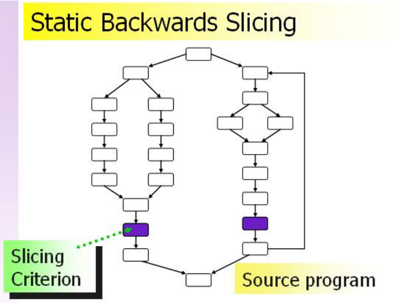
A Theory of Slicing for Probabilistic Control Flow Graphs

Torben Amtoft (Kansas State University)
Anindya Banerjee (IMDEA Software Institute)

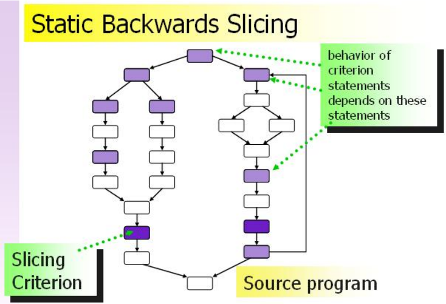
FoSSaCS, April 2016

What is Program Slicing?



Pick one or more program points of interest, called the slicing criterion

What is Program Slicing?

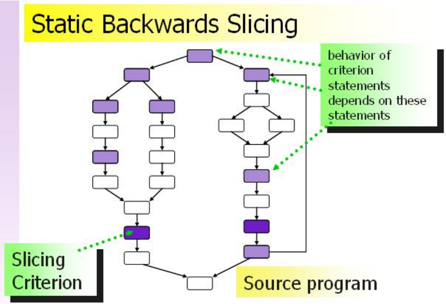


Walk backwards to find the nodes (the slice set) that the nodes in the slicing criterion depend on

- through data dependence, or
- through control dependence

Remove nodes not in the slice set.

What is Program Slicing?

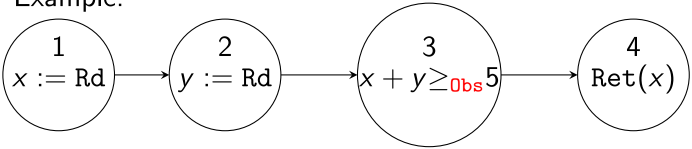


Applications include

- compiler optimizations
- debugging
- model checking
- protocol understanding

Probabilistic Setting

Example:

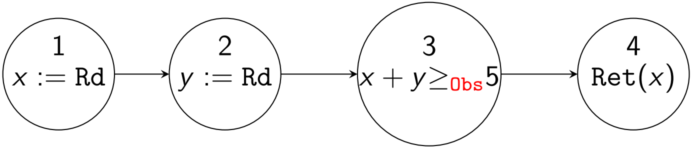


We shall work with CFGs (control flow graphs) with

- a unique End node, returning the final result
- random assignments from a given distribution
  - In this talk, we shall always use the one that assigns each of 0,1,2,3 equal probability (0.25)
- conditioning nodes (Observe) which remove "undesired" value combinations.

Applications: see excellent survey article [ICSE'2014] by Andrew Gordon et al

Probabilistic Semantics



Semantics is expressed using distributions which assign probabilities to stores.

- special deterministic case: one store has probability 1, all other stores have probability 0

Distribution D before node 3:

$$D(\{x \mapsto i,\ y \mapsto j\}) = 1/16 \text{ for } i, j \in 0..3$$

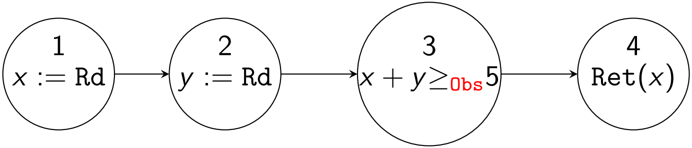
Distribution D after node 3:

$$\begin{aligned} D(\{x \mapsto 3,\ y \mapsto 2\}) &= 1/16 \\ D(\{x \mapsto 2,\ y \mapsto 1\}) &= 0 \end{aligned}$$

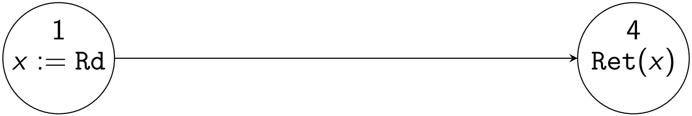
Thus $\sum D < 1$ is possible (can later be normalized)

