REGULAR PROGRAMMING OVER DATA STREAMS

Mukund Raghothaman
April 27, 2015
AN INTRODUCTION TO DREX
THE CASE OF `swapBibtex`

```latex
\@book{Gal1638,
  publisher={Elzevir},
  place={Leiden},
  year={1638},
  title={Two New Sciences},
  author={Galileo},
}

\@book{Gal1638,
  publisher={Elzevir},
  place={Leiden},
  year={1638},
  title={Two New Sciences},
  author={Galileo},
}
```
THE CASE OF \textit{swapBibtex}

- \textit{swapEntry} moves the title of a single entry to the top
- \textit{swapBibtex} = \textit{iter}($\textit{swapEntry}$)
THE CASE OF `swapBibtex`

- `swapEntry` moves the title of a single entry to the top
- `swapBibtex = iter(swapEntry)`

```
\begin{tikzpicture}
  \node (e) {e};
  \node [left of=e] (titleOnly) {titleOnly};
  \node [right of=e] (allButTitle) {allButTitle};
  \node [left of=titleOnly] (titleOnly_e) {titleOnly(e)};
  \node [right of=allButTitle] (allButTitle_e) {allButTitle(e)};
  \path (e) -- (titleOnly_e);
  \path (e) -- (allButTitle_e);
\end{tikzpicture}
```

- `swapEntry = combine(titleOnly, allButTitle)`
We propose a *simple, expressive* programming model for string transformations, with:

1. strong theoretical foundations,
2. fast evaluation algorithms, and
3. tools for static analysis.
Languages, $\Sigma^* \to \text{bool} \equiv \text{Regular expressions}$

Transformations, $\Sigma^* \to \Gamma^* \equiv \text{DReX}$
· Expressively equivalent to regular string transformations
· Multiple characterizations: two-way finite state transducers, MSO-definable graph transformations, streaming string transducers
· Closed under various operations: function composition, regular look-ahead etc.
Streaming evaluation algorithm for consistent expressions

\[ f(\sigma) \text{ can be computed in time } O(poly(|f|) \cdot |\sigma|) \]
• Is the transformation well-defined for all inputs?
• Does the output always have some “nice” property? 
  \( \forall \sigma, \text{ is it the case that } f(\sigma) \in L? \)
• Are two transformations equivalent?
FUNCTION COMBINATORS
BASE FUNCTIONS: \( a \mapsto \gamma \)

Map the single character input string \( \sigma = a \) to \( \gamma \), and undefined everywhere else

\[ "a" \mapsto "Vowel" \]

Analogue of basic regular expressions: \( \{a\} \), for \( a \in \Sigma \)
If \( f(\sigma) \) is defined, then output \( f(\sigma) \), and otherwise output \( g(\sigma) \)

\[
[0-9]^* \mapsto \text{“Number”} \quad \text{else} \quad [a-z]^* \mapsto \text{“Name”}
\]

Analogue of unambiguous regex union
Split $\sigma$ into $\sigma = \sigma_1\sigma_2$ with both $f(\sigma_1)$ and $g(\sigma_2)$ defined. If the split is unambiguous then $\text{split}(f, g)(\sigma) = f(\sigma_1)g(\sigma_2)$

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

Kleene-*

If $echo$ echoes a single character, then $id = \text{iter}(echo)$ is the identity function
Split $\sigma = \sigma_1\sigma_2 \cdots \sigma_k$, with all $f(\sigma_i)$ defined. If the split is unambiguous, then output $f(\sigma_k)f(\sigma_{k-1}) \cdots f(\sigma_1)$

Think of string reversal: $left-iter(echo)$
“REPEATED” SUM: \( \text{combine}(f, g) \)

\[
\text{combine}(f, g)(\sigma) = f(\sigma)g(\sigma)
\]

