A Theory of Slicing for Probabilistic Control-Flow Graphs*

Torben Amtoft
Kansas State University
Manhattan, KS, USA
tamtoft@ksu.edu

Anindya Banerjee
IMDEA Software Institute
Madrid, Spain
anindya.banerjee@imdea.org

November 3, 2017

Abstract

We present a theory for slicing probabilistic imperative programs —containing random assignments, and "observe" statements (for conditioning) — represented as probabilistic control-flow graphs (pCFGs) whose nodes modify probability distributions. We show that such a representation allows direct adaptation of standard machinery such as data and control dependence, postdominators, relevant variables, *etc.* to the probabilistic setting. We separate the specification of slicing from its implementation: first we develop syntactic conditions that a slice must satisfy; next we prove that any such slice is semantically correct; finally we give an algorithm to compute the least slice. To generate smaller slices, we may in addition take advantage of knowledge that certain loops will terminate (almost) always. A key feature of our syntactic conditions is that they involve two disjoint slices such that the variables of one slice are *probabilistically independent* of the variables of the other. This leads directly to a proof of correctness of probabilistic slicing. In a companion article we show adequacy of the semantics of pCFGs with respect to the standard semantics of structured probabilistic programs.

1 Introduction

The task of program slicing [22, 20] is to remove the parts of a program that are irrelevant in a given context. This paper addresses slicing of probabilistic imperative programs which, in addition to the usual control structures, contain "random assignment" and "observe" (or conditioning) statements. The former assign random values from a given distribution to variables. The latter remove undesirable combinations of values, a feature which can be used to bias (or condition) the variables according to real world observations. The excellent survey by Gordon *et al.* [10] depicts how probabilistic programs can be used in a variety of contexts, such as: encoding applications from machine learning, biology, security; representing probabilistic models (Bayesian Networks, Markov Chains); estimating probability distributions through probabilistic inference algorithms (like the Metropolis-Hastings algorithm for Markov Chain Monte Carlo sampling); *etc.*

Program slicing of deterministic imperative programs is increasingly well understood [17, 4, 18, 1, 8]. A basic notion is that if the slice contains a program point which depends on some other program points then these also should be included in the slice; here "depends" typically encompasses data dependence and control dependence. However, Hur *et al.* [11] recently demonstrated that in the presence of random assignments and observations, standard notions of data and control dependence no longer suffice for semantically correct

*Expanded and revised version of a paper originally appearing in *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016.*

(backward) slicing. They develop a denotational framework in which they prove correct an algorithm for program slicing. In contrast, this paper shows how *classical notions of dependence can be extended to give a semantic foundation for the (backward) slicing of probabilistic programs*. The paper's key contributions are:

- A formulation of probabilistic slicing in terms of probabilistic control-flow graphs (pCFGs) (Section 3) that allows direct adaptation of standard machinery such as data and control dependence, postdominators, relevant variables, *etc.* to the probabilistic setting. We also provide a novel operational semantics of pCFGs (Section 4): the semantic function $\omega^{(v,v')}$ transforms a probability distribution at node v into a probability distribution at node v', so as to model what happens when "control" moves from v to v' in the control-flow graph.
- Syntactic conditions for correctness (Section 5) that in a non-trivial way extend classical work on program slicing [8] and whose key feature is that they involve *two* disjoint slices; in order for the first to be a correct final result of slicing, the other must contain any "observe" nodes sliced away and all nodes on which they depend. We show that the variables of one slice are *probabilistically independent* of the variables of the other, and this leads directly to the correctness of probabilistic slicing (Theorem 1 in Section 6).
- An algorithm (Section 7), with running time at most cubic in the size of the program, that (given an approximation of which loops terminate with probability 1) computes the best possible slice in that it is contained in any other (syntactic) slice of the program.

Our approach separates the specification of slicing from algorithms to compute the best possible slice. The former is concerned with defining what is a correct syntactic slice, such that the behavior of the sliced program is equivalent to that of the original. The latter is concerned with how to compute the best possible syntactic slice; this slice is automatically a semantically correct slice —no separate proof is necessary.

A program's behavior is its final probability distribution; we demand equality modulo a constant factor so as to allow the removal of "observe" statements that do not introduce any bias in the final distribution. This will be the case if the variables tested by "observe" statements are independent, in the sense of probability theory, of the variables relevant for the final value.

Compared to the conference version of this paper [2], the additional contributions are:

- We allow to slice away certain loops if they are known (through some analysis, or an oracle) to terminate with probability 1.
- In a companion article [3] we establish adequacy of the operational semantics of pCFGs with respect to the "classical" semantics [10, 11] for structured probabilistic programs.

We prove all non-trivial results; some proofs are in the main text but most are relegated to Appendix B. Our development is based on domain theory whose basic concepts we recall in Appendix A.

2 Motivating Examples

2.1 Probabilistic programs

Whereas in deterministic languages, a variable has only one value at a given time, we consider a language where a variable may have many different values at a given time, each with a certain probability. (Determinism is a special case where one value has probability one, and all others have probability zero.) We assume, to keep our development simple, that each possible value is an integer. A more general development, somewhat orthogonal to the aims of this article, would allow real numbers and would employ measure theory (as explained in [16]); we conjecture that much will extend naturally (with summations becoming integrals).

Similarly to [10], probabilities are introduced by the construct $x := \mathbf{Random}(\psi)$ which assigns to variable

x a value with probability given by the random distribution ψ which in our setting is a mapping from $\mathbb{Z}$ (the set of integers) to $[0,1]$ such that $\sum_{z\in\mathbb{Z}} \psi(z) = 1$. A program phrase modifies a distribution into another distribution, where a distribution assigns a probability to each possible store. This was first formalized by Kozen [14] in a denotational setting. As also in [10], we shall use the construct **Observe**(B) to "filter out" values which do not satisfy the boolean expression B. That is, the resulting distribution assigns zero probability to all stores not satisfying B, while stores satisfying B keep their probability.

2.2 The examples

Slicing amounts to picking a set Q of "program points" (satisfying certain conditions as we shall soon discuss) and then removing the program points not in Q (as we shall formalize in Section 4.4). The examples all use a random distribution ψ_4 over $\{0,1,2,3\}$ where $\psi_4(0) = \psi_4(1) = \psi_4(2) = \psi_4(3) = \frac{1}{4}$ whereas $\psi_4(i) = 0$ for $i \notin \{0,1,2,3\}$. The examples all consider whether it is correct to let Q contain exactly $x := \textbf{Random}(\psi_4)$ and **Return**(x), and thus slice into a program P_x with straightforward semantics: after execution, the probability of each possible store is given by the distribution Δ' defined as $\Delta'(\{x \mapsto i\}) = \frac{1}{4}$ if $i \in \{0,1,2,3\}$; otherwise $\Delta'(\{x \mapsto i\}) = 0$.

Example 2.1 *Consider the program P_1 given by*

$$\begin{array}{ll} 1: & x := \textbf{Random}(\psi_4) \\ 2: & y := \textbf{Random}(\psi_4) \\ 3: & \textbf{Observe}(y \geq 2) \\ 4: & \textbf{Return}(x) \end{array}$$

The distribution produced by the first two assignments will assign probability $\frac{1}{4}\cdot\frac{1}{4} = \frac{1}{16}$ to each possible store $\{x \mapsto i,\ y \mapsto j\}$ with $i,j \in \{0,1,2,3\}$. In the final distribution D_1, a store $\{x \mapsto i,\ y \mapsto j\}$ with $j < 2$ is impossible, and for each $i \in \{0,1,2,3\}$ there are thus only two possible stores that associate x with i: the store $\{x \mapsto i,\ y \mapsto 2\}$, and the store $\{x \mapsto i,\ y \mapsto 3\}$. Restricting to the variable x that is ultimately returned,

$$D_1(\{x \mapsto i\}) = \sum_{j=2}^{3} D_1(\{x \mapsto i,\ y \mapsto j\}) = \frac{1}{16} + \frac{1}{16} = \mathbf{\frac{1}{8}}$$

if $i \in \{0,1,2,3\}$ (otherwise, $D_1(\{x \mapsto i\}) = 0$). We see that the probabilities in D_1 do not *add up to 1 which reflects that the purpose of an* **Observe** *statement is to cause undesired parts of the current distribution at that node to "disappear" (which may give certain branches more relative weight than other branches). We also see that D_1 equals Δ' except for a constant factor: $D_1 = 0.5 \cdot \Delta'$. That is, Δ' gives the same* relative *distribution over the values of x as D_1 does. We shall therefore say that P_x is a* correct *slice of P_1.*

Thus the **Observe** statement is irrelevant to the final relative distribution of x. This is because y and x are *independent* in D_1, as formalized in Definition 4.7.

Example 2.2 *Consider the program P_2 given by*

$$\begin{array}{ll} 1: & x := \textbf{Random}(\psi_4) \\ 2: & y := \textbf{Random}(\psi_4) \\ 3: & \textbf{Observe}(x + y \geq 5) \\ 4: & \textbf{Return}(x) \end{array}$$

Here the final distribution D_2 allows only 3 stores: $\{x \mapsto 2,\ y \mapsto 3\})$, $\{x \mapsto 3,\ y \mapsto 2\})$ and $\{x \mapsto 3,\ y \mapsto 3\})$, all with probability $\frac{1}{16}$, and hence $D_2(\{x \mapsto 2\}) = \frac{1}{16}$ and $D_2(\{x \mapsto 3\}) = \frac{1}{16} + \frac{1}{16} = \frac{1}{8}$. Thus the program

is biased towards x having value 2 or 3; in particular we cannot write D_2 in the form $c\Delta'$. Hence it is incorrect *to slice P_2 into P_x.*

In this example, x and y are *not* independent in D_2; this is as expected since the **Observe** statement in P_2 depends on something (the assignment to x) on which the returned variable x also depends.

Example 2.3 *Consider the program P_3 given by*

$$\begin{array}{ll} 1: & x := \mathbf{Random}(\psi_4) \\ 2: & \mathbf{if}\ x \geq 2 \\ 3: & \quad z := \mathbf{Random}(\psi_4) \\ 4: & \quad \mathbf{Observe}(z \geq 3) \\ 5: & \mathbf{Return}(x) \end{array}$$

Since three quarters of the distribution disappears when $x \geq 2$, P_3 is biased in that it is more likely to return 0 or 1 than 2 or 3; in fact, the final distribution D_3 is given by $D_3(\{x \mapsto i\}) = \frac{1}{4}$ when $i \in \{0,1\}$ and $D_3(\{x \mapsto i\}) = D_3(\{x \mapsto i,\ z \mapsto 3\}) = \frac{1}{16}$ when $i \in \{2,3\}$. (And, when $i \notin \{0,1,2,3\}$, $D_3(\{x \mapsto i\}) = 0$.) Hence it is incorrect *to slice P_3 into P_x.*

We conclude that the **Observe** statement cannot be removed; this is because it is *control dependent* on the assignment to x, on which the returned x also depends.

The discussion so far suggests the following tentative correctness condition for the set Q picked by slicing:

- Q is "closed under dependence", *i.e.*, if a program point in Q depends on another program point then that program point also belongs to Q;
- Q is part of a "slicing pair": any **Observe** statement that is sliced away belongs to a set Q_0 that is also closed under dependence and is disjoint from Q.

The above condition will be made precise in Definition 5.6 (Section 5) which contains a further requirement, necessary since an **Observe** statement may be encoded as a potentially non-terminating loop, as the next example illustrates.

Example 2.4 *Consider the program P_4 given by*

$$\begin{array}{ll} 1: & x := \mathbf{Random}(\psi_4) \\ 2: & y := 0 \\ 3: & \mathbf{if}\ x \geq 2 \\ 4: & \quad \mathbf{while}\ y < 3 \\ 5: & \quad\quad C \\ 6: & \mathbf{Return}(x) \end{array}$$

where C is a (random) assignment. If C is "$y := y+1$" then the loop terminates after 3 iterations, and y's final value is 3. In the resulting distribution D', for $i \in \{0,1,2,3\}$ we have $D'(\{x \mapsto i\}) = D'(\{x \mapsto i,\ y \mapsto 3\}) = \frac{1}{4} = \Delta'(\{x \mapsto i\})$. Thus it is correct to slice P_4 into P_x.

But if C is "$y := 1$" then the program will not terminate when $x \geq 2$ (and hence the conditional encodes **Observe**$(x < 2)$*). Thus the resulting distribution D_4 is given by $D_4(\{x \mapsto i\}) = \frac{1}{4}$ when $i \in \{0,1\}$ and $D_4(\{x \mapsto i\}) = 0$ when $i \notin \{0,1\}$. Thus it is incorrect to slice P_4 into P_x. Indeed, Definition 5.6 rules out such a slicing.*

Now assume that C is "$y := \mathbf{Random}(\psi_4)$". Then the loop may iterate arbitrarily many times, but will yet terminate with probability 1, and y's final value is 3. Again, it is correct to slice P_4 into P_x.

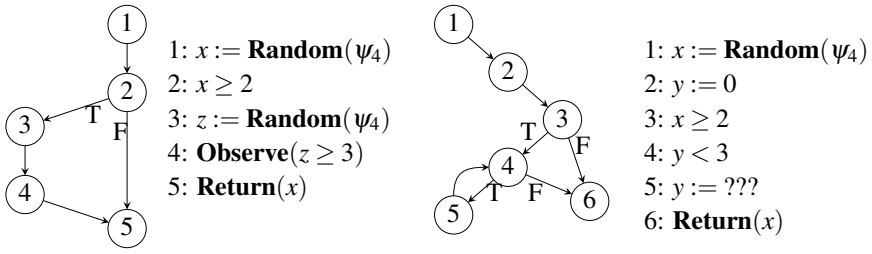

Figure 1: The pCFGs for P_3 (left) and P_4 (right) from Examples 2.3 and 2.4.

3 Probabilistic Control-Flow Graphs

This section precisely defines the kind of probabilistic control-flow graphs (pCFGs) we consider, as well as some key concepts that are mostly standard (see, *e.g.*, [17, 4]). However, we also introduce a notion (Definition 3.15) specific to our approach.

Figure 1 depicts, with the nodes numbered, the pCFGs corresponding to the programs P_3 and P_4 from Examples 2.3 and 2.4. We see that a node $v \in \mathscr{V}$ can be labeled (the label is called $Lab(v)$) with an assignment $x := E$ (x a program variable and E an arithmetic expression), with a random assignment $x :=$ **Random**(ψ) (we shall assume that the probability distribution ψ contains no program variables though it would be straightforward to allow it as in [11]), with **Observe**(B) (B is a boolean expression), or (though not part of these examples) with **Skip**; a node of the abovementioned kinds has exactly one outgoing edge. Also, there are branching nodes with *two* outgoing edges. (If v has an outgoing edge to v' we say that v' is a successor of v; a branching node has a *true*-successor and a *false*-successor.) Finally, there is a unique `End` node to which there must be a path from all other nodes but which has no outgoing edges, and a special node `Start` (which is numbered 1 in the examples) from which there is a path to all other nodes. It will often be the case that the `End` node has a label of the form **Return**(x).

We let $Def(v)$ be the variable occurring on the left hand side if v is a (random) assignment, and let $Use(v)$ be the variables used in v, that is: if v is an assignment then those occurring on the right hand side; if v is an **Observe** node or a branching node then those occurring in the boolean expression; and if v is **Return**(x) then $\{x\}$. We demand that all variables be defined before they are used: for all nodes $v \in \mathscr{V}$, for all $x \in Use(v)$, and for all paths π from `Start` to v, there must exist $v_0 \in \pi$ with $v_0 \neq v$ such that $x \in Def(v_0)$.

Definition 3.1 (Postdomination) *We say that v_1* **postdominates** *v, also written $(v, v_1) \in \mathsf{PD}$, if v_1 occurs on all paths from v to* `End`*; if also $v_1 \neq v$, v_1 is a* **proper postdominator** *of v.*

It is easy to see that the relation "postdominates" is reflexive, transitive, and antisymmetric. We say that v_1 is the ***first proper postdominator*** of v if whenever v_2 is another proper postdominator of v then all paths from v to v_2 contain v_1.

Lemma 3.2 *For any v with $v \neq$* `End`*, there is a unique first proper postdominator of v.*

Proof: This follows from Lemma 3.3 since the set of proper postdominators is non-empty (will at least contain `End`). □

We shall use the term $1PPD(v)$ for the unique first proper postdominator of v. In Figure 1(right), $1PPD(1) = 2$ (while also nodes 3 and 6 are proper postdominators of 1) and $1PPD(3) = 6$.

