REGULAR PROGRAMMING OVER DATA STREAMS

Mukund Raghothaman

April 27, 2015

AN INTRODUCTION TO DREX

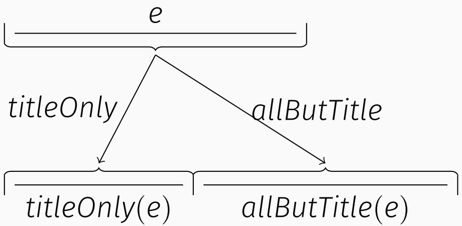
THE CASE OF *swapBibtex*

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

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

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

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



- $swapEntry = combine(titleOnly, allButTitle)$

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.

$$\text{Languages, } \Sigma^* \to \texttt{bool} \quad \equiv \quad \text{Regular expressions}$$
$$\text{Tranformations, } \Sigma^* \to \Gamma^* \quad \equiv \quad \text{DReX}$$

- 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

$f(\sigma)$ 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?
  $\forall \sigma$, is it the case that $f(\sigma) \in L$?
- Are two transformations equivalent?

FUNCTION COMBINATORS

Map the single character input string $\sigma = a$ to γ, and undefined everywhere else

“`a`” $\mapsto$ “`Vowel`”

Analogue of basic regular expressions: $\{a\}$, for $a \in \Sigma$

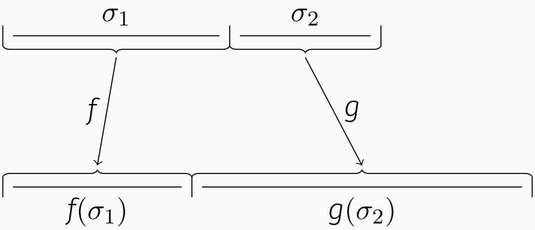
CONDITIONALS: *f else g*

If $f(\sigma)$ is defined, then output $f(\sigma)$, and otherwise output $g(\sigma)$

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

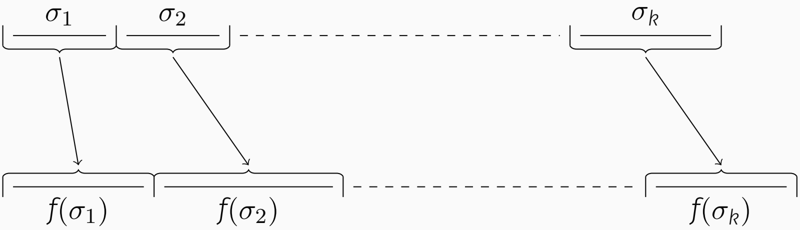
Analogue of unambiguous regex union

Split σ into $\sigma = \sigma_1\sigma_2$ with both $f(\sigma_1)$ and $g(\sigma_2)$ defined. If the split is unambiguous then $\mathit{split}(f, g)(\sigma) = f(\sigma_1)g(\sigma_2)$



Analogue of regex concatenation

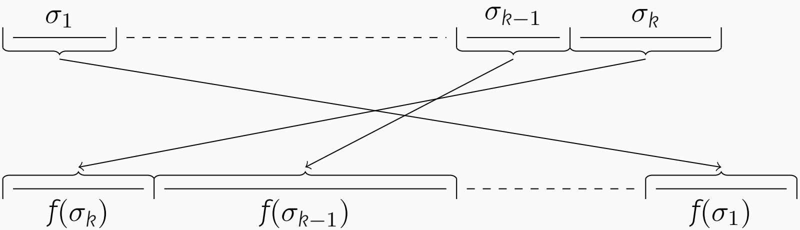
Split $\sigma = \sigma_1\sigma_2\cdots\sigma_k$, with all $f(\sigma_i)$ defined. If the split is unambiguous, then output $f(\sigma_1)f(\sigma_2)\cdots f(\sigma_k)$



Kleene-*

If *echo* echoes a single character, then $id = iter(echo)$ is the identity function