No regex equivalent

\( \sigma \mapsto \sigma\sigma: \text{combine}(id, id) \)
...
Given two entries, \( e_1 \) and \( e_2 \), makeEntry outputs the title of \( e_2 \) and the remaining body of \( e_1 \)
alignBibtex = chain(makeEntry, R_{Entry})

Describing alignBibtex

\[ e_1 \quad e_2 \quad e_3 \quad \cdots \quad e_{k-1} \quad e_k \]

\[ \text{makeEntry}(e_1 e_2) \quad \text{makeEntry}(e_2 e_3) \quad \cdots \quad \text{makeEntry}(e_{k-1} e_k) \]

18
**CHAINED SUM:** $\text{chain}(f, R)$

And similarly for $\text{left-chain}(f, R)$
## Summary of Combinators

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Transformations</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>(\text{bottom}, \epsilon \mapsto \gamma, \ a \mapsto \gamma)</td>
<td>(\emptyset, {\epsilon}, {a})</td>
</tr>
<tr>
<td>Concatenation</td>
<td>(\text{split}(f, g), \text{left-split}(f, g))</td>
<td>(R_1 \cdot R_2)</td>
</tr>
<tr>
<td>Union</td>
<td>(f \text{ else } g)</td>
<td>(R_1 \cup R_2)</td>
</tr>
<tr>
<td>Kleene-*</td>
<td>(\text{iter}(f), \text{left-iter}(f))</td>
<td>(R^*)</td>
</tr>
<tr>
<td>Repetition</td>
<td>(\text{combine}(f, g))</td>
<td></td>
</tr>
<tr>
<td>Chained sum</td>
<td>(\text{chain}(f, R), \text{left-chain}(f, R))</td>
<td>New!</td>
</tr>
</tbody>
</table>
Sequence of bids from an auction

...;
Bid $25; Bid $18; Bid $42; Bid $37; Sold;
Bid $32; Bid $19; Bid $29; Sold;
...

Potential query: “What was the lowest bid to ever win?”
Loosely inspired by NEXMark (Tucker et al 2002)
... Card #217 swiped at entrance
Door opens
Person detected in doorway
Door closes... 

... Person #217 enters building
...
String-to-string transformations (Completed work)

1. Evaluation algorithms
2. Regular string transformations

Ongoing work

1. Static analysis tools
2. Quantitative properties
EVALUATION ALGORITHMS
THE ANATOMY OF A STREAMING EVALUATOR

((a, 1) (b, 2) (b, 3) (a, 4) (b, 5) \( (\sigma_n, n) \))

\[ \left( \begin{array}{c}
\text{Evaluator for } f \\
(\sigma_i, i) \\
\end{array} \right) \rightarrow (Result, \gamma) \]

\[ \left( \begin{array}{c}
(\text{Result}, \gamma) \\
(\text{Result}, \gamma') \\
\end{array} \right) \]
THE CASE OF $\text{split}(f, g)$

\[ T_{\text{split}(f,g)} \]

\[ f \text{ defined} \quad \quad g \text{ defined} \]

\[
\begin{array}{cccc}
1 & i & j & n \\
\end{array}
\]

\[ T_f \xrightarrow{(\sigma_i, i)} (\text{Result}, \gamma) \]

\[ T_g \xrightarrow{(\sigma_i, i)} (\text{Result}, \gamma) \]
THE CASE OF $\text{split}(f, g)$

$f$ defined \hspace{5cm} $g$ defined

$1 \quad i \quad j \quad n$

$(\text{Start, } i) \xrightarrow{T_f} (\text{Result, } \gamma)$ \hspace{2cm} $(\text{Start, } i) \xrightarrow{T_g} (\text{Result, } \gamma)$

