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

(argmax $0 (elevation:i $0) (and (mountain:t $0) (loc:t $0 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

Three letter word starting with 'a'

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

([A-Za-z]){3} & (\b[A-Za-z]+\b) & (a.*)

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

Semantic Equivalence

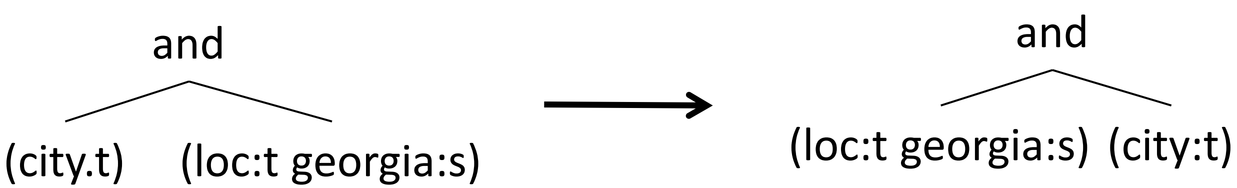
Exact String Equivalence

Zelle & Mooney 1996, Thompson & Mooney 2003, Kate & Mooney 2005, Liang et al. 2009

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

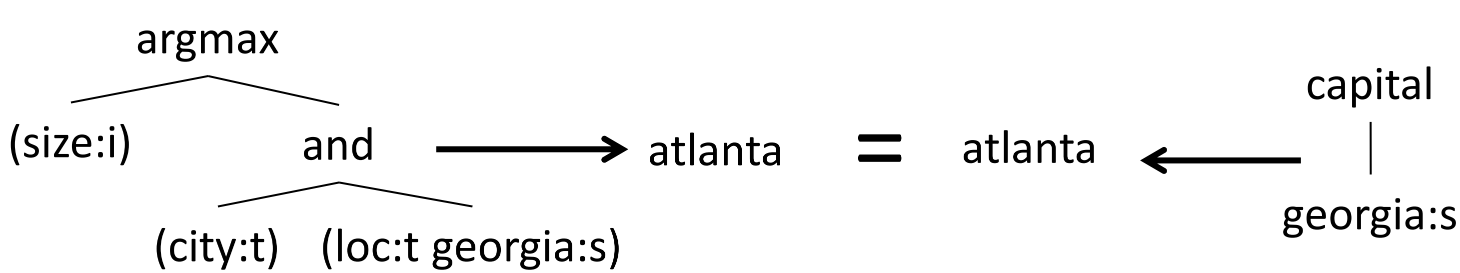
Local Transformations

Ge & Mooney 2005, Wong & Mooney 2007, Kwiakowski et al. 2010, 2011



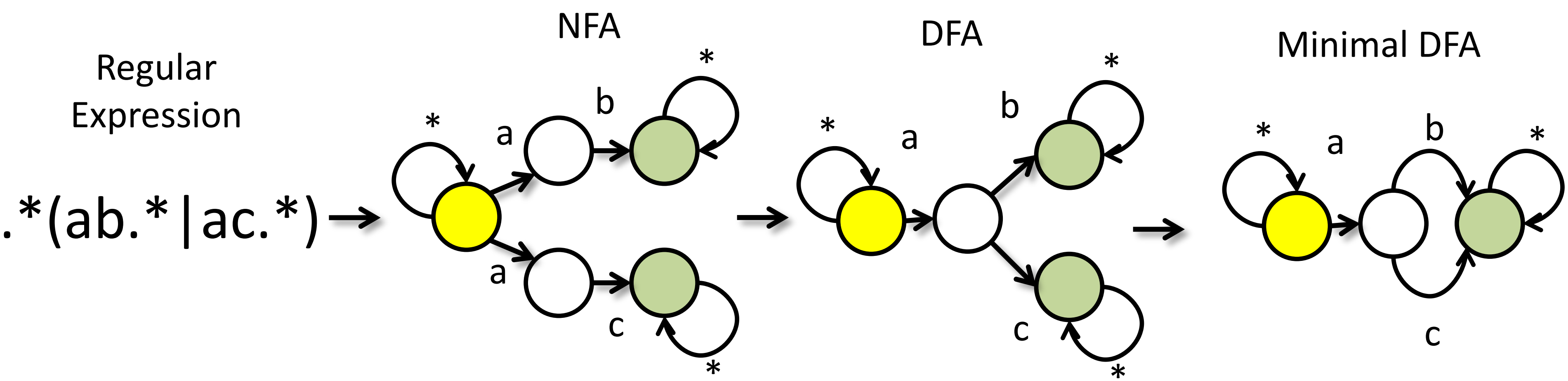
Execution Based Equivalence

Branavan et al. 2009, 2010, 2011, Clark & Roth 2010, Chen & Mooney, 2011
Liang et al. 2011, Artzi et al. 2013