Lemma 3.3 *For given v, let $\prec$ be an ordering among proper postdominators of v, by stipulating that $v_1 \prec v_2$ iff in all acyclic paths from v to* `End`*, v_1 occurs strictly before v_2. Then $\prec$ is transitive, antisymmetric, and total. Also, if $v_1 \prec v_2$ then for* all *paths from v to* `End` *it is the case that the first occurrence of v_1 is before the first occurrence of v_2.*

Definition 3.4 (LAP) *For $(v,v') \in \mathsf{PD}$, we define $\mathsf{LAP}(v,v')$ as the maximum length of an acyclic path from v to v'. (The length of a path is the number of edges.)*
Thus $\mathsf{LAP}(v,v) = 0$ for all nodes v. As expected, we have:
Lemma 3.5 *If $(v,v_1) \in \mathsf{PD}$ and $(v_1,v_2) \in \mathsf{PD}$ (and thus $(v,v_2) \in \mathsf{PD}$) then $\mathsf{LAP}(v,v_2) = \mathsf{LAP}(v,v_1) + \mathsf{LAP}(v_1,v_2)$.*

To reason about cycles, it is useful to pinpoint the kind of nodes that cause cycles:
Definition 3.6 (Cycle-inducing) *A node v is* cycle-inducing *if with $v' = 1PPD(v)$ there exists a successor v_i of v such that $\mathsf{LAP}(v_i,v') \geq \mathsf{LAP}(v,v')$.*
Note that if v is cycle-inducing then v must be a branching node (since if v has only one successor then that successor is v').
Example 3.7 *In Figure 1(right), there are two branching nodes, 3 and 4, both having node 6 as their first proper postdominator. Node 4 is cycle-inducing, since 5 is a successor of 4 with $\mathsf{LAP}(5,6) = 2 > 1 = \mathsf{LAP}(4,6)$. On the other hand, node 3 is not cycle-inducing, since $\mathsf{LAP}(3,6) = 2$ which is strictly greater than $\mathsf{LAP}(4,6)$ $(= 1)$ and $\mathsf{LAP}(6,6)$ $(= 0)$.*
Lemma 3.8 *If v is cycle-inducing then there exists a cycle that contains v but not $1PPD(v)$.*
Proof: With $v' = 1PPD(v)$, by assumption there exists a successor v_i of v such that $\mathsf{LAP}(v_i,v') \geq \mathsf{LAP}(v,v')$; observe that v' is a postdominator of v_i. Let π be an acyclic path from v_i to v' with length $\mathsf{LAP}(v_i,v')$; then the path $v\pi$ is a path from v to v' that is longer than $\mathsf{LAP}(v_i,v')$, and thus also longer than $\mathsf{LAP}(v,v')$. This shows that $v\pi$ cannot be acyclic; hence $v \in \pi$ and thus $v\pi$ contains a cycle involving v but not v'. □

Lemma 3.9 *All cycles will contain at least one node which is cycle-inducing.*
Proof: Let a cycle π be given. For each $v \in \pi$, define $f(v)$ as $\mathsf{LAP}(v,\mathtt{End})$. For a node v that has only one successor, v_1, we have $f(v_1) < f(v)$ (since by Lemma 3.5, $f(v) = \mathsf{LAP}(v,v_1) + f(v_1) = 1 + f(v_1)$). Thus π must contain a branching node v_0 with a successor v_i such that $f(v_i) \geq f(v_0)$. But with $v' = 1PPD(v_0)$ we then have (by Lemma 3.5)

$$\mathsf{LAP}(v_i,v') = f(v_i) - f(v') \geq f(v_0) - f(v') = \mathsf{LAP}(v_0,v')$$

which shows that v_0 is cycle inducing. □

Definition 3.10 (Data dependence) *We say that v_2 is* **data dependent** *on v_1, written $v_1 \overset{dd}{\to} v_2$, if there exists $x \in Use(v_2) \cap Def(v_1)$, and there exists a path π (with at least one edge) from v_1 to v_2 such that $x \notin Def(v)$ for all nodes v that are interior in π.*
In Figure 1(right), $2 \overset{dd}{\to} 4$ and $5 \overset{dd}{\to} 4$. A set of nodes Q is ***closed under data dependence*** if whenever $v_2 \in Q$ and $v_1 \overset{dd}{\to} v_2$ then also $v_1 \in Q$.
Definition 3.11 (Relevant variable) *We say that x is (Q-)relevant for v, written $x \in rv_Q(v)$, if there exists $v' \in Q$ such that $x \in Use(v')$, and an acyclic path π from v to v' such that $x \notin Def(v_1)$ for all $v_1 \in \pi \setminus \{v'\}$.*
For example, in Figure 1(left), $rv_{\{4,5\}}(4) = \{x,z\}$ but $rv_{\{4,5\}}(3) = \{x\}$. The following two lemmas follow from the above definition.
Lemma 3.12 *Assume that v is an assignment, of the form $x := E$, with successor v'. If $v \in Q$ then (with $fv(E)$ the free variables in E)*

$$rv_Q(v) = (rv_Q(v') \setminus \{x\}) \cup fv(E).$$

This follows since the variables free in E are relevant before v (as $v \in Q$), and all variables relevant after v are also relevant before v except for x as it is being redefined.

Lemma 3.13 *Assume that v is a branching node, with condition B and with successors v_1 and v_2. If $v \in Q$ then*

$$rv_Q(v) = fv(B) \cup rv_Q(v_1) \cup rv_Q(v_2).$$

Lemma 3.14 *For all nodes v, and all node sets Q_1 and Q_2*
- $rv_{Q_1 \cup Q_2}(v) = rv_{Q_1}(v) \cup rv_{Q_2}(v)$
- $rv_{Q_1}(v) \cap rv_{Q_2}(v) = \emptyset$ *if Q_1 and Q_2 are both closed under data dependence, and $Q_1 \cap Q_2 = \emptyset$.*

Proof: The first claim is obvious. For the second claim, assume that Q_1 and Q_2 are closed under data dependence, that $Q_1 \cap Q_2 = \emptyset$; further assume, to get a contradiction, that $x \in rv_{Q_1}(v) \cap rv_{Q_2}(v)$. That is, there exists $v_1 \in Q_1$ with $x \in Use(v_1)$ and a path from v to v_1 that does not define x until possibly v_1, and there exists $v_2 \in Q_2$ with $x \in Use(v_2)$ and a path from v to v_2 that does not define x until possibly v_2. As we have demanded that x is defined before it is used, we infer that with π a path from `Start` to v, at least one of the nodes in π defines x; let v_x be the last such node. As Q_1 and Q_2 are closed under data dependence, we infer that $v_x \in Q_1$ and $v_x \in Q_2$, yielding the desired contradiction since $Q_1 \cap Q_2 = \emptyset$. □

Next, a concept we have discovered useful for the subsequent development:

Definition 3.15 (Staying outside until) *With v' a postdominator of v, and Q a set of nodes, we say that v* **stays outside** *Q* **until** *v' iff whenever π is a path from v to v' where v' occurs only at the end, π will contain no node in Q except possibly v'.*

In Figure 1(right), node 4 stays outside $\{1,6\}$ until 6 but does not stay outside $\{1,5,6\}$ until 6. Trivially, v stays outside Q until v for all Q and v.

Lemma 3.16 *If v stays outside Q until v' and Q is closed under data dependence then $rv_Q(v) = rv_Q(v')$.*

Proof: The claim is trivial if $v = v'$, so assume $v \neq v'$. We shall show inclusions each way.

First assume that $x \in rv_Q(v)$. Thus there exists $v_0 \in Q$ with $x \in Use(v_0)$ and a path π from v to v_0 such that $x \notin Def(v_1)$ for all $v_1 \in \pi \setminus \{v_0\}$. Since v' postdominates v, and v stays outside Q until v', we infer that v' belongs to π and thus a suffix of π is a path from v' to v_0 which shows that $x \in rv_Q(v')$.

Conversely, assume that $x \in rv_Q(v')$. Thus there exists $v_0 \in Q$ with $x \in Use(v_0)$ and a path π' from v' to v_0 such that $x \notin Def(v_1)$ for all $v_1 \in \pi' \setminus \{v_0\}$. With π an acyclic path from v to v', the concatenation of π and π' is a path from v to v_0 which will show the desired $x \in rv_Q(v)$, provided that π does not contain a node $v_1 \neq v'$ with $x \in Def(v_1)$. Towards a contradiction, assume that such a node does exist; with v_1 the last such node we would have $v_1 \xrightarrow{dd} v_0$ so from $v_0 \in Q$ and Q closed under data dependence we could infer $v_1 \in Q$ which contradicts the assumption that v stays outside Q until v'. □

4 Semantics

In this section we shall define the meaning of the pCFGs introduced in the previous section, in terms of an operational semantics that manipulates distributions which assign probabilities to stores (Section 4.1). Section 4.2 defines what it means for sets of variables to be independent wrt. a given distribution.

The semantics of pCFGs is defined in a number of steps: first (Section 4.3) we define transfer functions for traversing one edge of the pCFG, and next (Section 4.4) we present a functional, the fixed point of which provides the meaning of a pCFG. The semantics also applies to sliced programs and hence provides the meaning of slicing.

In the companion article [3], we show that for a pCFG that is the translation of a "structured" program, the semantics given in this section is adequately related to the semantics given in [10, 11] for structured probabilistic programs.

4.1 Stores and Distributions

Let $\mathscr{U}$ be the universe of variables. A store s is a partial mapping from $\mathscr{U}$ to $\mathbb{Z}$. We write $s[x \mapsto z]$ for the store s' that is like s except $s'(x) = z$, and write $dom(s)$ for the domain of s. We write $\mathscr{S}(R)$ for the set of stores with domain R, and also write $\mathscr{F}$ for $\mathscr{S}(\mathscr{U})$. If $s_1 \in \mathscr{S}(R_1)$ and $s_2 \in \mathscr{S}(R_2)$ with $R_1 \cap R_2 = \emptyset$, we may define $s_1 \oplus s_2$ with domain $R_1 \cup R_2$ the natural way. If $s \in \mathscr{S}(R')$ and $R \subseteq R'$ we define $s\,|_R$ as the restriction of s to R. With R a subset of $\mathscr{U}$, we say that s_1 agrees with s_2 on R, written $s_1 \stackrel{R}{=} s_2$, iff $R \subseteq dom(s_1) \cap dom(s_2)$ and for all $x \in R$, $s_1(x) = s_2(x)$. We assume that there is a function $[\![\ ]\!]$ such that $[\![E]\!]s$ is the integer result of evaluating E in store s and $[\![B]\!]s$ is the boolean result of evaluating B in store s (the free variables of E and B must be in $dom(s)$).

A distribution $D \in \mathscr{D}$ (we shall later also use the letter Δ) is a mapping from $\mathscr{F}$ to non-negative reals. We shall often expect that D is *bounded*, that is $\sum D \leq 1$ where $\sum D$ is a shorthand for $\sum_{s \in \mathscr{F}} D(s)$. Thanks to our assumption that values are integers, and since $\mathscr{U}$ can be assumed finite, $\mathscr{F}$ is a countable set and thus $\sum D$ is well-defined even without measure theory. We define $D_1 + D_2$ by stipulating $(D_1 + D_2)(s) = D_1(s) + D_2(s)$ (if $\sum D_1 + \sum D_2 \leq 1$ then $D_1 + D_2$ is bounded), and for $c \geq 0$ we define cD by stipulating $(cD)(s) = cD(s)$ (if D is bounded and $c \leq 1$ then cD is bounded); we write $D = 0$ when $D(s) = 0$ for all s. We say that D is *concentrated* if there exists $s_0 \in \mathscr{F}$ such that $D(s) = 0$ for all $s \in \mathscr{F}$ with $s \neq s_0$; for that s_0, we say that D is concentrated on s_0. (Thus the distribution 0 is concentrated on everything.)

Note that the set of real numbers in $[0..1]$ form a *pointed cpo* with the usual ordering, as 0 is the bottom element and the supremum operator yields the least upper bound of a chain. (We refer to Appendix A for the basics of domain theory.) Hence also the set $\mathscr{D}$ of distributions form a pointed cpo, with ordering defined pointwise ($D_1 \leq D_2$ iff $D_1(s) \leq D_2(s)$ for all stores s), with 0 the bottom element, and the least upper bound defined pointwise.

The following result is often convenient; in particular, it shows that if each distribution in a chain $\{D_k \mid k\}$ is bounded then also the least upper bound is a bounded distribution.

Lemma 4.1 *Assume that $\{D_k \mid k\}$ is a chain of distributions (not necessarily bounded). With S a (countable) set of stores, we have*

$$\sum_{s \in S} (lim_{k \to \infty} D_k)(s) = lim_{k \to \infty} \sum_{s \in S} D_k(s).$$

As suggested by the calculation in Example 2.1, we have

Definition 4.2 For $s \in \mathscr{S}(R)$, let $D(s) = \sum_{s_0 \in \mathscr{F} \mid s_0 \stackrel{R}{=} s} D(s_0)$.

Observe that $D(\emptyset) = \sum D$.

Lemma 4.3 *If $R \subseteq R'$ then for $s \in \mathscr{S}(R)$ we have*

$$D(s) = \sum_{s' \in \mathscr{S}(R') \mid s' \stackrel{R}{=} s} D(s').$$

Lemma 4.3 amounts to Definition 4.2 if $R' = \mathscr{U}$, and is trivial if $R' = R$ (as the right hand side is then the sum of the singleton set $\{D(s)\}$).

By letting $R = \emptyset$ in Lemma 4.3 we get $D(\emptyset) = \sum_{s' \in \mathscr{S}(R')} D(s')$ and thus (by renaming)

Lemma 4.4 *For all distributions D, and all R, $\sum D = \sum_{s \in \mathscr{S}(R)} D(s)$.*

Definition 4.5 (agrees with) *We say that D_1 agrees with D_2 on R, written $D_1 \stackrel{R}{=} D_2$, if $D_1(s) = D_2(s)$ for all $s \in \mathscr{S}(R)$.*

Agreement on a set implies agreement on a subset:

Lemma 4.6 *If $D_1 \stackrel{R'}{=} D_2$ and $R \subseteq R'$ then $D_1 \stackrel{R}{=} D_2$.*

Proof: For $s \in \mathscr{S}(R)$ we infer from Lemma 4.3 that

$$D_1(s) = \sum_{s' \in \mathscr{S}(R') \mid s' \stackrel{R}{=} s} D_1(s') = \sum_{s' \in \mathscr{S}(R') \mid s' \stackrel{R}{=} s} D_2(s') = D_2(s).$$

4.2 Probabilistic Independence

Some variables of a distribution D may be *independent* of other variables. That is, knowing the values of the former gives no extra information about the values of the latter, or vice versa. Formally:

Definition 4.7 (independence) Let R_1 and R_2 be disjoint sets of variables. We say that R_1 and R_2 are ***independent*** in D iff for all $s_1 \in \mathscr{S}(R_1)$ and $s_2 \in \mathscr{S}(R_2)$, we have $D(s_1 \oplus s_2)\sum D = D(s_1)D(s_2)$.

To motivate the definition, first observe that if $\sum D = 1$ it amounts to the well-known definition of probabilistic independence; next observe that if $0 < \sum D$, it is equivalent to the well-known definition of "normalized" probabilities:

$$\frac{D(s_1 \oplus s_2)}{\sum D} = \frac{D(s_1)}{\sum D} \cdot \frac{D(s_2)}{\sum D}$$

Trivially, R_1 and R_2 are independent in D if $D = 0$ or $R_1 = \emptyset$ or $R_2 = \emptyset$.

Example 4.8 *In Example 2.1, $\{x\}$ and $\{y\}$ are independent in D_1. To see this, first note that D_1 produces a non-zero value for only 8 stores: for $i = 0 \ldots 3$, these are the stores $\{x \mapsto i, y \mapsto 2\}$, $\{x \mapsto i, y \mapsto 3\}$.*

For $i \in \{0,1,2,3\}$ and $j \in \{2,3\}$ we have $D_1(\{x \mapsto i,\ y \mapsto j\}) = \frac{1}{16}$ and thus $D_1(\{x \mapsto i\}) = \frac{1}{8}$ and $D_1(\{y \mapsto j\}) = \frac{1}{4}$. As $\sum D_1 = \sum_{s \in \mathscr{F}} D_1(s) = 8 \cdot \frac{1}{16} = \frac{1}{2}$, we have the desired equality $D_1(\{x \mapsto i,\ y \mapsto j\})\sum D_1 = \frac{1}{32} = D_1(\{x \mapsto i\}) \cdot D_1(\{y \mapsto j\})$.

The equality holds trivially if $i \notin \{0,1,2,3\}$ or $j \notin \{2,3\}$ since then $D_1(\{x \mapsto i,\ y \mapsto j\}) = 0$ and either $D_1(\{x \mapsto i\}) = 0$ or $D_1(\{y \mapsto j\}) = 0$.

Example 4.9 *In Example 2.2, $\{x\}$ and $\{y\}$ are* not *independent in D_2: $D_2(\{x \mapsto 3,\ y \mapsto 3\})\sum D_2 = \frac{3}{256}$, while $D_2(\{x \mapsto 3\})D_2(\{y \mapsto 3\}) = \frac{4}{256}$.*

4.3 Transfer Functions

To deal with traversing a single edge in the pCFG, we shall define a number of functions with functionality $\mathscr{D} \to \mathscr{D}$. Each such ***transfer function*** f will be

additive if $D_1 + D_2$ is a distribution then $f(D_1 + D_2) = f(D_1) + f(D_2)$ (this reflects that a distribution is not more than the sum of its components);

multiplicative $f(cD) = cf(D)$ for all distributions D and all real $c \geq 0$;

continuous $f(lim_{k\to\infty} D_k) = lim_{k\to\infty} f(D_k)$ when $\{D_k \mid k\}$ is a chain of distributions (this is is a key property for functions on cpos, cf. Appendix A);

non-increasing $\sum f(D) \leq \sum D$ for all distributions D (this reflects that distribution may disappear, as we have seen in our examples, but cannot be created ex nihilo).

Some functions will even be

sum-preserving $\sum f(D) = \sum D$ for all distributions D (if D is such that this equation holds we say that f is sum-preserving for D).

Transfer functions for **Observe** nodes are not sum-preserving (unless the condition is always true), and neither is the semantic function for a loop that has a non-zero probability of non-termination; a primary

contribution of this article is to show that if a loop is sum-preserving it may be safe to slice it away, even if it occurs in a branch (cf. Example 2.4).

