

Extracting STRIPS Representations of Actions and Events

Avirup Sil and Alexander Yates

Center for Data Analytics and Biomedical Informatics

Temple University

Broad St. and Montgomery Ave.

Philadelphia, PA 19122

{avirup.sil, yates}@temple.edu

Abstract

Knowledge about how the world changes over time is a vital component of common-sense knowledge for Artificial Intelligence (AI) and natural language understanding. Actions and events are fundamental components to any knowledge about changes in the state of the world: the states before and after an event differ in regular and predictable ways. We describe a novel system that tackles the problem of extracting knowledge from text about how actions and events change the world over time. We leverage standard language-processing tools, like semantic role labelers and coreference resolvers, as well as large-corpus statistics like pointwise mutual information, to identify STRIPS representations of actions and events, a type of representation commonly used in AI planning systems. In experiments on Web text, our extractor’s Area under the Curve (AUC) improves by more than 31% over the closest system from the literature for identifying the preconditions and add effects of actions. In addition, we also extract significant aspects of STRIPS representations that are missing from previous work, including delete effects and arguments.

1 Introduction

Common-sense knowledge about the changes in the state of the world over time is one of the most crucial forms of knowledge for an intelligent agent, since it informs an agent of the ways in which it can act upon the world. A recent survey of the common-sense knowledge involved in the recognizing textual entailment task demonstrates that knowledge about action and event semantics, in particular, constitutes a major component of the knowledge involved in understanding

natural language (LoBue and Yates, 2011). This knowledge is also vital for central AI tasks like planning, plan recognition (Kautz, 1991; Geib and Steedman, 2007), and dialogue processing (Carberry, 1990; Litman and Allen, 1987).

In this paper we explore text mining approaches to extracting common-sense knowledge about action and event semantics. Our previous approach (Sil et al., 2010) (henceforth, S10) identifies the preconditions and effects of actions. We describe how we extend S10’s approach by identifying additional kinds of effects; by connecting this knowledge to an external ontology and generalizing the preconditions and effects; and by identifying argument variables for each predicate. Our experiments show that our novel extractor can identify the fully-formed STRIPS representations of actions with precision 0.73 and recall 0.72, and it improves on S10’s AUC for tasks that both systems can handle by over 31%.

The next section discusses previous work. Section 3 introduces STRIPS representation and the challenges involved in extracting such representations. Section 4 details our extraction techniques. Section 5 presents our experiments. Section 6 concludes and discusses future work.

2 Previous Work

The most closely related work has investigated how to extract “scripts” or “narrative event schemas” (Chambers and Jurafsky, 2009) — sets of events that often occur together. Schank and Abelson’s (1977) famous example of a restaurant script includes events such as sitting down, ordering, eating, and paying the bill. Script knowledge is distinct from STRIPS representations in that a script relates one event e to a subsequent event e' , whereas STRIPS relates an event e to a state of the world s before or after e . Our extracted knowledge could complement the standard restaurant script, for example, with knowl-

edge that `is hungry(diner)` is true before the diner eats, and `¬is hungry(diner)` is true afterwards. Neither of these statements constitutes an event in a script, but they do fall into the STRIPS paradigm.

Other research into extracting the relationships between events has investigated causal relationships (Girju, 2003) and, more generally, paraphrases, such as in the DIRT system (Lin and Pantel, 2001). Such systems typically do not distinguish between event-event relationships that appear in scripts — *e.g.*, a flooding event e_2 can follow a raining event e_1 — and event-state relationships — *e.g.*, `is wet(grass)` follows a raining event e_1 . Our system is focused only on the latter: we are concerned how the state of the world changes with the occurrence of an event rather than how one event influences another event. Furthermore, existing systems do not consider precondition relationships, which are neither causal nor paraphrases, and which are central to AI representations of actions and events.

Extracting and representing selectional preferences has attracted significant attention recently, especially using latent-variable probabilistic models like Latent Dirichlet Allocation (Ritter et al., 2010). Preconditions are a more general type of restriction on the arguments to actions than selectional preferences — *e.g.* `asleep(x)` is a precondition to `awaken`, but would not be considered a selectional preference because it does not constitute a class or type, but rather a property, of x .