Our Approach: *Exact Semantic Equivalence*

Generate the same output for any input



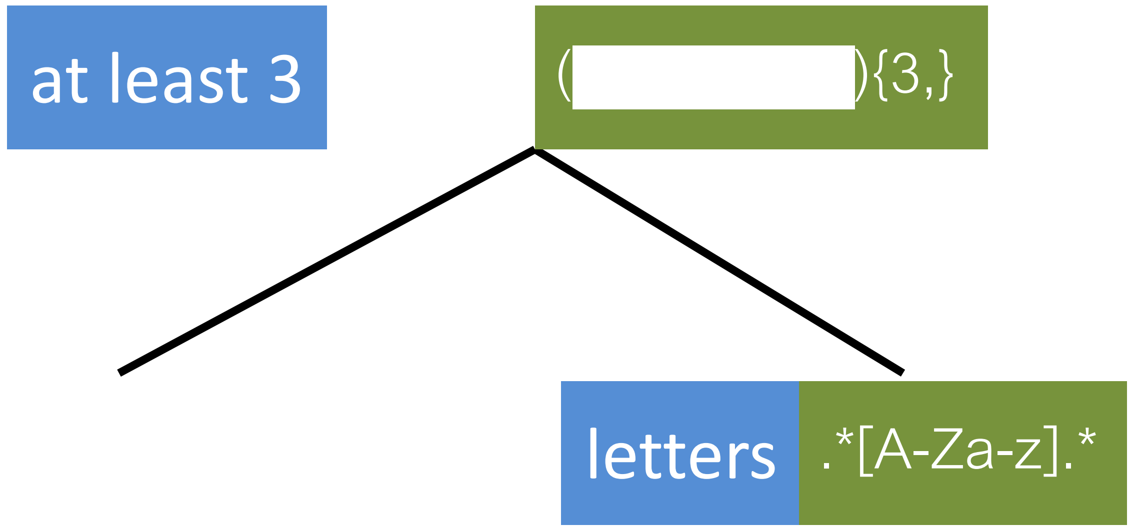
Minimal DFAs are unique → `DFA-EQUAL`

Representation

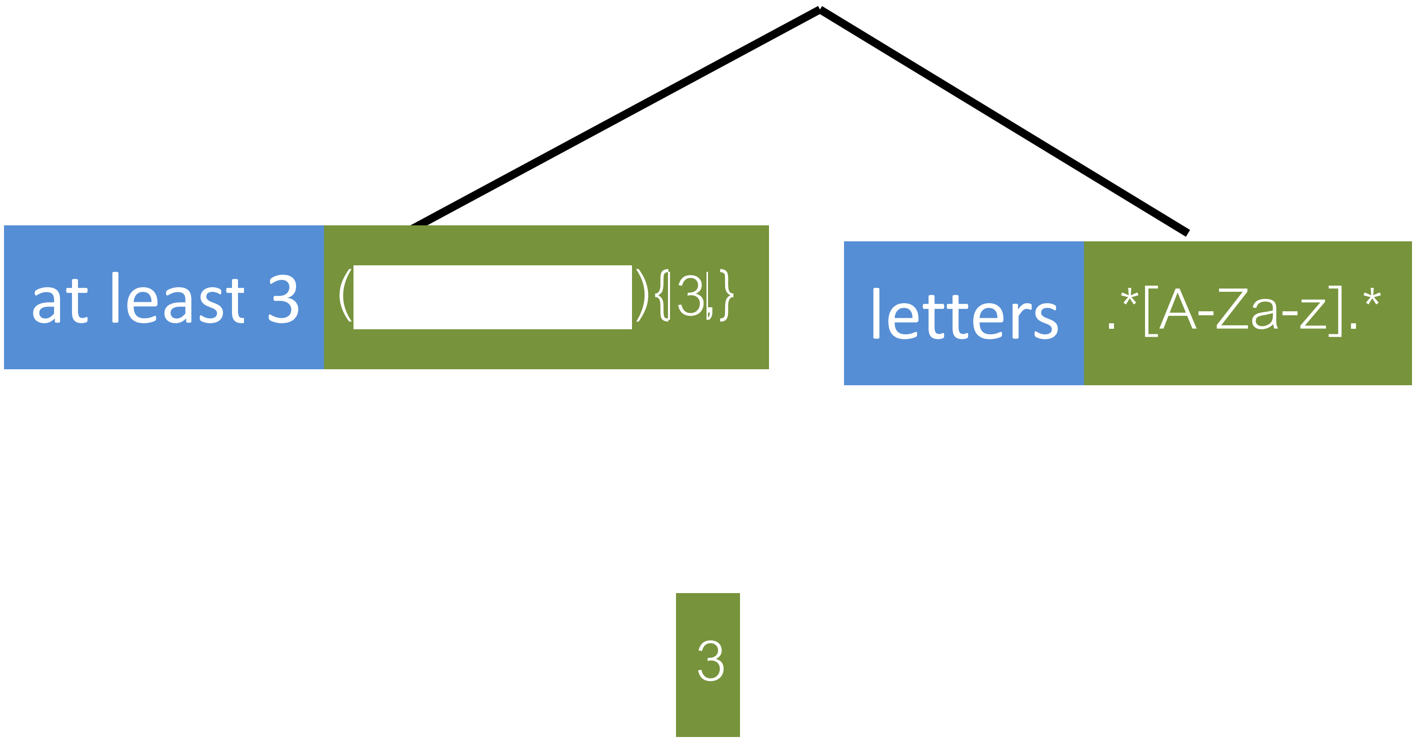
at least 3 letters (.*[A-Za-z].*){3,}

