REGULAR PROGRAMMING OVER DATA STREAMS

Mukund Raghothaman (Joint work with Rajeev Alur, Loris D’Antoni, Dana Fisman, Adam Freilich and Arjun Radhakrishna)
ExCAPE PI Meeting, 2015
abc@gmail.com → gmail.com

- $swallowUserName = (ALNUM^*) \cdot @ \cdot ? \cdot \epsilon$
- $echoDomain = \text{Identity function over strings}$
- $getDomain = \text{split}(swallowUserName, echoDomain)$
“What was the highest winter-time temperature?”

...;
Temp 32°C;
AutumnEquinox);
Temp 27°C;
LeonidMeteorShowers;
Temp 26°C;
Temp 21°C;
WinterSolstice;
Temp 15°C;
CalendarYearEnd;
Temp 12°C;
SpringEquinox;
...

· allWinterHi = iter-max(annualWinterHi)
· The function annualWinterHi returns the highest winter-time temperature of one summer-winter cycle
“What was the highest winter-time temperature?”

...;
Temp 32°C;
AutumnEquinox;
Temp 27°C;
LeonidMeteorShowers;
Temp 26°C;
Temp 21°C;
WinterSolstice;
Temp 15°C;
CalendarYearEnd;
Temp 12°C;
SpringEquinox;
...

- \( allWinterHi = \) iter-max(annualWinterHi)
- The function annualWinterHi returns the highest winter-time temperature of one summer-winter cycle
- \( annualWinterHi = \) split-plus(winterHi, \( r_{summer} \) ? 0)
The customer and the bank

...; Deposit $32; Deposit $7; Deposit $9;
...; Withdraw $2; ...
EndOfMonth;

“What was the customer reward?”

\[ reward = \text{if} (\text{goodMonth}) \text{ then } \text{hiReward} \text{ else } \text{loReward} \]

The functions \textit{hiReward} and \textit{loReward} compute the customer reward in qualifying and non-qualifying months respectively.
DReX is a simple, expressive programming model for string transformations, with:

1. robust expressiveness,
2. fast evaluation algorithms, and
3. tools for static analysis
Small collection of core **combinators**

**Basic functions:**  \texttt{data ? cost} \\
**Conditional choice:**  \texttt{f else g}
Small collection of core combinators

Basic functions:  \( data \ ? \ cost \)
Conditional choice:  \( f \ else \ g \)

Cost operations:  \( op(f_1, f_2, \ldots, f_k) \)
What about concatenation?

\[ w_1 \in \mathbb{D}^* \]  \[ w_2 \in \mathbb{D}^* \]

\[ f \]  \[ g \]  \[ \text{Output} \]
What about concatenation?

\[ w_1 \in \mathbb{D}^* \quad \text{and} \quad w_2 \in \mathbb{D}^* \]

\[ f \quad x \quad g \quad \text{Output} \]
Small collection of core **combinators**

Basic functions: \( data ? cost \)

Conditional choice: \( f \text{ else } g \)

Cost operations: \( op(f_1, f_2, \ldots, f_k) \)
Small collection of core **combinators**

Basic functions: `data ? cost, regex ? term`

Conditional choice: `f else g`

Cost operations: `op(f_1, f_2, \ldots, f_k)`
Small collection of core **combinators**

**Basic functions:**  \( \text{data} ? \text{cost}, \text{regex} ? \text{term} \)

**Conditional choice:**  \( f \text{ else } g \)

**Concatenation:**  \( \text{split}(f \rightarrow^x g), \text{split}(f \leftarrow^x g) \)

**Cost operations:**  \( \text{op}(f_1, f_2, \ldots, f_k) \)
Small collection of core **combinators**

**Basic functions:** $data \; ? \; cost$, $regex \; ? \; term$

**Conditional choice:** $f \; else \; g$

**Concatenation:** $\text{split}(f \xhookrightarrow{X} g)$, $\text{split}(f \xhookleftarrow{X} g)$

**Function iteration:** $\text{iter}^{\rightarrow}(f)$, $\text{iter}^{\leftarrow}(f)$

**Cost operations:** $\text{op}(f_1, f_2, \ldots, f_k)$
Function descriptions are modular

isWinter: bool, currHi: int
while exists next event e:
  if e is AutumnEquinox:
    isWinter := true
  elif e is SpringEquinox:
    isWinter := false
  elif isWinter:
    currHi := max(currHi, e.temp)
DReX allows regular parsing of the input data stream

- Unrestricted global choice: $f \text{ else } g$
- Iteration patterns part of query, **not of the data**: iter-max(weeklyAverageTemp)
DREX IS A ...

“...WITH ROBUST EXPRESSIVENESS, ...”

- Expressively equivalent to regular string-to-term 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.
DREX is a …

“…with fast evaluation algorithms, …”

Input string length

Evaluation time (s)

- deleteComments
- insertQuotes
- getTags
- reverse
- swapBibtex
- alignBibtex

$10^3$
Over arbitrary cost domains \((\mathbb{D}, G)\)

- \(\mathbb{D} = \mathbb{N} \text{ or } \mathbb{Z} \text{ or } \ldots\)
- Cost operations, \(G = \{+ \text{, } \text{min, } \text{max, } \text{avg, } \text{median}, \ldots\}\)
- DReX expression maps the input data word \(w\) to a term \(t\)
- When \(t\) admits a **succinct** representation, we can evaluate quickly
  - \(\Gamma^*\) with concatenation: ✓
  - \(\mathbb{Z}\) with \(\{\text{min, } +\}\), \(\text{min}(x + a, b)\): ✓
  - \(\{\text{avg}\}\) (set total, set size): ✓
  - ...
  - \(\mathbb{Z}\) with \text{median}: ✗?
• Is the transformation defined for all inputs?
• In the case of string-to-string transformations:
  • Does the output always have some “nice” property?
    \( \forall w, \text{ is it the case that } f(w) \in L? \)
  • Are two transformations equivalent?
Can we synthesize DReX expressions equivalent to given sed / Bash scripts and audit requirements?

```bash
#!/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
```
STATUS

Done:  \( \Sigma^* \rightarrow \Gamma^* \) (LICS 2014, POPL 2015)

Ongoing:  \( \Sigma^* \rightarrow \mathbb{Z} \) (and \( \mathbb{N} \) and \( \mathbb{R} \) and ...)
THANK YOU!

TRY IT OUT AT HTTP://DREXONLINE.COM