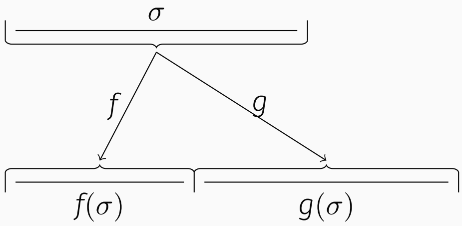
Split $\sigma = \sigma_1\sigma_2\cdots\sigma_k$, with all $f(\sigma_i)$ defined. If the split is unambiguous, then output $f(\sigma_k)f(\sigma_{k-1})\cdots f(\sigma_1)$



Think of string reversal: *left-iter*(*echo*)

“REPEATED” SUM: *combine*(f, g)

$$combine(f, g)(\sigma) = f(\sigma)g(\sigma)$$



No regex equivalent

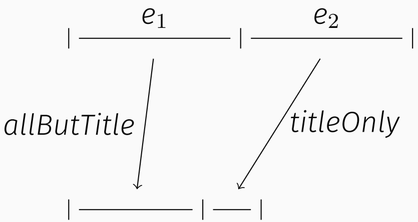
$\sigma \mapsto \sigma\sigma$: $combine(id, id)$

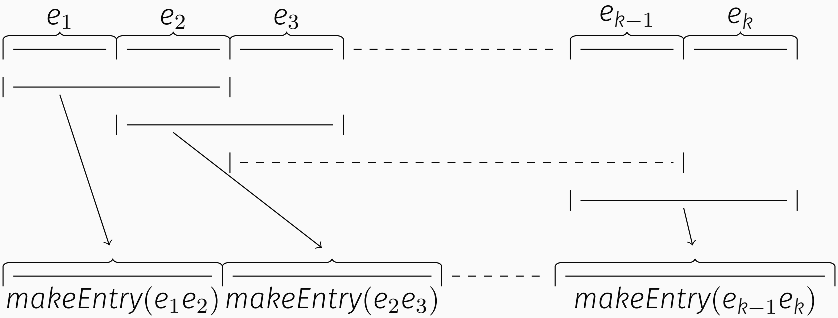
THE CASE OF *alignBibtex*

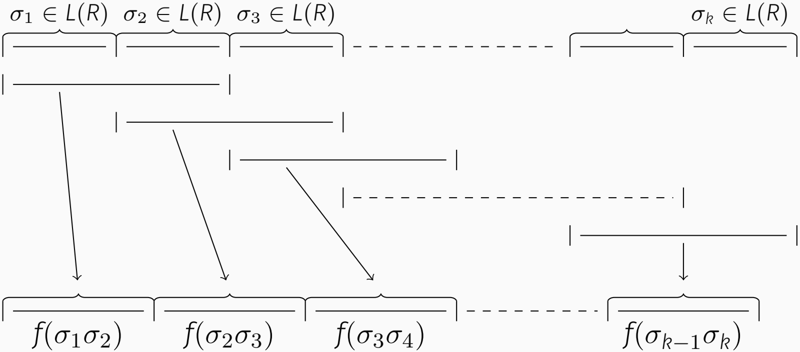

```
...

@book{Book1,
  title = {Title0},
  author = {Author1},
  year = {Year1},
}

@book{Book2,
  title = {Title1},
  author = {Author2},
  year = {Year2},
}

...
```

```
...

@book{Book1,
  title = {Title1},
  author = {Author1},
  year = {Year1},
}

...
```

Given two entries, e_1 and e_2, *makeEntry* outputs the title of e_2 and the remaining body of e_1




e_1
e_2
e_3
e_{k-1}
e_k
$makeEntry(e_1 e_2)$
$makeEntry(e_2 e_3)$
$makeEntry(e_{k-1} e_k)$



And similarly for *left-chain*(f, R)