.*[A-Za-z].*

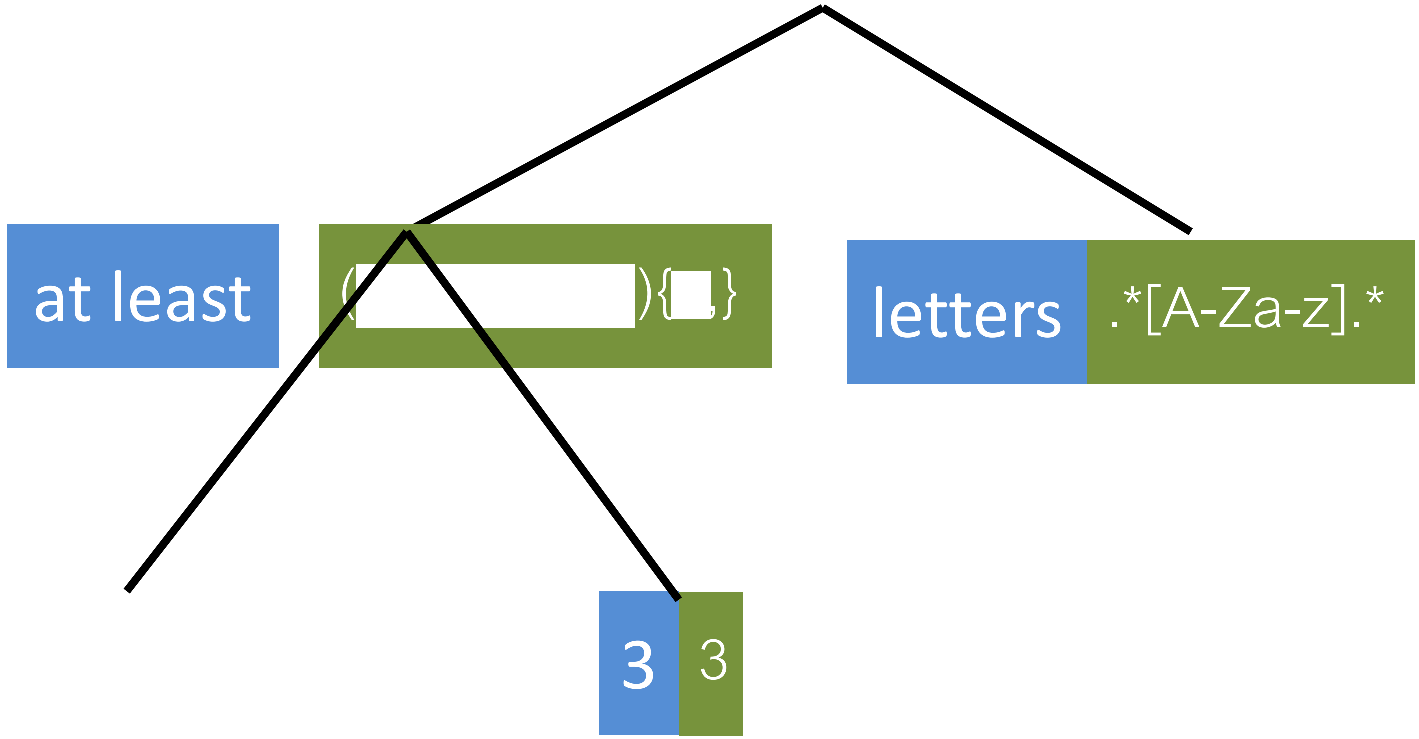
Representation



Representation



Representation



Representation

Lexicon:

letters	.*[A-Za-z].*	R	Full Regular Expression
3	3	I	Integers
at least	λxy.(y){x,}	R/I/R	Regular Expression Functions

