Lecture 6
Structured Explicit Search

Armando Solar-Lezama and Xiaokang Qiu
Sets of programs as graphs
E-Graph

\((x+2*y)+4*y\)
Flashfill

Spreadsheet Data Manipulations Using Examples
Gulwani, Harris, Singh, CACM August 2012
With slides from Sumit Gulwani and Rishabh Singh
Core language

\[
\begin{align*}
\text{Trace expr } e & := \text{Concatenate}(f_1, \ldots, f_n) \mid f \\
\text{Atomic expr } f & := \text{ConstStr}(s) \mid \text{SubStr}(v_i, p_1, p_2) \mid \text{Loop}(\lambda w : e) \\
\text{Position } p & := \text{CPos}(k) \mid \text{Pos}(r_1, r_2, c) \\
\text{Integer expr } c & := k \mid k_1 w + k_2 \\
\text{Regular expr } r & := \text{TokenSeq}(T_1, \ldots, T_n) \mid T \mid \varepsilon
\end{align*}
\]

Additional shorthand

\[\text{SubStr2}(v_i, r, k) = \text{SubStr}(v_i, \text{Pos}(\varepsilon, r, k), \text{Pos}(r, \varepsilon, k))\]

- \(k^{th}\) occurrence of regular expression \(r\) in \(v_i\)
Let \( w = \text{SubString}(s, p, p') \)
where \( p = \text{Pos}(r_1, r_2, k) \) and \( p' = \text{Pos}(r_1', r_2', k') \)

Two special cases:
- \( r_1 = r_2' = \epsilon \): This describes the substring
- \( r_2 = r_1' = \epsilon \): This describes boundaries around the substring

The general case allows for the combination of the two and is thus a very powerful operator!
Additional Control Structure

String program $P$ ::= $\text{Switch } ((b_1, e_1), \ldots, (b_n, e_n)) \mid e$

Boolean condition $b$ ::= $d_1 \lor \ldots \lor d_n$

Conjunction $d$ ::= $\pi_1 \land \ldots \land \pi_n$

Predicate $\pi$ ::= $\text{Match}(v, r, k) \mid \neg \text{Match}(v, r, k)$
Syntactic String Transformations: Example

<table>
<thead>
<tr>
<th>Input $v_1$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(425)-706-7709</td>
<td>425-706-7709</td>
</tr>
<tr>
<td>510.220.5586</td>
<td>510-220-5586</td>
</tr>
<tr>
<td>235 7654</td>
<td>425-235-7654</td>
</tr>
<tr>
<td>745-8139</td>
<td>425-745-8139</td>
</tr>
</tbody>
</table>

$\text{Switch}((b_1, e_1), (b_2, e_2))$, where

\[
\begin{align*}
    b_1 &\equiv \text{Match}(v_1, \text{NumTok}, 3), & b_2 &\equiv \neg\text{Match}(v_1, \text{NumTok}, 3), \\
    e_1 &\equiv \text{Concatenate} \left( \begin{array}{c} 
        \text{SubStr2}(v_1, \text{NumTok}, 1), \\
        \text{ConstStr}(-), \\
        \text{SubStr2}(v_1, \text{NumTok}, 2), \\
        \text{ConstStr}(-), \\
        \text{SubStr2}(v_1, \text{NumTok}, 3) 
    \end{array} \right), & e_2 &\equiv \text{Concatenate} \left( \begin{array}{c} 
        \text{ConstStr}(425-), \\
        \text{ConstStr}(-), \\
        \text{SubStr2}(v_1, \text{NumTok}, 1), \\
        \text{ConstStr}(-), \\
        \text{SubStr2}(v_1, \text{NumTok}, 2) 
    \end{array} \right)
\end{align*}

\[ \text{Slide by Sumit Gulwani} \]
How does it work

Reuse ideas from version space algebra

- Start with simple version spaces
- Define combinators to construct complex version spaces from simple ones

New twist:

- Move beyond simple lattice representations
Guarded Expressions

$D$ is a set of examples $i_j, o_j$. $H$ corresponds to trace expressions. Compute $S_j = VS_{(i_jo_j)H}$ and $S = VS_{DH} = \cap S_j$

Solution = $S \cup Switch((p_i, VS_{D_iH}))$

- Either you can find a solution to $D$ in $H$, or you can split $D$ and find solutions for subsets $D_i \subset D$
- Partition into subsets $D_i$ must satisfy the following properties:
  - $VS_{D_iH} \neq \emptyset$
  - Partition is minimal
  - You can learn predicates $p_i$ to create the partitions
## Learning Trace Expressions

<table>
<thead>
<tr>
<th>Full Name</th>
<th>Title Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rob Miller</td>
<td>Mr. Rob</td>
</tr>
<tr>
<td>Saman Amarsinghe</td>
<td></td>
</tr>
<tr>
<td>Sumit Gulwani</td>
<td></td>
</tr>
<tr>
<td>Armando Solar-Lezama</td>
<td></td>
</tr>
<tr>
<td>Martin Rinard</td>
<td></td>
</tr>
</tbody>
</table>

Slide by Rishabh Singh
Learning Trace Expressions

Top level is always concatenation
Concat Expression (Associative)

Rob Miller \rightarrow Mr. Rob
Concat Expression (Associative)

Slide by Rishabh Singh
Concat Expression (Associative)
Mr. Rob
Concat Expression (Associative)
DAG-based Sharing

Any Path is a valid program

\[ \gamma_1 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \quad \gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \quad \gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \quad \gamma_1 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \quad \gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \quad \gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7 \]

Exponential Number of paths

Slide by Rishabh Singh
Learning atomic expressions

Can only be constants, substring or loop expressions
Constants are trivial to learn
Substring can be factored!
$145.67 \rightarrow 145.67$

**Substring Expression**

\[ \text{Substr(left, right)} \]

1 \rightarrow 7

- dollar, $\epsilon$, 1
- dollar, $\epsilon$, -1
- $\epsilon$, decimal, -1
- $\epsilon$, number, 1
- ... constant 1

- $\epsilon$, End of line, 1
- alphanumeric, $\epsilon$, -1
- decimal, $\epsilon$, -1
- ... constant 7

---

*Slide by Rishabh Singh*
Ranking

Prefer shorter programs.
  - Fewer number of conditionals.
  - Shorter string expression, regular expressions.

Prefer programs with fewer constants.

Strategies

**Baseline**: Pick any minimal sized program using minimal number of constants.

**Manual**: Break conflicts using a weighted score of program features.

**Machine Learning**: Weights are learned from training data.

Adapted from slide by Sumit Gulwani
Experimental Comparison of various Ranking Strategies

Reference: Predicting a correct program in Programming by Example, Technical Report, Singh, Gulwani