Purpose	Transformations	Languages
Base	*bottom*, $\epsilon \mapsto \gamma$, $a \mapsto \gamma$	$\emptyset$, $\{\epsilon\}$, $\{a\}$
Concatenation	*split*(f, g), *left-split*(f, g)	$R_1 \cdot R_2$
Union	f *else* g	$R_1 \cup R_2$
Kleene-*	*iter*(f), *left-iter*(f)	R^*
Repetition	*combine*(f, g)	New!
Chained sum	*chain*(f, R), *left-chain*(f, R)	

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)

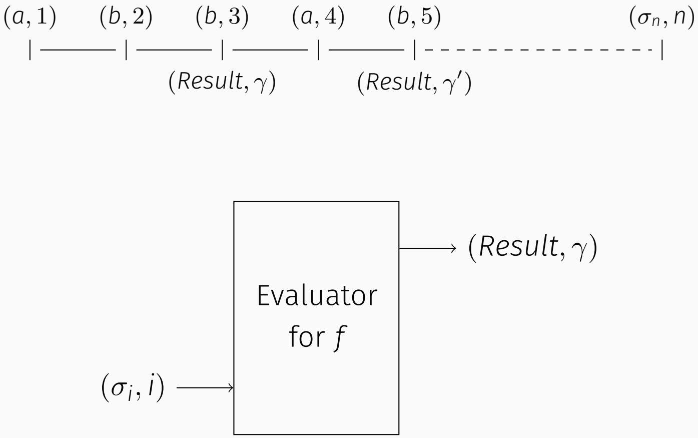
1. Evaluation algorithms
2. Regular string transformations

Ongoing work

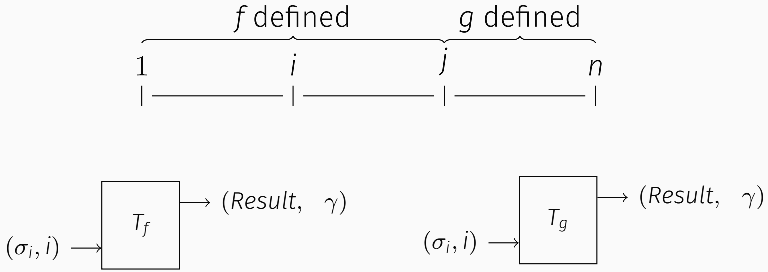
1. Static analysis tools
2. Quantitative properties

