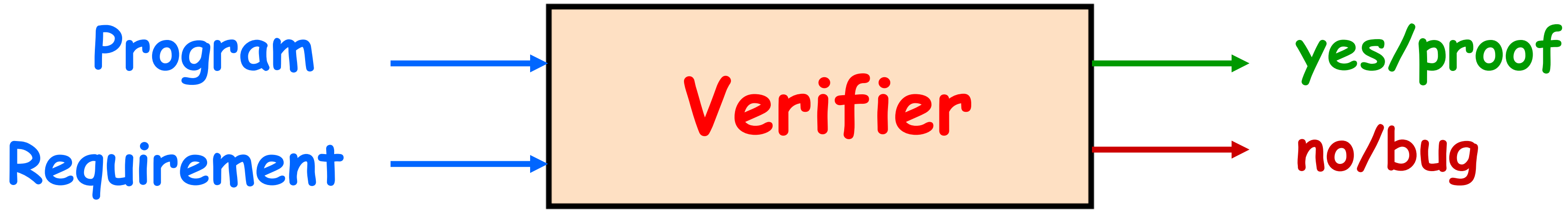
Streaming Transducers

Rajeev Alur
University of Pennsylvania

Penn Engineering

Can Software Verification be Automated?



Improving reliability of software: Grand challenge for computer science

Recent Success Story: Software Model Checking

```
do{
 KeAcquireSpinLock();
 nPacketsOld = nPackets;
 if(request){
   request = request->Next;
   KeReleaseSpinLock();
   nPackets++;
 }
}while(nPackets!=
   nPacketsOld);
KeReleaseSpinLock();
```

Do lock operations, acquire and release strictly alternate on every program execution?

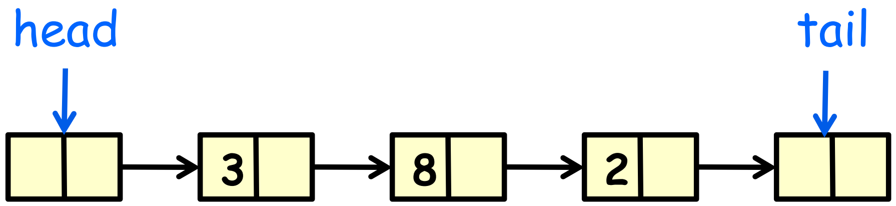
Microsoft success (SLAM, SDV)

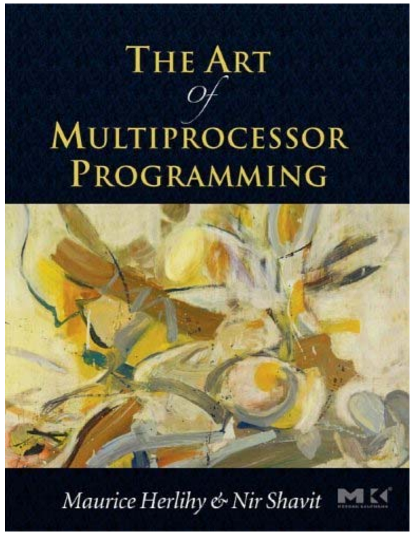
- Theoretical advances +
- Tool engineering +
- Target choice (device drivers)

New Opportunity: Concurrent Data Structures

```
boolean dequeue(queue *queue, value *pvalue)
{
  node *head;
  node *tail;
  node *next;

  while (true) {
    head = queue->head;
    tail = queue->tail;
    next = head->next;
    if (head == queue->head) {
      if (head == tail) {
        if (next == 0)
          return false;
        cas(&queue->tail, tail, next);
      } else {
        *pvalue = next->value;
        if (cas(&queue->head, head, next))
          break;
      }
    }
  }
  delete_node(head);
  return true;
}
```




The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Programs Manipulating Heap-allocated Data

- Heap consists of cells containing data, with a graph structure induced by next pointers
- Operations on data structures traverse and update heap
- All existing results show undecidability for simple properties (e.g. aliasing: can two pointers point to same cell?)
- Why can't we view a program as a transducer?
  - Operations such as insert, delete, reverse map sequences of data items to sequences of data items
  - Automata (NFA, pushdown, Buchi, tree) theory has provided foundations to algorithmic verification

String Transducers

- A transducer maps a string over input alphabet to a string over output alphabet

- Simplest transducer model: (Finite-state) Mealy Machines

- At every step, read an input symbol, produce an output symbol and update state

$$q \xrightarrow{a/0} q'$$

- Example: Replace every a and b by 0, and every c by 1

- Analyzable like finite automata, but not very expressive

  What about "delete all a symbols"?