3 STRIPS Representations

3.1 Background and Terminology

We define *actions* as observable phenomena, or *events*, that are brought about by rational agents. Because actions and events are central to AI, there is a long history of work in representing their semantics. One of the best-known, and still widely used, representations for action semantics is the STRIPS representation (Fikes and Nilsson, 1971); two examples of STRIPS representations are given in Figure 1. We use STRIPS to represent both actions and events. Formally, a STRIPS representation is a 5-tuple $(a, args, pre, add, del)$ consisting of the action name a , a list $args$ of argument variables that range over the set of objects in the world, and three sets of predicates that reference the argument variables. The first, the *precondition* list pre , is a set of conditions

		awaken	insert
STRIPS	args:	x	o, p
	pre:	$asleep(x)$	$object\#1(o),$ $opening\#1(p),$ $\neg in(o, p)$
	add:	$awake(x)$	$in(o, p)$
	del:	$asleep(x)$	$\neg in(o, p)$
S10	pre:	$asleep$	$person, slot$
	add:	$awake$	in

Figure 1: Two example STRIPS representations (above), and corresponding examples of the representation extracted in our prior work, S10 (below). In contrast with S10, the STRIPS representations require extracting delete effects and resolving coreference relationships among arguments to predicates. Also, our version of STRIPS uses WordNet synsets to unambiguously specify predicate names.

that must be met in order for the action to be allowed to take place. For instance, in order for someone to *awaken*, she or he must first be *asleep*. The other two sets of conditions specify how the world changes when the action takes place: the *add* list describes the set of new conditions that must be true afterwards (*e.g.*, after the event `insert(pencil24, sharpener3)`, `in(pencil24, sharpener3)` holds true), and the *del* list specifies the conditions that were true before the action happened but are no longer true. These *add* and *del* conditions are sometimes collectively referred to as *effects* or *postconditions*.

Formally, the precondition, add, and delete lists correspond to a set of rules describing the logical consequences of observing an event. To describe these rules, we assume a representation of the world grounded in a logical form, such as situation logic (Barwise, 1989) or episodic logic (Schubert and Hwang, 2000). For simplicity, we represent the passage of time by discrete time points t , together with a temporal-ordering relation $after(t_1, t_2)$. This is the same notion of time traditionally adopted by AI planning systems, although recent work has gone into elaborating this representation (Bresina et al., 2002; Younes et al., 2003). A set of constants identify

the *objects* that exist in the world, and at each time point, a set of logical *predicates* describes the *state* of the world at that time, for instance $\text{on}(\text{book1}, \text{shelf4}, t_9)$.

Let t_1 be the time point immediately preceding an event e with arguments \mathbf{args} , t_2 the time of event e , and t_3 the time immediately following e . For each precondition p , each add effect a , and each delete effect d , the following rules hold:

$$\begin{aligned}\forall_{\mathbf{args}} e(\mathbf{args}, t_2) &\Rightarrow p(\mathbf{args}_p, t_1) \\ \forall_{\mathbf{args}} e(\mathbf{args}, t_2) &\Rightarrow a(\mathbf{args}_a, t_3) \\ \forall_{\mathbf{args}} e(\mathbf{args}, t_2) &\Rightarrow \neg d(\mathbf{args}_d, t_3)\end{aligned}$$

where \mathbf{args}_x represents the subset of the arguments to which the predicate x applies. Finally, we assume a second-order *frame axiom* that states that unless explicitly updated by an event’s effects, predicates that were true (false) before an event remain true (false) afterwards.

3.2 Problem Formulation: STRIPS Extraction

The STRIPS extraction task takes as input a word or phrase e naming a type of event, like `insert`, and a large collection D of documents that mention the action at least once. As output, systems produce a STRIPS representation of the event: the argument list for the event; three sets of predicates representing the preconditions, add effects, and delete effects; and for each predicate, the list of variables that the predicate applies to.

This problem formulation is a first step towards extracting knowledge of dynamics, although it certainly does not cover the full scope of the problem. For instance, we do not attempt to extract representations for durative or repetitive events, or actions like `escalate` or `accelerate` that change quantities or numerical attributes. Furthermore, we restrict our attention in this paper to extracting predicates with only a single argument. Despite the restrictions from the full problem of extracting knowledge of dynamics, our problem formulation involves a number of difficult technical challenges which together constitute a substantial extraction problem.