Combinators:

at least λxy.(y){x,} + 3 3 → at least 3 λy.(y){3,} Forward Application

‘b’ b + or ‘a’ λx.(x|a) → ‘b’ or ‘a’ (b|a) Backward Application

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

$$O = \sum_i log \sum_{t|regexp(t)=r_i} p(t|\overrightarrow{w};\Theta,\Lambda)$$

String Equivalence

Our Objective

$$O = \sum_i log \sum_{t|\mathrm{DFA\!-\!EQUAL}(regexp(t),r_i)} p(t|\overrightarrow{w};\Theta,\Lambda)$$

Semantic Equivalence

Parameter Estimation

Stochastic Gradient Descent

Expected feature counts in correct parses

Expected feature counts in all parses

$$\frac{\partial O_i}{\partial \Theta_j} = E_{p(t|DFA\text{-}EQUAL(regexp(t),r_i),\vec{w}_i;\Theta,\Lambda)}[\varphi(t,\vec{w}_i)] - E_{p(t|\vec{w}_i;\Theta,\Lambda)}[\varphi(t,\vec{w}_i)]$$

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 Λ with one lexical entry for each training sample
- Each iteration split lexical entries used by correct parses

---

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

All possible phrase splits:

at least 3 | letters

at least | 3 letters

at | least 3 letters

X

All possible logical form splits:

λx.(.*[A-Za-z].*){x,} | 3

λx.(.*[A-Za-z]x){3,} | .*

λx.(.*[A-Za-z]x*){3,} | .

λx.(.*x.*){3,} | [A-Za-z]

⋮

Learning algorithm

Initialize: Λ with an entry for each training sample

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

BEST = set of n-best parses

CORRECT = parses in BEST that are DFA-EQUAL to r_i

- Update lexicon:
  - Λ = Λ ∪ SPLIT-LEX(CORRECT)
- Update parameters:
  - $\Delta = E_{p(t|t \in CORRECT)}[\varphi(t, \vec{w}_i)] - E_{p(t|t \in BEST)}[\varphi(t, \vec{w}_i)]$

Dataset

880 natural language, regular-expression pairs

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

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]