$(\sigma_i, i) \xrightarrow{T_f} (\text{Result, } \gamma)$

$(\sigma_i, i) \xrightarrow{T_g}$
THE CASE OF $\text{split}(f, g)$

\[
\begin{align*}
&\text{f defined} \quad \text{g defined} \\
&\begin{cases}
1 & \text{f defined} \\
\ \ \ i & \ \ \ j & \ \ \ n
\end{cases}
\end{align*}
\]

\[
\begin{align*}
(\text{Start, } i) & \rightarrow T_f & (\text{Result, } \gamma) & \rightarrow (\text{Start, } i) \\
(\sigma_i, i) & \rightarrow T_f & \quad & \rightarrow T_g \\
& \rightarrow (\text{Result, } \gamma)
\end{align*}
\]
THE CASE OF $\text{split}(f, g)$

$T_{\text{split}(f, g)}$

\[
\begin{array}{ccc}
1 & f \text{ defined} & i \\
\hline
& j & n \\
\end{array}
\]

\[
(\text{Start}, i) \rightarrow T_f \rightarrow (\text{Result}, j, \gamma)
\]

\[
(\sigma_i, i) \rightarrow T_f \rightarrow (\text{Result}, j, \gamma)
\]

\[
(\text{Start}, i) \rightarrow T_g \rightarrow (\text{Result}, j, \gamma)
\]

\[
(\sigma_i, i) \rightarrow T_g \rightarrow (\text{Result}, j, \gamma)
\]
The case of \( \text{split}(f, g) \)

\[
\begin{array}{ccc}
\text{\( f \) defined} & \text{\( g \) defined} \\
1 \begin{array}{ccc}
\text{\( f \) defined} & \text{\( i \)} & j \\
\hline
n
\end{array}
\end{array}
\]

\((\text{Start, } i)\) \rightarrow \text{\( T_f \)} \rightarrow (\text{Result, } j, \gamma) \rightarrow (\text{Start, } i) \rightarrow \text{\( T_g \)} \rightarrow (\text{Result, } j, \gamma)

<table>
<thead>
<tr>
<th>Thread starting at index</th>
<th>Index at which ( T_f ) responded</th>
<th>Result reported by ( T_f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>\textit{aaab}</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>\textit{abbab}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Thread starting at index: 2
Index at which \( T_f \) responded: 9
Result reported by \( T_f \): \textit{aaab}

Thread starting at index: 3
Index at which \( T_f \) responded: 7
Result reported by \( T_f \): \textit{abbab}

Thread starting at index: ...
Index at which \( T_f \) responded: ...
Result reported by \( T_f \): ...
THE CASE OF \( \text{split}(f, g) \)

\[
\begin{array}{c}
\text{\( f \) defined} \\
1 \quad \text{\( f \) defined} \\
\quad i \\
\quad j \\
\quad n
\end{array}
\]

\[
\begin{array}{c}
\text{\( g \) defined}
\end{array}
\]

\[
\begin{array}{c}
(\text{Start}, i) \rightarrow T_f \\
(\sigma, i) \rightarrow T_f
\end{array}
\]

\[
\begin{array}{c}
(\text{Result}, j, \gamma) \\
(Kill, j)
\end{array}
\]

\[
\begin{array}{c}
(\text{Start}, i) \rightarrow T_g \\
(\sigma, i) \rightarrow T_g
\end{array}
\]

\[
\begin{array}{c}
(\text{Result}, j, \gamma) \\
(Kill, j)
\end{array}
\]

<table>
<thead>
<tr>
<th>Thread starting at index</th>
<th>Index at which ( T_f ) responded</th>
<th>Result reported by ( T_f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>\text{aaab}</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>\text{abbab}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
THE CASE OF $split(f, g)$

- Consistency assumed in correctness proof
- $split(f, g)$ is consistent iff
  - both $f$ and $g$ are consistent, and
  - their domains are unambiguously concatenable
- Statically disallow two threads of $T_g$ simultaneously reporting results

\[
\begin{align*}
\text{f defined} & \quad \text{g defined} \\
\hline
\text{f defined} & \quad \text{g defined}
\end{align*}
\]
Theorem (POPL 2015)