3.3 Challenges

Word sense ambiguity, synonyms, and syntactic ambiguity plague our system, as they do all extraction systems, but in contrast to S10 we expect our extractor to identify sense-disambiguated

entries in an ontology for predicates, rather than ambiguous terms. Hence, we want to extract *liquid#3* (fluid matter having no fixed shape but a fixed volume) in Wordnet (Fellbaum, 1998) as a precondition for action *boil* as opposed to *liquid#4* (a frictionless continuant that is not a nasal consonant). Like the KNOWITALL system and related Web IE systems (Etzioni et al., 2005; Downey et al., 2005), we rely on the redundancy inherent in large document collections to help address these issues. In addition, we face these challenges:

Lack of Explicitly Stated Knowledge: Commonsense knowledge, like preconditions and postconditions of events, is often taken for granted by the author and reader, and thus does not need to be stated explicitly. Our biggest challenge is to create a system that can extract this knowledge even though it is never stated explicitly.

Temporality: Our patterns must distinguish between implications that are true before an event vs. after an event.

Generalization: The most common example of a cut event in text may be of a scissors cutting paper, but we do not want to conclude from these examples that scissors and paper are preconditions for cutting. Instead, some larger class of objects, like the set of sharp objects, is a better description of the precondition for the cutting instrument. Unlike S10, we expect a STRIPS extractor to extract appropriately-generalized predicates.

Rule Extraction: Like the DIRT system (Lin and Pantel, 2001), a STRIPS extraction system must identify rules rather than grounded facts. Instead of discovering $\text{asleep}(\text{person1})$, we want to discover patterns like $\forall_{x,t_1,t_2} \text{awaken}(x, t_2) \wedge \text{after}(t_2, t_1) \Rightarrow \text{asleep}(x, t_1)$. In contrast, S10 does not identify predicate arguments, which enable the use of preconditions and effects as inference rules.

4 Extraction Methods

4.1 Extracting Preconditions and Add Effects

Our previous system, S10 identifies the names of preconditions and add effects. We briefly review S10’s approach here.

Given a corpus where each document contains an event e , S10 begins by identifying relations and arguments in a large text corpus using an open-

domain semantic role labeler (Huang and Yates, 2010) and OpenNLP’s noun-phrase coreference resolution system¹. Taking a set of candidate predicate words, we then define different features of the labeled corpus that measure the proximity in the annotated corpus between a candidate word and the action word. Using a small sample of labeled action words with their correct preconditions and effects, we then train an RBF-kernel Support Vector Machine (SVM) to rank the candidate predicate words by their proximity to the action word.

S10 use three different types of features for measuring proximity: first, we compute the point-wise mutual information (PMI) (Turney, 2002) between the event e and the candidate word c using the document set D . For any set of words W , let D_W represent the set of documents containing all words in W .

$$PMI(e, c) = \log \frac{|D_{\{e,c\}}|}{|D_{\{e\}}||D_{\{c\}}|} \quad (1)$$

Second, we compute the three-way PMI between e , c , and discriminator features f :

$$PMI(e, c, f) = \log \frac{|D_{\{e,c,f\}}|}{|D_{\{e\}}||D_{\{c\}}||D_{\{f\}}|} \quad (2)$$

By using discriminator features f like `before` and `requires`, these three-way PMI features can measure if e and c relate to one another in a way that is indicative of preconditions, in particular. Likewise, discriminator features like `after` and `causes`, can measure whether c relates to e in the manner of an effect. In practice, approximately 200 discriminator features for preconditions and 200 for add effects are selected using greedy, χ^2 feature selection.

The third kind of feature for measuring proximity between e and c relies on semantic role and coreference annotations. For instance, one such measure counts how often c occurs as an argument to a predicate e , as indicated by the semantic role annotations. Another feature counts how often c corefers with an argument to a predicate e , and another counts how often c appears within a window of text near a predicate e . See S10 for full details on these features.

