Using Semantic Unification to Generate Regular Expressions from Natural Language

Nate Kushman and Regina Barzilay

MIT
Task

Map natural language to regular expressions

Question: How do I write the regular expression for 3 letter words starting with ‘a’?

Answer: \ba[A-Za-z]\{2\}\b

Motivation: Hard even for humans
Many semantically equivalent correct answers
Semantic Parsing

Rely on a fragment-by-fragment mapping

What is the highest mountain in Alaska?

$\text{argmax} \; 0 \; (\text{elevation:i} \; 0) \; (\text{and} \; (\text{mountain:t} \; 0) \; (\text{loc:t} \; 0 \; \text{alaska:s})))$
Key Challenge:

Fragment-by-fragment mapping not possible

Three letter word starting with ‘a’
\ba[A-Za-z]{2}\b

Three letter word starting with ‘a’
\ba[A-Za-z]{2}\b

Three letter word starting with ‘a’
\ba[A-Za-z]{2}\b
Key Challenge:
Fragment-by-fragment mapping not possible

\[ \text{Three letter word starting with 'a'} \]
\[ \backslash b[a-zA-z]\{2\}\backslash b \]
\[ ([a-zA-z])\{3\} \& (\backslash b[a-zA-z]+\backslash b) \& (a.*) \]

Key Idea: Take advantage of semantic equivalence to overcome the lack of syntactic isomorphism
Semantic Equivalence

Exact String Equivalence

```
strcmp(expr1, expr2)
```

Local Transformations

Execution Based Equivalence
Liang et al. 2011, Artzi et al. 2013

```
argmax
(size:i) and (city:t) (loc:t georgia:s) 
```

```
and
(city:t) (loc:t georgia:s) 
```

```
(locate:capital)
(city:t) (loc:t georgia:s) 
```

```
and
(loc:t georgia:s) (city:t) 
```

```
argmax
(size:i) and atlanta 
```

```
 atlanta 
```

```
 atlanta 
```

```
 atlanta 
```

```
(locate:capital)
(city:t) (loc:t georgia:s) 
```
Our Approach: Exact Semantic Equivalence

Generate the same output for any input

\[.*\text{(ab.*)\mid\text{ac.})} \rightarrow \]

Regular Expression

NFA

DFA

Minimal DFA

Minimal DFAs are unique $\Rightarrow$ DFA\text{-EQUAL}
at least 3 letters  

\( .*[A-Za-z].* \) \( \{3,\} \)
Representation

at least 3

(letters){3,}

.*[A-Za-z].*
Representation

at least 3 (\text{\{3\}}) \rightarrow \text{letters} \cdot [A-Za-z] \cdot *
Representation

at least \((\text{letters})\{3\}\) letters \.*[A-Za-z]*\.

3 3
Representation

Lexicon:

- **letters**: .*[A-Za-z].*
- **R**: Full Regular Expression
- **I**: Integers
- **R/I/R**: Regular Expression Functions

Combinators:

- **Forward Application**: at least λxy.(y){x,} + 3 3 \rightarrow at least 3 λy.(y){3,}
- **Backward Application**: 'b' b + or 'a' λx.(x|a) \rightarrow 'b' or 'a' (b|a)
Key Departures

1. Parameters updates use semantic equivalence

2. Parser based on n-best parsing which more effectively represents high probability parses
Model

Parse Probability

\[ p(t|\overrightarrow{w}; \Theta, \Lambda) = \frac{e^{\Theta \cdot \varphi(t,\overrightarrow{w})}}{\sum_{t'} e^{\Theta \cdot \varphi(t',\overrightarrow{w})}} \]

Standard Maximum Log-Likelihood Objective

\[ O = \sum_{i} \log \left( \sum_{t|\text{regexp}(t)=r_{i}} p(t|\overrightarrow{w}; \Theta, \Lambda) \right) \]

Our Objective

\[ O = \sum_{i} \log \left( \sum_{t|\text{DFA-EQUAL}(\text{regexp}(t),r_{i})} p(t|\overrightarrow{w}; \Theta, \Lambda) \right) \]
Parameter Estimation

Stochastic Gradient Descent

**Expected feature counts in correct parses**

\[
\frac{\partial O_i}{\partial \theta_j} = E_p(t|\text{DFA-EQUAL}(\text{regexp}(t), r_i), \bar{w}_i; \theta, \Delta) [\varphi(t, \bar{w}_i)] - E_p(t|\bar{w}_i; \theta, \Delta) [\varphi(t, \bar{w}_i)]
\]

**Expected feature counts in all parses**

**Conditioned on generating correct regular expression**

**Exact calculation computationally intractable**
Approximating the Gradient

Our Approach: N-Best Parses

- Always includes the highest probability parses
- Runtime performance scales well with N

Traditional Approach: Beam Search Inside-Outside

- Myopic pruning in beam search removes high probability parses
- Runtime performance scales badly with beam size
Learning the Lexicon