Slicing: the Challenge

Question: can we slice away nodes 2 and 3 in



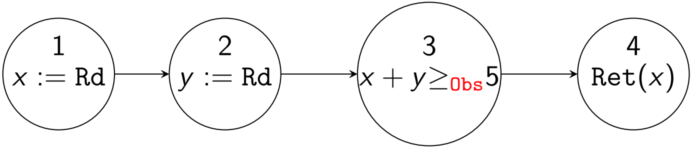
so as to arrive at



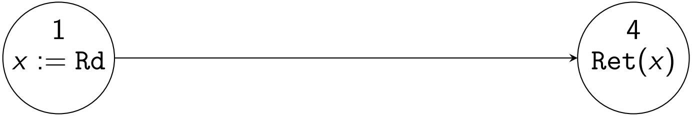
In the original program:

Slicing: the Challenge

Question: can we slice away nodes 2 and 3 in



so as to arrive at

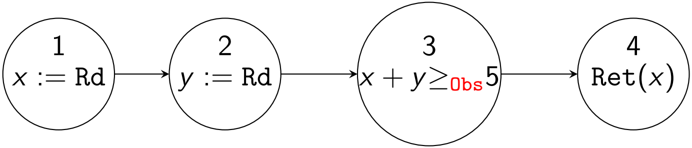


In the original program:

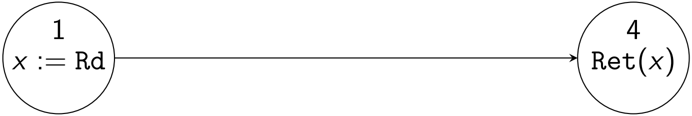
- classically, 4 depends only on 1, so yes

Slicing: the Challenge

Question: can we slice away nodes 2 and 3 in



so as to arrive at



In the original program:

- classically, 4 depends only on 1, so yes
- but the final distribution of x is skewed ($x \leq 1$ is impossible) so NO

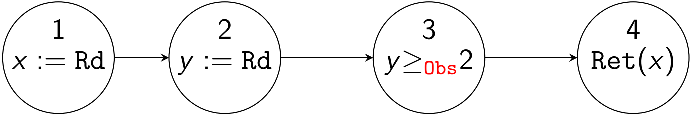
We need to be more careful!

Our Goals

- state what is means for slicing to be correct in a probabilistic setting
- give syntactic conditions that guarantee semantic correctness
- present algorithm to find best syntactic slice.

When Slicing is Correct: Semantic Definition

Question: can we slice away nodes 2 and 3 in



Final distribution D:

$$D(\{x \mapsto i,\ y \mapsto j\}) = 1/16 \text{ for } i \in 0..3, j \in 2..3$$

which restricted to the returned variable x is

$$D(\{x \mapsto i\}) = 1/8 \text{ for } i \in 0..3$$

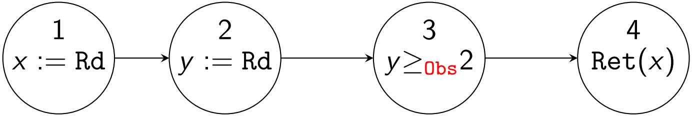
Final distribution Δ of sliced program:

$$\Delta(\{x \mapsto i\}) = 1/4 \text{ for } i \in 0..3$$

With $c = 0.5$ we have $D = c \cdot \Delta$ and shall therefore say that slicing is correct as it does not skew the distribution of the relevant variable x.

Why Slicing is Correct: Syntactic Criterion

Question: how to infer, without semantic calculations, that it is correct to slice away nodes 2 and 3 in



The correctness relies on the fact that

- at node 4, x and y are probabilistically independent

which will be the case since

- the set of nodes $\{1\}$ that may influence x is disjoint from the set of nodes $\{2\}$ that may influence y.

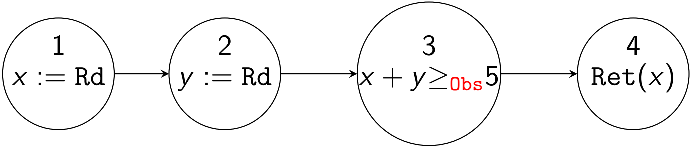
Initial Finding: A conditioning can be sliced away if

- the nodes that the End node depends on, and
- the nodes that the conditioning depends on

have nothing in common.