Sequential Transducers

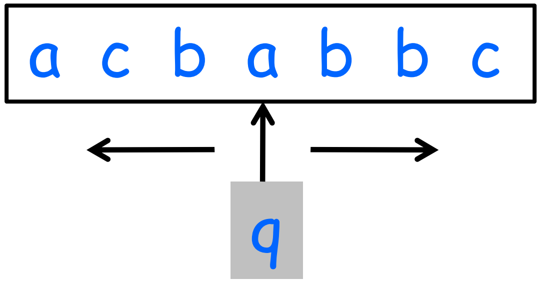
- At every step, read an input symbol, output zero or more symbols, and update state

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

- 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?

Deterministic Two-way Transducers

- Input stored on tape, with a read head
- At each step, produce 0 or more output symbols, update state, move left/right, or stop



- Examples:

  Reverse

  Copy entire string (map w to w.w)

  Delete a symbols if string ends with b (regular look-ahead)

  Swapping of substrings

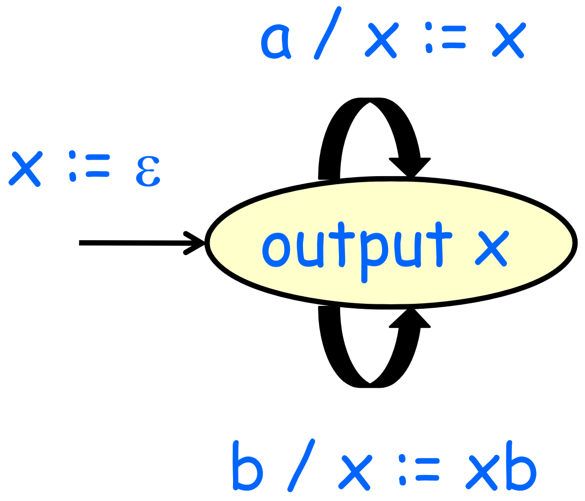
- More expressive than det. sequential transducers, and define the class of regular transductions

Two-way Transducers

- Closed under operations such as
  - Sequential composition
  - Regular look-ahead: $f(w)$ = if w in L then $f_1(w)$ else $f_2(w)$
- Equivalent characterization using MSO (monadic second-order logic) definable graph transductions
- Checking equivalence is decidable !
- But not much used in program verification, Why?
  - Not a suitable abstraction for programs over linked lists
  - A C program and a two-way transducer reverse a list in very different ways

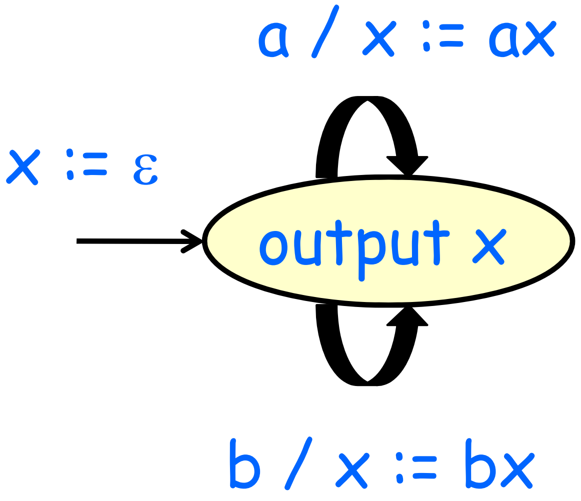
Streaming Transducer: Delete

- Finite state control + variable x ranging over output strings
- String variables explicitly updated at each step
- Delete all a symbols