To show that a function is sum-preserving, it suffices to consider concentrated distributions:

Lemma 4.10 *Let $f \in \mathscr{D} \to \mathscr{D}$ be continuous and additive. Assume that for all D that are concentrated, f is sum-preserving for D. Then f is sum-preserving.*

For a boolean expression B, we define select_B by letting $\mathsf{select}_B(D) = D'$ where

$$\begin{array}{rcll} D'(s) & = & D(s) & \text{if } [\![B]\!]s \\ D'(s) & = & 0 & \text{otherwise} \end{array}$$

Lemma 4.11 *For all B, select_B is continuous, additive, multiplicative, and non-increasing; also, for all D we have*

$$\mathsf{select}_B(D) + \mathsf{select}_{\neg B}(D) = D.$$

Assignments For a variable x and an expression E, we define $\mathsf{assign}_{x:=E}$ by letting $\mathsf{assign}_{x:=E}(D)$ be a distribution D' such that for each $s' \in \mathscr{F}$,

$$D'(s') = \sum_{s \in \mathscr{F} \mid s' = s[x \mapsto [\![E]\!]s]} D(s)$$

That is, the "new" probability of a store s' is the sum of the "old" probabilities of the stores that become like s' after the assignment (this will happen for a store s if $s' = s[x \mapsto [\![E]\!]s]$).

Lemma 4.12 *Assume that $\mathsf{assign}_{x:=E}(D) = D'$ and $x \notin R$. Then $D \overset{R}{=} D'$.*

Lemma 4.13 *Each $\mathsf{assign}_{x:=E}$ is additive, multiplicative, non-increasing, and sum-preserving.*

Proof: That $\mathsf{assign}_{x:=E}$ is sum-preserving, and hence non-increasing, follows from Lemma 4.12, with $R = \emptyset$. Additivity and multiplicativity are trivial. □

Lemma 4.14 *$\mathsf{assign}_{x:=E}$ is continuous.*

Random Assignments For a variable x and a random distribution ψ, we define $\mathsf{rassign}_{x:=\psi}$ by letting $\mathsf{rassign}_{x:=\psi}(D)$ be a distribution D' such that for each $s' \in \mathscr{F}$,

$$D'(s') = \sum_{s \in \mathscr{F} \mid s' \overset{\mathscr{V} \setminus \{x\}}{=} s} \psi(s'(x)) D(s)$$

Lemma 4.15 *Assume that $\mathsf{rassign}_{x:=E}(D) = D'$ and $x \notin R$. Then $D \overset{R}{=} D'$.*

Lemma 4.16 *Each $\mathsf{rassign}_{x:=E}$ is additive, multiplicative, non-increasing, and sum-preserving.*

Proof: That $\mathsf{rassign}_{x:=E}$ is sum-preserving, and hence non-increasing, follows from Lemma 4.15, with $R = \emptyset$. Additivity and multiplicativity is trivial. □

Lemma 4.17 *$\mathsf{rassign}_{x:=E}$ is continuous.*

4.4 Fixed-point Semantics

Having expressed the semantics of a single edge, we shall now express the semantics of a full pCFG. Our goal is to compute "modification functions" to express how a distribution is modified as "control" moves

from `Start` to `End`. To accomplish this, we shall solve a more general problem: for each $(v,v') \in \mathsf{PD}$, state how a given distribution is modified as "control" moves from v to v' along paths that may contain multiple branches and even loops but which do *not* contain v' until the end.

We would have liked to have a definition of the modification function that is inductive in $\mathsf{LAP}(v,v')$, but this is not possible due to cycle-inducing nodes (cf. Definition 3.6). For such nodes, the semantics cannot be expressed by recursive calls on the successors, but the semantics of (at least) one of the successors will have to be provided as an *argument*. This motivates that our main semantic function be a *functional* that transforms a modification function into another modification function, with the desired meaning being the *fixed point* (cf. Lemma A.2 in Appendix A) of this functional.

We shall now specify a functional $\mathscr{H}_X$ which is parametrized on a set X of nodes; the idea is that only the nodes in X are taken into account. To get a semantics for the original program, we must let X be the set $\mathscr{V}$ of all nodes; to get a semantics for a sliced program, we must let X be the set Q of nodes included in the slice.

$\mathscr{H}_X$ operates on $\mathsf{PD} \to \mathscr{D} \to \mathscr{D}$ and we shall show (Lemma 4.20) that it even operates on $\mathsf{PD} \to \mathscr{D} \to_c \mathscr{D}$ (we let $\to_c$ denote the set of continuous functions) which, as stated in Lemma A.1 in Appendix A, is a pointed cpo:

Lemma 4.18 *$\mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$ is a pointed cpo, with the ordering given pointwise, and with least element 0 given as $\lambda(v_1,v_2).\lambda D.0$.*

Definition 4.19 ($\mathscr{H}_X$) *The functionality of $\mathscr{H}_X$ is given by*

$$\mathscr{H}_X : (\mathsf{PD} \to \mathscr{D} \to \mathscr{D}) \to (\mathsf{PD} \to \mathscr{D} \to \mathscr{D})$$

where, given

$$h_0 : \mathsf{PD} \to \mathscr{D} \to \mathscr{D}$$

we define

$$h = \mathscr{H}_X(h_0) : \mathsf{PD} \to \mathscr{D} \to \mathscr{D}$$

by letting $h(v,v')$, written $h^{(v,v')}$, be stipulated by the following rules that are inductive in $\mathsf{LAP}(v,v')$:

1. *if $v' = v$ then $h^{(v,v')}(D) = D$;*
2. *otherwise, if $v' \neq v''$ with $v'' = 1PPD(v)$ then*

$$h^{(v,v')}(D) = h^{(v'',v')}(h^{(v,v'')}(D))$$

   (this is well-defined by Lemma 3.5);
3. *otherwise, that is if $v' = 1PPD(v)$:*
   (a) *if $v \notin X$ or $Lab(v) = \mathbf{Skip}$ then $h^{(v,v')}(D) = D$;*
   (b) *if $v \in X$ with $Lab(v)$ of the form $x := E$ then $h^{(v,v')}(D) = \mathsf{assign}_{x:=E}(D)$;*
   (c) *if $v \in X$ with $Lab(v)$ of the form $x := \mathbf{Random}(\psi)$ then $h^{(v,v')}(D) = \mathsf{rassign}_{x:=\psi}(D)$;*
   (d) *if $v \in X$ with $Lab(v)$ of the form $\mathbf{Observe}(B)$ then $h^{(v,v')}(D) = \mathsf{select}_B(D)$;*
   (e) *otherwise, that is if v is a branching node with condition B, we compute $h^{(v,v')}$ as follows: with v_1 the true-successor of v and v_2 the false-successor of v, let $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$; we then let $h^{(v,v')}(D)$ be $D'_1 + D'_2$ where for each $i \in \{1,2\}$, D'_i is computed as*
      - *if $\mathsf{LAP}(v_i,v') < \mathsf{LAP}(v,v')$ then $D'_i = h^{(v_i,v')}(D_i)$;*
      - *if $\mathsf{LAP}(v_i,v') \geq \mathsf{LAP}(v,v')$ (and thus v is cycle-inducing) then $D'_i = {h_0}^{(v_i,v')}(D_i)$.*

Lemma 4.20 *Assume that ${h_0}^{(v,v')}$ is continuous for all $(v,v') \in \mathsf{PD}$ and let $h = \mathscr{H}_X(h_0)$. Then $h^{(v,v')}$ is continuous for all $(v,v') \in \mathsf{PD}$.*

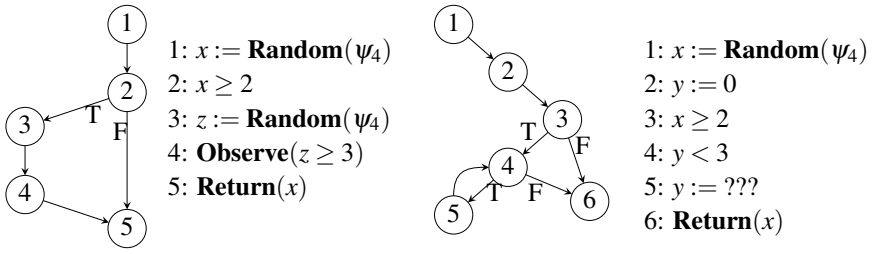

Figure 2: The pCFGs for P_3 (left) and P_4 (right) from Examples 2.3 and 2.4 (copied from Figure 1).

Proof: This follows by an easy induction in $\mathsf{LAP}(v,v')$, using Lemmas 4.11, 4.14 and 4.17, and the fact that the composition of two continuous functions is continuous. □

Thus $\mathscr{H}_X$ is a mapping from $\mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$ to itself.

Lemma 4.21 *The functional $\mathscr{H}_X$ is continuous on $\mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$.*

Lemmas 4.18 and 4.21, together with Lemma A.2 in Appendix A, give

Proposition 4.22 *The functional $\mathscr{H}_X$ has a least fixed point (belonging to $\mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$), called $fix(\mathscr{H}_X)$, and given as $lim_{k\to\infty} \mathscr{H}_X^k(0)$, that is the limit of the chain $\{\mathscr{H}_X^k(0) \mid k\}$ (where $\mathscr{H}_X^k(0)$ denotes k applications of $\mathscr{H}_X$ to the modification function that maps all distributions to 0).*

We can now define the meaning of the original program:

Definition 4.23 (Meaning of Original Program) *Given a pCFG, with $\mathscr{V}$ the set of its nodes, we define its meaning ω as $\omega = fix(\mathscr{H}_\mathscr{V})$. Thus $\omega = lim_{k\to\infty} \omega_k$ where $\omega_k = \mathscr{H}_\mathscr{V}^k(0)$ (thus $\omega_0 = 0$).*

Thus for all $k > 0$ we have $\omega_k = \mathscr{H}_\mathscr{V}(\omega_{k-1})$. Intuitively speaking, ω_k is the meaning of the program assuming that control is allowed to loop, that is move "backwards", at most $k-1$ times.

A *slice set* is a set Q of nodes (which must satisfy certain conditions, *cf.* Definition 5.6) to be included in the slice:

Definition 4.24 (Meaning of Sliced Program) *Given a pCFG, and given a slice set Q, we define the meaning of the sliced program as $\phi = fix(\mathscr{H}_Q)$. Thus $\phi = lim_{k\to\infty} \phi_k$ where $\phi_k = \mathscr{H}_Q^k(0)$.*

Example 4.25 *Consider Example 2.1, with Q containing nodes 1 and 4. Thus nodes 2 and 3 are treated like* **Skip** *nodes, and for D we thus have $\phi^{(1,4)}(D) = \mathsf{rassign}_{x:=\psi_4}(D)$.*

The following result is applicable to the original program as well as to the sliced program:

Lemma 4.26 *Let $h = fix(\mathscr{H}_X)$. Then for each $(v,v') \in \mathsf{PD}$, $h^{(v,v')}$ is additive, multiplicative and non-increasing (as is also ${\omega_k}^{(v,v')}$ for each $k \geq 0$).*

On the other hand, $h^{(v,v')}$ may fail to be sum-preserving, due to **Observe** nodes (*cf.* Examples 2.1–2.3), or due to infinite loops (*cf.* Example 2.4).

Example 4.27 *Consider Example 2.4, with pCFG depicted in the right of Figure 2 (which for the reader's convenience we copy from Figure 1). We shall consider various possibilities for the assignment at node 5; we shall prove that $\omega^{(4,6)}$ is sum-preserving when Lab(5) is an assignment $y := y+1$ or a random assignment $y := \mathbf{Random}(\psi_4)$ (but not when it is $y := 1$).*

In all cases, it is convenient to define, for integers i, j and for real $r \geq 0$, the concentrated distribution $D^r_{i,j}$ by stipulating that

$$\begin{aligned} D^r_{i,j}(s) &= r \text{ if } s = \{x \mapsto i,\ y \mapsto j\} \\ D^r_{i,j}(s) &= 0 \text{ otherwise} \end{aligned}$$

When control first reaches node 4, the distribution is $\sum_{i=2,3} D^{0.25}_{i,0}$ since y is zero, and for x, only the values 2 and 3 lead to node 4 while the values 0 and 1 do not.

To show that $\omega^{(4,6)}$ is sum-preserving, by Lemma 4.10 it is enough to show that $\omega^{(4,6)}$ is sum-preserving for each $D^r_{i,j}$. For that purpose, we shall consider a given i, and a given $r \geq 0$. Recall (Example 3.7) that $\mathsf{LAP}(5,6) > \mathsf{LAP}(4,6)$,

First consider $j \geq 3$. Then $\mathsf{select}_{y<3}(D^r_{i,j}) = 0$ and $\mathsf{select}_{\neg(y<3)}(D^r_{i,j}) = D^r_{i,j}$; thus we see from clause 3e in Definition 4.19 (substituting ω_{k-1} for h_0) that for $k \geq 1$ we have

$$\omega_k{}^{(4,6)}(D^r_{i,j}) = \omega_{k-1}{}^{(5,6)}(0) + \omega_k{}^{(6,6)}(D^r_{i,j}) = D^r_{i,j}$$

and we thus infer that

$$\forall j \geq 3 : \ \omega^{(4,6)}(D^r_{i,j}) = D^r_{i,j}. \tag{1}$$

Similarly, clause 3e in Definition 4.19 also gives us

$$\forall j < 3, \forall k \geq 1 : \ \omega_k{}^{(4,6)}(D^r_{i,j}) = \omega_{k-1}{}^{(5,6)}(D^r_{i,j}). \tag{2}$$

Also, clause 2 in Definition 4.19 (and the definition of ω_0) gives us

$$\forall k \geq 0, \forall D \in \mathit{Dist} : \ \omega_k{}^{(5,6)}(D) = \omega_k{}^{(4,6)}(\omega_k{}^{(5,4)}(D)). \tag{3}$$

We shall now look at the various cases for the assignment at node 5.

$y := 1$ *For all $k \geq 1$, and all $j < 3$, we have $\omega_k{}^{(5,4)}(D^r_{i,j}) = D^r_{i,1}$ and by (3) thus $\omega_k{}^{(5,6)}(D^r_{i,j}) = \omega_k{}^{(4,6)}(D^r_{i,1})$ (which also holds for $k \geq 0$). Thus from (2) we get that $\omega_k{}^{(4,6)}(D^r_{i,j}) = \omega_{k-1}{}^{(4,6)}(D^r_{i,1})$ for all $k \geq 1$ and $j < 3$. As $\omega_0 = 0$, we see by induction that $\omega_k{}^{(4,6)}(D^r_{i,j}) = 0$ for all $k \geq 0$ and $j < 3$, and for all $j < 3$ we thus have*

$$\omega^{(4,6)}(D^r_{i,j}) = 0$$

which confirms that from node 4 the probability of termination is zero (actually termination is impossible) and that certainly $\omega^{(4,6)}$ is not sum-preserving.

$y := y+1$ *For all $k \geq 1$, and all $j < 3$, we have $\omega_k{}^{(5,4)}(D^r_{i,j}) = D^r_{i,j+1}$ and by (3) thus $\omega_k{}^{(5,6)}(D^r_{i,j}) = \omega_k{}^{(4,6)}(D^r_{i,j+1})$ (which also holds for $k \geq 0$). Thus from (2) we get that $\omega_k{}^{(4,6)}(D^r_{i,j}) = \omega_{k-1}{}^{(4,6)}(D^r_{i,j+1})$ for all $k \geq 1$ and $j < 3$. We infer that for all $j < 3$, and all $k > 3 - j$,*

$$\omega_k{}^{(4,6)}(D^r_{i,j}) = \omega_{k-(3-j)}{}^{(4,6)}(D^r_{i,3}) = D^r_{i,3}$$

and thus we infer that for all $j < 3$ we have

$$\omega^{(4,6)}(D^r_{i,j}) = D^r_{i,3}$$

which together with (1) confirms that $\omega^{(4,6)}$ is sum-preserving as any loop from node 4 will eventually terminate.