1. Consistency can be checked in $O(poly(|f|, |\Sigma|))$.
2. If $f$ is a consistent function expression, then $f(\sigma)$ can be computed in time $O(poly(|f|) \cdot |\sigma|)$.
EXPERIMENTAL PERFORMANCE

![Graph showing experimental performance](image-url)

- **Input string length**
- **Evaluation time (s)**

Legend:
- `deleteComments`
- `insertQuotes`
- `getTags`
- `reverse`
- `swapBibtex`
- `alignBibtex`
REGULAR STRING TRANSFORMATIONS
Languages, $\Sigma^* \rightarrow \text{bool} \equiv$ Finite automata

Transformations, $\Sigma^* \rightarrow \Gamma^* \equiv$ Finite state transducers
One-way transducers: Mealy machines

Folk knowledge (Aho et al 1969)

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

Gap includes many transformations of interest

String reversal, copy, substring swap, etc.
Known results

- Closed under composition (Chytil, Jákl 1977)
- Decidable equivalence checking (Gurari 1980)
- Equivalent to MSO-definable string transformations (Engelfriet, Hoogeboom 2001)

Recent result: Equivalent one-way deterministic model with applications to the analysis of list-processing programs (Alur, Černý 2011)

Streaming string transducers are our notion of regularity
STREAMING STRING TRANSDUCERS (SST)

If input ends with a \( b \), then reverse, else identity

- \( str \) contains the input string seen so far
- \( rev \) contains the reverse
STREAMING STRING TRANSDUCERS (SST)

- Finitely many locations
- Finite set of registers
- Transitions test-free
- Registers concatenated (copyless updates only)
- Final states associated with output functions

\[
\begin{align*}
\text{Output } str & : \text{ } str := str \cdot a \\
\text{Output } rev & : \text{ } rev := a \cdot rev \\
\text{Output } str & : \text{ } str := str \cdot b \\
\text{Output } rev & : \text{ } rev := b \cdot rev
\end{align*}
\]
Theorem (LICS 2014)

All regular string transformations can be expressed as a consistent function expression using:

1. **Base functions**: bottom, $\epsilon \mapsto \gamma$, $a \mapsto \gamma$,
2. $f$ else $g$, $\text{split}(f, g)$, $\text{combine}(f, g)$, and
3. **chained sums**: $\text{chain}(f, R)$, and $\text{left-chain}(f, R)$. 
REGULAR STRING TRANSFORMATIONS

A TASTE OF THE COMPLETENESS PROOF
Quick Recap

- \( Q = \{ q_1, q_2, \ldots, q_n \} \)
- Iterative algorithm
- In step \( i \), summarize all strings, \( q \rightarrow q_{\leq i}^* \rightarrow q' \)
SUMMARIZE EFFECT OF (INDIVIDUAL) STRINGS

1. $a$: $x := xy$, $y := a$, $z := zb$
2. $b$: $x := bxa$, $y := zy$, $z := a$

For $ab$: $x := bxya$, $y := zba$, $z := a$
SUMMARIZE EFFECT OF (INDIVIDUAL) STRINGS

\[
\begin{align*}
\text{ab} & \quad x := bxya \\
& \quad y := ab \\
\text{ba} & \quad x := bxa \\
& \quad y := yba
\end{align*}
\]

\[
\begin{align*}
\text{ab} & \quad x := bbxayb \\
& \quad y := b \\
\end{align*}
\]

\[
\begin{align*}
x := & \quad \longleftrightarrow x \quad \longleftrightarrow y \quad \longleftrightarrow \\
y := & \quad \longleftrightarrow \\
\end{align*}
\]
“Summarize” = “Give function expression for each patch”

\[ \begin{align*}
\gamma_{x1} : x & \leftrightarrow x \\
\gamma_{x2} : x & \leftrightarrow y \\
\gamma_{x3} : y & \leftrightarrow y
\end{align*} \]
Summarize with function expressions all paths $q \rightarrow q_{\leq i}^* \rightarrow q'$ with shape $S$

- Start with $i = 0$
- Iterative until $i = n$
- If shape of whole path of $S$, what can be the possible subpath shapes?
- Inner induction over shapes
Value appended to $x$ at the end of this loop iteration ($\gamma_1$) depends on value computed in $y$ during the previous iteration.

