A Domain-independent Rule-based Framework for Event Extraction

Marco A. Valenzuela-Escárcega Gus Hahn-Powell Thomas Hicks Mihai Surdeanu
University of Arizona, Tucson, AZ, USA
{marcov,hahnpowell,msurdeanu,hickst}@email.arizona.edu

Abstract

We describe the design, development, and API of ODIN (Open Domain INformer), a domain-independent, rule-based event extraction (EE) framework. The proposed EE approach is: *simple* (most events are captured with simple lexico-syntactic patterns), *powerful* (the language can capture complex constructs, such as events taking other events as arguments, and regular expressions over syntactic graphs), *robust* (to recover from syntactic parsing errors, syntactic patterns can be freely mixed with surface, token-based patterns), and *fast* (the runtime environment processes 110 sentences/second in a real-world domain with a grammar of over 200 rules). We used this framework to develop a grammar for the biochemical domain, which approached human performance. Our EE framework is accompanied by a web-based user interface for the rapid development of event grammars and visualization of matches. The ODIN framework and the domain-specific grammars are available as open-source code.

1 Introduction

Rule-based information extraction (IE) has long enjoyed wide adoption throughout industry, though it has remained largely ignored in academia, in favor of machine learning (ML) methods (Chiticariu et al., 2013). However, rule-based systems have several advantages over pure ML systems, including: (a) the rules are interpretable and thus suitable for rapid development and domain transfer; and (b) humans and machines can contribute to the same model. Why then have such systems failed to hold the attention of the academic community? One argument raised by Chiticariu et al. is that, despite notable efforts (Appelt and Onyshkevych, 1998; Levy and Andrew, 2006; Hunter et al., 2008; Cunningham et al., 2011; Chang and Manning, 2014), there is not a standard language for this task, or a "standard way to express rules", which raises the entry cost for new rule-based systems.

Here we aim to address this issue with a novel event extraction (EE) language and framework called ODIN (Open Domain INformer). We follow the simplicity principles promoted by other natural language processing toolkits, such as Stanford's CoreNLP, which aim to "avoid over-design", "do one thing well", and have a user "up and running in ten minutes or less" (Manning et al., 2014). In particular, our approach is:

Simple: Taking advantage of a syntactic dependency[1] representation (de Marneffe and Manning, 2008), our EE language has a simple, declarative syntax (see Examples 1 & 2) for *n*-ary events, which captures single or multi-word event predicates (`trigger`) with lexical and morphological constraints, and event arguments (e.g., `theme`) with (generally) simple syntactic patterns and semantic constraints.

Powerful: Despite its simplicity, our EE framework can capture complex constructs when necessary, such as: (a) recursive events[2], (b) complex regular expressions over syntactic patterns for event arguments. Inspired by Stanford's Semgrex[3], we have extended a standard regular expression language to describe patterns over directed graphs[4], e.g., we introduce new < and > operators to specify the direction of edge traversal in the dependency graph. Finally, we allow for (c) optional arguments[5] and multiple arguments with the same name.

Robust: To recover from unavoidable syntactic errors, SD patterns (such as the ones in Examples 1 and 2) can be can be freely mixed with surface, token-based patterns, using a language inspired by the Allen Insti-

[1] Hereafter abbreviated as SD.

[2] Events that take other events as arguments (see Figure 1 and the corresponding Example (2) for such an event in the biochemical domain. The `Positive_Regulation` takes a `Phosphorylation` event as the `Controlled` argument)

[3] `nlp.stanford.edu/software/tregex.shtml`

[4] Here we use syntactic dependencies.

[5] `cause` in Example 1.

