences from internet weblog stroeis. In *FLAIRS*.

Majercik, S., and Littman, M. 1998. Maxplan: A new approach to probabilistic planning. In *Proceedings of the 5th Int'l Conf. on AI Planning and Scheduling (AIPS'98)*.

Rabiner, L. R. 1989. A tutorial on HMM and selected applications in speech recognition. *IEEE* 77(2).

Reiter, R. 2001. *Logical Foundations for Describing and Implementing Dynamical Systems*.

Riley, P., and Veloso, M. M. 2004. Advice generation from observed execution: Abstract markov decision process learning. In *AAAI*, 631–637.

Sadilek, A., and Kautz, H. 2010. Recognizing multi-agent activities from gps data. In *AAAI*.

Schuler, K. K. 2005. *Verbnet: a broad-coverage, comprehensive verb lexicon*. Ph.D. Dissertation. AAI3179808.

Vogel, A., and Jurafsky, D. 2010. Learning to follow navigational directions. In *ACL*.

Webber, B. L. 1978. *A Formal Approach to Discourse Anaphora*. Ph.D. Dissertation, Harvard. publ. Garland 1979.

Zettlemoyer, L., and Collins, M. 2009. Learning context-dependent mappings from sentences to logical forms. In *ACL-IJCNLP*, 976–984.

Zettlemoyer, L. S.; Pasula, H. M.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *AAAI*.