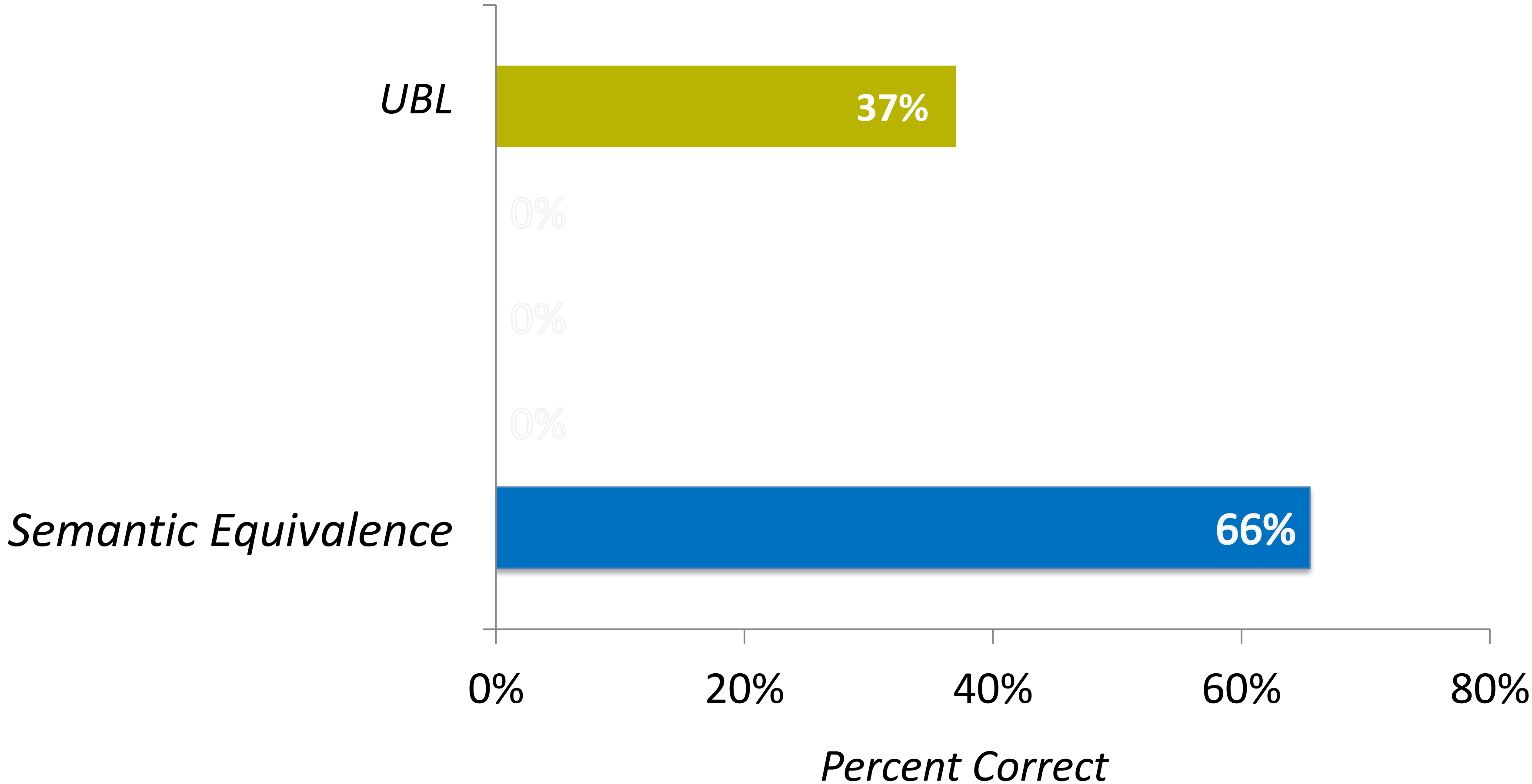
Test Train

3 fold cross validation → 587 train/239 test

Evaluation

Evaluate on accuracy using DFA-EQUAL

Results



Significantly outperform baseline

Alternative Equivalence Techniques

Initialize: Λ with an entry for each training sample

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

BEST = n-best parses

CORRECT = parses in BEST that are DFA-EQUAL to r_i