- Initialize $\Lambda$ with one lexical entry for each training sample
- Each iteration split lexical entries used by correct parses

at least 3 letters $\lambda x.([A-Za-z].*)^3$}

All possible phrase splits:
- at least 3 letters
- at least 3 letters
- at least 3 letters

All possible logical form splits:
- $\lambda x.([A-Za-z].*)^3$
- $\lambda x.([A-Za-z]x)^3$
- $\lambda x.([A-Za-z]x^*)^3$
- $\lambda x.([A-Za-z].*)^3$
Learning algorithm

Initialize: $\Lambda$ with an entry for each training sample

For: each iteration and each training example: $\overrightarrow{w_i}, r_i$

$\text{BEST} = \text{set of n-best parses}$

$\text{CORRECT} = \text{parses in BEST that are DFA-EQUAL to } r_i$

- Update lexicon:
  - $\Lambda = \Lambda \cup \text{SPLIT-LEX(CORRECT)}$

- Update parameters:
  - $\Delta = E_{p(t|t\in\text{CORRECT})}[\varphi(t, \overrightarrow{w_i})] - E_{p(t|t\in\text{BEST})}[\varphi(t, \overrightarrow{w_i})]$
## Dataset

880 natural language, regular-expression pairs

<table>
<thead>
<tr>
<th>Lines that contain words with 'ru'</th>
<th>.*\b[A-Za-z]<em>ru[A-Za-z]\b.</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines using two or more words comprised of 5 letters</td>
<td>.<em>\b[A-Za-z]{5}\b.</em>{2,}</td>
</tr>
<tr>
<td>Lines containing words that begin with ‘G’ and end with ‘y’</td>
<td>.*\bG[A-Za-z]<em>y\b.</em></td>
</tr>
<tr>
<td>Lines that contain the numbers ‘9’ and ‘10’</td>
<td>*(9.*10</td>
</tr>
<tr>
<td>Lines that have 3 numbers and contain the word “Columbia”</td>
<td>(.<em>[0-9].</em>){3}&amp;(.<em>\bColumbia\b.</em>)</td>
</tr>
<tr>
<td>Lines that contain a word starting with the letter ‘a’ and a word starting with the letter ‘z’</td>
<td>(.\ba[A-Za-z]<em>\b.</em>&amp;.<em>\bz[A-Za-z]\b.</em>)</td>
</tr>
</tbody>
</table>

Amazon Mechanical Turk oDesk
Experimental Setup

Baseline

*UBL* – state-of-the-art semantic parser based on CCG
- string equality plus simple transformations

[Kwiatkowski et al., 2010]

Test Train

3 fold cross validation $\rightarrow$ 587 train/239 test

Evaluation

Evaluate on accuracy using DFA-EQUAL
Results

**Semantic Equivalence**

- **UBL:** 37%
- **Semantic Equivalence:** 66%

Significantly outperform baseline
Alternative Equivalence Techniques

Initialize: \( \Lambda \) with an entry for each training sample

For: each iteration and each training example: \( \vec{w}_i, r_i \)

- BEST = n-best parses
- CORRECT = parses in BEST that are DFA-EQUAL to \( r_i \)

• Update lexicon:
  - \( \Lambda = \Lambda \cup \text{SPLIT-LEX}(\text{CORRECT}) \)

• Update parameters:
  - \( \Delta = \sum_{t \in \text{CORRECT}} E_p(t | t \in \text{CORRECT})[\varphi(t, \vec{w}_i)] - \sum_{t \in \text{BEST}} E_p(t | t \in \text{BEST})[\varphi(t, \vec{w}_i)] \)
Alternative Equivalence Techniques

Exact String Equivalence

`strcmp`

Execution Based Equivalence

details in the paper

Heuristic Equivalence (local transformations)

\[(a|(b|c)) \rightarrow (a|b|c)\]  \[(b|a) \rightarrow (a|b)\]  \[(a|b) \rightarrow (a|b)\]
Results

Semantic Equivalence

Percent Correct

UBL

37%

Semantic Equivalence

66%
Results

- **UBL**: 37%
- **String Equality**: 31%
- **Semantic Equivalence**: 66%

Percent Correct
Results

- **UBL**: 37%
- **String Equality**: 31%
- **Execution Equality**: 52%
- **Semantic Equivalence**: 66%
Results

- **UBL**: 37%
- **String Equality**: 31%
- **Execution Equality**: 52%
- **Heuristic Equality**: 57%
- **Semantic Equivalence**: 66%

Percent Correct
Beyond Regular Expressions

• Semantic unification for regular expressions is typically very efficient
• Semantic unification for fully general domains is undecidable
• In many interesting domains it’s decidable and heuristics make it efficient:
  – Theorem proving
  – Hardware verification
  – SAT Solver
Conclusion

• Using the inference capabilities of a domain improves the performance of semantic parsing

• Shown in the domain of regular expressions

• Motivates its use in more general domains

Code and Data available at:
http://groups.csail.mit.edu/rbg/code/regexp/