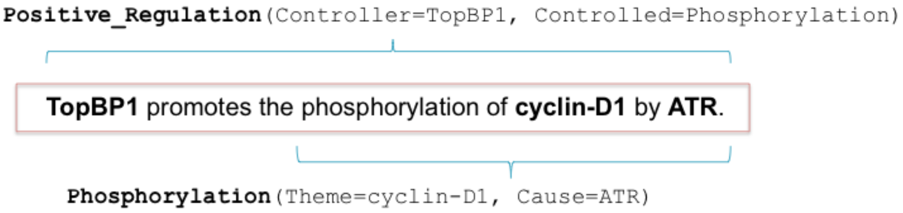

Positive_Regulation(Controller=TopBP1, Controlled=Phosphorylation)
TopBP1 promotes the phosphorylation of cyclin-D1 by ATR.
Phosphorylation(Theme=cyclin-D1, Cause=ATR)

Figure 1: An example sentence containing a recursive event.

tute of Artificial Intelligence's Tagger[6]. These patterns match against information extracted in our text processing pipeline[7] , namely a token's part of speech, lemmatized form, named entity label, and the immediate incoming and outgoing edges in the SD graph. Example 3 shows an equivalent rule to the one in Example 1 using surface patterns (i.e. a pattern that is independent of a token sequence's underlying syntactic structure).

Fast: Our EE runtime is fast because our rules use event trigger phrases, captured with shallow lexico-morphological patterns, as starting points. Only when event triggers are detected is the matching of more complex syntactic patterns for arguments attempted. This guarantees quick executions. For example, in the biochemical domain (discussed in Section 2), our framework processes an average of 110 sentences/second[8] with a grammar of 211 rules on a laptop with an i7 CPU and 16GB of RAM.

2 Building a Domain from Scratch

We next describe how to use the proposed framework to build an event extractor for the biochemical domain (Ohta et al., 2013) from scratch.

Rule-based systems have been shown to perform at the state-of-the-art for event extraction in the biology domain (Peng et al., 2014; Bui et al., 2013). The domain, however, is not without its challenges. For example, it is not uncommon for biochemical events to contain other events as arguments. Consider the example sentence in Figure 1. The sentence contains two events, one event referring to the biochemical process known as phosphorylation, and a recursive event describing a biochemical regulation that controls the mentioned phosphorylation. We will introduce a minimal set of rules that capture these two events. Here, we will assume the simple entities (denoted in bold in Figure 1) have already been detected through a named entity recognizer.[9]

When a rule matches, the extracted token spans for trigger and arguments, together with the corresponding event and argument labels (here the event

```
1  - name: Phosphorylation_1
2    priority: 2
3    label: [Phosphorylation, Event]
4    pattern: |
5      trigger = [lemma="phosphorylation"]
6      theme:PhysicalEntity = prep_of
7          (nn|conj|cc)*
8      cause:PhysicalEntity? = prep_by
9          (nn|conj|cc)*
```

Example 1: An example of a rule using syntactic structure. For the phosphorylation event, our selected event `trigger` (LINE 5) is a nominal predicate with the lemma *phosphorylation*. This trigger serves as the starting point for the syntactic patterns that extract event arguments. When searching for a `theme` to the Phosphorylation event, we begin at the specified `trigger` and look for an incoming dependent that is the object of the preposition *of*. The pattern fragment `(nn|conj_and|cc)*` targets entities that appear as modifiers in noun phrases (e.g., . . . *of the* **cyclin-D1** *protein*), or a series of arguments in a coordinated phrase. The entity mention associated with our `theme` must be a named entity with the label `PhysicalEntity` (LINE 7), a hypernym of several more specialized types identified in an earlier iteration. The `cause` argument is marked as optional (denoted by the `?` symbol).

label is `Phosphorylation`, and the argument labels are `theme` & `cause`) are dispatched to a labeling action. By default, these actions simply create an `EventMention` Scala object with the corresponding event label, and the extracted named arguments. Example 5 summarizes the `EventMention` class. Custom actions may be defined as Scala code, and be attached to specific rules. For example, a custom action may trigger coreference resolution when a rule matches a common noun, e.g., *the protein*, instead of the expected named entity.