?

- Update lexicon:
  - Λ = Λ ∪ SPLIT-LEX(CORRECT)
- Update parameters:
  - $\Delta = E_{p(t|t \in CORRECT)}[\varphi(t, \vec{w}_i)] - E_{p(t|t \in BEST)}[\varphi(t, \vec{w}_i)]$

Alternative Equivalence Techniques

Exact String Equivalence

strcmp

Execution Based Equivalence

details in the paper

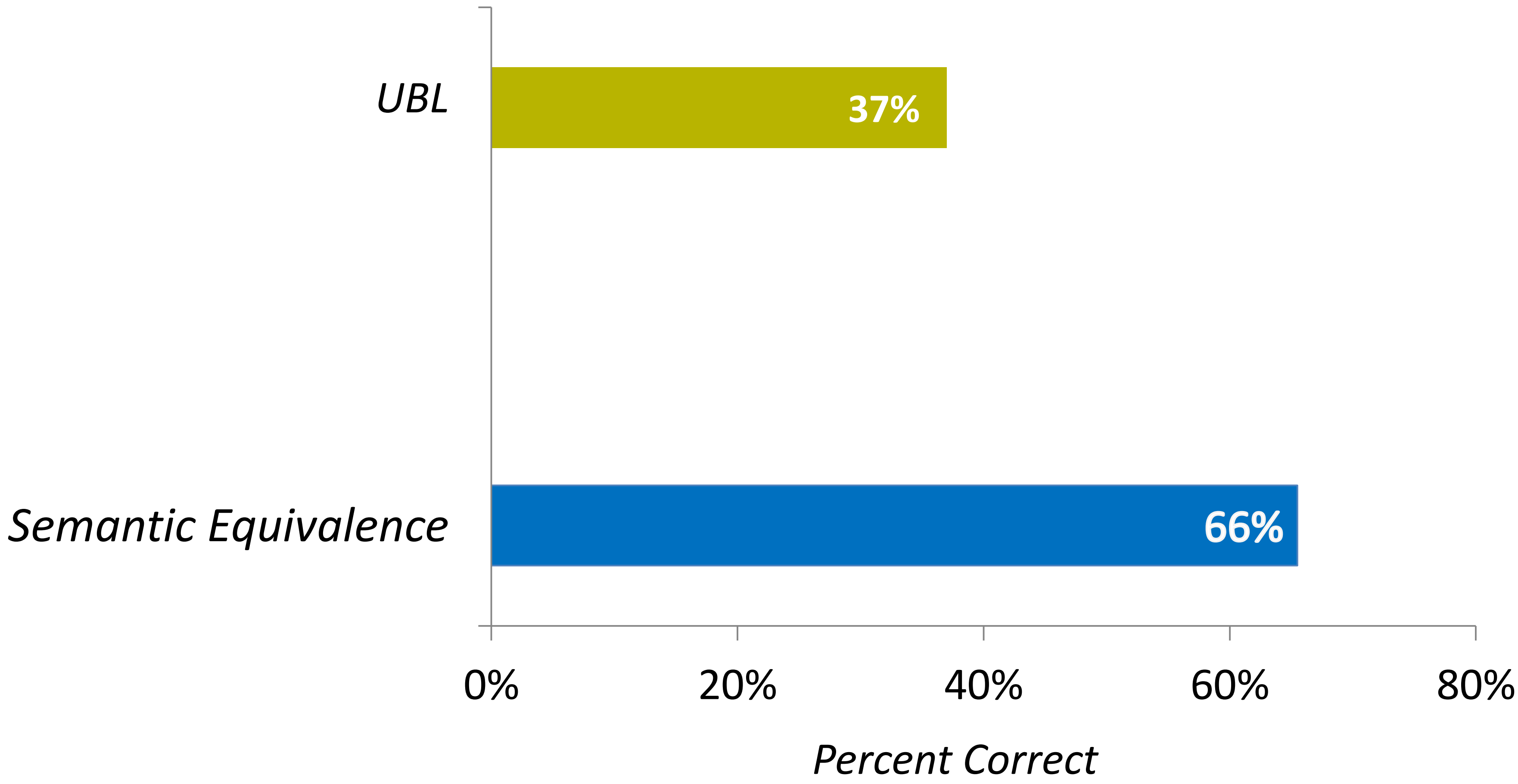
Heuristic Equivalence (local transformations)