$y := \mathbf{Random}(\psi_4)$ *For all $k \geq 1$, and all $j < 3$, we have*

$$\omega_k{}^{(5,4)}(D^r_{i,j}) = 0.25 \cdot (D^r_{i,0} + D^r_{i,1} + D^r_{i,2} + D^r_{i,3})$$

and by (3), together with the fact (Lemma 4.26) that $\omega_k{}^{(4,6)}$ is additive and multiplicative, thus

$$\omega_k{}^{(5,6)}(D^r_{i,j}) = 0.25 \cdot (\omega_k{}^{(4,6)}(D^r_{i,0}) + \omega_k{}^{(4,6)}(D^r_{i,1}) + \omega_k{}^{(4,6)}(D^r_{i,2}) + \omega_k{}^{(4,6)}(D^r_{i,3}))$$

(which also holds for $k \geq 0$) so from (2) we get that

$$\forall k \geq 1, j < 3 : \omega_k{}^{(4,6)}(D^r_{i,j}) = 0.25 \cdot \left(\sum_{q=0,1,2,3} \omega_{k-1}{}^{(4,6)}(D^r_{i,q}) \right). \tag{4}$$

One can easily prove by induction in k that if $j_1 < 3$ and $j_2 < 3$ then $\omega_k{}^{(4,6)}(D^r_{i,j_1}) = \omega_k{}^{(4,6)}(D^r_{i,j_2})$ so if we define $D_k = \omega_k{}^{(4,6)}(D^r_{i,0})$ we have $\omega_k{}^{(4,6)}(D^r_{i,j}) = D_k$ for all $j < 3$. We shall now establish

$$lim_{k\to\infty} D_k = D^r_{i,3} \tag{5}$$

which together with (1) will demonstrate that $\omega^{(4,6)}$ is sum-preserving as any loop from node 4 will terminate with probability 1.
To show (5), observe that (4) together with (1) makes it easy to prove by induction that $D_k(s) = 0 = D^r_{i,3}(s)$ for all $k \geq 0$ when $s \neq \{x \mapsto i,\ y \mapsto 3\}$, and also gives the recurrences

$$\begin{aligned} D_0(s_3) &= 0 \\ D_1(s_3) &= 0 \\ D_k(s_3) &= 0.75 \cdot D_{k-1}(s_3) + 0.25 \cdot r \text{ for } k \geq 2 \end{aligned}$$

when $s_3 = \{x \mapsto i,\ y \mapsto 3\}$. We must prove that $lim_{k\to\infty} D_k(s_3) = r$ (as $D^r_{i,3}(s_3) = r$) but this follows, with $a = 0.75$ and $b = 0.25$, from a general result:
Lemma 4.28 *If $\{x_i \mid i\}$ is a sequence of non-negative reals, satisfying $x_0 = x_1 = 0$ and $x_k = a\,x_{k-1} + b\,r$ for $k > 1$ where a, b, r are non-negative reals with $b > 0$ and $a + b = 1$, then $lim_{i\to\infty} x_i = r$.*
Proof: Observe that: (i) *$\{x_i \mid i\}$ is a chain (as can be seen by induction since $x_0 = x_1 \leq x_2$ and if $x_k \leq x_{k+1}$ then $x_{k+1} \leq x_{k+2}$);* (ii) *$x_i \leq r$ for all i since if $x_k > r$ for some k then $x_{k+1} = ax_k + br = (1-b)x_k + br = x_k + b(r - x_k) < x_k$ which contradicts $\{x_i \mid i\}$ being a chain;* (iii) *thus $lim_{i\to\infty} x_i < \infty$ and since $lim_{i\to\infty} x_i = a \cdot lim_{i\to\infty} x_i + br$ we get $b \cdot lim_{i\to\infty} x_i = (1-a) lim_{i\to\infty} x_i = br$ from which we infer the desired $lim_{i\to\infty} x_i = r$.* □

The final two lemmas below provide justification that Definition 3.15 is indeed useful: if v stays outside Q until v' then the sliced program behaves as the identity from v to v', and so does the original program — at least on the relevant variables — if it is sum-preserving,

Lemma 4.29 *Given a pCFG, a slice set Q, and $(v, v') \in \mathsf{PD}$ such that v stays outside Q until v'. We then have $\mathscr{H}_Q(h)^{(v,v')}(D) = D$ for all $D \in \mathscr{D}$ and all modification functions h.*

Lemma 4.30 *Let $(v, v') \in \mathsf{PD}$. Assume that Q is closed under data dependence and that v stays outside Q until v' (by Lemma 3.16 it thus makes sense to define $R = rv_Q(v) = rv_Q(v')$).*

For all distributions D, if $\sum \omega^{(v,v')}(D) = \sum D$ then $\omega^{(v,v')}(D) \overset{R}{=} D$.

5 Conditions for Slicing

With Q the slice set, we now develop conditions for Q that ensure semantic correctness. It is standard to require Q to be closed under data dependence, cf. Def. 3.10, and additionally also under some kind of "control dependence", a concept which in this section we shall elaborate on and then study the extra conditions needed in our probabilistic setting. Eventually, Definition 5.6 gives conditions that involve not only Q but also another slice set Q_0 containing all **Observe** nodes to be sliced away. As stated in Proposition 6.4 (Section 6.1), these conditions are sufficient to establish probabilistic independence of Q and Q_0. This in turn is crucial for establishing the correctness of slicing, as stated in Theorem 1 (Section 6.2).

Weak Slice Sets Danicic *et al.* [8] showed that various kinds of control dependence can all be elegantly expressed within a general framework whose core is the following notion:

Definition 5.1 (next visible) With Q a set of nodes, v' is a ***next visible*** in Q of v iff $v' \in Q \cup \{\texttt{End}\}$, and v' occurs on all paths from v to a node in $Q \cup \{\texttt{End}\}$.

A node v can have at most one next visible in Q. It thus makes sense to write $v' = next_Q(v)$ if v' is a next visible in Q of v. We say that Q ***provides next visibles*** iff $next_Q(v)$ exists for all nodes v. If $v' = next_Q(v)$ then v' is a postdominator of v, and if $v \in Q \cup \{\texttt{End}\}$ then $next_Q(v) = v$.

In the pCFG for P_3 (Figure 2(left)), letting $Q = \{1,3,5\}$, node 5 is a next visible in Q of 4: all paths from 4 to a node in Q will contain 5. But no node is a next visible in Q of 2: node 3 is not since there is a path from 2 to 5 not containing 3, and node 5 is not since there is a path from 2 to 3 not containing 5. Therefore Q cannot be the slice set: node 1 can have only one successor in the sliced program but we have no reason to choose either of the nodes 3 and 5 over the other as that successor. This motivates the following definition:

Definition 5.2 (weak slice set) We say that Q is a ***weak slice set*** iff it provides next visibles, and is closed under data dependence.

While the importance of "provides next visible" was recognized already in [18, 1], Danicic *et al.* were the first to realize that it is *the* key property (together with data dependence) to ensure semantically correct slicing. They call the property "weakly committing" (thus our use of "weak") and our definition differs slightly from theirs in that we always consider `End` as "visible".

Observe that the empty set is a weak slice set, since it is vacuously closed under data dependence, and since for all v we have $\texttt{End} = next_\emptyset(v)$; also the set $\mathcal{V}$ of all nodes is a weak slice set since it is trivially closed under data dependence, and since for all v we have $v = next_\mathcal{V}(v)$. The property of being a weak slice set is also closed under union:

Lemma 5.3 *If Q_1 and Q_2 are weak slice sets, also $Q_1 \cup Q_2$ is a weak slice set.*

The following result is frequently used:

Lemma 5.4 *Assume that Q and v are such that $next_Q(v)$ exists, and that $v \notin Q \cup \{\texttt{End}\}$. Then v stays outside Q until $1PPD(v)$.*

Proof: Let $v' = 1PPD(v)$ and assume, to get a contradiction, that π is a path from v to v' where v' occurs only at the end and which contains a node in $Q \setminus \{v'\}$; let v_0 be the first such node. We infer that $v_0 \neq v$ (as $v \notin Q$) and $v_0 \neq v'$, and that $v_0 = next_Q(v)$. Thus v_0 is a proper postdominator of v, which entails (since $v' = 1PPD(v)$) that v' occurs on all paths from v to v_0. For the path π it is thus the case that v' occurs before v_0, which contradicts our assumption that v' occurs only at the end. □

A weak slice set that covers all cycles must include all nodes that are cycle-inducing (cf. Definition 3.6):

Lemma 5.5 *Assume that Q provides next visibles, and that each cycle contains a node in Q. Then all cycle-inducing nodes belong to Q.*

Proof: Let v be a cycle-inducing node, and let $v' = 1PPD(v)$ and $v'' = next_Q(v)$. By Lemma 3.8 there is a cycle π which contains v but not v'. By assumption, there exists v_1 that belongs to π and also to Q. Since there is a path within π from v to v_1, v'' will occur on that path; hence v'' belongs to π, and $v'' \in Q$ (as a cycle cannot contain `End`).

We know that v'' is a postdominator of v. Assume, to get a contradiction, that v'' is a proper postdominator of v; then v' will occur on all paths from v to v'', and hence v' belongs to π which is a contradiction. We infer $v'' = v$ and thus the desired $v \in Q$. □

5.1 Adapting to the Probabilistic Setting

As already motivated through Examples 2.1–2.4, the key challenge in slicing probabilistic programs is how to handle **Observe** nodes. In Section 2 we hinted at some tentative conditions a slice set Q must satisfy; we can now phrase them more precisely:

1. Q must be a weak slice set that contains `End`, and
2. there exists another weak slice set Q_0 such that (a) Q and Q_0 are disjoint and (b) all **Observe** nodes belong to either Q or Q_0.

We shall now see how these conditions work out for our example programs.

For programs P_1, P_2, the control flow is linear and hence all nodes have a next visible, no matter the choice of Q; thus a node set is a weak slice set iff it is closed under data dependence.

For P_1 we may choose $Q = \{1,4\}$ and $Q_0 = \{2,3\}$ as they are disjoint, and both closed under data dependence. As can be seen from Defs. 4.24 and 4.19, the resulting sliced program has the same meaning as the program that results from P_1 by replacing all nodes not in Q by **Skip**, that is

$$\begin{array}{ll} 1: & x := \mathbf{Random}(\psi_4); \\ 2: & \mathbf{Skip}; \\ 3: & \mathbf{Skip}; \\ 4: & \mathbf{Return}(x) \end{array}$$

which is obviously equivalent to P_x as defined in Section 2.

Next consider the program P_2 where Q should contain 4 and hence (by data dependence) also contain 1. Now assume, in order to remove the **Observe** node (and produce P_x), that Q does not contain 3. Then Q_0 must contain 3, and (as Q_0 is closed under data dependence) also 1. But then Q and Q_0 are not disjoint, which contradicts our requirements. Thus Q does contain 3, and hence also 2. That is, $Q = \{1,2,3,4\}$. We see that the only possible slicing is the trivial one.

Any slice for P_3 (Figure 2) will also be trivial. From $5 \in Q$ we infer (by data dependence) that $1 \in Q$. Assume, to get a contradiction, that $4 \notin Q$. As 4 is an **Observe** node we must thus have $4 \in Q_0$, and for node 2 to have a next visible in Q_0 we must then also have $2 \in Q_0$ which by data dependence implies $1 \in Q_0$ which contradicts Q and Q_0 being disjoint. This shows $4 \in Q$ which implies $3 \in Q$ (by data dependence) and $2 \in Q$ (as otherwise 2 has no next visible in Q).

For P_4, we need $6 \in Q$ and by data dependence thus also $1 \in Q$; actually, our tentative conditions can be satisfied by choosing $Q = \{1,6\}$ and $Q_0 = \emptyset$, as for all $v \neq 1$ we would then have $6 = \mathit{next}_Q(v)$. From Definitions 4.24 and 4.19 we see that the resulting sliced program has the same meaning as

$$\begin{array}{ll} 1: & x := \mathbf{Random}(\psi_4); \\ 2: & \mathbf{Skip}; \\ 3: & \mathbf{Skip}; \\ 6: & \mathbf{Return}(x) \end{array}$$

Yet, in Example 2.4 we saw that in general (as when node C is labeled $y := 1$) this is *not* a correct slice of P_4. This reveals a problem with our tentative correctness conditions; they do not take into account that **Observe** nodes may be "encoded" as infinite loops.

To repair that, we shall demand that just like all **Observe** nodes must belong to either Q or Q_0, also all cycles must touch either Q or Q_0, except if the cycle is known to terminate with probability 1. Allowing this exception is an added contribution to the conference version of this article [2].

Observe that all cycles touch either Q or Q_0 iff all cycle-inducing nodes belong to Q or Q_0: "only if" follows from Lemma 5.5 (and 5.3), and "if" follows from Lemma 3.9.

We are now done motivating how to arrive at the following definition which shall serve as our final stipulation of what is a correct slice:

Definition 5.6 (slicing pair) *Let Q, Q_0 be sets of nodes. (Q, Q_0) is a* **slicing pair** *iff*

1. *Q, Q_0 are both weak slice sets with $\texttt{End} \in Q$;*
2. *Q, Q_0 are disjoint;*
3. *all* **Observe** *nodes are in $Q \cup Q_0$; and*
4. *for all cycle-inducing nodes v, either*
   (a) *$v \in Q \cup Q_0$, or*
   (b) *$\omega^{(v,1PPD(v))}$ is sum-preserving (in which case we shall say that v is sum-preserving).*

For Example 2.4, we saw in Example 3.7 that 4 is the only cycle-inducing node, and we must demand either $4 \in Q \cup Q_0$ or that $\omega^{(4,6)}$ is sum-preserving. Recalling the findings in Example 4.27, we see that:

1. If node 5 is labeled $y := y + 1$ or $y := \mathbf{Random}(\psi_4)$ then we don't need $4 \in Q \cup Q_0$, and thus $(\{1,6\}, \emptyset)$ is a valid slicing pair.
2. If node 5 is labeled $y := 1$ then we must require $4 \in Q \cup Q_0$. But $4 \in Q_0$ is impossible, as then $3 \in Q_0$ (since otherwise 3 has no next visible in Q_0) which by data dependence implies $1 \in Q_0$ which contradicts $Q \cap Q_0 = \emptyset$, since $1 \in Q$. Thus $4 \in Q$, and then $3 \in Q$ (since otherwise 3 has no next visible in Q) and $2, 5 \in Q$ (by data dependence). We see that Q contains all nodes, giving a trivial slice.

The concepts involved in Definition 5.6 help towards establishing a result showing that a larger class of semantic functions are sum-preserving:

Lemma 5.7 *Assume Q' is a node set which contains all* **Observe** *nodes, and that for each cycle-inducing node v_0, either $v_0 \in Q'$ or $\omega^{(v_0,1PPD(v_0))}$ is sum-preserving. If v stays outside Q' until v' then $\sum \omega^{(v,v')}(D) = \sum D$ for all D.*

We can now state a key result which shows that nodes not in a slicing pair are not relevant for computing the final result:

Lemma 5.8 *Assume that (Q, Q_0) is a slicing pair, and that v stays outside $Q \cup Q_0$ until v'. With $R = rv_{Q \cup Q_0}(v) = rv_{Q \cup Q_0}(v')$ (equality holds by Lemma 3.16), we have $\omega^{(v,v')}(D) \stackrel{R}{=} D$ for all D.*

Proof: With the given assumptions, Lemma 5.7 is applicable (with $Q' = Q \cup Q_0$) to establish that for all D we have $\sum \omega^{(v,v')}(D) = \sum D$. The claim now follows from Lemma 4.30. □

6 Slicing and its Correctness

In this section, we shall embark on proving the semantic correctness of the slicing conditions developed in Section 5. This will involve reasoning about the behavior of $\omega^{(v,v')}$ for $(v, v') \in \mathsf{PD}$, but which reasoning principle should we employ? Just doing induction in $\mathsf{LAP}(v, v')$ will obviously not work for a cycle-inducing node; instead, the following approach is often feasible:

1. prove results about ${\omega_k}^{(v,v')}$ by induction in k, where for each k we do an inner induction on $\mathsf{LAP}(v, v')$ (possible since $\omega_{k+1} = \mathscr{H}_{\mathscr{V}}(\omega_k)$ where $\mathscr{H}_{\mathscr{V}}(\omega_k)^{(v,v')}$ is defined inductively in $\mathsf{LAP}(v, v')$);
2. lift the results about ${\omega_k}^{(v,v')}$ to results about the limit $\omega^{(v,v')}$.

Unfortunately, this approach does not work for certain properties, such as being sum-preserving as this will hold only in the limit. We need to be more clever!

Our idea is, given a slicing pair (Q, Q_0) which will serve as implicit parameters, to introduce a family ($k \geq 0$) of functions

$$\gamma_k : \mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$$

such that we can prove that $\{\gamma_k \mid k\}$ is a chain with $lim_{k\to\infty}\gamma_k = \omega$. Of course, already $\{\omega_k \mid k\}$ is such a chain, but we shall define γ_k in a way such that we can still reason by induction, but also get certain properties already from the beginning of the fixed point iteration, not just at the limit. This is achieved by this inductive definition:

Definition 6.1 *Given a slicing pair (Q,Q_0), for $k \geq 0$ define γ_k as follows:*

$$\begin{aligned} \gamma_0{}^{(v,v')} &= \omega^{(v,v')} \text{ if } v \text{ stays outside } Q \cup Q_0 \text{ until } v' \\ \gamma_0{}^{(v,v')} &= 0 \quad \text{otherwise} \\ \gamma_k &= \mathscr{H}_{\mathscr{V}}(\gamma_{k-1}) \text{ for } k > 0 \end{aligned}$$

Lemma 6.2 *Assume that v stays outside $Q \cup Q_0$ until v'. Then $\gamma_k{}^{(v,v')} = \omega^{(v,v')}$ for all $k \geq 0$.*

Observe that $\gamma_0{}^{(v,v')} \leq \gamma_1{}^{(v,v')}$ holds for all $(v,v') \in \mathsf{PD}$ since by Lemma 6.2 we have equality when v stays outside $Q \cup Q_0$ until v', and the left hand side is 0 otherwise. Thus $\gamma_0 \leq \gamma_1$ which enables us (since $\mathscr{H}_{\mathscr{V}}$ is monotone) to infer inductively that $\{\gamma_k \mid k\}$ is a chain. Moreover, since $0 = \omega_0 \leq \gamma_0 \leq \omega$ trivially holds, we can inductively (since ω is a fixed point of $\mathscr{H}_{\mathscr{V}}$) infer that $\omega_k \leq \gamma_k \leq \omega$ for all $k \geq 0$. This allows us to deduce that $lim_{k\to\infty}\gamma_k = \omega$; we have thus proved:

Proposition 6.3 *The sequence $\{\gamma_k \mid k\}$ is a chain, with $lim_{k\to\infty}\gamma_k = \omega$.*

6.1 Probabilistic Independence

We shall now show a main contribution of this paper: that we have provided (in Definition 5.6) syntactic conditions for probabilistic independence, in that the Q-relevant variables are probabilistically independent (as defined in Definition 4.7) of the Q_0-relevant variables:

Proposition 6.4 (Independence) *Let (Q,Q_0) be a slicing pair. Assume that $\omega^{(v,v')}(D) = D'$. If $rv_Q(v)$ and $rv_{Q_0}(v)$ are independent in D then $rv_Q(v')$ and $rv_{Q_0}(v')$ are independent in D'.*

This follows from a more general result:

Lemma 6.5 *Let (Q,Q_0) be a slicing pair. Assume that $\omega^{(v,v')}(D) = D'$. Let $R = rv_Q(v)$, $R' = rv_Q(v')$, $R_0 = rv_{Q_0}(v)$, and $R'_0 = rv_{Q_0}(v')$. If R and R_0 are independent in D then*

1. *R' and R'_0 are independent in D'*
2. *if v stays outside Q until v' (and by Lemma 3.16 thus $R' = R$) then for all $s \in \mathscr{S}(R)$ we have*
$$D(s)\sum D' = D'(s)\sum D$$
3. *if v stays outside Q_0 until v' (and thus $R'_0 = R_0$) then for all $s_0 \in \mathscr{S}(R_0)$ we have*
$$D(s_0)\sum D' = D'(s_0)\sum D$$

6.2 Correctness of Slicing

We can now precisely phrase the desired correctness result, which (as hinted at in Section 2) states that the sliced program produces the same *relative* distribution over the values of the relevant variables as does the original program, and will be at least as "defined":

Theorem 1 *For a given pCFG, let (Q,Q_0) be a slicing pair, and let $\phi = fix(\mathscr{H}_Q)$ (cf. Definition 4.24) be the meaning of the sliced program. For a given $(v,v') \in \mathsf{PD}$, and a given $D \in \mathscr{D}$ such that $rv_Q(v)$ and $rv_{Q_0}(v)$ are independent in D, there exists a real number c (depending on v,v' and D) with $0 \leq c \leq 1$ such that*

$$\omega^{(v,v')}(D) \overset{rv_Q(v')}{=} c \cdot \phi^{(v,v')}(D).$$

Moreover, if v stays outside Q_0 until v' then $c = 1$.