Chained sum
FROM TRANSDUCERS TO FUNCTION EXPRESSIONS

RECALL $\text{chain}(f, R)$

\[ f(\sigma_1 \sigma_2) \quad f(\sigma_2 \sigma_3) \quad f(\sigma_3 \sigma_4) \quad \ldots \quad f(\sigma_{k-1} \sigma_k) \]
ONGOING WORK
String-to-string

- Achieving expressive parity
- Fast evaluation algorithms
- Practical static analysis
- Parallel evaluation
- Tightening the completeness proof

Stream-to-cost

- What are regular cost functions?
- Obtaining the calculus
- Fast evaluation algorithms
ONGOING WORK

STATIC ANALYSIS TOOLS
Precondition computation: Given $f, L$, find $\sigma$ so that $f(\sigma) \in L$

“Does this sanitizer ever emit an unescaped backslash character?”

“Do login and logout events always alternate?”

PSPACE-complete

Equivalence checking: For all $\sigma$, is it true that $f_1(\sigma) = f_2(\sigma)$?

PSPACE
#!/usr/bin/env bash
./run-experiments
for f in ‘ls *.tmp’
do
  BASE=‘echo $f | sed s/\.[^\./\]*$///’
  ./process-log "$BASE.log" >> outfile
  rm "$BASE”* 
done
· Implementation of precondition computation and equivalence checking routines
· Suspicious program identifier for Bash scripts
ONGOING WORK

QUANTITATIVE FUNCTION EXPRESSIONS
Sequence of bids from an auction

...;
Bid $25; Bid $18; Bid $42; Bid $37; Sold;
Bid $32; Bid $19; Bid $29; Sold;
...

“What was the lowest bid to ever win?”

\[
\text{winBid} = \text{split-plus}(\text{iter-max}(\text{Bid } n \mapsto n), \text{Sold} \mapsto 0)
\]

\[
\text{winBid}_{\text{low}} = \text{iter-min}(\text{winBid})
\]
QUANTITATIVE FUNCTION EXPRESSIONS

![Graph showing price variations over time]

- *Signal* ∈ \{up, down, month\}
- Current price
- Historical low

**Day #**

<table>
<thead>
<tr>
<th>Day #</th>
<th>Signal</th>
<th>Current price</th>
<th>Historical low</th>
</tr>
</thead>
</table>
Signal sequence $\sigma \in \{\text{up}, \text{down}, \text{month}\}^*$. What is the largest intra-month price swing?

$$
p = \text{iter-plus}(\text{up} \mapsto 1 \text{ else } \text{down} \mapsto -1)
$$

$$
m_{hi} = \text{split-plus}(\text{max-prefix}(p), \text{month} \mapsto 0)
$$

$$
m_{lo} = \text{split-plus}(\text{min-prefix}(p), \text{month} \mapsto 0)
$$

$$
\text{bigSwing} = \text{iter-max}(\text{minus}(m_{hi}, m_{lo}))
$$
Fixing the calculus

1. What are regular cost functions? (LICS 2013)
2. Choosing combinators to achieve expressive parity (Conjectures)

Fast evaluation algorithms
- Parameterized by:
  1. cost domain $\mathbb{D}$, and
  2. operations $G = \{+,-,\min,\max,\ldots\}$

- Stream-to-term function implicitly defines a stream-to-cost function

```
Bid $n/\quad wb := \max(wb, n)$

Sold $/\quad wb_{lo} := \min(wb_{lo}, wb)$

$wb := 0$
```
Appealing properties

Equivalent to MSO-definable string-to-term transformations
Closed under choice, regular look-ahead, input reversal, etc.