(a|(b|c)) → (a|b|c)

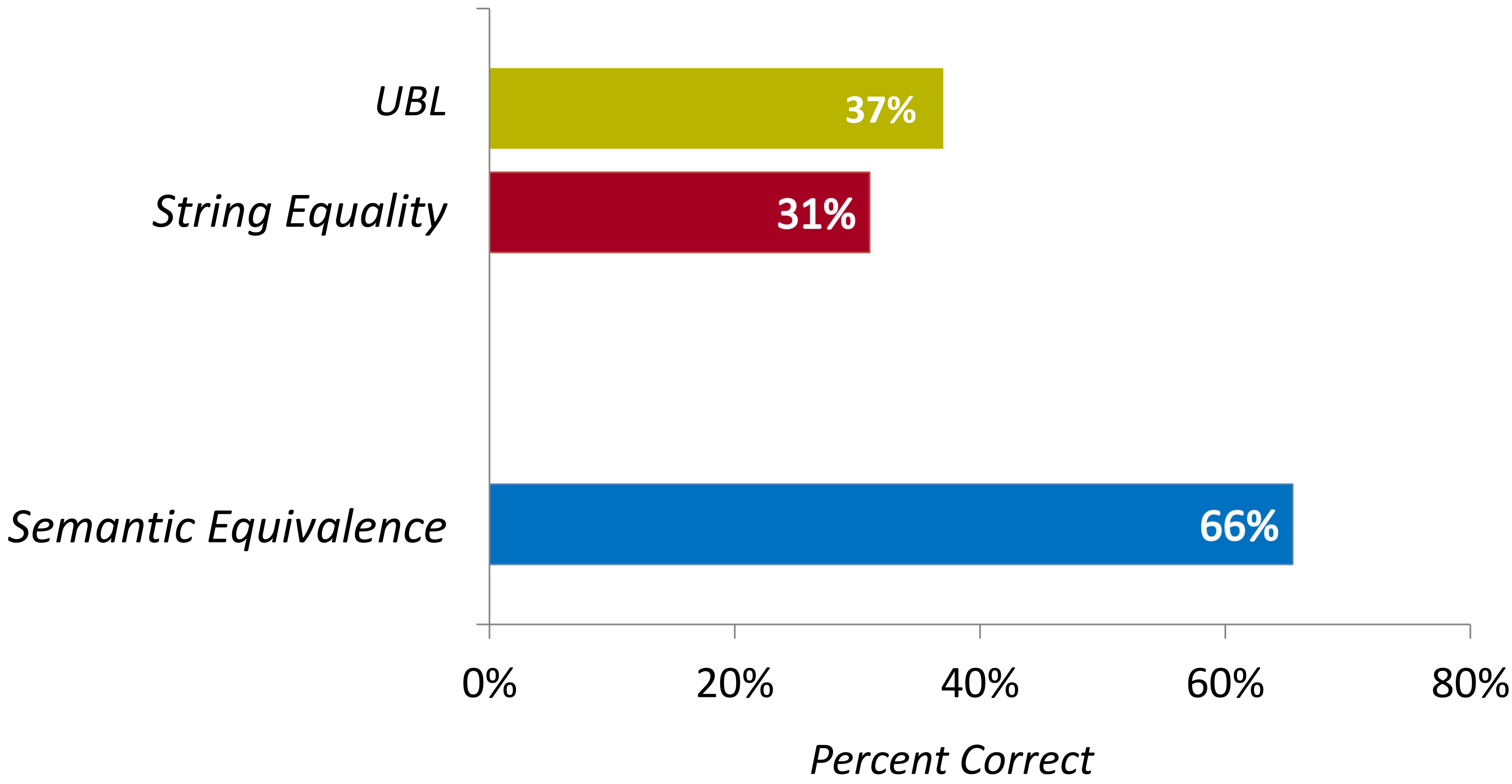
(b|a) → (a|b)