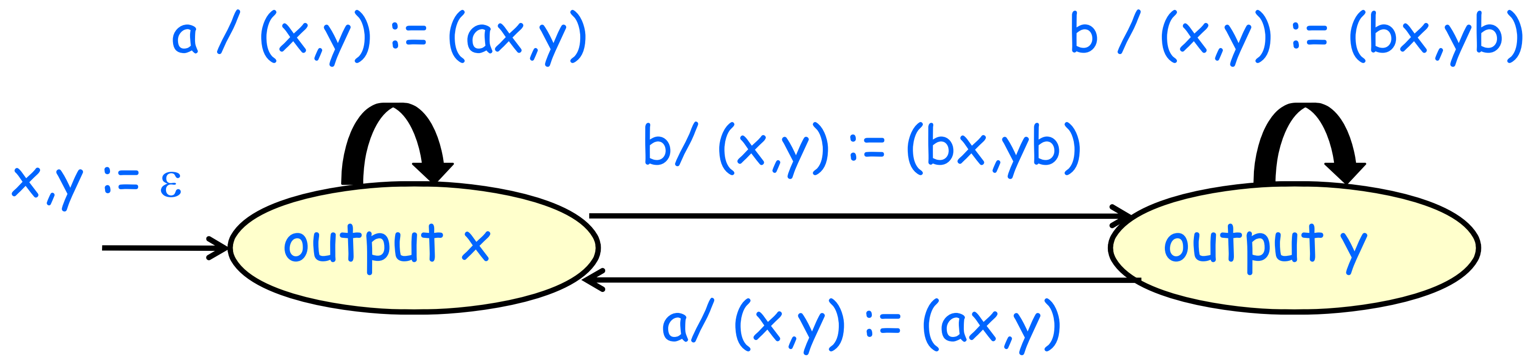
Streaming Transducer: Reverse

- Symbols may be added to string variables at both ends



Streaming Transducer: Regular Look Ahead

- Multiple string variables are allowed (and needed)
- If input ends with b, then delete all a symbols, else reverse



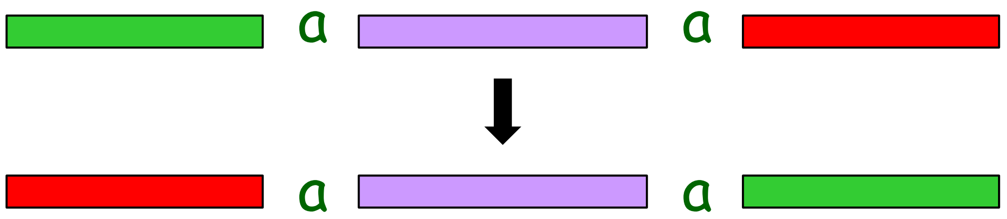
Variable x equals reverse of the input so far

Variable y equals input so far with all a's deleted

Streaming Transducer: Concatenation

- String variables can be concatenated

- Example: Swap substring before first a with substring following last a



- Key restriction: a variable can appear at most once on RHS

  (x,y) := (xy, ε) allowed

  (x,y) := (xy, y) not allowed

Streaming String Transducer (SST)

1. Finite set Q of states
2. Input alphabet A
3. Output alphabet B
4. Initial state q_0
5. Finite set X of string variables
6. Partial output function F : Q -> (B U X)*
7. State transition function δ : Q x A -> Q
8. Variable update function ρ : Q x A x X -> (B U X)*

- ❑ Output function and variable update function required to be copyless: each variable x can be used at most once
- ❑ Configuration = (state q, valuation α from X to B*)
- ❑ Semantics: Partial function from A* to B*

Transducer Application: String Sanitizers

- BEK: A domain specific language for writing string manipulating sanitizers on untrusted user data

- Analysis tool translates BEK program into (symbolic) transducer and checks properties such as
  - Is transduction idempotent: $f(f(w)) = f(w)$
  - Do two transductions commute: $f_1(f_2(w)) = f_2(f_1(w))$

- Recent success in analyzing IE XSS filters and other web apps

- Example sanitizer that BEK cannot capture (but SST can):

  Rewrite input w to suffix following the last occurrence of "dot"

Fast and precise sanitizer analysis with BEK.
Hooimeijer et al. USENIX Security 2011

Transducer Application: Program Synthesis

- Programming by examples to facilitate end-user programming
- Microsoft prototype to learn the transformation for Excel Spreadsheet Macros: success reported in practice, but no theoretical foundation (e.g. convergence of learning algorithm)
- Example transformation (swapping substrings requires SST !)

Aceto, Luca	Luca Aceto
Monika R. Henzinger	Monika Henzinger
Jiri Sgall	Jiri Sgall

Automating string processing in spreadsheets using input-output examples. Gulwani. POPL 2011

SST Properties

- At each step, one input symbol is processed, and at most a constant number of output symbols are newly created
- Output is bounded: Length of output = O(length of input)
- SST transduction can be computed in linear time
- Finite-state control: String variables not examined
- SST cannot implement merge

  $f(u_1u_2....u_k\#v_1v_2...v_k) = u_1v_1u_2v_2....u_kv_k$
- Multiple variables are essential

  For $f(w)=w^k$, k variables are necessary and sufficient

Decision Problem: Type Checking

Pre/Post condition assertion: { L } S { L' }

Given a regular language L of input strings (pre-condition), an SST S, and a regular language L' of output strings (post-condition), verify that for every w in L, S(w) is in L'

Thm: Type checking is solvable in polynomial-time

Key construction: Summarization

Decision Problem: Equivalence

Functional Equivalence;

Given SSTs S and S' over same input/output alphabets, check whether they define the same transductions.