4.2 Connecting Extractions to an Ontology

One obvious shortcoming of the S10 system is that it fails to generalize adequately. For instance,

¹<http://opennlp.sourceforge.net>

s	CW_s
nurse#1	{nurse}
doctor#1	{doctor,allergist}
health_prof.#1	{doctor,nurse,allergist}
person#1	{doctor,nurse,poet,...}

Table 1: Sample candidate preconditions from CS for action ‘heal’, together with the set of words in the corpus for ‘heal’ that have the candidate synset as a hypernym.

the system extracts `hammer` as a precondition for the action `crush`. While it is true that if one has a hammer, then one can crush things, this is too strict of a precondition. Using this incorrect knowledge, a system might conclude from the text “Jane crushed the soda can with her hands” that *hands* are a kind of *hammer*.

Our first extension to the baseline S10 system is to give it the capacity to generalize the predicates it finds, by giving it more general candidate predicates. Let $\text{synsets}(w)$ denote the set of WordNet synsets for a word w , and let CW be the set of candidate words used by S10. For each $c \in CW$, we add each $s \in \text{synsets}(c)$ to a new candidate predicate list of synsets CS ; if c does not appear in WordNet, we add c itself to CS . We then add all direct and indirect hypernyms of the synsets in CS to CS . In Table 1, we show a sample of the candidate preconditions s from CS for action `heal`. We also show the subset CW_s of words from CW that have s as a hypernym.

Our second extension to S10 is to modify the definition of our features so that they apply to the synsets in CS rather than the words in CW . To compute the PMI-based features, we set $|D_{\{s\}}|$ to $|D_{CW_s}|$, and $|D_{\{e,s,f\}}|$ to be $|D_{\{e,f\}} \cap D_{CW_s}|$. For semantic role-based features, let $F(e, c)$ denote one of the counts we compute for candidate word c and event e . For hypernyms, we change this to $F(e, s) = \sum_{c \in CW_s} F(e, c)$. We refer to S10 with the new candidates CS and the modified features as S10’.

Correctly ranking the elements of CS is significantly harder than ranking CW (the problem for S10), because the new list has far more elements — multiple synsets and hypernyms for each element of CW . The feature set in the S10’ system is unable to handle these new challenges. In particular, S10’ tends to choose overly

feature	description
root-dist	1. $\max_{r \in R} d(s, r)$ 2. $\min_{r \in R} d(s, r)$ 3. $\frac{\sum_{r \in R} d(s, r)}{ R }$
max-dist	$\max_{c \in CW_s} \min_{s' \in \text{synsets}(c)} d(s, s')$
avg-dist	$\frac{\sum_{s' \in \text{synsets}(c) c \in CW_s} d(s, s')}{ CW_s }$
weighted dist	$\frac{\sum_{c \in CW_s, s' \in \text{synsets}(c)} d(s, s') C(c)}{\sum_{c \in CW_s} C(c)}$

Table 2: Features added to S10' to create HYPER.

general hypernyms far too often. For example, synsets like `physical_entity#1` tend to rank highly as preconditions and add effects according to S10', as many words in CW are hyponyms of `physical_entity#1`, and thus this synset has high scores for count and PMI-based features.

To compensate, we include several new features that measure the generality of hypernyms. Table 2 lists the new features we add to S10' to create our new extractor, which we call the HYPER model. Here, $d(s, s')$ is the distance between s and s' , or the number of hyponym relationships separating s and s' ; R is the set of root nodes in the WordNet hierarchy; and $C(w)$ is the frequency of word w in our corpus. The first three features calculate the maximum, minimum and average distance separating s and any root node of the WordNet hierarchy. The second and third features find the maximum and average distance between s and the terms in CW_s . The final feature computes a weighted distance between s and the elements $c \in CW_s$, where each weight is the frequencies of c . Each of these features helps to differentiate between very general synsets and more specific synsets (or synsets for terms appearing frequently in the corpus). Adding these features to HYPER allows the SVM to balance between candidate synsets that score highly on the standard S10' features and candidate synsets that are less general.