EVALUATION ALGORITHMS

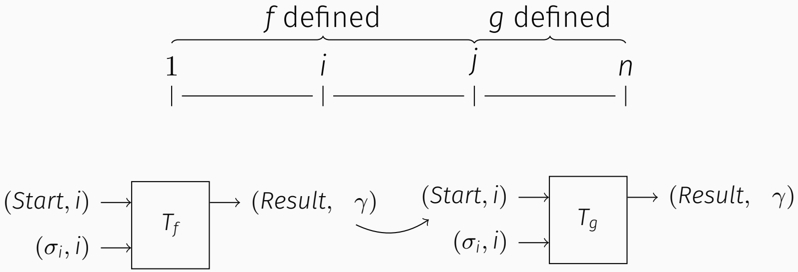
THE ANATOMY OF A STREAMING EVALUATOR



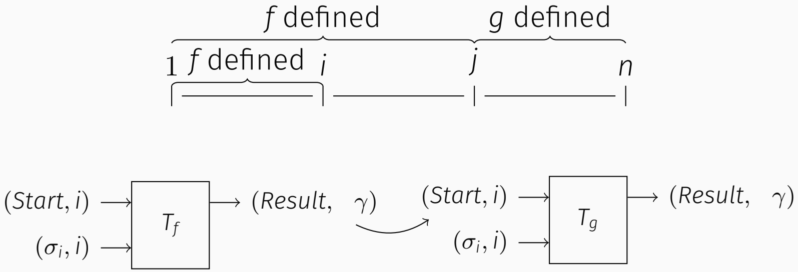
THE CASE OF $split(f,g)$

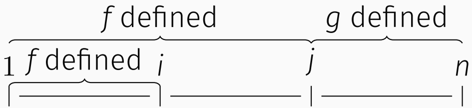


THE CASE OF $split(f, g)$

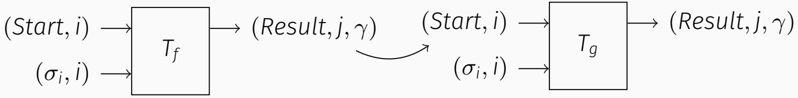


THE CASE OF $split(f, g)$



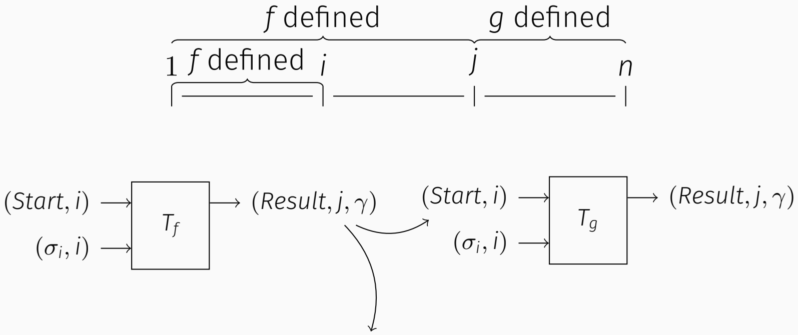


f defined
g defined
1
f defined
i
j
n

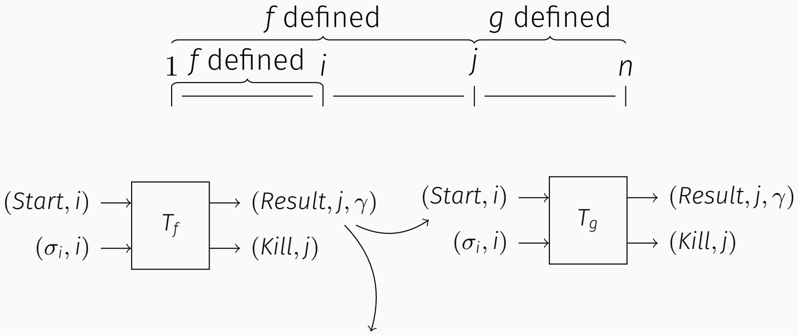


$(Start, i)$
(σ_i, i)
T_f
$(Result, j, \gamma)$
$(Start, i)$
(σ_i, i)
T_g
$(Result, j, \gamma)$

THE CASE OF $split(f, g)$



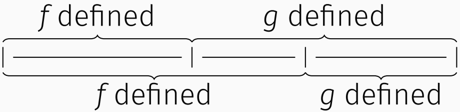
Thread starting at index	Index at which T_f responded	Result reported by T_f
2	9	*aaab*
3	7	*abbab*
...	...	...



Thread starting at index	Index at which T_f responded	Result reported by T_f
2	9	*aaab*
3	7	*abbab*
...	...	...