Thm: Equivalence is solvable in PSPACE

(polynomial in states, but exponential in # of string variables)

Expressiveness

Thm: A string transduction is definable by an SST iff it is regular

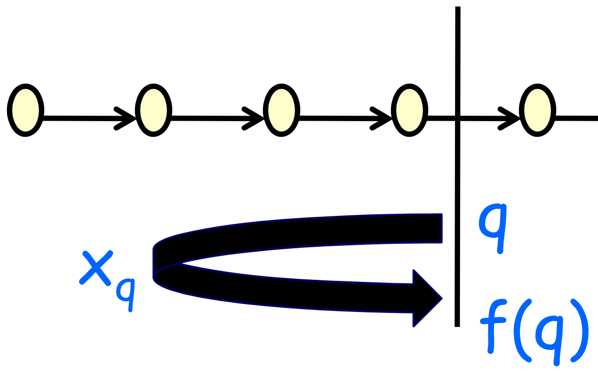
1. SST definable transduction is MSO definable
2. MSO definable transduction can be captured by a two-way transducer (Engelfriet/Hoogeboom 2001)
3. SST can simulate a two-way transducer

Evidence of robustness of class of regular transductions

Closure properties

1. Sequential composition: $f_1(f_2(w))$
2. Regular conditional choice: if w in L then $f_1(w)$ else $f_2(w)$

From Two-Way Transducers to SSTs



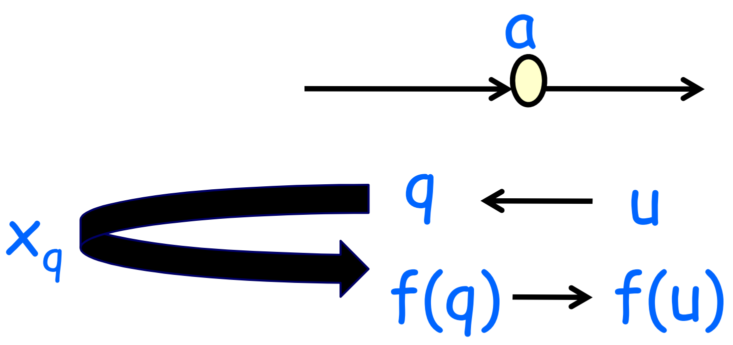
Two-way transducer *A* visits each position multiple times

What information should SST *S* store after reading a prefix?

For each state q of *A*, *S* maintains summary of computation of *A* started in state q moving left till return to same position

1. The state f(q) upon return
2. Variable x_q storing output emitted during this run

Challenge for Consistent Update



Map f: Q -> Q and variables x_q need to be consistently updated at each step

If transducer A moving left in state u on symbol a transitions to q, then updated f(u) and x_u depend on current f(q) and x_q

Problem: Two distinct states u and v may map to q

Then x_u and x_v use x_q, but assignments must be copyless !

Solution requires careful analysis of sharing (required value of each x_q maintained as a concatenation of multiple chunks)