We need to assume that the Q-relevant and Q_0-relevant variables are independent, so as to allow observe nodes in Q_0 to be sliced away (since then such nodes will not change the relative distribution of the Q-relevant variables), and also to allow certain branching nodes to be sliced away.

To prove Theorem 1 (as done at the end of this section), we need a result that involves γ_k as introduced in Definition 6.1, and which also (so as to facilitate a proof by induction in $\mathsf{LAP}(v,v')$) allows the sliced program to be given a distribution that, while agreeing on the relevant variables, may differ from the distribution given to the original program:

Lemma 6.6 *For a given pCFG, let (Q,Q_0) be a slicing pair. For all $k \geq 0$, all $(v,v') \in \mathsf{PD}$ with $R = rv_Q(v)$ and $R' = rv_Q(v')$ and $R_0 = rv_{Q_0}(v)$, all $D \in \mathscr{D}$ such that R and R_0 are independent in D, and all $\Delta \in \mathscr{D}$ such that $D \stackrel{R}{=} \Delta$, we have*

$$\gamma_k{}^{(v,v')}(D) \stackrel{R'}{=} c_{k,D}^{v,v'} \cdot \Phi_k{}^{(v,v')}(\Delta).$$

Here the numbers $c_{k,D}^{v,v'}$, and the modification functions Φ_k, are defined below (Definitions 6.7 and 6.11).

Definition 6.7 *For $k \geq 0$, $(v,v') \in \mathsf{PD}$, and $D \in \mathscr{D}$, the number $c_{k,D}^{v,v'}$ is given by the following rules that are inductive in $\mathsf{LAP}(v,v')$:*

1. *if $v = v'$ then $c_{k,D}^{v,v'} = 1$*
2. *otherwise, if $v' \neq v''$ where $v'' = 1PPD(v)$ then*

$$c_{k,D}^{v,v'} = c_{k,D}^{v,v''} \cdot c_{k,\gamma_k{}^{(v,v'')}(D)}^{v'',v'}$$

3. *otherwise, if v stays outside Q_0 until v' then $c_{k,D}^{v,v'} = 1$*
4. *otherwise, if $D = 0$ then $c_{k,D}^{v,v'} = 1$ else*

$$c_{k,D}^{v,v'} = \frac{\sum \gamma_k{}^{(v,v')}(D)}{\sum D}.$$

We could have swapped the order of the first three clauses of Definition 6.7, since it is easy to prove by induction in $\mathsf{LAP}(v,v')$ that

Lemma 6.8 *If v stays outside Q_0 until v' then $c_{k,D}^{v,v'} = 1$ for all $k \geq 0$ and $D \in \mathscr{D}$.*

Since each γ_k is non-increasing (cf. Lemma B.11), it is easy to prove by induction in $\mathsf{LAP}(v,v')$ that

Lemma 6.9 *We have $0 \leq c_{k,D}^{v,v'} \leq 1$ for all $k \geq 0$, $(v,v') \in \mathsf{PD}$, $D \in \mathscr{D}$.*

Since we know (Proposition 6.3) that $\{\gamma_k \mid k\}$ is a chain, we get:

Lemma 6.10 *$\{c_{k,D}^{v,v'} \mid k\}$ is a chain for each $(v,v') \in \mathsf{PD}$ and $D \in \mathscr{D}$.*

Definition 6.11 *Given a slicing pair (Q,Q_0), for $k \geq 0$ define Φ_k as follows:*

$$\begin{aligned} \Phi_0{}^{(v,v')}(D) &= D \text{ if } v \text{ stays outside } Q \cup Q_0 \text{ until } v' \\ \Phi_0{}^{(v,v')}(D) &= 0 \quad \text{otherwise} \\ \Phi_k &= \mathscr{H}_Q(\Phi_{k-1}) \text{ for } k > 0 \end{aligned}$$

Lemma 6.12 *$\{\Phi_k \mid k\}$ is a chain, with $lim_{k\to\infty} \Phi_k = lim_{k\to\infty} \phi_k = \phi$.*

Proof: By Lemma 4.29 we get $\phi_0 \leq \Phi_0 \leq \phi_1$ so by the monotonicity of $\mathscr{H}_Q$ we inductively get

$$\phi_k \leq \Phi_k \leq \phi_{k+1} \text{ for all } k \geq 0$$

which yields the claim. □

Proof of Theorem 1 We are given $(v,v') \in \mathsf{PD}$, and D such that $rv_Q(v)$ and $rv_{Q_o}(v)$ are independent in D; let $R' = rv_Q(v')$. For each $s' \in \mathscr{S}(R')$ we have the calculation

$$\begin{aligned} & \omega^{(v,v')}(D)(s') \\ (\textit{Proposition } 6.3) \quad &= \quad lim_{k\to\infty}\, \gamma_k{}^{(v,v')}(D)(s') \\ (\textit{Lemma } 6.6) \quad &= \quad lim_{k\to\infty}\, (c_{k,D}^{v,v'} \cdot \Phi_k{}^{(v,v')}(D)(s')) \\ (\textit{Lemma } 6.12 \quad &= \quad (lim_{k\to\infty}\, c_{k,D}^{v,v'}) \cdot \phi^{(v,v')}(D)(s'). \end{aligned}$$

With $c = lim_{k\to\infty}\, c_{k,D}^{v,v'}$ (well-defined by Lemma 6.10) we thus have

$$\omega^{(v,v')}(D) \stackrel{R'}{=} c \cdot \phi^{(v,v')}(D)$$

which yields the result since if v stays outside Q_0 until v' then $c = 1$ (by Lemma 6.8).

7 Computing the (Least) Slice

There always exists at least one slicing pair, with Q the set of all nodes and with Q_0 the empty set; in that case, the sliced program is the same as the original. Our goal, however, is to find a slicing pair (Q,Q_0) where Q is as small as possible. This section describes an algorithm for doing so.

Looking at Definition 5.6, we see a couple of potential obstacles:

1. Detecting whether a node is sum-preserving is undecidable, as it is easy to see that the halting problem can be reduced to it (see [12] for more results about the decidability of termination in a probabilistic setting).
2. Since finding the longest acyclic path is in general an NP-hard problem (as the Hamiltonian path problem can be reduced to it), it may not be feasible in polynomial time to detect whether a node is cycle-inducing — but since we only consider graphs where each node has at most two outgoing edges, and since we do not need to actually compute the longest acyclic paths but only to compare their lengths, there may still exist a polynomial algorithm for checking if a node is cycle-inducing (finding such an algorithm is a topic for future work).

Therefore, our approach shall be to assume that we have been provided (perhaps by an *oracle*) a list `ESS` that approximates the *essential* nodes:

Definition 7.1 (essential nodes) *A node v is essential iff*

1. *v is an* **Observe** *node, or*
2. *v is cycle-inducing but not sum-preserving.*

We can now provide a computable version of Definition 5.6:

Definition 7.2 *Let* `ESS` *be a set of nodes that contains all essential nodes. Then (Q,Q_0) is a* **slicing pair wrt.** `ESS` *iff*

1. *Q,Q_0 are both weak slice sets with* $\texttt{End} \in Q$*;*
2. *Q,Q_0 are disjoint;*
3. $\texttt{ESS} \subseteq Q \cup Q_0$.

If we find (Q,Q_0) satisfying Definition 7.2 then (Q,Q_0) will also satisfy Definition 5.6 (and hence Theorem 1, etc, will apply):

Proposition 7.3 *If (Q,Q_0) is a slicing pair wrt.* `ESS` *then (Q,Q_0) is a slicing pair.*

On the other hand, the converse does not necessarily hold as $\texttt{ESS} \not\subseteq Q \cup Q_0$ may happen if non-essential nodes are included in `ESS`. For example, in Example 2.4 with C as "$y := y+1$" there are no essential nodes,

and thus (cf. the discussion after Definition 5.6) $(\{1,6\},\emptyset)$ is a slicing pair. However if we were unable to infer that 4 is sum-preserving, we may have $\texttt{ESS} = \{4\}$, in which case $(\{1,6\},\emptyset)$ is not a slicing pair wrt. ESS.

To approximate the essential nodes (which is outside the scope of this article) one may use techniques from [15, 7, 9] for detecting that loops terminate with probability one, or techniques from [13] for detecting a stronger property: that the expected run-time is finite.

If the pCFG in question is a translation of a structured command (cf. the companion article [3]), it will be safe to let ESS contain (in addition to the **Observe** nodes) the branching nodes created when translating while loops (but ESS does not need to contain the branching nodes created when translating conditionals since such nodes will not be cycle-inducing).

With the set ESS given, we can now develop our algorithm to find the least Q that for some Q_0 satisfies the conditions in Definition 7.2. We shall measure its running time in terms of $|\mathscr{V}|$, the number of nodes in the pCFG; we shall often write n instead of $|\mathscr{V}|$ (note that the number of edges is at most $2n$ and thus in $O(n)$).

Our approach has four stages:

1. to compute (Section 7.1) the data dependences (in time $O(n^3)$);
2. to construct an algorithm PNV? (Section 7.2) that (in linear time) checks is a given set of nodes provides next visibles, and if not, returns a set of nodes that definitely needs to be added;
3. to construct an algorithm LWS (Section 7.3) that computes the least weak slice set that contains a given set of nodes (each call to LWS takes time in $O(n^2)$);
4. to compute (Section. 7.4) the best slicing pair wrt. the given ESS.

The resulting algorithm BSP (for best slicing pair) has a total running time in $O(n^3)$.

7.1 Computing Data Dependencies

Our algorithms use a boolean table $\texttt{DD}^*$ such that $\texttt{DD}^*(v,v')$ is true iff $v \stackrel{dd}{\rightarrow}^* v'$ where $\stackrel{dd}{\rightarrow}^*$ is the reflexive and transitive closure of $\stackrel{dd}{\rightarrow}$ defined in Definition 3.10.

Lemma 7.4 *There exists an algorithm that computes* $\texttt{DD}^*$ *in time* $O(n^3)$.

Proof: First, for each node v with $Def(v) \neq \emptyset$, we find the nodes v' with $v \stackrel{dd}{\rightarrow} v'$ which can be done in time $O(n)$ by a depth-first search which does not go past the nodes that redefine the variable defined in v. Thus in time $O(n^2)$, we can compute a boolean table DD such that $\texttt{DD}(v,v')$ is true iff $v \stackrel{dd}{\rightarrow} v'$. To compute $\texttt{DD}^*$ we now take the reflexive and transitive closure of DD which can be done in time $O(n^3)$ (for example using Floyd's algorithm). □

Given $\texttt{DD}^*$, it is easy to ensure that sets are closed under data dependence, and we shall do that in an incremental way, as stated by the following result:

Lemma 7.5 *There exists an algorithm* $\texttt{DD}^{\text{close}}$ *which given a node set Q that is closed under data dependence, and a node set Q_1, returns the least set containing Q and Q_1 that is closed under data dependence. Moreover, assuming* $\texttt{DD}^*$ *is given,* $\texttt{DD}^{\text{close}}$ *runs in time $O(n \cdot |Q_1|)$.*

7.2 Checking For Next Visibles

A key ingredient in our approach is the function PNV?, presented in Figure 3, that for a given Q checks if it provides next visibles, and if not, returns a non-empty set of nodes which must be part of any set that provides next visibles and contains Q. The function PNV? works by doing a backward breadth-first search (with F being the current "frontier") from $Q \cup \{\texttt{End}\}$ to find (using the table N that approximates "next

```
PNV?(Q)
    F ← Q ∪ {End}
    C ← ∅
    foreach v ∈ 𝒱 \ F
        N[v] ← ⊥
    foreach v ∈ F
        N[v] ← v
    while F ≠ ∅ ∧ C = ∅
        F′ ← ∅
        foreach edge from v ∉ Q to v′ ∈ F
            if N(v) = ⊥
                N(v) ← N(v′)
                F′ ← F′ ∪ {v}
            else if N(v) ≠ N(v′)
                C ← C ∪ {v}
        F ← F′
    return C
```

Figure 3: An algorithm to check if Q provides next visibles.

visible") the first node(s), if any, from which two nodes in $Q \cup \texttt{End}$ are reachable without going through Q; such "conflict" nodes are stored in C and must be included in any superset providing next visibles.

Example 7.6 *Consider the program P_1 from Example 2.1.*

- *Calling* PNV? *on* $\{1,4\}$ *returns* $\emptyset$ *after a sequence of iterations where F is first* $\{1,4\}$ *and next* $\{3\}$ *and next* $\{2\}$ *and finally* $\emptyset$.
- *Calling* PNV? *on* $\{2,3\}$ *returns* $\emptyset$ *after a sequence of iterations where F is first* $\{1\}$ *and finally* $\emptyset$.

Example 7.7 *Consider the program P_4 from Example 2.4, with pCFG depicted in Figure 2(right). Then*

- PNV?$(\{1,6\})$ *returns* $\emptyset$*, after a sequence of iterations where F is first* $\{1,6\}$ *and next* $\{3,4\}$ *and next* $\{2,5\}$ *and finally* $\emptyset$.
- PNV?$(\{2,4,5\})$ *returns* $\{3\}$*, as initially* $F = \{2,4,5,6\}$ *which causes the first iteration of the while loop to put* 3 *in* C.

The following result establishes the correctness of PNV?:

Lemma 7.8 *The function* PNV? *runs in time $O(n)$ and, given Q, returns C such that $C \cap Q = \emptyset$ and*

- *if C is empty then Q provides next visibles*
- *if C is non-empty then all supersets of Q that provide next visibles will contain C.*

7.3 Computing Least Weak Slice Set

We are now ready to define, in Figure 4, a function LWS which constructs the least weak slice set that contains a given set $\hat{Q}$; it works by successively adding nodes to the set until it is closed under data dependence, and provides next visibles.

Example 7.9 *We shall continue Example 7.6 (which considers the program P_1 from Example 2.1). First observe that the non-trivial true entries of* $\texttt{DD}^*$ *are* $(1,4)$ *(since* $1 \xrightarrow{dd} 4$*) and* $(2,3)$.

- *When running* LWS *on* $\{4\}$*, initially* $Q = \{1,4\}$ *which is also the final value of Q since* PNV?$(\{1,4\})$ *returns* $\emptyset$.
- *When running* LWS *on* $\{3\}$*, initially* $Q = \{2,3\}$ *which is also the final value of Q since* PNV?$(\{2,3\})$ *returns*

```
LWS(Q̂)
  Q ← DD^close(∅, Q̂)
  C ← PNV?(Q)
  while C ≠ ∅
    Q ← DD^close(Q, C)
    C ← PNV?(Q)
  return Q
```

Figure 4: An algorithm that finds the least weak slice set containing $\hat{Q}$.

$\emptyset$.

Example 7.10 *We shall continue Example 7.7 (which considers the program P_4 from Example 2.4, with pCFG depicted in Figure 2(right)).*

First observe that $\overset{dd}{\to}$ is given as follows: $1 \overset{dd}{\to} 3$, $1 \overset{dd}{\to} 6$, $2 \overset{dd}{\to} 4$, $2 \overset{dd}{\to} 5$, $5 \overset{dd}{\to} 4$, and $5 \overset{dd}{\to} 5$.

- *When running* LWS *on $\{6\}$, initially $Q = \{1,6\}$ which is also the final value of Q since* PNV?$(\{1,6\})$ *returns $\emptyset$.*
- *When running* LWS *on $\{4\}$, we initially have $Q = \{2,4,5\}$. The first call to* PNV? *thus (Example 7.7) returns $\{3\}$. Since $1 \overset{dd}{\to} 3$ holds, the next iteration of* LWS *will have $Q = \{1,2,3,4,5\}$ which is also the final value of Q since* PNV? *will return $\emptyset$ on that set.*

The following result establishes the correctness of LWS:

Lemma 7.11 *The function* LWS*, given $\hat{Q}$, returns Q such that*

- *Q is a weak slice set*
- *$\hat{Q} \subseteq Q$*
- *if Q' is a weak slice set with $\hat{Q} \subseteq Q'$ then $Q \subseteq Q'$.*

Moreover, assuming DD* *is given,* LWS *runs in time $O(n^2)$.*

7.4 Computing The Best Slicing Pair

We are now ready to define, in Figure 5, an algorithm BSP which given a set ESS that contains all essential nodes (for an implicitly given pCFG) returns a slicing pair (Q, Q_0) such that $Q \subseteq Q'$ for any other slicing pair $(Q', \_)$. The idea is to build Q incrementally, with Q initially containing only End; each iteration will process the nodes in ESS that are not already in Q, and add them to Q (via F) if they cannot be placed in Q_0 without causing Q and Q_0 to overlap.