4.3 Detecting Delete Effects

S10' and HYPER can identify preconditions and add effects, but they do not handle delete effects. We extend the system with a separate extractor for delete effects. By far the most common kind

feature	description
prefix	1 if $p = \{\text{un-,im-,in-}\}$ concatenated with an add effect
loose count	a separate feature $ D_{\{\text{neg}, p, f\}} $ for each $\text{neg} \in \{\text{"no"}, \text{"not"}\}$ and each $f \in \{\text{"after"}, \text{"during"}, \text{"as"}, \text{"before"}\}$
strict count	for each neg and f , $ D_{\{\text{"neg } p f\}} $
simple PMI	for each neg , $\text{PMI}(\text{neg}, p)$ and $\text{PMI}(\text{"neg } p", e)$
ratio PMI	for each neg , $\frac{\text{PMI}(\text{"neg } p", e)}{\text{PMI}(p, e)}$

Table 3: Features for classifying whether a precondition predicate p is a delete effect of an event e .

of delete effect is one that falsifies a precondition predicate: *e.g.*, before someone puts a book down, they are holding the book, and afterwards they are not. So far, we have restricted our attention to this common case, although more general extractors are possible for conditional delete effects, which falsify a predicate only on the condition that the predicate was true before the event.

We create a binary SVM classifier that predicts for each precondition predicate whether or not the precondition turns false after the event. For each precondition predicate p , we construct features that measure how strongly p is associated with negation in the context of the event e . We include a mix of orthographic features, count features, and PMI-based features. The full set of our features for this classifier is listed in Table 3.

As an example of the delete effects classifier in action, consider the event `maim`. HYPER can extract `unhurt` as a precondition and `hurt` as an add effect. In general, whenever we see an add effect that contradicts a precondition, we expect to delete the precondition. The `prefix` feature in Table 3 for `maim` flags `unhurt` as a possible precondition to be deleted because it matches 'un' + add effect `hurt`.

4.4 Determining Arguments

The last subtask for our STRIPS extractor is to "relation-ify" our extracted representation by assigning arguments to the event e and each predicate. S10 makes no attempt to identify arguments to extracted predicates. As a result, for action

awaken, the S10 representation does not distinguish between a case where one entity x is asleep and another entity y wakes up, and the case where x is asleep and then x awakens.

This is a complex, structured-prediction problem involving coreference resolution between the arguments to extracted relationships. As a first attempt, we resort to an effective heuristic solution. We use the argument role labels supplied by our propbank-style semantic role labeler as candidate variables for our representation. For an extracted predicate p for e , we assign arguments to p based on the semantic role label or labels with which it is most commonly associated in the annotated corpus. That is, for each possible semantic role r , we count how often p occurs in a phrase that is an argument to e and is annotated with role r . We also count how often p occurs as part of any phrase that is annotated with role r . Let $score(e, r, p)$ denote the sum of these two counts. We choose an argument variable $r^* = \arg \max_r score(e, r, p)$, and write p as the predicate $p(r^*)$. Finally, we set the arguments of e to be the set of unique arguments chosen for all of its extracted predicates.

Figure 2 shows an example of this technique and two baselines. The input to each system is a STRIPS representation without arguments and the output adds arguments. For action `maim`, the semantic role heuristic finds that `person#1` and `unhurt#1` occur most often in phrases marked with a propbank A1 role. Hence, it concludes that they both should have the same argument label. `object#1` occurs more in phrases with A2 roles, and is given a separate argument variable as a result. These two roles then constitute the argument set for event `maim`.

5 Experiments

5.1 Experimental Setup

We use the same experimental setup as in S10: we use the dataset of 40 actions from the lexical units in the frames that inherit from the `Transitive_action` frame in FrameNet (Johnson et al., 2003). We use the same document collection of 15,088 documents that we downloaded from the Web for these 40 action words. For each action word, candidate predicates for precondition and add effect extraction were the top 500 words ranked by PMI with the action word. This list was augmented with the superclasses from WordNet, as described above. For

	<u>action</u>	<u>pre</u>	<u>add</u>
S10' :	<code>maim</code>	<code>person#1</code> <code>unhurt</code> <code>object#1</code>	<code>hurt</code>
Distinct var. baseline:	<code>maim(a,b,c,d)</code>	<code>person#1(a)</code> <code>unhurt(b)</code> <code>object#1(c)</code>	<code>hurt(d)</code>
Same var. baseline:	<code>maim(a)</code>	<code>person#1(a)</code> <code>unhurt(a)</code> <code>object#1(a)</code>	<code>hurt(a)</code>
Semantic Role heuristic:	<code>maim(A1,A2)</code>	<code>person#1(A1)</code> <code>unhurt(A1)</code> <code>object#1(A2)</code>	<code>hurt(A1)</code>

Figure 2: Addition of arguments to predicates for action ‘maim’.

each action word, we hand-constructed a STRIPS representation (we did not use S10’s labeled data because it did not include the WordNet superclasses as candidate words, or as part of its hand-constructed representations). On average, our labeled data had 2.6 preconditions, 0.8 add effects, 0.5 delete effects, and 3 argument variables per action word. In all of our extraction experiments, we take care to test the extractors on different action words from the ones on which they are trained (for any components that require training), so that results should generalize to new action words beyond the ones in our current collection.

