Regular Combinators for String Transformations

Rajeev Alur  Adam Freilich  Mukund Raghothaman

ExCAPE Annual Meeting, 2014

Tuesday 11th March, 2014
String Transformations

... are all over the place

- Find and replace
  Rename variable foo to bar
String Transformations

... are all over the place

- Find and replace
  Rename variable foo to bar

- Spreadsheet macros
  Convert phone numbers like
  "(123) 456-7890" to
  "123-456-7890"
String Transformations

... are all over the place

- Find and replace
  Rename variable `foo` to `bar`
- Spreadsheet macros
  Convert phone numbers like “(123) 456-7890” to “123-456-7890”
- String sanitization
- ...
String Transformations
Tool and theory support

- Good tool support: Sed, Awk, Perl, domain-specific tools, ...
- But unsatisfactory theory ...
- Expressibility: Can I even express (favorite transformation) using (favorite tool)?
- Analysis questions:
  - Is the transformation well-defined for all inputs?
  - Does the output of the transformation always have some “nice” property? \( \forall \sigma, \text{ is it the case that } f(\sigma) \in L? \)
  - Are two transformations equivalent?
Historical Context

Regular languages

Beautiful theory

Regular expressions $\equiv$ DFA

Analysis questions (mostly) efficiently decidable

Lots of practical implementations
One-way transducers: Mealy machines

Folk knowledge

Two-way transducers strictly more powerful than one-way transducers

Think of $\sigma \mapsto \sigma^\text{rev}, \sigma \mapsto \sigma\sigma, \sigma \# \tau \mapsto \tau \# \sigma$
Regular String Transformations

- Pick two-way finite state transducers as notion of regularity
- Nice properties
  - Decidable equivalence and type-checking (Does $\sigma \in L$ imply $f(\sigma) \in L'$?)
  - Closed under composition and regular look-ahead
  - Equivalent characterization as MSO
- Long hunt for single-pass representation
Regular String Transformations

Streaming string transducers

\[ b / \begin{align*}
    x & := xy \\
    y & := za \\
    z & := \epsilon
\end{align*} \]

Streaming string transducers

- Finite state space, registers
- Transitions test-free
- Registers concatenated, and values appended at register extremities
- Copyless updates
Regular String Transformations

\[
\text{DFA} \equiv \text{Regular expressions} \\
\text{SST} \equiv ?
\]
Can We Find an Equivalent Regex-like Characterization?
Function Combinators

Base functions: $L/\gamma$

If $\sigma \in L$, then $\gamma$, and otherwise undefined

$$(\text{".c" + ".cpp")/".cpp"}$$

Think of basic regular expressions: $\{a\}$, for $a \in \Sigma$
Function Combinators

If-then-else: $\text{ite } L f g$

If $\sigma \in L$, then $f(\sigma)$, and otherwise $g(\sigma)$

$\text{ite } [0 - 9]^* (\Sigma^*/\text{"Number"}) (\Sigma^*/\text{"Non-number"})$

Similar to unambiguous regex union
Split sum: $f \oplus g$

Split $\sigma$ into $\sigma = \sigma_1 \sigma_2$ with both $f(\sigma_1)$ and $g(\sigma_2)$ defined. If the split is unambiguous then $f \oplus g(\sigma) = f(\sigma_1)g(\sigma_2)$

Think of regex concatenation
Function Combinators

Iterated sum: $\sum f$

Split $\sigma = \sigma_1\sigma_2\ldots\sigma_k$, with all $f(\sigma_i)$ defined. If the split is unambiguous, then output $f(\sigma_1)f(\sigma_2)\ldots f(\sigma_k)$

- Kleene-*$^*$
- If $echo$ echoes a single character, then $\sum echo$ is the identity function
Function Combinators

Left-iterated sum: \( \sum f \)

Split \( \sigma = \sigma_1 \sigma_2 \ldots \sigma_k \), with all \( f(\sigma_i) \) defined. If the split is unambiguous, then output \( f(\sigma_k) f(\sigma_{k-1}) \ldots f(\sigma_1) \)

Think of \( \sigma \mapsto \sigma^{rev} \): \( \sum \text{echo} \)
Function Combinators

“Repeated” sum: $f + g$

$$f + g(\sigma) = f(\sigma)g(\sigma)$$

- No regex equivalent
- $\sigma \mapsto \sigma\sigma$: $id + id$
Function Combinators

Function composition: $f \circ g$

$$f \circ g(\sigma) = f(g(\sigma))$$

Regular string transformations are closed under composition
Theorem (Completeness)

All regular string transformations can be expressed using the combiners just described

$L/d, \text{ite}\ L f\ g, f \oplus g, \sum f, \sum f, f + g, f \circ g$
Conclusion

Introduced a declarative specification for regular string transformations
### Conclusion

#### Summary of operators

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Regular Expressions</th>
<th>Regular Transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>( { a }, \text{ for } a \in \Sigma )</td>
<td>( L/\gamma )</td>
</tr>
<tr>
<td>Union</td>
<td>( L_1 \cup L_2 )</td>
<td>( \text{ite } L f g )</td>
</tr>
<tr>
<td>Concatenation</td>
<td>( L_1 \cdot L_2 )</td>
<td>( f \oplus g )</td>
</tr>
<tr>
<td>Kleene-*</td>
<td>( L^* )</td>
<td>( \sum f ) (also ( \sum f ))</td>
</tr>
<tr>
<td>Repetition</td>
<td></td>
<td>( f + g )</td>
</tr>
<tr>
<td>Composition</td>
<td></td>
<td>( f \circ g )</td>
</tr>
</tbody>
</table>
Conclusion

Future Work

- Build lots of practical implementations (and analysis tools)
Conclusion
Future Work

▶ Build lots of practical implementations (and analysis tools)
▶ Theoretical questions about combinators
  ▶ How fast can we evaluate these functions?
  ▶ Lower bounds on expressibility of certain functions
Conclusion

Future Work

- Build lots of practical implementations (and analysis tools)
- Theoretical questions about combinators
  - How fast can we evaluate these functions?
  - Lower bounds on expressibility of certain functions
- Automatically learn transformations
  - from input/output examples
  - from teachers (L*)

Extensions to tree transducers
Processing hierarchical data, XML documents, etc.
Conclusion

Future Work

- Build lots of practical implementations (and analysis tools)
- Theoretical questions about combinators
  - How fast can we evaluate these functions?
  - Lower bounds on expressibility of certain functions
- Automatically learn transformations
  - from input/output examples
  - from teachers (L*)
- Extensions to tree transducers
  Processing hierarchical data, XML documents, etc.
Thank you! Questions?
Thank you! Questions? Suggestions? Brickbats?