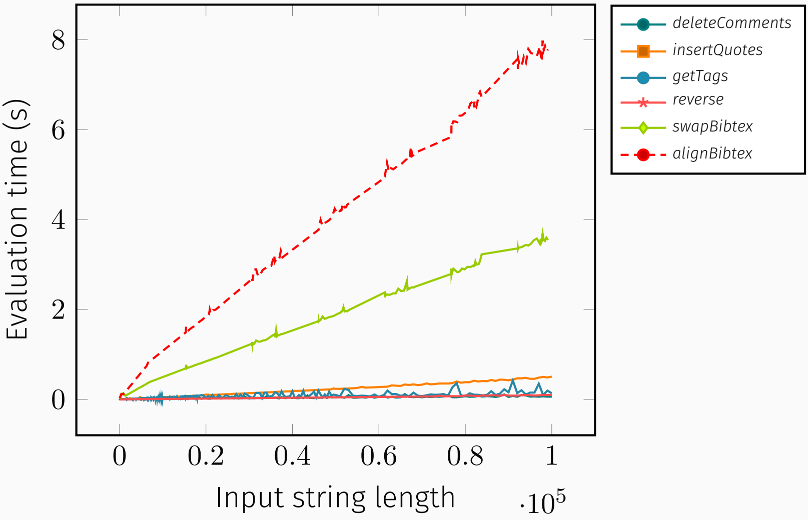
THE CASE OF $split(f, g)$

- 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



Theorem (POPL 2015)

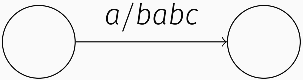
1. *Consistency can be checked in* $O(poly(|f|, |\Sigma|))$.
2. *If* f *is a consistent function expression, then* $f(\sigma)$ *can be computed in time* $O(poly(|f|) \cdot |\sigma|)$.


Evaluation time (s)
8
6
4
2
0
0
0.2
0.4
0.6
0.8
1
Input string length
$\cdot 10^5$
deleteComments
insertQuotes
getTags
reverse
swapBibtex
alignBibtex

REGULAR STRING TRANSFORMATIONS

Languages, $\Sigma^* \to \texttt{bool}$	$\equiv$	Finite automata
Tranformations, $\Sigma^* \to \Gamma^*$	$\equiv$	Finite state transducers

One-way transducers: Mealy machines



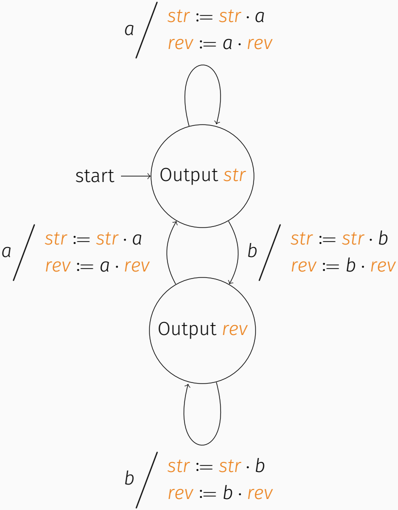
Folk knowledge (Aho et al 1969)

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

Gap includes many transformations of interest

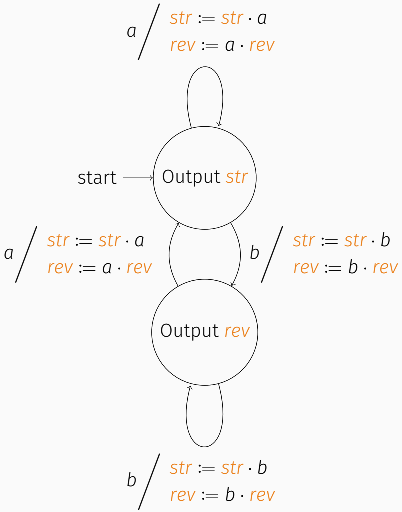
String reversal, copy, substring swap, etc.

- 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



If input ends with a b, then reverse, else identity

- *str* contains the input string seen so far
- *rev* contains the reverse



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

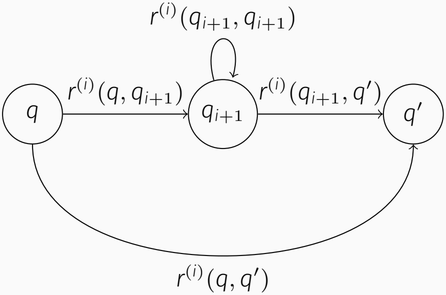
Theorem (LICS 2014)

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, *split*(f, g), *combine*(f, g), *and*
3. *chained sums: chain*(f, R), *and left-chain*(f, R).