The second rule, shown in Example 2, captures the recursive event in Figure 1. Importantly, this rule takes other events as arguments, e.g., the `controlled` argument must be an event mention, here generated by the rule in Example 1. To guarantee correct execution, the runtime repeatedly applies the given EE grammar on each sentence until no rule matches. For example, here the rule in Example 2 would not match in the first

[6] https://github.com/allenai/taggers

[7] https://github.com/sistanlp/processors

[8] after the initial text processing pipeline

[9] Though the discussion focuses on event extraction, our framework can also be applied to the task of entity recognition.

```
- name: Positive_regulation_1
  label: [Positive_regulation, Event]
  priority: 3
  pattern: |
    trigger =
        [lemma=/promot|induc|increas
        |stimul|lead|enhanc|up-regulat/
        & tag=/^V|RB/]
    controller:PhysicalEntity = nsubj
        nn*
    controlled:Event = dobj nn*
```

Example 2: An example of a rule designed to capture a recursive event. The rule detects a relevant verbal or adverbial trigger and expects its arguments to be in a SUBJECT ↔ DIRECT OBJECT relationship. The `controlled` argument must be the mention of another event.

```
- name: Phosphorylation_surface_1
  priority: 2
  type: token
  label: [Phosphorylation, Event]
  pattern: |
    (?<trigger>
    [lemma="phosphorylation"]) of []*?
    @theme:PhysicalEntity []*?
    (by @cause:PhysicalEntity)?
```

Example 3: An alternative rule to Example 1 that uses a surface pattern. Surface patterns match event triggers and arguments over sequences of tokens and other mentions (e.g., the `theme` matches over an entire named entity of type `PhysicalEntity`). Event triggers (`trigger`) match the whole sequence of tokens encompassed in parentheses. Argument names preceded by the `@` symbol, e.g., `@theme`, require the specification of an event type (denoted by `:type`). This pattern is shorthand for matching the span of an entire named entity with the specified `type`.

iteration because no event mentions have been created yet, but would match in the second iteration. This process can optionally be optimized with rule priorities (as shown in the figure). For example, the priorities assigned to Examples 1 and 2 enforce that the second rule is executed only in an iteration following the first rule. Utilizing rule priorities allows for a derivational construction of complex events or complete grammars from their components.

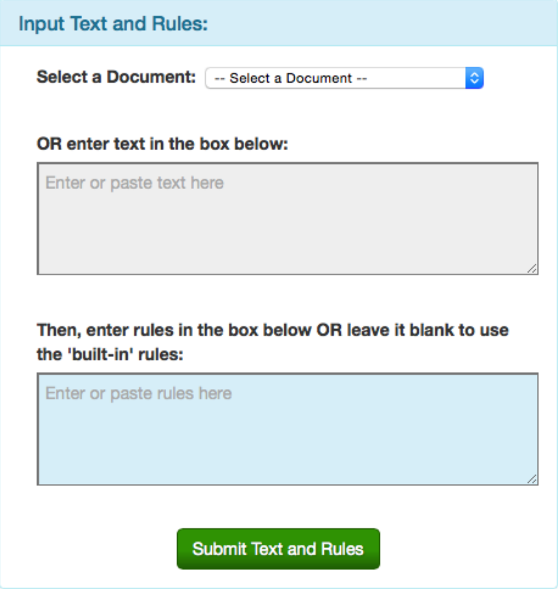
Once the grammar has been defined, the entire system can be run in less than 10 lines of code, as shown in Example 4. The output of this code is a collection of event mentions, i.e., instances of the `EventMention` class outlined in Example 5.