5.2 Results and Discussion

Our first experiment compared predicate extraction (preconditions and add effects) between S10’ and HYPER. We use 5-fold cross-validation, with each run training on 32 action words and testing on the remaining 8. The training data consists of action words, candidate words, feature values, and a +1 label for candidates matching our hand-constructed representation, and -1 for those that did not match. We train a regression model, so that our SVMs produce real-valued predictions for (action word, candidate word) pairs. We construct a list of all such pairs and rank them according to the SVM output. Figure 3 shows our results. The area under the curve (AUC) for both preconditions and add effects is significantly higher (0.34 improvement in AUC for preconditions, 0.17 for add effects) for the full model, largely because the S10’ model ranks very general WordNet classes, like `physical_entity`, very highly for most action words, simply because they appear so often as the

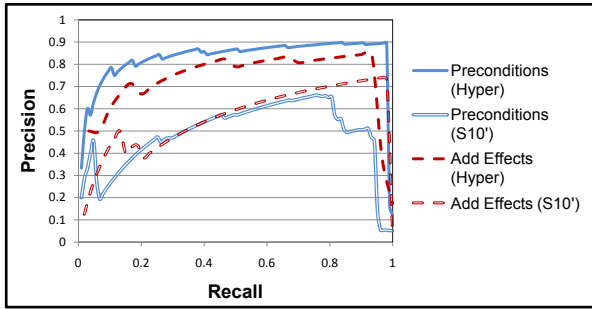


Figure 3: Precision-Recall curves for extracting preconditions and add effects.

superclasses of words in the documents. By incorporating the features that measure the generality of classes, the full extraction model can learn to rank these very general classes much lower, except when strongly supported by evidence from the documents. The absolute performance of the full extractor is quite strong, with AUC 0.82 for preconditions and 0.72 for add effects, compared with AUCs for S10' of 0.48 for preconditions and 0.52 for add effects.

We measured the performance of our delete effect extractor using the same 5-fold cross-validation setup. Recall that our delete classifier predicts which preconditions become false after the action. To separate the evaluation of this classifier's performance from our precondition extractor, we use gold-standard preconditions as input to the classifier. As before, we construct train and test sets consisting of the action word, the precondition, values for our features, and a label of +1 if the precondition is in fact a delete effect, and -1 otherwise. We train an SVM classifier, and measure its precision and recall on detecting true delete effects for each of the five folds. Table 4 shows the results for extracting this kind of knowledge. The average precision across the folds was 72.2%, and recall was 52.6%, for an F1 of 60.8. In contrast, a baseline that predicts all preconditions are also delete effects achieves an F1 of 41.3 (26% precision, 100% recall), and other baselines (random, no preconditions are delete effects) performed worse. Thus, the delete effect classifier is able to reliably detect negative knowledge, which is rarely stated explicitly, using co-occurrence statistics and other simple features.

For argument matching, we measured performance by the overall quality of the extracted STRIPS representations, including arguments. We first computed a maximal matching

Technique	Prec.	Recall	F1
All pre. are deleted	26	90	40.3
No pre. are deleted	100	10	18.2
SVM trained model	72.2	52.6	60.8

Table 4: Precision and recall for our system which extracts delete effects. The final SVM trained model has gold standard preconditions as input to the classifier. For an action with no delete effects, if the system predicts no delete effects, we judged precision and recall to be 100%, which is why the recall of the second baseline is nonzero. Precision and recall numbers are macro-averaged across actions.