Example 7.12 *We shall continue Examples 7.6 and 7.9 (which consider the program P_1 from Example 2.1). Here 3 is the only essential node so we may assume that* ESS $= \{3\}$*;* BSP *thus needs to run* LWS *on $\{4\}$ and on $\{3\}$ and from Example 7.9 we see that we get $Q_4 = \{1,4\}$ and $Q_3 = \{2,3\}$. When the members of $W = \{3\}$ are first examined in the* BSP *algorithm, we have $Q = Q_4$ and thus $Q_3 \cap Q = \emptyset$. Hence the while loop terminates after one iteration, with $Q = \{1,4\}$, and subsequently we get $Q_0 = Q_3 = \{2,3\}$.*

Example 7.13 *We shall continue Examples 7.7 and 7.10 (which consider the program P_4 from Example 2.4, with pCFG depicted in Figure 2(right)). We know from Example 3.7 that node 4 is cycle-inducing but node 3 is not; in Example 4.27 we showed that node 4 is essential when Lab(5) is an assignment $y := 1$ (as then $\omega^{(4,6)}$ is not sum-preserving) and that node 4 is not essential when Lab(5) is an assignment $y := y+1$ or a random assignment $y :=$ **Random**(ψ_4) (as then $\omega^{(4,6)}$ is sum-preserving).*

There are thus two natural possibilities for ESS*: the set $\{4\}$, and the empty set; we shall consider both:*

- *First assume that* ESS $= \emptyset$*.* BSP *thus needs to run* LWS *on only $\{6\}$, and from Example 7.10 we see that*

```
BSP(ESS)
    W ← ESS
    foreach v ∈ W ∪ {End}
        Q_v ← LWS({v})
    Q ← ∅
    F ← Q_End
    while F ≠ ∅
        Invariants:
            Q and F are both weak slice sets, with End ∈ Q ∪ F
            W ⊆ ESS and if v ∈ W then Q_v ∩ Q = ∅
            if v ∈ ESS but v ∉ W then v ∈ Q ∪ F
            if (Q', Q'_0) is a slicing pair wrt. ESS then Q ∪ F ⊆ Q'
        Q ← Q ∪ F
        F ← ∅
        foreach v ∈ W
            if Q_v ∩ Q ≠ ∅
                W ← W \ {v}
                F ← F ∪ Q_v;
    Q_0 ← ⋃_{v∈W} Q_v
    return (Q, Q_0)
```

Figure 5: Finding the best slicing pair (BSP).

we get $Q_6 = \{1,6\}$. As $W = \emptyset$, the while loop terminates after one iteration with $Q = Q_6 = \{1,6\}$, and subsequently we get $Q_0 = \emptyset$.

- *Next assume that* $\texttt{ESS} = \{4\}$. BSP *thus needs to run* LWS *on* $\{4\}$ *and* $\{6\}$*, and from Example 7.10 we see that we get* $Q_4 = \{1,2,3,4,5\}$ *and* $Q_6 = \{1,6\}$.
  When the members of $W = \{4\}$ *are first examined in the* BSP *algorithm, we have* $Q = Q_6$ *and thus* $Q_4 \cap Q = \{1\} \neq \emptyset$. *Hence W will become empty, and eventually the loop will terminate with* $Q = Q_6 \cup Q_4 = \{1,2,3,4,5,6\}$ *(and we also get* $Q_0 = \emptyset$*).*

That BSP produces the *best slicing pair* is captured by the following result:

Theorem 2 *The algorithm* BSP *returns, given a pCFG and a set of nodes* ESS*, sets Q and* Q_0 *such that*

- (Q, Q_0) *is a slicing pair wrt.* ESS
- *if* (Q', Q'_0) *is a slicing pair wrt.* ESS *then* $Q \subseteq Q'$.

Moreover, BSP *runs in time* $O(n^3)$ *(with n the number of nodes in the pCFG).*

We do not expect that there exists an algorithm wih lower asymptotic complexity, since we need to compute data dependencies which is known to involve computing a transitive closure.

8 Improving Precision

Section 7 presented an algorithm for computing the least slice satisfying Definition 7.2; such a slice will also satisfy Definition 5.6 and hence be semantically correct (as phrased in Theorem 1). Still, a smaller semantically correct slice may exist; in this section we briefly discuss two approaches for finding such slices: semantic analysis of the pCFG, and syntactic transformation of the pCFG. (Obviously, it is undecidable to always find the smallest semantically correct slice.)

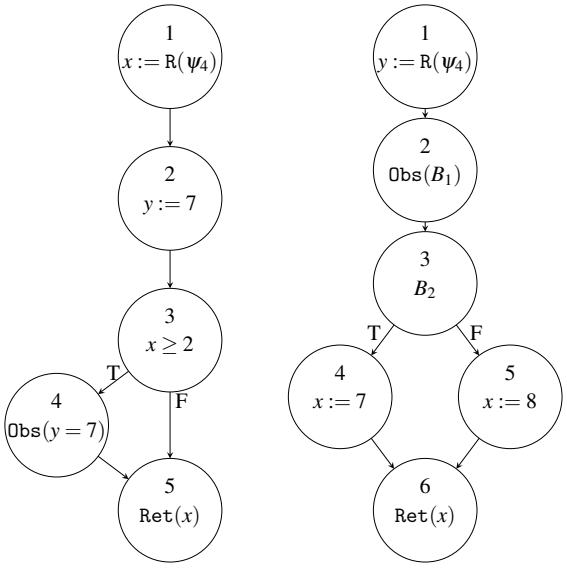

Figure 6: A redundant **Observe** node (left) and a potentially redundant branch (right).

8.1 Improvement by Semantic Analysis

Already in Section 7 we discussed how a precise (termination) analysis may help us to construct a set ESS that contains fewer (if any) non-essential nodes which in turn may enable us to slice away some loops.

The size of the slice may also be reduced if a semantic analysis can determine that a boolean expression always evaluates to *true*. This is illustrated by the pCFGs in Figure 6, as we shall now discuss.

First consider the pCFG on the left. As $y = 7$ holds at node 4, the **Observe** statement can be discarded, and indeed, the pCFG is semantically equivalent to the pCFG containing only nodes 1 and 5. Yet it has no smaller syntactic slice, since if (Q, Q_0) is a slicing pair, implying $5 \in Q$ and thus $1 \in Q$, then $Q = \{1,2,3,4,5\}$ as we now show. If $4 \in Q_0$ then $3 \in Q_0$ (as Q_0 provides next visibles) and thus $1 \in Q_0$ (by data dependence) which contradicts $Q \cap Q_0 = \emptyset$. As 4 (as it is essential) must belong to $Q \cup Q_0$, we see that $4 \in Q$; but then $2 \in Q$ (by data dependence) and $3 \in Q$ (as Q provides next visibles).

Next consider the (generic) pCFG on the right, where B_1 and B_2 are expressions involving y. There exists no smaller syntactic slice, since if (Q, Q_0) is a slicing pair and thus $6 \in Q$ then (by data dependence) $4, 5 \in Q$ and thus (as Q provides next visibles) $3 \in Q$ and thus (by data dependence) $1 \in Q$; also $2 \in Q$ as otherwise $2 \in Q_0$ and thus (by data dependence) $1 \in Q_0$ which contradicts $Q \cap Q_0 = \emptyset$. Still, if say B_2 is a logical consequence of B_1, then it is semantically sound to slice away nodes 3 and 5. Thus, even though $(Q, Q_0) = (\{1,2,4,6\}, \emptyset)$ is not a slicing pair according to Definition 5.6 as 3 has no next visible in $\{1,2,4,6\}$, it may be considered a "semantically valid slicing pair".

8.2 Improvement by Syntactic Transformation

Simple analyses like constant propagation may improve the precision of slicing even in a deterministic setting, but the probabilistic setting gives an extra opportunity: after an **Observe**(B) node, we know that B holds. As richly exploited in [11], a simple syntactic transformation often suffices to get the benefits of that information, as we illustrate on the program from [11, Figure 4] whose pCFG (in slightly modified form) is depicted in Figure 7. In our setting, if (Q, Q_0) with $18 \in Q$ is the best slicing pair, then Q will contain

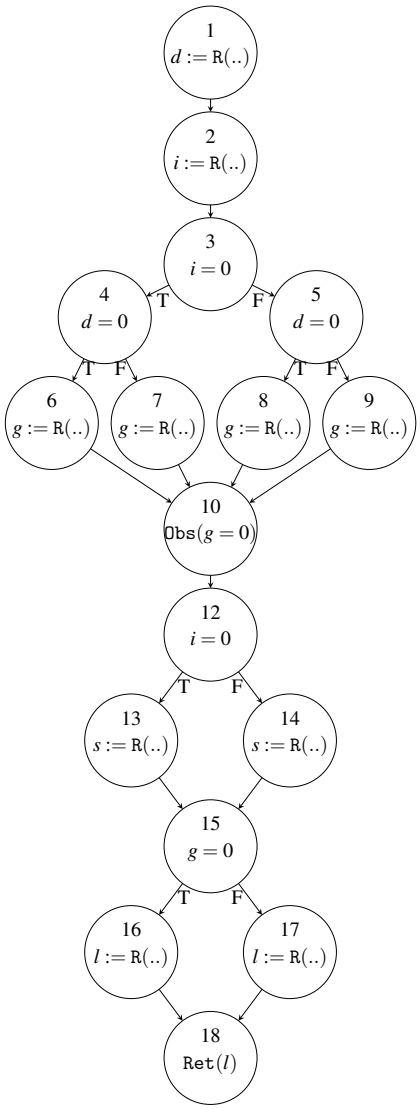

Figure 7: The program from Figure 4 of Hur et al. (modified).

everything except nodes 12, 13, 14, as can be seen as follows: $16, 17 \in Q$ by data dependence; $15 \in Q$ as Q provides next visibles; $6, 7, 8, 9 \in Q$ by data dependence; $3, 4, 5 \in Q$ as Q provides next visibles; $1, 2 \in Q$ by data dependence; also $10 \in Q$ as otherwise $10 \in Q_0$ and thus also $9 \in Q_0$ which contradicts $Q \cap Q_0 = \emptyset$.

Alternatively, suppose we insert a node 11 labeled $g := 0$ between nodes 10 and 12. This clearly preserves the semantics, but allows a much smaller slice: choose $Q = \{11, 15, 16, 17, 18\}$ and $Q_0 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. This is much like what is arrived at (through a more complex process) in [11, Figure 15].

Future work involves exploring a larger range of examples, and (while somewhat orthogonal to the current work) investigating useful techniques for computing slices that are smaller than the least syntactic slice yet semantically correct.

9 Conclusion and Related Work

We have developed a theory for the slicing of probabilistic imperative programs. We have used and extended techniques from the literature [17, 4, 18, 1] on the slicing of deterministic imperative programs. These frameworks, some of which have been partly verified by mechanical proof assistants [21, 5], were recently coalesced by Danicic *et al.* [8] who provide solid semantic foundations for the slicing of a large class of deterministic programs. Our extension of that work is non-trivial in that we need to capture probabilistic independence between two sets of variables, as done in Proposition 6.4, which requires *two* slice sets rather than one. The technical foundations of our work rest on a novel semantics of pCFGs. In a companion article [3] we establish an adequacy result that shows that for pCFGs that are translations of programs in a structured probabilistic language, our semantics is suitably related to that language's denotational semantics as formulated first by Kozen [14] and later augmented by Gordon *et al.* [10] (in particular to handle conditioning).

We were directly inspired by Hur *et al.* [11] who point out the challenges involved in the slicing of probabilistic programs, and present an algorithm which constructs a semantically correct slice. The paper does not state whether it is in some sense the least possible slice; neither does it address the complexity of the algorithm. While Hur *et al.*'s approach differs from ours, for example it is for a structured language and uses the denotational semantics presented by Gordon *et al.* [10], it is not surprising that their correctness proof also has probabilistic independence (termed "decomposition") as a key notion. Our theory separates specification and implementation which we believe provides for a cleaner approach. But as mentioned in Section 8, they incorporate powerful optimizations that we do not (yet) allow.

Future work includes investigating how our techniques can be used to analyze which sets of variables in a given probabilistic program are probabilistically independent of each other (a topic explored in, for example, [6]).

Acknowledgements. We much appreciate the feedback we have received on earlier versions; in addition to anonymous conference reviewers, we would in particular like to thank Gordon Stewart.

References

[1] T. Amtoft. Slicing for modern program structures: A theory for eliminating irrelevant loops. *Information Processing Letters*, 106(2):45–51, Apr. 2008.

[2] T. Amtoft and A. Banerjee. A theory of slicing for probabilistic control flow graphs. In B. Jacobs and C. Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference (FoSSaCS)*, volume 9634 of *Lecture Notes in Computer Science*, pages 180–196. Springer-Verlag, 2016.

[3] T. Amtoft and A. Banerjee. A semantics for probabilistic control flow graphs. Submitted for publication, 2017.

[4] T. Ball and S. Horwitz. Slicing programs with arbitrary control flow. In P. Fritzon, editor, *Proceedings of the First International Workshop on Automated and Algorithmic Debugging (AADEBUG '93)*, volume 749 of *LNCS*, pages 206–222, London, UK, 1993. Springer-Verlag.

[5] S. Blazy, A. Maroneze, and D. Pichardie. Verified validation of program slicing. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15, pages 109–117, New York, NY, USA, 2015. ACM.

[6] O. Bouissou, E. Goubault, S. Putot, A. Chakarov, and S. Sankaranarayanan. Uncertainty propagation using probabilistic affine forms and concentration of measure inequalities. In M. Chechik and J.-F.

Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016*, volume 9636 of *LNCS*, pages 225–243. Springer Berlin Heidelberg, 2016.

[7] A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 511–526. Springer Berlin Heidelberg, 2013.

[8] S. Danicic, R. W. Barraclough, M. Harman, J. D. Howroyd, Á. Kiss, and M. R. Laurence. A unifying theory of control dependence and its application to arbitrary program structures. *Theoretical Computer Science*, 412(49):6809–6842, Nov. 2011.

[9] L. M. F. Fioriti and H. Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 489–501, 2015.

[10] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In M. B. Dwyer and J. Herbsleb, editors, *ICSE, Future of Software Engineering track*, FOSE 2014, pages 167–181, New York, NY, USA, 2014. ACM.

[11] C.-K. Hur, A. V. Nori, S. K. Rajamani, and S. Samuel. Slicing probabilistic programs. In K. Pingali, editor, *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 133–144, New York, NY, USA, 2014. ACM.

[12] B. L. Kaminski and J.-P. Katoen. On the hardness of almost–sure termination. In G. F. Italiano, G. Pighizzini, and D. T. Sannella, editors, *Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, pages 307–318, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[13] B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. Weakest precondition reasoning for expected runtimes of probabilistic programs. In *ESOP'16*, volume 9632 of *LNCS*, pages 364–389. Springer Verlag, 2016.

[14] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[15] D. Monniaux. An abstract analysis of the probabilistic termination of programs. In *8th International Static Analysis Symposium (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 111–126. Springer Verlag, 2001.

[16] P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

[17] A. Podgurski and L. A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Transactions on Software Engineering*, 16(9):965–979, Sept. 1990.

[18] V. P. Ranganath, T. Amtoft, A. Banerjee, J. Hatcliff, and M. B. Dwyer. A new foundation for control dependence and slicing for modern program structures. *ACM Trans. Program. Lang. Syst. (TOPLAS)*, 29(5), Aug. 2007. A special issue with extended versions of selected papers from the 14th European Symposium on Programming (ESOP'05).

[19] D. A. Schmidt. *Denotational Semantics, a Methodology for Language Development*. Allyn and Bacon, Boston, 1986.

[20] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, 1995.

[21] D. Wasserrab. *From Formal Semantics to Verified Slicing*. PhD thesis, Karlsruher Institut für Technologie, 2010.

[22] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[23] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

A Domain Theory

This section summarizes key aspects of domain theory, as presented in, *e.g.*, [19, 23].

A domain is a set D equipped with a partial order $\sqsubseteq$, that is $\sqsubseteq$ is reflexive, transitive, and anti-symmetric. A chain $\{x_k \mid k\}$ is a mapping from the natural numbers into D such that if $i < j$ then $x_i \sqsubseteq x_j$. We say that D is a *cpo* if each chain $\{x_k \mid k\}$ has a *least upper bound* (also called *limit*), that is $x \in D$ such that $x_k \sqsubseteq x$ for all k and such that if also $x_k \sqsubseteq y$ for all k then $x \sqsubseteq y$; we shall often write $lim_{k\to\infty} x_k$ for that least upper bound. We say that a cpo is a *pointed* cpo if there exists a least element, that is an element $\bot$ such that $\bot \sqsubseteq x$ for all $x \in D$.

We say that a domain D is *discrete* if $x \sqsubseteq y$ implies $x = y$; a discrete domain is trivially a cpo (but not a pointed cpo unless a singleton).

A function f from a cpo D_1 to a cpo D_2 is *continuous* if for each chain $\{x_k \mid k\}$ in D_1 the following holds: $\{f(x_k) \mid k\}$ is a chain in D_2, and $lim_{k\to\infty} f(x_k) = f(lim_{k\to\infty} x_k)$. We let $D_1 \to_c D_2$ denote the set of continuous functions from D_1 to D_2. A continuous function f is also monotone, that is $f(x_1) \sqsubseteq f(x_2)$ when $x_1 \sqsubseteq x_2$ (for then $x_1, x_2, x_2, x_2 \ldots$. is a chain and by continuity thus $f(x_2)$ is the least upper bound of $f(x_1), f(x_2)$ implying $f(x_1) \sqsubseteq f(x_2)$).