When Slicing is Incorrect, and Why

We saw it is incorrect to slice away nodes 2 and 3 in



since the final distribution D has say

$$D(\{x \mapsto 1\}) = 0 \text{ but } D(\{x \mapsto 2\}) = 1/16$$

and with Δ the uniform distribution of x in the sliced program we thus for all c have

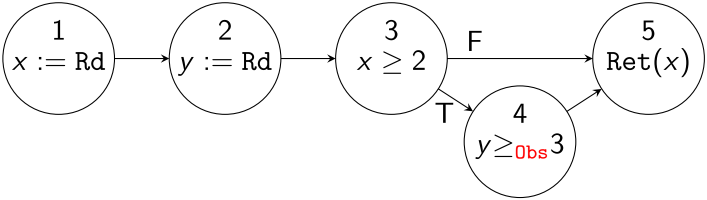
$$D \neq c\Delta$$

And indeed, our tentative syntactic correctness criterion will disallow slicing, since

- the End node (data) depends on node 1, and
- the conditioning node (data) depends also on node 1.

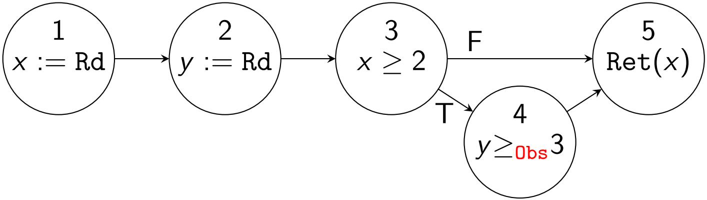
When Slicing is Incorrect, and Why (II)

Can we slice away nodes 2, 3 and 4 in



When Slicing is Incorrect, and Why (II)

Can we slice away nodes 2, 3 and 4 in



No, since the final distribution D is skewed:

$$D(\{x \mapsto 1\}) = 1/4 \text{ but } D(\{x \mapsto 2\}) = 1/16$$

And indeed, our tentative syntactic correctness criterion will disallow slicing, since

- the End node depends on node 1, and
- the conditioning node control depends on node 3 which data depends on node 1.

Tentative Syntactic Correctness Criterion

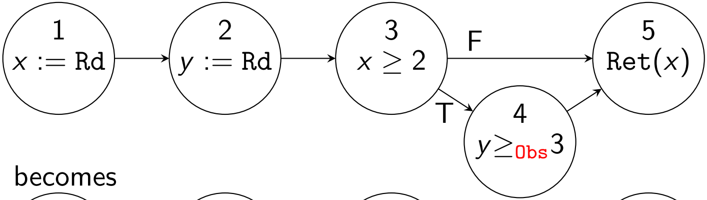
It appears that for Q to be a correct slice, we must require that

- Q is "closed under dependency"
- all conditioning nodes not in Q must belong to another set, Q_0, such that
  - Q and Q_0 are disjoint
  - Q_0 is also closed under dependency.

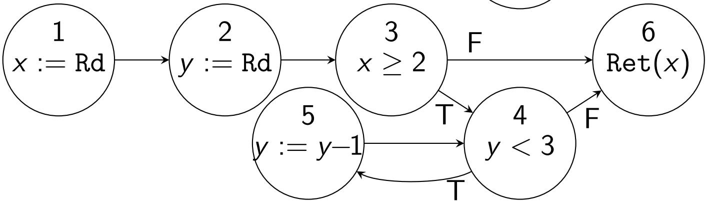
We shall soon refine these conditions.

Why Slicing Away Loops may be Incorrect

We can encode conditioning using loops: the example



becomes



Above we can thus not slice away all nodes but 1 and 6

- though node 6 appears to depend on node 1 only.

We thus need to augment correctness criterion:

- all loops contain at least one node in $Q \cup Q_0$.

Weak Slice Sets

We shall now formalize "Q is closed under dependency":

- data dependency (DD): straight-forward
- control dependency:
  each node has exactly one next Q-observable

v' is a next Q-observable of v (there is at most one) iff

- $v' \in Q \cup \{\texttt{End}\}$
- all paths from v to a node in $Q \cup \{\texttt{End}\}$ contain v'.