Heap-manipulating Programs

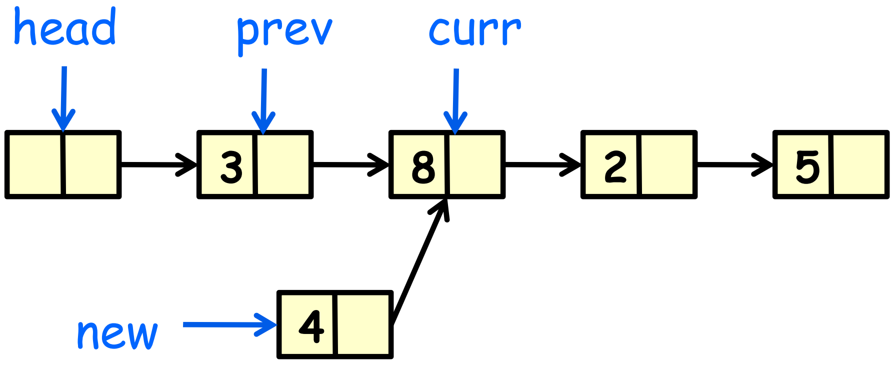
Sequential program +

Heap of cells containing data and next pointers +

Boolean variables +

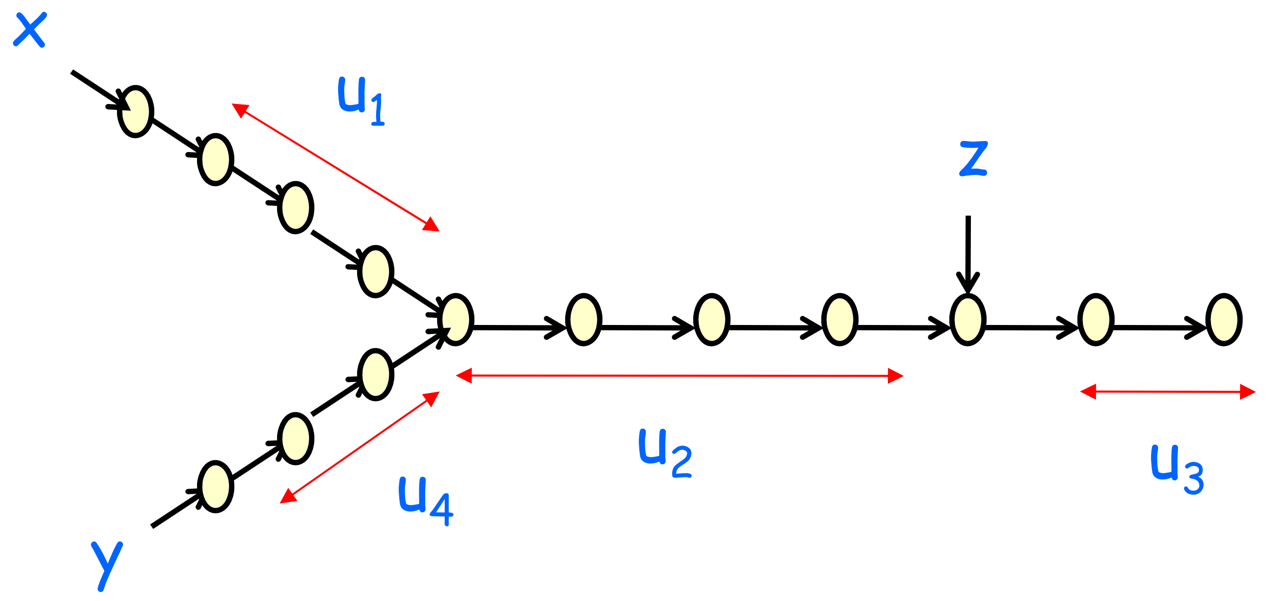
Pointer variables that reference heap cells

Program operations can add cells, change next pointers, and traverse the heap by following next pointers



How to restrict operations to capture exactly regular transductions

Representing Heaps in SST



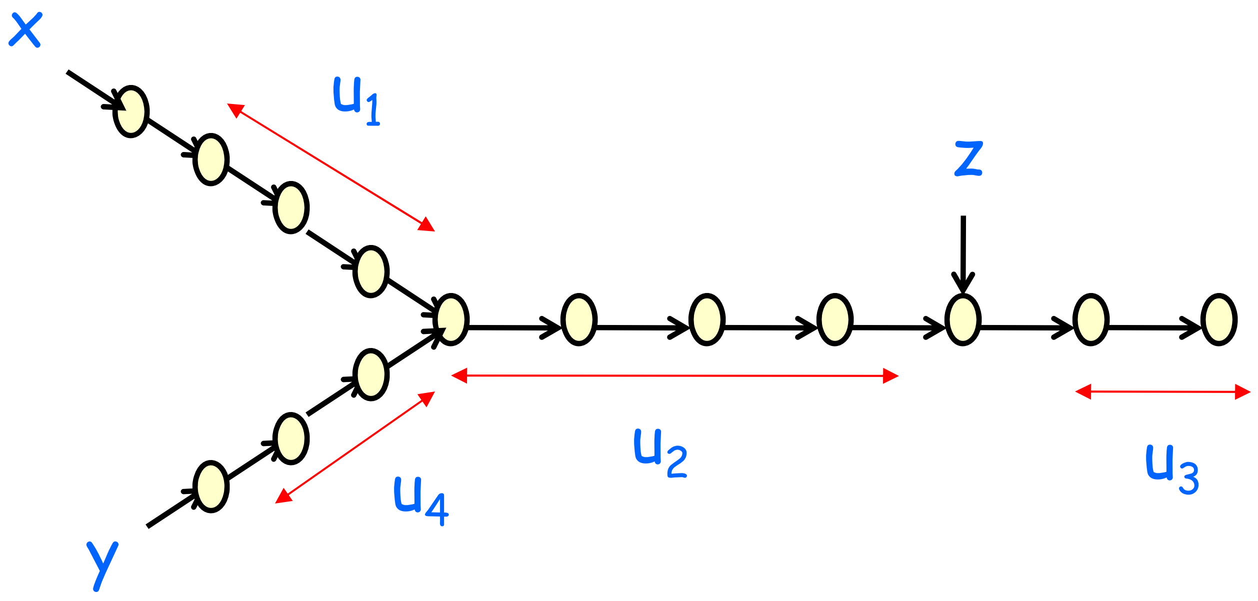
Shape (encoded in state of SST):