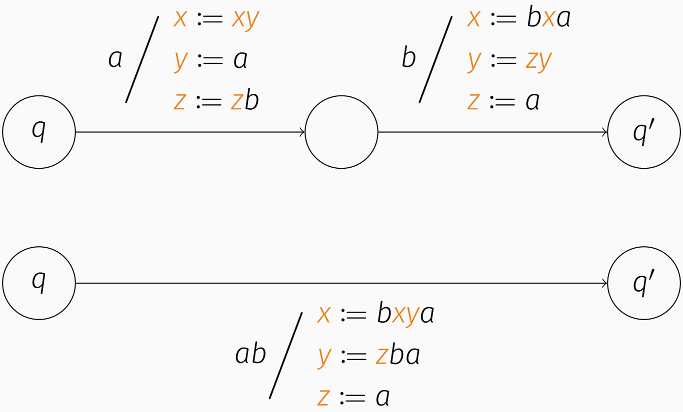
REGULAR STRING TRANSFORMATIONS

A TASTE OF THE COMPLETENESS PROOF

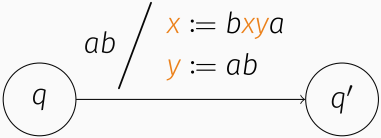


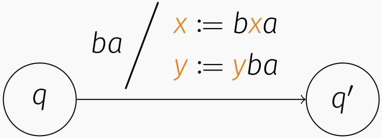
- $Q = \{q_1, q_2, \ldots, q_n\}$
- Iterative algorithm
- In step i, summarize all strings, $q \to q^*_{\leq i} \to q'$

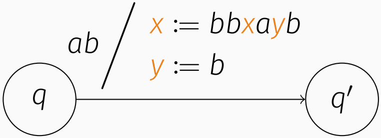
SUMMARIZE EFFECT OF (INDIVIDUAL) STRINGS



SUMMARIZE EFFECT OF (INDIVIDUAL) STRINGS







$x := \longleftrightarrow x \longleftrightarrow y \longleftrightarrow$

$y := \longleftrightarrow$

$x := \longleftrightarrow x \longleftrightarrow$

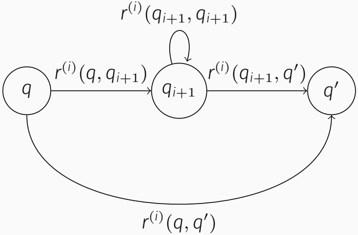
$y := \longleftrightarrow y \longleftrightarrow$

"Summarize" = "Give function expression for each patch"

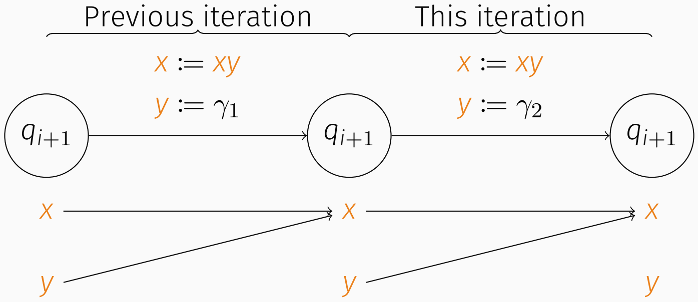
$$x := \xleftrightarrow{\gamma_{x1}} x \xleftrightarrow{\gamma_{x2}} y \xleftrightarrow{\gamma_{x3}}$$

$$y := \xleftrightarrow{\gamma_{y1}}$$

Summarize with function expressions all paths $q \to q^*_{\leq i} \to q'$ with shape S