Function grammars

\{\text{min}, +\} \text{ semiring over } \mathbb{Z} \text{ or } \mathbb{N}

\{\text{max, min, +}\} \text{ over } \mathbb{Z} \cup \{\pm \infty\}

\{\cdot \leq \cdot \ ?, \cdot : \cdot, +\} \text{ over } \mathbb{Z} \cup \{\pm \infty\}
## Quantitative Function Combinators

<table>
<thead>
<tr>
<th>Purpose</th>
<th>String-to-string</th>
<th>String-to-cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base</strong></td>
<td><em>bottom, $\epsilon \mapsto d$, $a \mapsto d$</em></td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td>$(\text{left-})\text{split}(f, g)$</td>
<td>$\text{split-min}(f, g)$, $\text{split-plus}(f, g)$</td>
</tr>
<tr>
<td>Union</td>
<td>$f \text{ else } g$</td>
<td>$f \text{ else } g$</td>
</tr>
<tr>
<td>Kleene-*</td>
<td>$(\text{left-})\text{iter}(f)$</td>
<td>$\text{iter-min}(f)$, $\text{iter-plus}(f)$</td>
</tr>
<tr>
<td>Repetition</td>
<td>$\text{combine}(f, g)$</td>
<td>$\min(f, g)$, $\sum(f, g)$</td>
</tr>
<tr>
<td>Chained sum</td>
<td>$(\text{left-})\text{chain}(f, R)$</td>
<td></td>
</tr>
<tr>
<td>Prefix / suffix</td>
<td></td>
<td>$\text{min-prefix}(f)$, $\text{min-suffix}(f)$</td>
</tr>
</tbody>
</table>
• Identification of expressively equivalent combinator calculi for regular cost functions over:
  • \{\text{min}, +\},
  • \{\text{min}, \text{max}, +\}, and
  • \{\cdot \leq \cdot, ?:\cdot, +\}

• Fast evaluation algorithms for all these calculi
### Ongoing Work

<table>
<thead>
<tr>
<th>Activity</th>
<th>Need ... months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantitative function expressions</td>
<td>≈ 3</td>
</tr>
<tr>
<td>Practical static analysis tools</td>
<td>≈ 3</td>
</tr>
<tr>
<td>Parallel evaluation algorithms</td>
<td>2-3</td>
</tr>
<tr>
<td>Tightening the completeness proof</td>
<td>1</td>
</tr>
<tr>
<td>Thesis writing</td>
<td>2-3</td>
</tr>
</tbody>
</table>

Not mentioned in timeline: Front-end improvements, proof fixes, new basic combinator (*restrict*), etc.
RELATED WORK
Well-established research area

Quantile computation (Munro, Paterson 1980)
Counting distinct elements (Flajolet, Martin 1984)
Finding frequency moments (Alon et al 2006)
...

Textbooks: Muthukrishnan 2005

Orthogonal goals to us

Algorithms usually clearly streamable
Proof of correctness usually difficult
∙ Finite automata with edges annotated with numbers

∙ Semantics:
  ∙ Add the weights along each path
  ∙ Take the minimum over all paths

∙ Applicable to semirings

∙ Very mature research area: Krob 1992, Droste et al 2009, Almagor et al 2011, ...

∙ Regular cost functions over $\{\text{min}, +\}$ expressively equivalent to unambiguous weighted automata
· Various languages for data structured as XML documents
· Querying: XPath, XQuery
· Transformations: XSLT
  · Turing-complete
  · Potentially unbounded evaluation complexity
  · Static analysis is hard
Systems: Aurora (Abadi et al 2003), STREAM (Arasu et al 2003), Niagara (Chen et al 2000)

Query languages: Continuous Query Language (Arasu et al 2006)

Rich features: Multiple streams, query composition, etc.