Lemma A.1 *Let D_1 and D_2 be cpos. Then $D_1 \to_c D_2$ is a cpo, with ordering defined pointwise: $f_1 \sqsubseteq f_2$ iff $f_1(x) \sqsubseteq f_2(x)$ for all $x \in D_1$.*

If D_2 is a pointed cpo then also $D_1 \to_c D_2$ is a pointed cpo.

If D_1 is discrete then $D_1 \to_c D_2$ contains all functions from D_1 to D_2 (and thus we may just write $D_1 \to D_2$).

Proof: Let $\{f_k \mid k\}$ be a chain of continuous functions from D_1 to D_2, with f their pointwise limit, that is: $f(x) = lim_{k\to\infty} f_k(x)$ for all $x \in D_1$. We have to show that f is continuous. But if $\{x_k \mid k\}$ is a chain in D_1 then

$$\begin{aligned} f(lim_{k\to\infty} x_k) &= lim_{m\to\infty} f_m(lim_{k\to\infty} x_k) \\ &= lim_{m\to\infty} lim_{k\to\infty} f_m(x_k) \\ &= lim_{k\to\infty} lim_{m\to\infty} f_m(x_k) \\ &= lim_{k\to\infty} f(x_k). \end{aligned}$$

If D_2 has a bottom element $\bot$ then $\lambda x.\bot$ is the bottom element in $D_1 \to_c D_2$, and if D_1 is discrete then all functions from D_1 to D_2 are continuous since a chain in D_1 can contain only one element. □

Lemma A.2 *Let f be a continuous function on a pointed cpo D. Then*[1] *$\{f^k(\bot) \mid k\}$ is a chain, and $lim_{k\to\infty} f^k(\bot)$ is the least fixed point of f.*

Proof: From $\bot \sqsubseteq f(\bot)$ we by monotonicity of f infer that $f^k(\bot) \sqsubseteq f^{k+1}(\bot)$ for all k so $\{f^k(\bot) \mid k\}$ is indeed a chain. With $y = lim_{k\to\infty} f^k(\bot)$ we see by continuity of f that y is indeed a fixed point of f: $f(y) = lim_{k\to\infty} f^{k+1}(\bot) = y$. And if z is also a fixed point, we have $\bot \sqsubseteq z$ and by monotonicity of f thus $f^k(\bot) \sqsubseteq f^k(z) = z$ for all k, from which we infer $y \sqsubseteq z$. □

[1] Recall that f^k is defined by letting $f^0(x) = x$, and $f^{k+1}(x) = f(f^k(x))$ for $k \geq 0$.

B Miscellaneous Proofs

B.1 Proofs for Section 3

Lemma 3.3: For given v, let $\prec$ be an ordering among proper postdominators of v, by stipulating that $v_1 \prec v_2$ iff in all acyclic paths from v to `End`, v_1 occurs strictly before v_2. Then $\prec$ is transitive, antisymmetric, and total. Also, if $v_1 \prec v_2$ then for *all* paths from v to `End` it is the case that the first occurrence of v_1 is before the first occurrence of v_2.

Proof: The first two properties are obvious.

We next show that $\prec$ is total. Assume, to get a contradiction, that there exists an acyclic path π_1 from v to `End` that contains v_1 strictly before v_2, and also an acyclic path π_2 from v to `End` that contains v_2 strictly before v_1. But then the concatenation of the prefix of π_1 that ends with v_1, and the suffix of π_2 that starts with v_1, is a path from v to `End` that avoids v_2, yielding a contradiction as v_2 postdominates v.

Finally, assume that $v_1 \prec v_2$, and that π is a path from v to `End`; to get a contradiction, assume that there is a prefix π_1 of π that ends with v_2 but does not contain v_1. Since there exists an acyclic path from v to `End`, we infer from $v_1 \prec v_2$ that there is an acyclic path π_2 from v_2 that does not contain v_1. But the concatenation of π_1 and π_2 is a path from v to `End` that does not contain v_1, which contradicts v_1 being a proper postdominator of v. □

Lemma 3.5: If $(v,v_1) \in \mathsf{PD}$ and $(v_1,v_2) \in \mathsf{PD}$ (and thus $(v,v_2) \in \mathsf{PD}$) then $\mathsf{LAP}(v,v_2) = \mathsf{LAP}(v,v_1) + \mathsf{LAP}(v_1,v_2)$.

Proof: If $v = v_1$ or $v_1 = v_2$, the claim is obvious; we can thus assume that v_1 and v_2 are proper postdominators of v and by Lemma 3.3 we further infer that v_1 will occur before v_2 in all paths from v to `End`.

First consider an acyclic path π from v to v_2. We have argued that π will contain v_1, and hence π is the concatenation of an acyclic path from v to v_1, thus of length $\leq \mathsf{LAP}(v,v_1)$, and an acyclic path from v_1 to v_2, thus of length $\leq \mathsf{LAP}(v_1,v_2)$. Thus the length of π is $\leq \mathsf{LAP}(v,v_1) + \mathsf{LAP}(v_1,v_2)$; as π was an arbitrary acyclic path from v to v_2, this shows "$\leq$".

To show "$\geq$", let π_1 be an acyclic path from v to v_1 of length $\mathsf{LAP}(v,v_1)$, and π_2 be an acyclic path from v_1 to v_2 of length $\mathsf{LAP}(v_1,v_2)$. Let π be the concatenation of π_1 and π_2; π is an acyclic path from v to v_2 since if $v' \neq v_1$ occurs in both paths then there is a path from v to v_2 that avoids v_1 which is a contradiction. As π is of length $\mathsf{LAP}(v,v_1) + \mathsf{LAP}(v_1,v_2)$, this shows "$\geq$". □

Lemma B.1 *Assume that v' is a proper postdominator of v, that with $v'' = 1PPD(v)$ we have $v' \neq v''$, and that Q is a set of nodes.*

If v stays outside Q until v' then (i) *v stays outside Q until v'', and* (ii) *v'' stays outside Q until v'.*

Proof: For (i), let π be a path from v to v'' that contains v'' only at the end. Hence v' cannot be in π (as v'' occurs in all paths from v to v'), so we can extend π into a path π' from v to v' that contains v' only at the end. Since v stays outside Q until v', π' contains no node in Q except possibly v', and hence π contains no node in Q.

For (ii), let π be a path from v'' to v' that contains v' only at the end. There is a path from v to v'' that does not contain v', so we can extend π into a path π' from v to v' that contains v' only at the end. Since v stays outside Q until v', π' contains no node in Q except possibly v', and hence π contains no node in Q except possibly v'. □

Lemma B.2 *Assume that v' is a proper postdominator of v, that v_1 is a successor of v, and that Q is a set of*

nodes.

If v stays outside Q until v' then also v_1 stays outside Q until v'.

Proof: Let π be a path from v_1 to v' that contains v' only at the end. Since $v \neq v'$, we can extend π into a path π' from v to v' that contains v' only at the end. Since v stays outside Q until v', π' contains no node in Q except possibly v', and hence π contains no node in Q except possibly v'. □

B.2 Proofs for Section 4

Lemma 4.1: Assume that $\{D_k \mid k\}$ is a chain of distributions (not necessarily bounded) with $D' = lim_{k\to\infty} D_k$. With S a (countable) set of stores, we have

$$\sum_{s\in S} D'(s) = lim_{k\to\infty} \sum_{s\in S} D_k(s)$$

Proof: From $D_k \leq D'$ we get that $\sum_{s\in S} D'(s)$ is an upper bound for $\{\sum_{s\in S} D_k(s) \mid k\}$; as $lim_{k\to\infty} \sum_{s\in S} D_k(s)$ is the least upper bound, we get

$$lim_{k\to\infty} \sum_{s\in S} D_k(s) \leq \sum_{s\in S} D'(s).$$

To establish that equality holds, we shall assume $lim_{k\to\infty} \sum_{s\in S} D_k(s) < \sum_{s\in S} D'(s)$ so as to get a contradiction. Then there exists $\varepsilon > 0$ such that $lim_{k\to\infty} \sum_{s\in S} D_k(s) + \varepsilon < \sum_{s\in S} D'(s)$. We infer that there exists a finite set S_0 with $S_0 \subseteq S$ such that $lim_{k\to\infty} \sum_{s\in S} D_k(s) + \varepsilon < \sum_{s\in S_0} D'(s)$. For each $s \in S_0$ there exists K_s such that $D_k(s) > D'(s) - \varepsilon/|S_0|$ for $k \geq K_s$, and thus there exists K (the maximum element of the finite set $\{K_s \mid s \in S_0\}$) such that for each $s \in S_0$, and each $k \geq K$, $D_k(s) + \varepsilon/|S_0| > D'(s)$. But then we get the desired contradiction:

$$\begin{aligned} \sum_{s\in S_0} D'(s) &< \sum_{s\in S_0} (D_K(s) + \varepsilon/|S_0|) = \sum_{s\in S_0} D_K(s) + \varepsilon \leq lim_{k\to\infty} \sum_{s\in S} D_k(s) + \varepsilon \\ &< \sum_{s\in S_0} D'(s). \end{aligned}$$

□

Lemma 4.3: If $R \subseteq R'$ then for $s \in \mathscr{S}(R)$ we have

$$D(s) = \sum_{s' \in \mathscr{S}(R') \mid s' \overset{R}{=} s} D(s').$$

Proof: We have the calculation

$$\begin{aligned} \sum_{s' \in \mathscr{S}(R') \mid s' \overset{R}{=} s} D(s') &= \sum_{s' \in \mathscr{S}(R') \mid s' \overset{R}{=} s} \left(\sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R'}{=} s'} D(s_0) \right) \\ &= \sum_{s_0 \in \mathscr{F},\ s' \in \mathscr{S}(R') \mid s' \overset{R}{=} s,\ s_0 \overset{R'}{=} s'} D(s_0) = \sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R}{=} s} D(s_0) = D(s) \end{aligned}$$

where the third equality is justified as follows: for a given $s_0 \in \mathscr{F}$, exactly one $s' \in \mathscr{S}(R')$ will satisfy $s_0 \overset{R'}{=} s'$, and for that s' we will have (since $R \subseteq R'$) that $s' \overset{R}{=} s$ iff $s_0 \overset{R}{=} s$. □

Lemma 4.10: Let $f \in \mathscr{D} \to \mathscr{D}$ be continuous and additive. Assume that for all D that are concentrated, f is sum-preserving for D. Then f is sum-preserving.

Proof: Let s_1, s_2, ... be an enumeration of stores in $\mathscr{F}$. For given $D \in \mathscr{D}$, and for each $k \geq 1$, let D_k be given by stipulating $D_k(s_k) = D(s_k)$ but $D_k(s) = 0$ when $s \neq s_k$. Thus each D_k is concentrated, and $D = lim_{k\to\infty} D'_k$ where $D'_k = D_1 + \ldots + D_k$. Since f is assumed continuous and additive,

$$f(D) = f(lim_{k\to\infty} D'_k) = lim_{k\to\infty} f(D'_k) = lim_{k\to\infty} (f(D_1) + \ldots + f(D_k))$$

and thus the desired result follows from the calculation (where we use Lemma 4.1 twice, and exploit that f is sum-preserving for each D_k)

$$\begin{aligned}
\sum f(D) &= \sum_{s\in\mathscr{F}} f(D)(s) \\
&= \sum_{s\in\mathscr{F}} lim_{k\to\infty} (f(D_1)(s) + \ldots + f(D_k)(s)) \\
&= lim_{k\to\infty} \sum_{s\in\mathscr{F}} (f(D_1)(s) + \ldots + f(D_k)(s)) \\
&= lim_{k\to\infty} (\sum_{s\in\mathscr{F}} f(D_1)(s) + \ldots + \sum_{s\in\mathscr{F}} f(D_k)(s)) \\
&= lim_{k\to\infty} (\sum_{s\in\mathscr{F}} D_1(s) + \ldots + \sum_{s\in\mathscr{F}} D_k(s)) \\
&= lim_{k\to\infty} \sum_{s\in\mathscr{F}} D'_k(s) \\
&= \sum_{s\in\mathscr{F}} lim_{k\to\infty} D'_k(s) = \sum_{s\in\mathscr{F}} D(s) = \sum D.
\end{aligned}$$

□

Lemma 4.12: Assume that $\mathsf{assign}_{x:=E}(D) = D'$ and that $x \notin R$. Then $D \overset{R}{=} D'$.

Proof: Given $s_0 \in \mathscr{S}(R)$, we must show that $D'(s_0) = D(s_0)$. But this follows since

$$\begin{aligned}
D'(s_0) &= \sum_{s'\in\mathscr{F} \mid s' \overset{R}{=} s_0} D'(s') = \sum_{s'\in\mathscr{F} \mid s' \overset{R}{=} s_0} \left(\sum_{s\in\mathscr{F} \mid s' = s[x\mapsto[\![E]\!]s]} D(s) \right) \\
&= \sum_{s,s'\in\mathscr{F} \mid s' \overset{R}{=} s_0,\ s' = s[x\mapsto[\![E]\!]s]} D(s) \\
(\text{as } x \notin R) &= \sum_{s,s'\in\mathscr{F} \mid s \overset{R}{=} s_0,\ s' = s[x\mapsto[\![E]\!]s]} D(s) = \sum_{s\in\mathscr{F} \mid s \overset{R}{=} s_0} D(s) = D(s_0)
\end{aligned}$$

□

Lemma 4.14 $\mathsf{assign}_{x:=E}$ is continuous.

Proof: Obviously, $\mathsf{assign}_{x:=E}$ is monotone. To show continuity, let $\{D_k \mid k\}$ be a chain. With $D'_k = \mathsf{assign}_{x:=E}(D_k)$, monotonicity implies that also $\{D'_k \mid k\}$ is a chain; let $D = lim_{k\to\infty} D_k$ and $D' = lim_{k\to\infty} D'_k$. Our goal is to prove that $D' = \mathsf{assign}_{x:=E}(D)$. But this follows since by Lemma 4.1 for each s' we have the

calculation

$$\begin{aligned} D'(s') &= lim_{k\to\infty} D'_k(s') = lim_{k\to\infty} \sum_{s\in\mathscr{F} \mid s'=s[x\mapsto[\![E]\!]s]} D_k(s) \\ &= \sum_{s\in\mathscr{F} \mid s'=s[x\mapsto[\![E]\!]s]} lim_{k\to\infty} D_k(s) = \sum_{s\in\mathscr{F} \mid s'=s[x\mapsto[\![E]\!]s]} D(s) \end{aligned}$$

□

Lemma 4.15: Assume that $\mathsf{rassign}_{x:=E}(D) = D'$ and that $x \notin R$. Then $D \overset{R}{=} D'$.

Proof: Given $s_0 \in \mathscr{S}(R)$, we must show that $D'(s_0) = D(s_0)$. But this follows since

$$\begin{aligned} D'(s_0) &= \sum_{s'\in\mathscr{F} \mid s'\overset{R}{=}s_0} D'(s') = \sum_{s'\in\mathscr{F} \mid s'\overset{R}{=}s_0} \left(\sum_{s\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x))D(s) \right) \\ (\text{as } x\notin R) &= \sum_{s,s'\in\mathscr{F} \mid s\overset{R}{=}s_0,\ s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x))D(s) = \sum_{s\in\mathscr{F} \mid s\overset{R}{=}s_0} \left(\sum_{s'\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x))D(s) \right) \\ &= \sum_{s\in\mathscr{F} \mid s\overset{R}{=}s_0} \left(D(s) \left(\sum_{s'\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x)) \right) \right) = \sum_{s\in\mathscr{F} \mid s\overset{R}{=}s_0} \left(D(s) \left(\sum_{z\in\mathbb{Z}} \psi(z) \right) \right) \\ &= \sum_{s\in\mathscr{F} \mid s\overset{R}{=}s_0} (D(s)\cdot 1) = D(s_0) \end{aligned}$$

□

Lemma 4.17 $\mathsf{rassign}_{x:=E}$ is continuous.

Proof: Obviously, $\mathsf{rassign}_{x:=E}$ is monotone. To show continuity, let $\{D_k \mid k\}$ be a chain. With $D'_k = \mathsf{rassign}_{x:=E}(D_k)$, monotonicity implies that also $\{D'_k \mid k\}$ is a chain; let $D = lim_{k\to\infty} D_k$ and $D' = lim_{k\to\infty} D'_k$. Our goal is to prove that $D' = \mathsf{rassign}_{x:=E}(D)$. But this follows since by Lemma 4.1 for each s' we have the calculation

$$\begin{aligned} D'(s') &= lim_{k\to\infty} D'_k(s') = lim_{k\to\infty} \sum_{s\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x))D_k(s) \\ &= \sum_{s\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} lim_{k\to\infty}\, \psi(s'(x))D_k(s) \\ &= \sum_{s\in\mathscr{F} \mid s'\overset{\mathscr{U}\setminus\{x\}}{=}s} \psi(s'(x))D(s) \end{aligned}$$

□

Lemma 4.21 The functional $\mathscr{H}_X$ is continuous on $\mathsf{PD} \to (\mathscr{D} \to_c \mathscr{D})$.