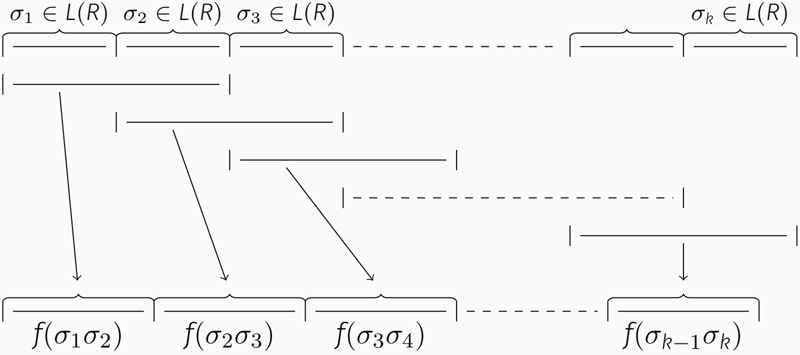


- Start with $i = 0$
- Iterative until $i = n$
- If shape of whole path of S, what can be the possible subpath shapes?
- Inner induction over shapes



Value appended to x at the end of *this* loop iteration (γ_1) depends on value computed in y during the *previous* iteration

Chained sum


$\sigma_1 \in L(R)$
$\sigma_2 \in L(R)$
$\sigma_3 \in L(R)$
$\sigma_k \in L(R)$
$f(\sigma_1\sigma_2)$
$f(\sigma_2\sigma_3)$
$f(\sigma_3\sigma_4)$
$f(\sigma_{k-1}\sigma_k)$

