Regular Functions

Rajeev Alur

University of Pennsylvania

In honor of Prof. Edmund M. Clarke

Clarke Symposium, Sept 2014

Penn Engineering

Regular Languages

- Natural
  - Intuitive operational model of finite-state automata
- Robust
  - Alternative characterizations and closure properties
- Analyzable
  - Decidable questions: emptiness, equivalence…
- Applications
  - Algorithmic verification, text processing …

What is the analog of regularity for defining functions?

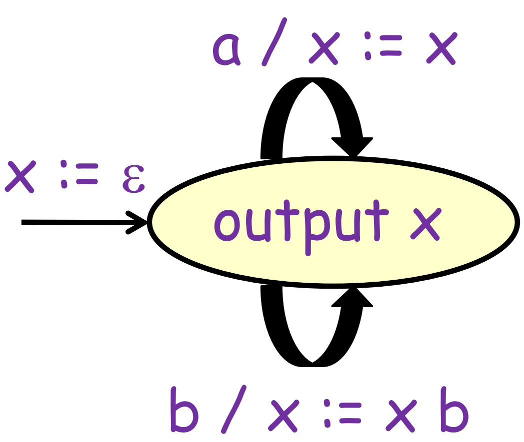
Sequential Transducers

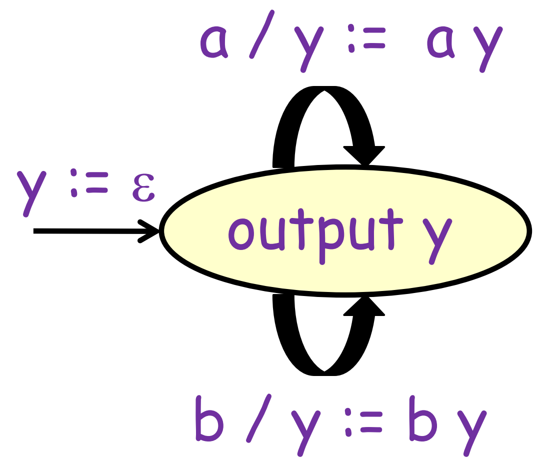
- At every step, read an input symbol, output zero or more symbols, and update state

$$q \xrightarrow{a/010} q'$$

- Examples:
  - Delete all a symbols, Duplicate each symbol
  - Insert 0 after first b
- Well-studied with some appealing properties
  - Equivalence decidable for deterministic case
  - Minimization possible
  - … but fragile theory
- Expressive enough ? What about reverse? swap ?
  - Model less expressive than two-way counterpart (Aho 69)

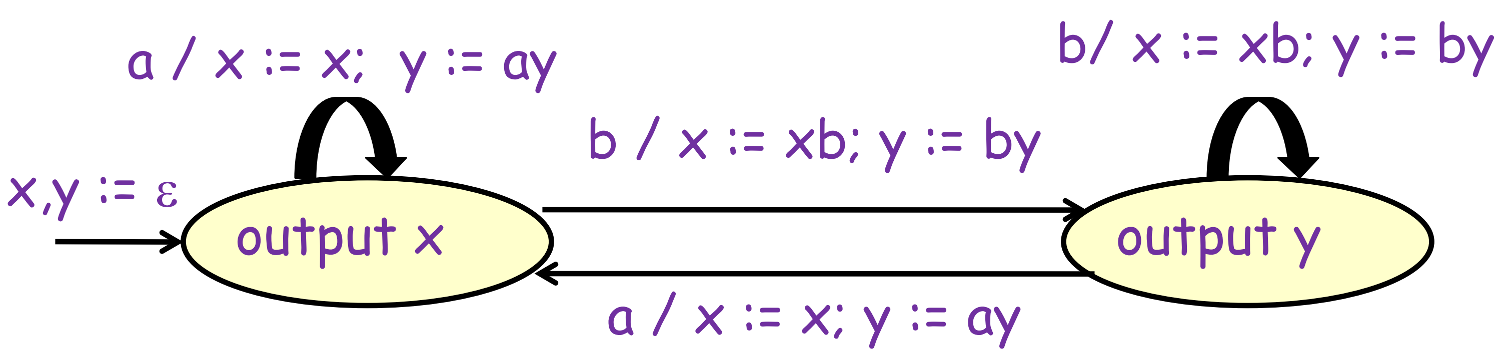
Streaming String Transducers





Del(w) = Delete all a's in w

Rev(w) = Reverse input w



f(w) = If w ends with b then Rev(w) else Del(w)

SST Model

- FSMs with write-only variables
  - Finite-state control
  - Finitely many string variables
  - Variables updated at each step, but no tests allowed
  - Copyless (single-use) assignment: $x := x.y; \ y := \varepsilon$
- Computes output in a single left-to-right pass over input string
  - Length of output is $O(|w|)$
- Example transformations
  - Insert, delete, substitute, reverse, swap, …
  - Copy(w) = w.w
- Regular string transformation = Computable by SST

Properties of Regular Functions

- Decidable analysis
  - Functional equivalence
  - Type checking
- Closed under many operations
  - Functional composition
  - Regular look-ahead
- Multiple equivalent characterizations
  - Two-way finite-state transducers
  - MSO-definable graph transformations
  - Declarative regular-expression-like language

Calculus of Regular Combinators

- Analog of regular expressions for regular (partial) functions
  - Base case: Constant γ
  - Choice: if r then f else g (here r is regular expression)
  - split(f,g): if there are unique u and v s.t. w=u.v and f(u) and g(v) are defined then return f(u).g(v)
  - left-split(f,g): similar to split, but return g(v).f(u)
  - iterate(f) and left-iterate(f)
  - combine(f,g): return f(w).g(w)
  - chain(f,r): allows mixing outputs from adjacent chunks

- Ongoing work: Language DReX based on this foundation
  - Type system to ensure consistency
  - Fast (linear-time) evaluation
  - Prototype implementation

Conclusions

- Class of string-to-string transformations with appealing theoretical foundations

- Defining regular functions using FSMs with write-only variables generalizes to many settings:
  - Strings to numerical costs
  - Infinite strings to infinite strings
  - Trees to strings/trees …

  Many results as well as many open/unexplored problems

- Potential applications
  - Analyzable language for document transformations (DReX)
  - Decidable sublcass of list processing programs
  - More expressive costs for quantitative analysis