3 Visualization

```
1  class SimpleExample extends App {
2    // read rules from file
3    val rules = Source.fromFile(
4      "rules.yml").mkString
5    // make extractor engine
6    val engine = new ExtractorEngine(rules)
7    // create text processor for biomedical
8    // domain: POS, NER, and syntax
9    val processor = new BioNLPProcessor
10   // make document from free text;
11   // the document includes POS, NER, and
12   // syntactic annotations
13   val text = "TopBP1 promotes the
         phosphorylation of cyclin-D1 by ATR."
14   val doc = processor.annotate(text)
15   // run the actual EE grammar
16   val mentions = engine.extractFrom(doc)
17 }
```

Example 4: The minimal Scala code required to run the system. The input (LINE 13) is raw text. The output is a list of event mentions of the type `EventMention`. Here we show the use of a text processor specific to the biomedical domain. The framework also includes an open-domain text processor that includes POS tagging, named entity recognition, syntactic parsing, and coreference resolution. Additional processors for domain-specific tasks can easily be added.

We accompany the above EE system with an interactive web-based tool for event grammar development and results visualization. Figure 2 shows the input fields for the user interface. The UI accepts free text to match against, and can be configured to run either a predefined domain grammar or one provided on-the-fly through a text box, allowing for the rapid development and tuning of rules.



Input Text and Rules:

Select a Document: -- Select a Document --

OR enter text in the box below:

Enter or paste text here

Then, enter rules in the box below OR leave it blank to use the 'built-in' rules:

Enter or paste rules here

Figure 2: Our interactive environment for rapid development of event grammars.The UI supports the input of rules and free text.

Figure 3 shows the output of the visualization tool on the example sentence from Figure 1 using the gram-

```
class EventMention(
  /** The ontological labels associated with
    * the event (specified in the rule) */
  val label: Seq[String],
  /** The starting point of our pattern */
  val trigger: TextBoundMention,
  /** A mapping of argument names to the
    * Mentions that contain them */
  val arguments: Map[String, Seq[Mention]],
  /** The name of the corresponding rule */
  val foundBy: String
  /** The span of the Mention
    * in the original document*/
  val tokenInterval: Interval)
```

Example 5: Example 4 produces a set of mentions. Here we focus on mentions of events (`EventMention`). This code block shows relevant fields in the `EventMention` class, which stores each event mention detected and assembled by the system. The `arguments` field captures the fact that the mapping from names to arguments is one-to-many (e.g., there may be multiple `theme` arguments). `Interval` stores a token span in the input text. `TextBoundMention` stores a simple mention, minimally a label and a token span.

mar discussed in the previous section. The web interface is implemented as a client-server Grails[10] web application which runs the EE system on the server and displays the results on the client side. The application's client-side code displays both entity and event mentions, as well as the output of the text preprocessor (to help with debugging) using Brat (Stenetorp et al., 2012).

4 Results

We extended the grammar introduced previously to capture 10 different biochemical events, with an average of 11 rules per event type. Using this grammar we participated in a recent evaluation by DARPA's Big Mechanism program[11], where systems had to perform deep reading of two research papers on cancer biology. Table 1 summarizes our results.

Our system was ranked above the median, with respect to overall F1 score. We find these results encouraging for two reasons. First, inter-annotator agreement on the task was below 60%, which indicates that our system roughly approaches human performance, especially for precision. Second, the lower recall is partially explained by the fact that annotators marked also indirect biological relations (e.g., *A activates B*), which do not correspond to actual biochemical reactions but, instead, summarize sequences of biochemical reactions. Our grammar currently recognizes only direct biochemical reactions.

System	Precision	Recall	F1
Submitted run	54%	29%	37.3%
Ceiling system	82.1%	81.8%	82%

Table 1: Results from the January 2015 DARPA Big Mechanism Dry Run evaluation on reading biomedical papers, against a known biochemical model. In addition to event extraction, this evaluation required participants to identify if the extracted information corroborates, contradicts, or extends the given model. Here, extending the model means proposing a biochemical reaction that is not contained in the model, but it involves at least a biochemical entity from the model. The ceiling system indicates idealized performance of the rule-based framework, after a *post-hoc* analysis.