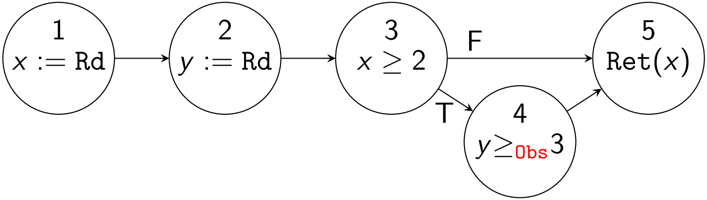
We say that Q is a weak slice set if

- Q is closed under DD
- each node has a next Q-observable

Weak Slice Sets, Example

Recall: v' is a next Q-observable of v iff

- $v' \in Q \cup \{\mathtt{End}\}$
- all paths from v to a node in $Q \cup \{\mathtt{End}\}$ contain v'.



- With $Q = \{2, 4\}$, Q is closed under DD but node 3 does not have a next Q-observable:
  - there is a path from 3 to 5 that does not contain 4
  - there is a path from 3 to 4 that does not contain 5
- With $Q = \{2, 3, 4\}$, all nodes have next Q-observables but is not closed under DD
- But $Q = \{1, 2, 3, 4\}$ is a weak slice set

Final Syntactic Correctness Criterion

We say that Q is a correct slice if there exists Q_0 such that (Q, Q_0) is a slicing pair in that

- Q and Q_0 are weak slice sets
- $Q \cap Q_0 = \emptyset$
- all conditioning nodes belong to $Q \cup Q_0$
- all loops contain a node in $Q \cup Q_0$.

Semantics of Probabilistic CFGs

One-step reduction $(v, D) \rightarrow (v', D')$

- v' successor of v
- transforms D into D'
- defined for (random) assignments, and conditioning

Multi-step reduction $(v, D) \Rightarrow (v', D')$

- v' postdominates v
- combines paths from v to v' that may contain cycles but do not contain v' until the very end
- defined as the limit (in D') of an inductively defined relation $(v, D) \stackrel{k}{\Rightarrow} (v', D')$ where k bounds the number of cycles.

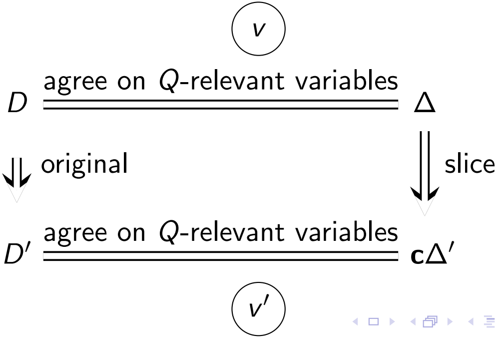
Conjecture: for CFGs produced from structured programs, this semantics will coincide with the standard denotational semantics [Kozen, Hur]

Syntactic Criteria Imply Semantic Correctness

Assume that

- we have slicing pair (Q, Q_0)
- v' postdominates v
- at v, the Q-relevant variables and the Q_0-relevant variables are probabilistically independent in D.

Then there always exists a real number c with $0 \leq c \leq 1$ such that the below diagram commutes:



Probabilistic Independence

When are R and R_0 independent in D?

- classical definition: if for all s/s_0 with domain R/R_0:

$$D(s \oplus s_0) = D(s) \cdot D(s_0)$$

- in our setting when $\sum D$ may not equal 1: instead

$$D(s \oplus s_0) \cdot \sum D = D(s) \cdot D(s_0).$$

Let (Q, Q_0) be a slicing pair, and assume $(v, D) \stackrel{k}{\Rightarrow} (v', D')$.

- If at v, the Q-relevant variables are independent of the Q_0-relevant variables in D
- then at v', the Q-relevant variables are independent of the Q_0-relevant variables in D'.

Algorithm to Compute Best Syntactic Slice

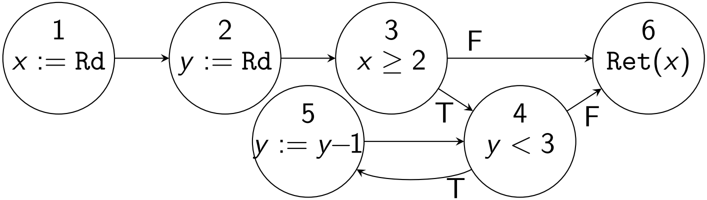
We have an $O(n^3)$ algorithm to find the best slice.
Subcomponents:

1. a function that given Q either
   - confirms that Q is a weak slice set, or
   - returns $C \neq \emptyset$ such that C is contained in all weak slice sets containing Q

