Synthesizing Extraction Rules from User Examples with SEER

Maeda F. Hanafi (maeda.hanafi@nyu.edu), Azza Abouzied (azza@nyu.edu),
Laura Chiticariu (chiti@us.ibm.com), Yunyao Li (yunyaoli@us.ibm.com)

Challenges in Information Extraction (IE)

- **IE is a key technology for understanding text**
  - Companies extract data to build knowledge-bases.
  - Journalists extract crime rates from FBI datasets to analyze changes.
  - Business analysts extract people and corporations from SEC filings to analyze relationships.

- **Two approaches to IE:**

1. Machine learning:
   - Cons: Requires large training datasets.
   - Cons: Produces hard-to-interpret statistical models.
   - Pros: Once training data is given, quick to generate results.
2. Programming:
   - Cons: Requires learning the language.
   - Cons: Labor-intensive and time-consuming.
   - Pros: Transparency

Solution: SEER

- **SEER: Combines the best of both approaches.**
  - Learning high-quality extraction rules from handful of examples.

- **SEER synthesizes and suggests visual rules to the user.**
  - Quick to get high-quality extraction rules and results.
  - Reduced manual development effort.

- Extraction Rules = sequences of *primitives*
  Primitives capture certain tokens.

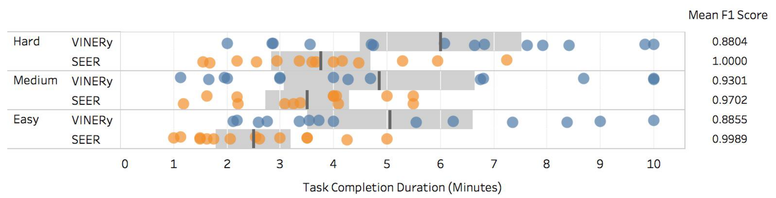
  Pre-builts extract entities: P: City
  Literals extract exact strings: L: 'Dubai'
  Dictionaries extract sets of exact strings: D: {Dubai, London}
  Regexes extract from a library of regexes: R: [A-Za-z]+
  Token gaps extract any token: T: 0-1

4. Evaluations

- **Baseline:VINERy Visual Programming IDE for IE** (VLDB '15)

- User Study: Each user used both tools on two datasets in random order to minimize learning effects.

- Black bars = mean duration.
  Gray bars = 95% mean confidence interval.



1. Highlight and Process Examples

- **The user highlights examples of text to extract.**

Law enforcement agencies submitted incident reports involving 5,479 criminal incidents. In cities with populations from 50,000 to 99,000 inhabitants, violent crimes rose 11 percent. In cities of populations above 100 thousand, hate crime offenses decreased 5 percent in 2010 alone.

Positive examples:
Negative examples:

- **SEER enumerates primitives for each positive token.**

Tokens:	Primitives:
rose	L: 'rose'
	R: [A-Za-z]+
11	P: Number
	P: Integer
	L: '11'
	R: [0-9]+
	T: 0-1
percent	L: 'percent'
	R: [A-Za-z]+
11 percent	P: Percentage

Tokens:	Primitives:
decreased	L: 'decreased'
	R: [A-Za-z]+
5	P: Number
	P: Integer
	L: '5'
	R: [0-9]+
	T: 0-1
percent	L: 'percent'
	R: [A-Za-z]+
5 percent	P: Percentage

- **Assign scores to the primitives to guide the search.**
  - Intuition: Rule developers prefer certain primitives for certain tokens.
  - If the primitive's tokens have inherent semantics:

    Semantic Token: **Dubai**

Primitive types:	Score:	Primitives:
Prebuilt	1.0 to 0.6	P: City = 1.0
Dictionary, Literal	0.4	L: 'Dubai' = 0.4
Regex, Token Gap	0.2	R: [A-Za-z]+ = 0.2

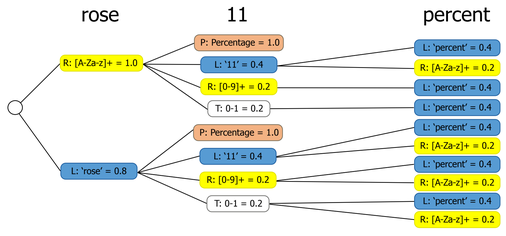
  - If the primitive's tokens are syntactic:

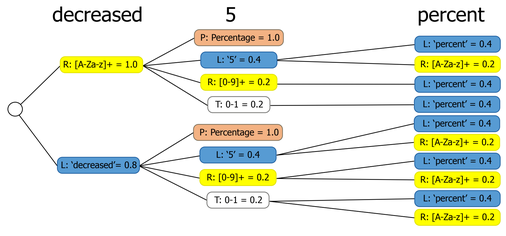
    Syntactic Token: **of**

Primitive types:	Score:	Primitives:
Regex, Token Gap	1.0	R: [A-Za-z]+ = 1.0
Dictionary, Literal	0.8	L: 'of' = 0.8

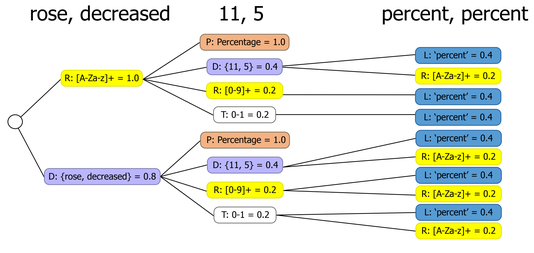
2. Synthesize Rules

- **For each positive example, generate a tree of rules.**
- Extraction Rule = path of primitives from root to leaf
- Rules capturing negative examples are removed.





- **Capture all user examples by intersecting trees.**
  - Generalize the rules to capture all user examples.



3. Prune and Suggest Rules

- **Group similar rules to ensure rule diversity.**
  - Similar rules have the same primitive types.
  - Pick the top rule of each group.
    Rule score = AVG(Primitve Scores)

Groups:

- Regex, Pre-built: R: [A-Za-z]+ = 1.0, P: Percentage = 1.0 = 1.00
- Regex, Dictionary, Literal: R: [A-Za-z]+ = 1.0, D: {11, 5} = 0.4, L: 'percent' = 0.4 = 0.60
  D: {rose, decreased} = 0.8, R: [0-9]+ = 0.2, L: 'percent' = 0.4 = 0.47
- Regex, Dictionary: R: [A-Za-z]+ = 1.0, D: {11, 5} = 0.4, R: [A-Za-z]+ = 0.2 = 0.53
  D: {rose, decreased} = 0.8, D: {11, 5} = 0.4, R: [A-Za-z]+ = 0.2 = 0.47
  D: {rose, decreased} = 0.8, R: [0-9]+ = 0.2, R: [A-Za-z]+ = 0.2 = 0.40
- Regex, Literal: R: [A-Za-z]+ = 1.0, R: [0-9]+ = 0.2, L: 'percent' = 0.4 = 0.53
- Regex, Token Gap, Literal: R: [A-Za-z]+ = 1.0, T: 0-1 = 0.2, L: 'percent' = 0.4 = 0.53
- Dictionary, Pre-built: D: {rose, decreased} = 0.8, P: Percentage = 1.0 = 0.90
- Dictionary, Literal: D: {rose, decreased} = 0.8, D: {11, 5} = 0.4, L: 'percent' = 0.4 = 0.53
- Dictionary, Token Gap, Literal: D: {rose, decreased} = 0.8, T: 0-1 = 0.2, L: 'percent' = 0.4 = 0.47
- Dictionary, Token Gap, Regex: D: {rose, decreased} = 0.8, T: 0-1 = 0.2, R: [A-Za-z]+ = 0.2 = 0.40

- **Show the top ten rules to the user.**

1. R: [A-Za-z]+ = 1.0, P: Percentage = 1.0 = 1.00
2. D: {rose, decreased} = 0.8, P: Percentage = 1.0 = 0.90
3. R: [A-Za-z]+ = 1.0, D: {11, 5} = 0.4, L: 'percent' = 0.4 = 0.60
4. R: [A-Za-z]+ = 1.0, D: {11, 5} = 0.4, R: [A-Za-z]+ = 0.2 = 0.53
5. R: [A-Za-z]+ = 1.0, R: [0-9]+ = 0.2, L: 'percent' = 0.4 = 0.53
6. R: [A-Za-z]+ = 1.0, T: 0-1 = 0.2, L: 'percent' = 0.4 = 0.53
7. D: {rose, decreased} = 0.8, D: {11, 5} = 0.4, L: 'percent' = 0.4 = 0.53
8. D: {rose, decreased} = 0.8, T: 0-1 = 0.2, L: 'percent' = 0.4 = 0.47
9. D: {rose, decreased} = 0.8, T: 0-1 = 0.2, R: [A-Za-z]+ = 0.2 = 0.40