More importantly, this evaluation offers a good platform to analyze the potential of the proposed rule-based framework, by estimating the ceiling performance of our EE system, when all addressable issues are fixed. We performed this analysis after the evaluation deadline, and we manually:

1. Removed the keys that do not encode direct biochemical reactions.
2. Corrected three rules, to better model one event and one entity type.
3. Fixed system bugs, including XML parsing errors, which caused some meta data to appear in text and be misinterpreted as biological entities, and a syntax error in one rule, which caused several false positives.

The results of this ceiling system are listed in the second row in Table 1. This analysis highlights an encouraging finding: the current rule framework is expressive: it can capture approximately 80% of the events in this complex domain. The remaining 20% require coreference resolution and complex syntactic patterns, which were not correctly captured by the parser.

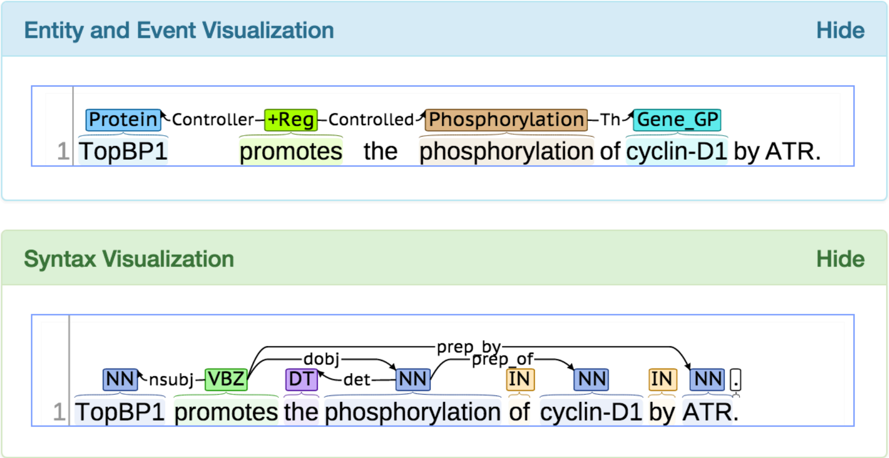
5 Related Work

Despite the dominant focus on machine learning models for IE in the literature, previous work includes several notable rule-based efforts. For example, GATE (Cunningham et al., 2011), and the Common Pattern Specification Language (Appelt and Onyshkevych, 1998) introduce a rule-based framework for IE, implemented as a cascade of grammars defined using surface patterns. The ICE system offers an active-learning system that learns named entity and binary relation patterns built on top of syntactic dependencies (He and Grishman, 2011). Stanford's Semgrex[12] and Tregex (Levy and Andrew, 2006) model syntactic patterns,

[10] `https://grails.org`

[11] `http://www.darpa.mil/Our_Work/I2O/Programs/Big_Mechanism.aspx`

[12] `http://nlp.stanford.edu/software/tregex.shtml`



1 TopBP1 promotes the phosphorylation of cyclin-D1 by ATR.

1 TopBP1 promotes the phosphorylation of cyclin-D1 by ATR.

Figure 3: A Brat-based visualization of the event mentions created from the example sentence in Figure 1. Not shown but included in the visualization: a table with token information (lemmas, PoS tags, NE labels, and character spans).

while a separate tool from the same group, TokensRegex (Chang and Manning, 2014), defines surface patterns over token sequences. Chiticariu et al. (2011) demonstrated that a rule-based NER system can match or outperform results achieved with machine learning approaches, but also showed that rule-writing is a labor intensive process even with a language specifically designed for the task.

In addition to the above domain-independent frameworks, multiple previous works focused on rule-based systems built around specific domains. For example, in bioinformatics, several dedicated rule-based systems obtained state-of-the-art performance in the extraction of protein-protein interactions (PPI) (Hunter et al., 2008; Huang et al., 2004).

Our work complements and extends the above efforts with a relatively simple EE platform that: (a) hybridizes syntactic dependency patterns with surface patterns, (b) offers support for the extraction of recursive events; (c) is coupled with a fast runtime environment; and (d) is easily customizable to new domains.