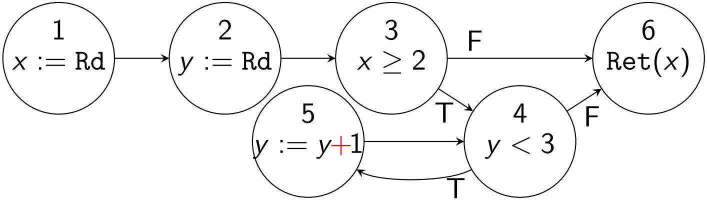
   and which works by a backwards breadth-first search through nodes not in Q
   - C will contain nodes reachable from two nodes
2. a function that given Q finds the least weak slice set containing Q

Semantic vs Syntactic Slices

Recall it is semantically unsound to slice away anything in



but if we change the loop body:



then all but 1 and 6 can be sliced away

- though $\{1, 6\}$ does not meet our syntactic criteria.

Computing the best semantic slice is clearly undecidable

Optimizations

One may in various ways transform the CFG so that the semantics is preserved while smaller slices may be generated:

- a conditioning $\texttt{Observe}(B)$ may be removed if B can be shown to always hold at that node
- after a conditioning of the form $\texttt{Observe}(x = c)$, insert an assignment $x := c$.

Related Work

Our main inspiration is [Hur et al, PLDI'14].

- they present algorithm for slicing of probabilistic structured programs
  - involving various preprocessing
  - doing various optimizations along the way.
- conditioning gives rise to a new kind of dependency:
  - if conditioning depends on x and y then any slice that includes y must also include x
- no separation between specification and implementation
  - this makes correctness proof more complex
- no analysis of asymptotic running time

Our Contributions

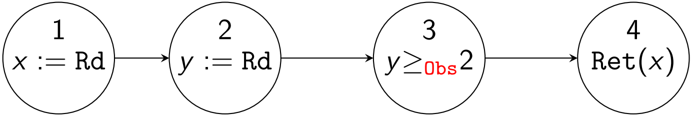
- framework for slicing of probabilistic programs which separates specification from implementation
- extends in non-trivial way classical slicing frameworks (as generalized by Danicic [TCS 2011])
- presents operational semantics for probabilistic CFGs
- presents cubic-time algorithm for finding best (syntactic) slice

Future/present work:

- allow to slice away loops that are know to always (with probability 1) terminate

When Algorithm Computes Non-Trivial Slice

We now show how the algorithm finds the best slicing pair for



1. Compute least weak slice set that contains 4:
   1.1 close under DD: $\{1, 4\}$
   1.2 nothing more is needed for unique next observable
2. Compute least weak slice set that contains 3 (the conditioning):
   2.1 close under DD: $\{2, 3\}$
   2.2 nothing more is needed for unique next observable

As $\{1, 4\} \cap \{2, 3\}$ are disjoint, this shows that $Q = \{1, 4\}$ is a valid slice (with $Q_0 = \{2, 3\}$).

How Algorithm Handles Loops

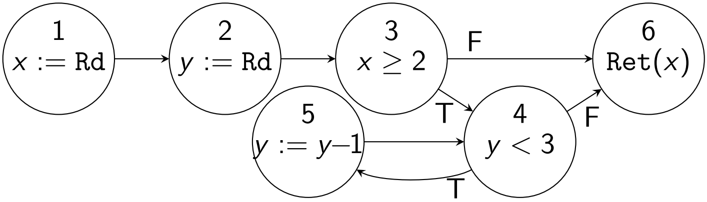
Recall that we require

for each loop, at least one node is in $Q \cup Q_0$

- this may appear to allow incompatible solutions
- but is equivalent to the below

for each loop,
the node(s) with "minimal height" is in $Q \cup Q_0$

When Algorithm Reveals No Non-Trivial Slice



1. Compute least weak slice set that contains 6:
   1.1 close under DD: $\{1, 6\}$
   1.2 nothing more is needed for unique next observable
2. Compute least weak slice set that contains 4 (the loop node closest to 6):
   2.1 close under DD: $\{2, 4, 5\}$
   2.2 a backwards search from $\{2, 4, 5\} \cup \{6\}$ will hit 3 from 4 and from 6 so we need to add 3: $\{2, 3, 4, 5\}$
   2.3 again close under DD: $\{1, 2, 3, 4, 5\}$

As the two results are not disjoint, we cannot put the second result in Q_0; instead, we must put it in Q and end up with a trivial slice: $Q = \{1, 2, 3, 4, 5, 6\}$