$x : u_1\, u_2\, z\, ;\ y : u_4\, u_2\, z\, ;\ z : u_3$

String variables: u_1, u_2, u_3, u_4

Shape + values of string vars enough to encode heap

Simulating Heap Updates

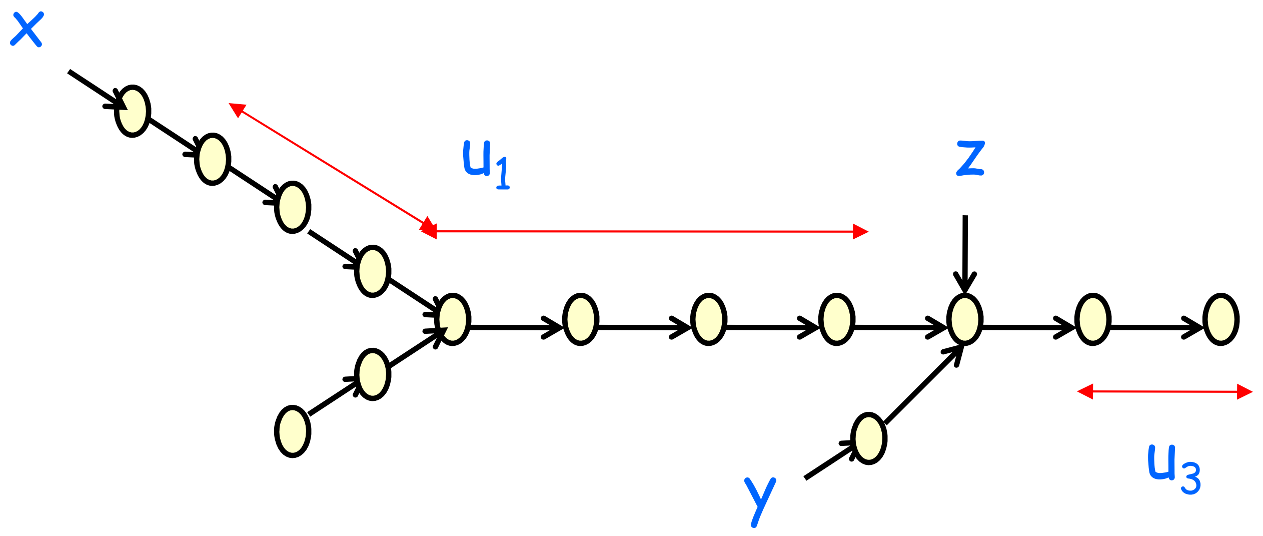


Consider program instruction

y.next := z

How to update shape and string variables in SST?

Simulating Heap Updates



New Shape: x: u_1 z ; y : z ; z : u_3

Variable update: $u_1 := u_1 u_2$

Special cells:

Cells referenced by pointer vars

Cells that 2 or more (reachable) next pointers point to

Contents between special cells kept in a single string var

Number of special cells = 2(# of pointer vars) - 1

Regular Heap Manipulating Programs

Update

```
x.next := y          (changes heap shape destructively)
x := new (a)         (adds new cell with data a and next nil)
```

Traversal

```
curr := curr.next    (traversal of input list)
x := y.next          (disallowed in general)
```

Theorem: Programs of above form can be analyzed by compiling into equivalent SSTs

- Single pass traversal of input list possible
- Pointers cannot be used as multiple read heads

Manipulating Data

- Each string element consists of (tag t, data d)
  - Tags are from finite set
  - Data is from unbounded set D that supports = and < tests
  - Example of D: Names with lexicographic order
- SSTs and list-processing programs generalized to allow
  - Finite set of data variables
  - Tests using = and < between current value and data vars
  - Input and output values
- Checking equivalence remains decidable (in PSPACE) !
- Many common routines fall in this class
  - Check if list is sorted
  - Insert an element in a sorted list
  - Delete all elements that equal input value