(a|b) → (a|b)

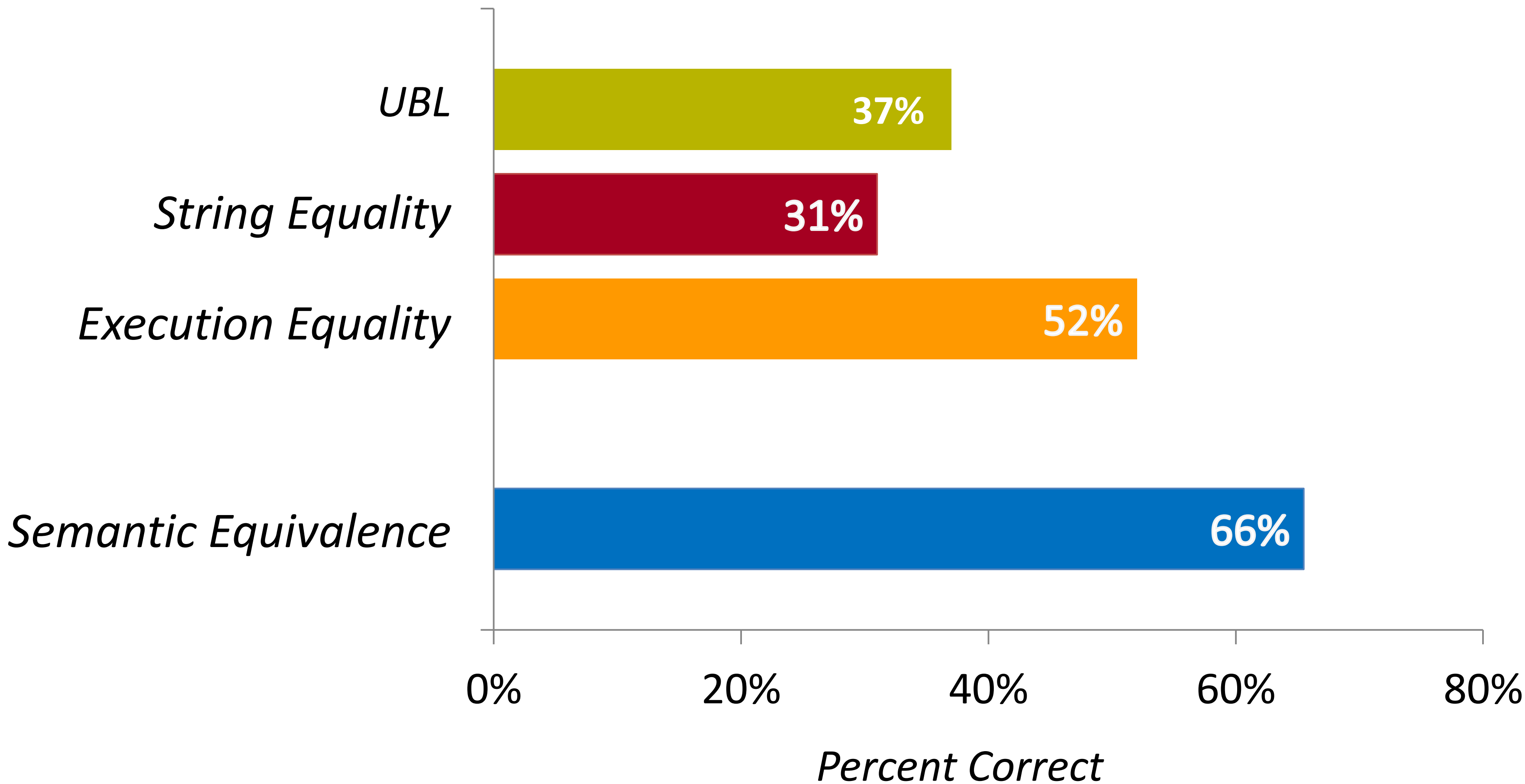
Results



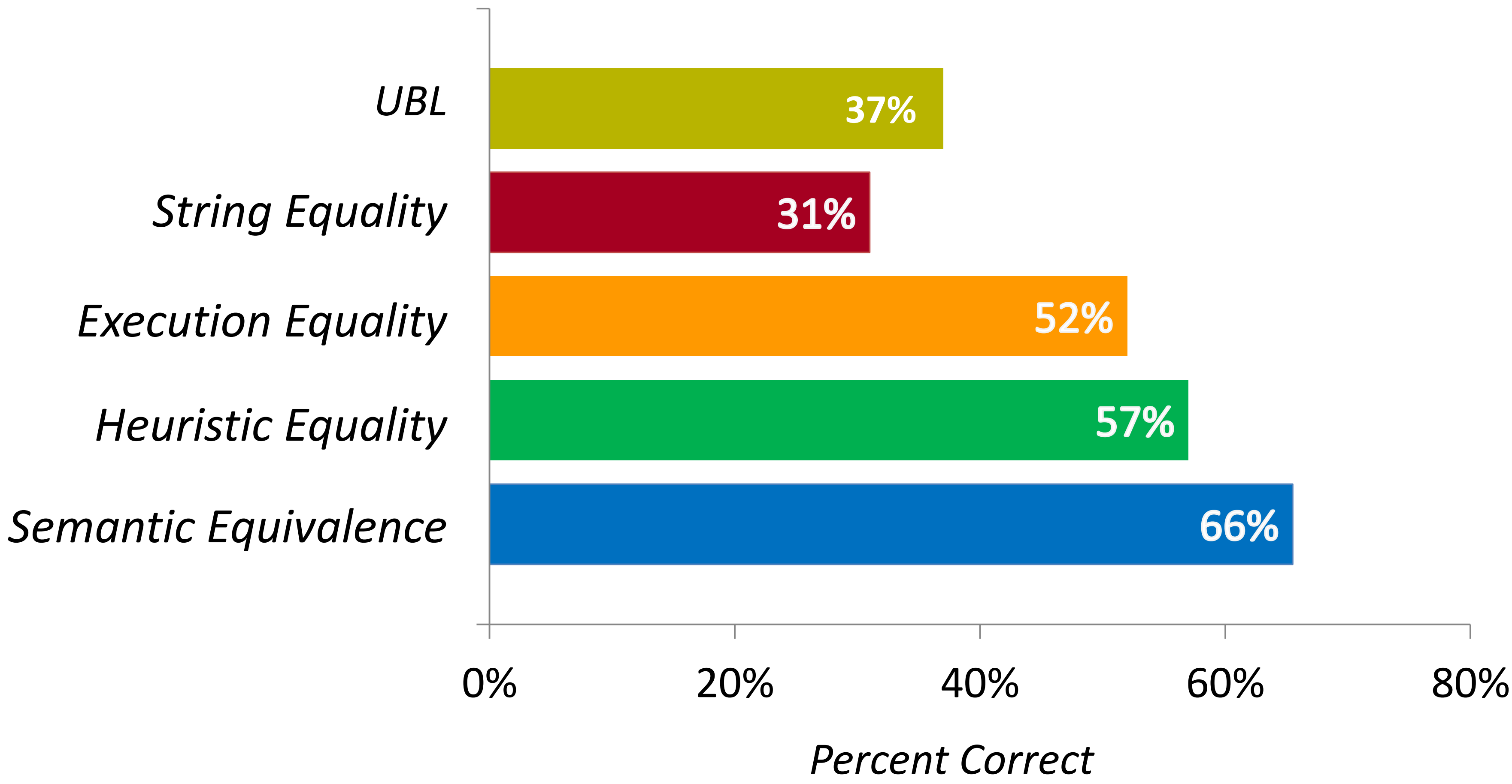
Results


UBL
37%
String Equality
31%
Semantic Equivalence
66%
0%
20%
40%
60%
80%
Percent Correct

Results


UBL
String Equality
Execution Equality
Semantic Equivalence
37%
31%
52%
66%
0%
20%
40%
60%
80%
Percent Correct

Results



UBL
String Equality
Execution Equality
Heuristic Equality
Semantic Equivalence
37%
31%
52%
57%
66%
0%
20%
40%
60%
80%
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/