between the argument variables selected by our method and the argument variables in the hand-constructed STRIPS representation. After computing the matching, we substituted the variables from the gold standard representation into the automatically-produced variables. We then measured the quality of our automatically-generated full STRIPS representation by measuring how many of the predicted predicates match exactly a predicate in the gold standard (precision), and how many of the gold standard predicates were found exactly in the automatically-generated representations (recall). For the purposes of this calculation, we used the top 3 automatically-generated preconditions and top 1 automatically-generated add effect per action word according to the HYPER extractor, regardless of the numeric scores for each predicate. (We found that recall increased but precision dropped more when we included a second add effect per action word.) We did not include delete effects in this experiment. We compared our heuristic technique to two baselines, one which predicts that all extracted predicates for an action share the same variable, and one which treats every argument as a distinct variable. Table 5 shows our results. The semantic role labeling heuristic improves dramatically over the closest baseline by 25 points in F1. Overall, our complete extraction system found precondition and add effect predicates and arguments for STRIPS representations with an F1 of 0.72, using only statistics over a small corpus collected from the Web and a small set of hand-labeled examples.

Technique	Prec.	Recall	F1
All preds. have same var.	32	33	32
Each pred. has distinct var.	56	58	57
Semantic role heuristic	73	72	72

Table 5: Precision and recall of our complete representation with extracted predicates and arguments.

6 Conclusion and Future Work

We have presented a system for extracting a complete STRIPS representation of 40 common actions from text, with an overall F1 of 0.72. We demonstrate that our system significantly outperforms the closest comparable one from the literature and extracts richer representations. Future directions include extracting more sophisticated representations of action semantics, especially multi-argument predicates and logical connectives between predicates, and extracting representations for more complex actions, like durative or repetitive actions.

References

- Jon Barwise. 1989. *The Situation in Logic*. CSLI.
- John Bresina, Richard Dearden, Nicolas Meuleau, David Smith, and Rich Washington. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. of the 18th Conference on Uncertainty in Artificial Intelligence*.
- Sandra Carberry. 1990. *Plan Recognition in Natural Language Dialogue*. MIT Press, Cambridge, MA, USA.
- Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of ACL-IJCNLP 2009*.
- Doug Downey, Oren Etzioni, and Stephen Soderland. 2005. A Probabilistic Model of Redundancy in Information Extraction. In *IJCAI*.
- O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134.
- Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.
- R. Fikes and N. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208.
- Christopher W. Geib and Mark Steedman. 2007. On natural language processing and plan recognition. In *IJCAI*, pages 1612–1617.
- Roxana Girju. 2003. Automatic detection of causal relations for question answering. In *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering*, pages 76–83, Morristown, NJ, USA.
- Fei Huang and Alexander Yates. 2010. Open-domain semantic role labeling by modeling word spans. In *ACL*.
- Christopher Johnson, Miriam Petruck, Collin Baker, Michael Ellsworth, Josef Ruppenhofer, and Charles Fillmore. 2003. *FrameNet: Theory and practice*.
- Henry A. Kautz. 1991. A formal theory of plan recognition and its implementation. In *Reasoning about plans*, pages 69–124. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- D. Lin and P. Pantel. 2001. DIRT – Discovery of Inference Rules from Text. In *KDD*.
- Diane J. Litman and James F. Allen. 1987. A plan recognition model for subdialogues in conversations. *Cognitive Science*, 11(2):163 – 200.
- Peter LoBue and Alexander Yates. 2011. Types of common-sense knowledge needed for recognizing textual entailment. In *ACL*.
- A. Ritter, Mausam, and Oren Etzioni. 2010. A latent dirichlet allocation method for selectional preferences. In *ACL*.
- R.C. Schank and R.P. Abelson. 1977. *Scripts, plans, goals and understanding: an inquiry into human knowledge structures*. Erlbaum.
- Lenhart K. Schubert and Chung Hee Hwang. 2000. Episodic Logic meets Little Red Riding Hood: A comprehensive natural representation for language understanding. In Lucja Iwanska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 111–174. MIT/AAAI Press.
- Avirup Sil, Fei Huang, and Alexander Yates. 2010. Extracting action and event semantics from web text. In *AAAI Fall Symposium on Common-Sense Knowledge (CSK)*.
- P. D. Turney. 2002. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Procs. of ACL*, pages 417–424.
- Håkan L. S. Younes, David J. Musliner, and Reid G. Simmons. 2003. A framework for planning in continuous-time stochastic domains. In *AAAI*.