Algorithmic Verification of List-processing Programs

```
function delete
  input ref curr;
  input data v;
  output ref result;
  output bool flag := 0;
  local ref prev;

  while (curr != nil) & (curr.data = v) {
      curr := curr.next;
      flag := 1;
      }
  result := curr;
  prev:= curr;
  if (curr != nil) then {
     curr := curr.next;
     prev.next := nil;
     while (curr != nil) {
         if (curr.data = v) then {
             curr := curr.next;
             flag := 1;
             }
         else {
             prev.next := curr;
             prev := curr;
             curr := curr.next;
             prev.next := nil;
             }
  }
```

Decidable Analysis:

1. Assertion checks
2. Pre/post condition
3. Full functional correctness

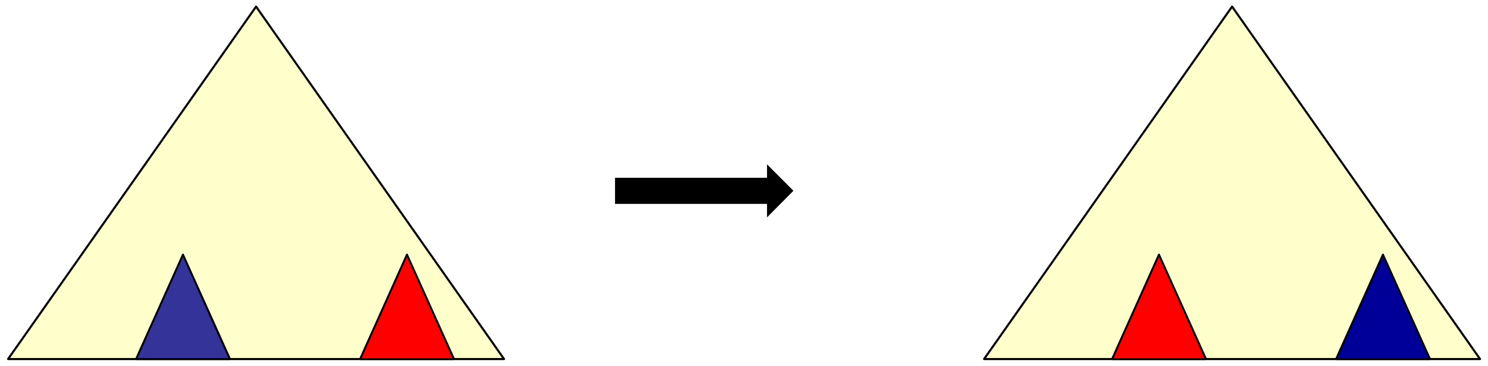
Recap

- Streaming String Transducers
  - New model for computing string transformations in a single pass
  - Key to expressiveness: multiple string variables
  - Key to analyzability: copyless updates and write-only output
- Decidable equivalence and type checking
- Robust expressiveness equivalent to MSO and two-way models
- Equivalent class of single pass list processing programs with solvable program analysis problems

Towards a Theory of Transducers

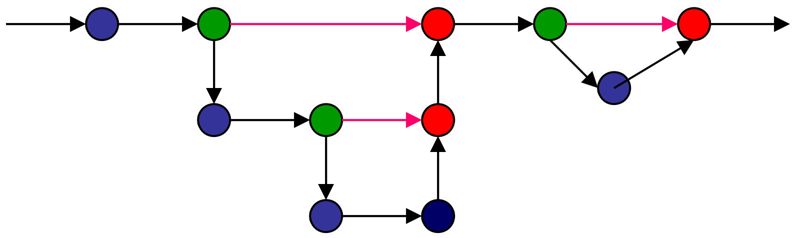
- ❑ Streaming String Transducers
  (with P. Cerny; POPL 2011, FSTTCS 2010)
- ❑ Transducers over Infinite Strings
  (with E. Filiot, A. Trivedi; LICS 2012)
- ❑ Nondeterministic Streaming Transducers
  (with J. Deshmukh; ICALP 2011)
- ❑ Streaming Tree Transducers
  (with L. D'Antoni; ICALP 2012)
- ❑ Regular Functions / Cost Register Automata
  (with L. D'Antoni, J. Deshmukh, M. Raghothaman, Y. Yuan)

Tree Transformations



- Motivation: XML transformations
- Existing models: Top-down / Bottom-up Tree transducers, Macro Tree Transducers
- Goal: Find a transducer model that can compute all “regular” (MSO-definable) tree transformations in a single pass