Sliding windows: Disallows general regular look-ahead

Interesting source of example applications: Linear Road, NEXMark (Tucker et al 2002)
• Vaziri et al 2014
• Calculus to express stream transformations with spreadsheets
• Basic combinators include switches (@) and latches (latch)
  • $s_1@s_2$ produces the next element of $s_1$ whenever $s_2$ evaluates to true
  • $latch(s_1, s_2)$ ticks whenever $s_2$ does, and produces the value of $s_1$ at the last tick of $s_2$
• Conjecture: Equivalent to append-on-the-right SSTs
THANK YOU!
BACKUP SLIDES
• (a \mapsto \gamma)-style base functions do not scale well Unicode, data payloads, etc.

Map single character input strings \( \sigma \) which satisfy \( \varphi \) to \( d(\sigma) \), and undefined everywhere else

\[
isLowerCase(x) \mapsto toUpperCase(x)
\]

\( \varphi \) is a character predicate, possibly symbolic
\( d : \Sigma \to \Gamma^* \) is a character-to-string transformation

Consistent iff \( \varphi \) is satisfiable
SUMMARY OF CONSISTENCY RULES

PART 1

- **bottom, \( \epsilon \rightarrow \gamma, a \rightarrow \gamma \) always consistent**
- **split\((f, g)\) and left-split\((f, g)\) are consistent iff**
  - \( f \) and \( g \) are consistent, and
  - \( \text{Dom}(f) \) and \( \text{Dom}(g) \) are unambiguously concatenable
- **\( f \) else \( g \) is consistent iff**
  - \( f \) and \( g \) are consistent, and
  - \( \text{Dom}(f) \) and \( \text{Dom}(g) \) are disjoint
- **combine\((f, g)\) is consistent iff**
  - \( f \) and \( g \) are consistent, and
  - \( \text{Dom}(f) = \text{Dom}(g) \)
\[ \text{iter}(f) \text{ and } \text{left-iter}(f) \text{ are consistent iff } \]
\[ \begin{align*}
&\cdot \ f \text{ is consistent, and} \\
&\cdot \ \text{Dom}(f) \text{ is unambiguously iterable}
\end{align*} \]

\[ \text{chain}(f, R) \text{ and } \text{left-chain}(f, R) \text{ are consistent iff } \]
\[ \begin{align*}
&\cdot \ f \text{ is consistent, } R \text{ is an unambiguous regular expression,} \\
&\cdot \ \text{Dom}(f) \text{ is unambiguously iterable, and} \\
&\cdot \ \text{Dom}(f) = \lfloor R \cdot R \rfloor
\end{align*} \]
MORE ONGOING WORK
MORE ONGOING WORK

PARALLEL EVALUATION ALGORITHMS
A simple NFA evaluation algorithm

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

String \( \sigma \) summarized by \( n \times n \) boolean matrix \( M_\sigma \)

Entry \( e_{ij} \) true iff \( q_i \) can reach \( q_j \)

\[ M_{\sigma_1 \sigma_2} = M_{\sigma_1} M_{\sigma_2} \]
A simple NFA evaluation algorithm

\[ Q = \{q_1, q_2, \ldots, q_n\}\]

String \(\sigma\) summarized by \(n \times n\) boolean matrix \(M_\sigma\)

Entry \(e_{ij}\) true iff \(q_i\) can reach \(q_j\)

\[ M_{\sigma_1 \sigma_2} = M_{\sigma_1} M_{\sigma_2}\]

Can we do something similar for function expressions?

Applications to compression / decompression algorithms, etc.
MORE ONGOING WORK

TIGHTENING THE COMPLETENESS PROOF
<table>
<thead>
<tr>
<th>split</th>
<th>left-split</th>
<th>else</th>
<th>iter</th>
<th>left-iter</th>
<th>combine</th>
<th>chain</th>
<th>left-chain</th>
<th>What’s inexpressible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Complete</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td></td>
<td>( \sigma \mapsto \sigma^\text{rev} )</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
<td>( \sigma \mapsto \sigma \sigma )</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>shuffle, alignBibtex</td>
</tr>
</tbody>
</table>