6 Conclusion

We have described a domain-independent, rule-based event extraction framework and rapid development environment that is simple, fast, powerful, and robust. It is our hope that this framework reduces the entry cost in the development of rule-based event extraction systems.

We demonstrated how to build a biomedical domain from scratch, including rule examples and simple Scala code sufficient to run the domain grammar over free text. We recently extended this grammar to participate in the DARPA Big Mechanism evaluation, in which our system achieved an F1 of 37%. By modeling the underlying syntactic representation of events, our grammar for this task used an average of only 11 rules per event; this indicates that the syntactic structures of events are largely generalizable to a small set of predicate frames and that domain grammars can be constructed with relatively low effort. Our *post-hoc* analysis demonstrated that the system's true ceiling is 82%. This important result demonstrates that the proposed event extraction framework is expressive enough to capture most complex events annotated by domain experts.

Finally, to improve the user experience by aiding in the construction of event grammars, our framework is accompanied by a web-based interface for testing rules and visualizing matched events.

This whole effort is available as open-source code at: `https://github.com/sistanlp/processors`. See also: `https://github.com/sistanlp/processors/wiki/ODIN-(Open-Domain-INformer)`, for ODIN documentation.

Acknowledgments

This work was funded by the DARPA Big Mechanism program under ARO contract W911NF-14-1-0395.

References

Appelt, Douglas E and Boyan Onyshkevych. 1998. The common pattern specification language. In *Proc. of the TIPSTER Workshop*. pages 23–30.

Bui, Quoc-Chinh, Erik M Van Mulligen, David Campos, and Jan A Kors. 2013. A fast rule-based approach for biomedical event extraction. *Proc. of ACL* page 104.

Chang, Angel X. and Christopher D. Manning. 2014. TokensRegex: Defining cascaded regular expressions over tokens. Technical Report CSTR 2014-02, Computer Science, Stanford.

Chiticariu, Laura, R. Krishnamurthy, Y. Li, F. R. Reiss, and S. Vaithyanathan. 2011. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proc. of EMNLP*.

Chiticariu, Laura, Yunyao Li, and Frederick R Reiss. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proc. of EMNLP*.

Cunningham, Hamish, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Developing Language Processing Components with GATE (Version 6)*. University of Sheffield.

de Marneffe, Marie-Catherine and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Proc. of COLING Workshop on Cross-framework and Cross-domain Parser Evaluation*.

He, Yifan and Ralph Grishman. 2011. Ice: Rapid information extraction customization for nlp novices. In *Proc. of NAACL*.

Huang, Minlie, Xiaoyan Zhu, Yu Hao, Donald G. Payan, Kunbin Qu, and Ming Li. 2004. Discovering patterns to extract proteinprotein interactions from full texts. *Bioinformatics* 20(18):3604–3612.

Hunter, Lawrence, Zhiyong Lu, James Firby, William A Baumgartner, Helen L Johnson, Philip V Ogren, and K Bretonnel Cohen. 2008. Opendmap: an open source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC bioinformatics* 9(1):78.

Levy, Roger and Galen Andrew. 2006. Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proc. of LREC*.

Manning, C. D., M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proc. of ACL*.

Ohta, Tomoko, Sampo Pyysalo, Rafal Rak, Andrew Rowley, Hong-Woo Chun, Sung-Jae Jung, Sung-Pil Choi, Sophia Ananiadou, and Junichi Tsujii. 2013. Overview of the pathway curation (pc) task of bionlp shared task 2013. In *Proc. of the BioNLP-ST Workshop*.

Peng, Yifan, Manabu Torii, Cathy H Wu, and K Vijay-Shanker. 2014. A generalizable NLP framework for fast development of pattern-based biomedical relation extraction systems. *BMC bioinformatics* 15(1):285.

Stenetorp, Pontus, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. Brat: a web-based tool for nlp-assisted text annotation. In *Proc. of the Demonstrations at EACL*.