Nested Words



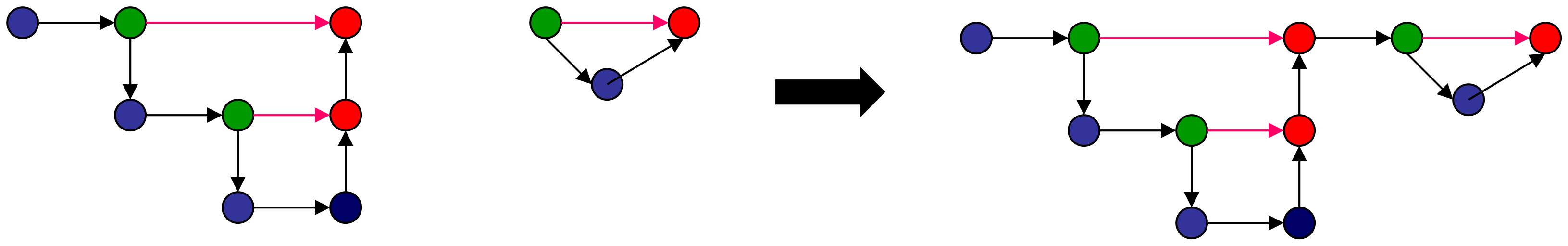
a <b c <a b c a> b> <c b c>

- Model of data with linear and hierarchical structure (encodes strings, ranked/unranked ordered trees)
- Nested Word Automata: Finite control + Visibly pushdown stack (push at calls, pop at returns)
- Many theoretical results and recent tools

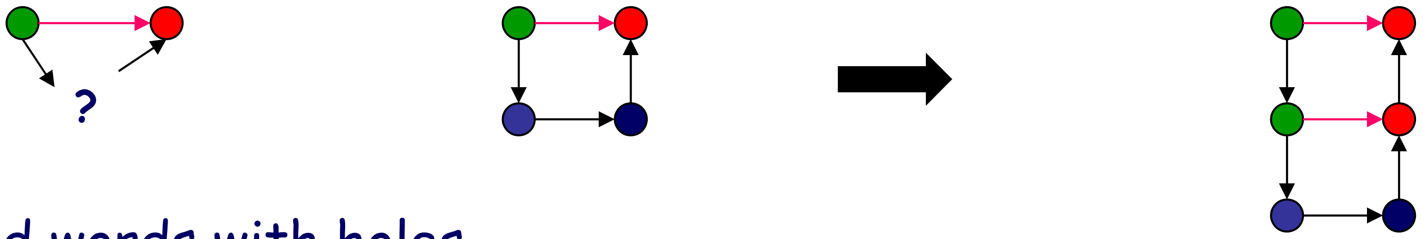
  see http://www.cis.upenn.edu/~alur/nw.html
- Transducer can push/pop variables also

Operations on Nested Word Variables

Concatenation



Substitution



Nested words with holes

Streaming Tree Transducer (STT)

- Finite state control + Stack
- Finitely many variables ranging over nested words with holes
- Processes input nested word left-to-right in single pass
- At internal symbol, updates state and variables
- At call, pushes stack symbol + vars, and resets vars
- At return, new state + vars depend on current state + vars and popped stack symbol + vars
- Updates using concatenation and substitution
- Updates obey Single-Use-Restriction
  - A variable may appear multiple times in RHS, but only one copy contributes to the output
  - Single use enforced using conflict relation over variables
  - Copyless limits expressiveness

STT Results

- Can capture many tree transformations naturally
  - Insert, Delete, Swap, Reverse, Bounded copy
- Computes output in linear-time in single pass
- STTs are closed under regular lookahead
- Expressiveness equals MSO-definable transformations
  - Relies on MSO equivalence of MTTs (Engelfriet/Maneth)
- Type checking solvable in EXPTIME
- Functional equivalence solvable in NEXPTIME
- Natural restrictions lead to MSO-equivalent models
  - Strings to Nested Words
  - Nested Words to Strings
  - Binary/Ranked Trees to Binary/Ranked Trees

Open Problems and Challenges

- Complexity of equivalence of SSTs and STTs
  - Large gap between lower and upper bounds
- Machine-independent characterization of "finite-state" transductions
  - To compute a function f : A* -> B* which auxiliary functions must be computed ?
- Regular functions
  - Decidability of min-cost for discounted cost automata
  - Decidability of equivalence for tropical semiring
- Learning algorithms