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 σ 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 σ, 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
```

DELIVERABLES

- Implementation of precondition computation and equivalence checking routines
- Suspicious program identifier for Bash scripts

ONGOING WORK

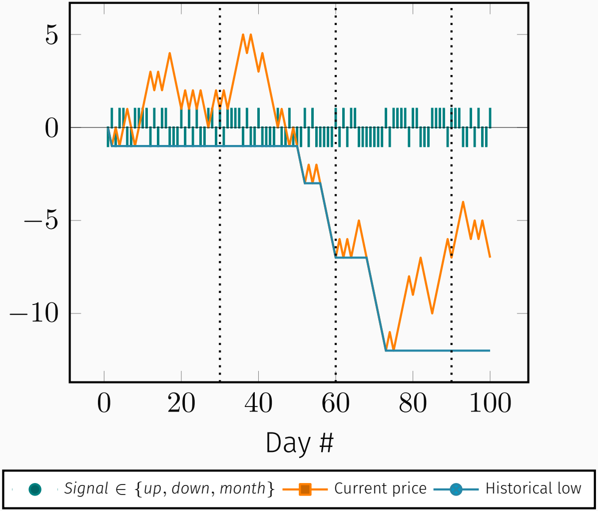
QUANTITATIVE FUNCTION EXPRESSIONS

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?"

$$\begin{aligned} \mathit{winBid} &= \mathit{split\text{-}plus}(\mathit{iter\text{-}max}(\texttt{Bid}\ \ n \mapsto n), \texttt{Sold} \mapsto 0) \\ \mathit{winBid}_{\mathit{low}} &= \mathit{iter\text{-}min}(\mathit{winBid}) \end{aligned}$$


5
0
−5
−10
0
20
40
60
80
100
Day #
Signal ∈ {up, down, month}
Current price
Historical low

Signal sequence $\sigma \in \{up, down, month\}^*$. What is the largest intra-month price swing?

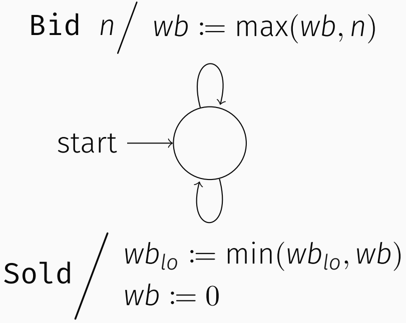
$$
\begin{aligned}
p &= \textit{iter-plus}(up \mapsto 1 \textit{ else } down \mapsto -1) \\
m_{hi} &= \textit{split-plus}(\textit{max-prefix}(p), month \mapsto 0) \\
m_{lo} &= \textit{split-plus}(\textit{min-prefix}(p), month \mapsto 0) \\
bigSwing &= \textit{iter-max}(minus(m_{hi}, m_{lo}))
\end{aligned}
$$

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



Appealing properties

Equivalent to MSO-definable string-to-term transformations

Closed under choice, regular look-ahead, input reversal, etc.

Function grammars

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

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

$\{\cdot \leq \cdot \,?\, \cdot : \cdot, +\}$ over $\mathbb{Z} \cup \{\pm\infty\}$

Purpose	String-to-string	String-to-cost
Base	*bottom*, $\epsilon \mapsto d$, $a \mapsto d$	
Concatenation	(*left*-)*split*(f, g)	*split-min*(f, g), *split-plus*(f, g)
Union	f *else* g	f *else* g
Kleene-*	(*left*-)*iter*(f)	*iter-min*(f), *iter-plus*(f)
Repetition	*combine*(f, g)	*min*(f, g), *sum*(f, g)
Chained sum	(*left*-)*chain*(f, R)	
Prefix / suffix		*min-prefix*(f), *min-suffix*(f)

- Identification of expressively equivalent combinator calculi for regular cost functions over:
  - $\{\min, +\}$,
  - $\{\min, \max, +\}$, and
  - $\{\cdot \leq \cdot \, ? \, \cdot : \cdot, +\}$
- Fast evaluation algorithms for all these calculi

Activity	Need … months
Quantitative function expressions	≈ 3
Practical static analysis tools	≈ 3
Parallel evaluation algorithms	2-3
Tightening the completeness proof	1
Thesis writing	2-3

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 $\{\min, +\}$ 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

BASE FUNCTIONS: $a \mapsto \gamma$ OR $\varphi \mapsto d$?

- ($a \mapsto \gamma$)-style base functions do not scale well
  Unicode, data payloads, etc.

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

$$\mathit{isLowerCase}(x) \mapsto \mathit{toUpperCase}(x)$$

φ is a character predicate, possibly symbolic

$d : \Sigma \to \Gamma^*$ is a character-to-string transformation

Consistent iff φ is satisfiable

- *bottom*, $\epsilon \mapsto \gamma$, $a \mapsto \gamma$ always consistent
- *split*(f, g) and *left-split*(f, g) are consistent iff
  - f and g are consistent, and
  - $Dom(f)$ and $Dom(g)$ are unambiguously concatenable
- f *else* g is consistent iff
  - f and g are consistent, and
  - $Dom(f)$ and $Dom(g)$ are disjoint
- *combine*(f, g) is consistent iff
  - f and g are consistent, and
  - $Dom(f) = Dom(g)$

- *iter*(f) and *left-iter*(f) are consistent iff
    - f is consistent, and
    - *Dom*(f) is unambiguously iterable
- *chain*(f, R) and *left-chain*(f, R) are consistent iff
    - f is consistent, R is an unambiguous regular expression,
    - *Dom*(f) is unambiguously iterable, and
    - $Dom(f) = [\![R \cdot R]\!]$

MORE ONGOING WORK

MORE ONGOING WORK

PARALLEL EVALUATION ALGORITHMS

A simple NFA evaluation algorithm

$Q = \{q_1, q_2, \ldots, q_n\}$

String σ summarized by $n \times n$ boolean matrix M_σ

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 σ summarized by $n \times n$ boolean matrix M_σ

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

TIGHTENING THE COMPLETENESS PROOF

split	*left-split*	*else*	*iter*	*left-iter*	*combine*	*chain*	*left-chain*	What's inexpressible?
Yes		Yes			Yes	Yes	Yes	Complete
Yes		Yes	Yes	No	Yes			$\sigma \mapsto \sigma^{rev}$
Yes	Yes	Yes	Yes	Yes	No			$\sigma \mapsto \sigma\sigma$
Yes	Yes	Yes	Yes	Yes	Yes	No	No	*shuffle*, *alignBibtex*