Proof: Consider a chain $\{g_k \mid k\}$, so as to prove that $\mathscr{H}_X(lim_{k\to\infty} g_k) = lim_{k\to\infty} \mathscr{H}_X(g_k)$. For all $(v,v') \in \mathsf{PD}$ and all D in $\mathscr{D}$, we must thus prove

$$\mathscr{H}_X(lim_{k\to\infty} g_k)(v,v')(D) = lim_{k\to\infty} \mathscr{H}_X(g_k)(v,v')(D)$$

and shall do so by induction in $\mathsf{LAP}(v,v')$, with a case analysis in Definition 4.19. We shall consider some sample cases:

- If $v \in X$ with $Lab(v)$ of the form $x := E$ then both sides evaluate to $\mathsf{assign}_{x:=E}(D)$.
- If $v' \neq v''$ where $v'' = 1PPD(v)$ then we have the calculation

$$\begin{array}{rcl} \mathscr{H}_X(lim_{k\to\infty} g_k)(v,v')(D) & = & \mathscr{H}_X(lim_{k\to\infty} g_k)(v'',v')(\mathscr{H}_X(lim_{k\to\infty} g_k)(v,v'')(D)) \\ & = & (lim_{k\to\infty} \mathscr{H}_X(g_k)(v'',v'))(lim_{k\to\infty} \mathscr{H}_X(g_k)(v,v'')(D)) \\ & = & lim_{k\to\infty} \mathscr{H}_X(g_k)(v'',v')(\mathscr{H}_X(g_k)(v,v'')(D)) \\ & = & lim_{k\to\infty} \mathscr{H}_X(g_k)(v,v')(D) \end{array}$$

  where the second equality follows from the induction hypothesis, and the third equality from continuity of $\mathscr{H}_X(g_k)(v'',v')$ (Lemma 4.20).
- If v is a branching node with condition B, *true*-successor v_1, and *false*-successor v_2, where $\mathsf{LAP}(v_1,v') \geq \mathsf{LAP}(v,v')$ and $\mathsf{LAP}(v_2,v') < \mathsf{LAP}(v,v')$ (other cases are similar), with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$ we have the calculation (where the second equality follows from the induction hypothesis):

$$\begin{array}{rcl} \mathscr{H}_X(lim_{k\to\infty} g_k)(v,v')(D) & = & lim_{k\to\infty} g_k(v_1,v')(D_1) + \mathscr{H}_X(lim_{k\to\infty} g_k)(v_2,v')(D_2) \\ & = & lim_{k\to\infty} (g_k(v_1,v')(D_1) + \mathscr{H}_X(g_k)(v_2,v')(D_2)) \\ & = & lim_{k\to\infty} \mathscr{H}_X(g_k)(v,v')(D) \end{array}$$

□

The following two lemmas are often convenient.

Lemma B.3 *Assume that* $\mathsf{assign}_{x:=E}(D) = D'$. *Assume that* R, R' *are such that* $x \in R'$, *and that* $R'' \cup fv(E) \subseteq R$ *where* $R'' = R' \setminus \{x\}$. *For* $s' \in \mathscr{S}(R')$ *we then have*

$$D'(s') = \sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s',\ s'(x) = [\![E]\!]s} D(s).$$

Proof: This follows from the calculation

$$\begin{array}{rcl} D'(s') & = & \displaystyle\sum_{s'_0 \in \mathscr{F} \mid s'_0 \overset{R'}{=} s'} D'(s'_0) \ = \ \sum_{s'_0 \in \mathscr{F} \mid s'_0 \overset{R'}{=} s'} \left(\sum_{s_0 \in \mathscr{F} \mid s'_0 = s_0[x \mapsto [\![E]\!]s_0]} D(s_0) \right) \\ & = & \displaystyle\sum_{s'_0, s_0 \in \mathscr{F} \mid s'_0 = s_0[x \mapsto [\![E]\!]s_0],\ s'_0 \overset{R''}{=} s',\ s'(x) = [\![E]\!]s_0} D(s_0) \ = \ \sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R''}{=} s',\ s'(x) = [\![E]\!]s_0} D(s_0) \\ & = & \displaystyle\sum_{s_0 \in \mathscr{F} \mid s_0|_R \overset{R''}{=} s',\ s'(x) = [\![E]\!](s_0|_R)} D(s_0) = \sum_{s_0 \in \mathscr{F}} \sum_{s \in \mathscr{S}(R) \mid s = s_0|_R,\ s \overset{R''}{=} s',\ s'(x) = [\![E]\!]s} D(s_0) \\ & = & \displaystyle\sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s',\ s'(x) = [\![E]\!]s} \left(\sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R}{=} s} D(s_0) \right) \ = \ \sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s',\ s'(x) = [\![E]\!]s} D(s) \end{array}$$

Lemma B.4 *Assume that* $\mathsf{rassign}_{x:=\psi}(D) = D'$. *Assume that* R, R' *are such that* $x \in R'$, *and that* $R'' \subseteq R$ *where* $R'' = R' \setminus \{x\}$. *For* $s' \in \mathscr{S}(R')$ *we then have*

$$D'(s') = \psi(s'(x)) \sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s'} D(s)$$

Proof: This follows from the calculation

$$\begin{aligned}
D'(s') &= \sum_{s'_0 \in \mathscr{F} \mid s'_0 \overset{R'}{=} s'} D'(s'_0) = \sum_{s'_0 \in \mathscr{F} \mid s'_0 \overset{R'}{=} s'} \left(\sum_{s_0 \in \mathscr{F} \mid s'_0 \overset{\mathscr{U} \setminus \{x\}}{=} s_0} \psi(s'_0(x)) D(s_0) \right) \\
&= \sum_{s_0, s'_0 \in \mathscr{F} \mid s_0 \overset{R''}{=} s',\ s'_0 = s_0[x \mapsto s'(x)]} \psi(s'(x)) D(s_0) \\
&= \sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R''}{=} s'} \psi(s'(x)) D(s_0) = \psi(s'(x)) \sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s'} \sum_{s_0 \in \mathscr{F} \mid s_0 \overset{R}{=} s} D(s_0) \\
&= \psi(s'(x)) \sum_{s \in \mathscr{S}(R) \mid s \overset{R''}{=} s'} D(s)
\end{aligned}$$

Next some results that prepare for the proof of Lemma 4.26.

Lemma B.5 *Let* $\{f_k \mid k\}$ *be a chain of additive functions. Then* $f = lim_{k \to \infty} f_k$ *is additive.*

Proof: For all distributions D_1 and D_2, we have

$$\begin{aligned}
f(D_1 + D_2) &= lim_{k \to \infty} f_k(D_1 + D_2) = lim_{k \to \infty} f_k(D_1) + f_k(D_2) \\
&= lim_{k \to \infty} f_k(D_1) + lim_{k \to \infty} f_k(D_2) = f(D_1) + f(D_2).
\end{aligned}$$

□

Lemma B.6 *Let* $\{f_k \mid k\}$ *be a chain of multiplicative functions. Then* $f = lim_{k \to \infty} f_k$ *is multiplicative.*

Proof: For all distributions D, and for all c with $c \geq 0$, we have

$$f(cD) = lim_{k \to \infty} f_k(cD) = lim_{k \to \infty} c f_k(D) = c\ lim_{k \to \infty} f_k(D) = c f(D).$$

□

Lemma B.7 *Let* $\{f_k \mid k\}$ *be a chain of non-increasing functions. Then* $f = lim_{k \to \infty} f_k$ *is a non-increasing function.*

Proof: For all distributions D, by assumption $\sum f_k(D) \leq \sum D$ for all k so by Lemma 4.1 we get:

$$\sum f(D) = \sum lim_{k \to \infty} f_k(D) = lim_{k \to \infty} \sum f_k(D) \leq lim_{k \to \infty} \sum D = \sum D.$$

□

Lemma B.8 *With* $\mathscr{H}_X$ *as defined in Def. 4.19, we have:*

- *if* ${h_0}^{(v,v')}$ *is additive for all* $(v,v') \in \mathsf{PD}$ *then* $\mathscr{H}_X(h_0)^{(v,v')}$ *is additive for all* $(v,v') \in \mathsf{PD}$*;*
- *if* ${h_0}^{(v,v')}$ *is multiplicative for all* $(v,v') \in \mathsf{PD}$ *then* $\mathscr{H}_X(h_0)^{(v,v')}$ *is multiplicative for all* $(v,v') \in \mathsf{PD}$*;*
- *if* ${h_0}^{(v,v')}$ *is non-increasing for all* $(v,v') \in \mathsf{PD}$ *then* $\mathscr{H}_X(h_0)^{(v,v')}$ *is non-increasing for all* $(v,v') \in \mathsf{PD}$.

Proof: An easy induction in $\mathsf{LAP}(v,v')$, using Lemmas 4.11, 4.13 and 4.16. □

Lemma 4.26 Let $h = \mathit{fix}(\mathscr{H}_X)$. Then for each $(v,v') \in \mathsf{PD}$, ${h}^{(v,v')}$ is additive, multiplicative and non-increasing (as is also ${\omega_k}^{(v,v')}$ for each $k \geq 0$).

Proof: The function 0 is obviously additive, multiplicative and non-increasing, so by Lemma B.8 we infer that for each k, letting $h_k = \mathscr{H}_X^k(h_0)$, and for each $(v,v') \in \mathsf{PD}$, ${h_k}^{(v,v')}$ is additive, multiplicative and non-increasing. The claim now follows from Lemmas B.5, B.6, and B.7. □

Lemma 4.29 Given a pCFG, a slice set Q, and $(v,v') \in \mathsf{PD}$ such that v stays outside Q until v'. We then have $\mathscr{H}_Q(h)^{(v,v')}(D) = D$ for all $D \in \mathscr{D}$ and all modification functions h.

Proof: We do induction in $\mathsf{LAP}(v,v')$. The claim is obvious if $v = v'$. If with $v'' = 1PPD(v)$ we have $v' \neq v''$, we can (by Lemmas B.1 and 3.5) apply the induction hypothesis to (v,v'') and (v'',v'), to get the desired $\mathscr{H}_Q(h)^{(v,v')}(D) = \mathscr{H}_Q(h)^{(v'',v')}(\mathscr{H}_Q(h)^{(v,v'')}(D)) = \mathscr{H}_Q(h)^{(v'',v')}(D) = D$.

We are left with the case when $v' = 1PPD(v)$, and since v stays outside Q until v' we see that $v \notin Q$ and thus clause 3a in Definition 4.19 gives the desired $\mathscr{H}_Q(h)^{(v,v')}(D) = D$. □

We now prepare for the proof of Lemma 4.30.

Definition B.9 *A function* $f : \mathscr{D} \to \mathscr{D}$ *is non-increasing wrt. R, a set of variables, if* $f(D)(s) \leq D(s)$ *holds for all* $D \in \mathscr{D}$ *and* $s \in \mathscr{S}(R)$.

Observe that with $R = \emptyset$, this reduces to the previous notion of non-increasing.

Lemma B.10 *With* $(v,v') \in \mathsf{PD}$*, assume that Q is closed under data dependence and that v stays outside Q until* v'*. Let* $R = rv_Q(v) = rv_Q(v')$ *(well-defined by Lemma 3.16), and let* $\omega_k = \mathscr{H}_\mathscr{V}^k(0)$ *for each* $k \geq 0$*. Then* ${\omega_k}^{(v,v')}$ *is non-increasing wrt. R for all* $k \geq 0$*.*

Proof: Induction in k, followed by induction in $\mathsf{LAP}(v,v')$. The case where $k = 0$ is trivial as then $\omega_k = 0$. Now let $k > 0$, in which case $\omega_k = \mathscr{H}_\mathscr{V}(\omega_{k-1})$. If $v' = v$, then ${\omega_k}^{(v,v')}(D) = D$ and the claim is trivial. Otherwise, let $v_0 = 1PPD(v)$; if $v_0 \neq v'$ then Lemma B.1 tells us that we can apply the induction hypothesis twice to infer that ${\omega_k}^{(v,v_0)}$ and ${\omega_k}^{(v_0,v')}$ are both non-increasing wrt. R which shows that ${\omega_k}^{(v,v')}$ is non-increasing wrt. R since for $s \in \mathscr{S}(R)$ we have

$$ {\omega_k}^{(v,v')}(D)(s) = {\omega_k}^{(v_0,v')}({\omega_k}^{(v,v_0)}(D))(s) \leq {\omega_k}^{(v,v_0)}(D)(s) \leq D(s). $$

We are left with case where $v' = 1PPD(v)$. If $Lab(v)$ is of the form $\mathbf{Observe}(B)$ the claim is trivial.

If $Lab(v)$ is of the form $x := E$ or of the form $x := \mathbf{Random}(\psi)$ we first infer that $x \notin R$ because otherwise, as Q is closed under data dependence, we would have $v \in Q$ which contradicts that v stays outside Q until v'. But then the claim follows from Lemmas 4.12 and 4.15.

The last case is if v is a branching node with condition B, with v_1 the *true*-successor of v and v_2 the *false*-successor of v. For each $D \in \mathscr{D}$, let $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$ and let (for $i \in \{1,2\}$) D_i' be defined as ${\omega_k}^{(v_i,v')}(D_i)$ if $\mathsf{LAP}(v_i,v') < \mathsf{LAP}(v,v')$, but otherwise as ${\omega_{k-1}}^{(v_i,v')}(D_i)$. For each $i \in \{1,2\}$ we can apply, as v_i stays outside Q until v' (Lemma B.2), either the outer induction hypothesis, or the inner induction hypothesis, to infer that for all $s \in \mathscr{S}(R)$, $D_i'(s) \leq D_i(s)$. For each $s \in \mathscr{S}(R)$ we thus infer the desired

$$ {\omega_k}^{(v,v')}(s) = D_1'(s) + D_2'(s) \leq D_1(s) + D_2(s) = D(s). $$

□

Lemma 4.30 With $(v,v') \in \mathsf{PD}$, assume that Q is closed under data dependence and that v stays outside Q until v' (by Lemma 3.16 it thus makes sense to define $R = rv_Q(v) = rv_Q(v')$).

For all distributions D, if $\sum \omega^{(v,v')}(D) = \sum D$ then $\omega^{(v,v')}(D) \stackrel{R}{=} D$.

Proof: Given $D \in \mathscr{D}$ with $\sum \omega^{(v,v')}(D) = \sum D$, for all $s \in \mathscr{S}(R)$, Lemma B.10 yields ${\omega_k}^{(v,v')}(D)(s) \leq D(s)$ for all $k \geq 0$, and as $\omega = lim_{k\to\infty}\, \omega_k$ this implies —since ω is the *least* upper bound— $\omega^{(v,v')}(D)(s) \leq D(s)$. We thus get (using Lemma 4.4)

$$\sum \omega^{(v,v')}(D) = \sum_{s\in\mathscr{S}(R)} \omega^{(v,v')}(D)(s) \leq \sum_{s\in\mathscr{S}(R)} D(s) = \sum D.$$

If $\sum \omega^{(v,v')}(D) = \sum D$ we infer from the above that

$$\sum_{s\in\mathscr{S}(R)} \omega^{(v,v')}(D)(s) = \sum_{s\in\mathscr{S}(R)} D(s)$$

which since $\omega^{(v,v')}(D)(s) \leq D(s)$ for all $s \in \mathscr{S}(R)$ is possible only if $\omega^{(v,v')}(D)(s) = D(s)$ for all $s \in \mathscr{S}(R)$, that is $\omega^{(v,v')}(D) \stackrel{R}{=} D$. □

B.3 Proofs for Section 5

Lemma 5.3 If Q_1 and Q_2 are weak slice sets, also $Q_1 \cup Q_2$ is a weak slice set. *Proof:* Let $Q = Q_1 \cup Q_2$. To see that Q is closed under data dependence, assume that $v \stackrel{dd}{\to} v'$ with $v' \in Q$; wlog. we can assume $v' \in Q_1$ which since Q_1 is closed under data dependence implies $v \in Q_1$ and thus $v \in Q$.

We shall now look at a node v, and argue that $next_Q(v)$ exists. If $next_{Q_1}(v) = v$ or $next_{Q_2}(v) = v$, then $v \in Q \cup \{\texttt{End}\}$ and thus $next_Q(v) = v$. Otherwise, let $v_1 = next_{Q_1}(v)$ and $v_2 = next_{Q_2}(v)$; both v_1 and v_2 are proper postdominators of v so by Lemma 3.3 we can wlog. assume that v_1 occurs before v_2 in all paths from v to `End` (or that $v_1 = v_2$).

We shall now show that $v_1 = next_Q(v)$, where we first observe that $v_1 \in Q \cup \{\texttt{End}\}$. Now consider a path π from v to $Q \cup \{\texttt{End}\}$. We must show that π contains v_1, which is obvious if the path π is to $Q_1 \cup \{\texttt{End}\}$. Otherwise, when π is to Q_2, we infer that π contains v_2 (which yields the claim if $v_1 = v_2$). Since all nodes have a path to `End`, π is a prefix of a path π' from v to `End`; as v_1 occurs before v_2 in all paths from v to `End`, we see that v_1 occurs before v_2 in π'. We infer that v_1 occurs also in π, as desired. □

Lemma 5.7 Assume Q' is a node set which contains all **Observe** nodes, and that for each cycle-inducing node v_0, either $v_0 \in Q'$ or $\omega^{(v_0,1PPD(v_0))}$ is sum-preserving. If v stays outside Q' until v' then $\sum \omega^{(v,v')}(D) = \sum D$ for all D.

Proof: We do induction in $\mathsf{LAP}(v,v')$. The claim is obvious if $v' = v$, so we can assume that v' is a proper postdominator of v.

With $v'' = 1PPD(v)$, let us first assume that $v' \neq v''$. By Lemma 3.5 we have $\mathsf{LAP}(v,v'') < \mathsf{LAP}(v,v')$ and $\mathsf{LAP}(v'',v') < \mathsf{LAP}(v,v')$, and by Lemma B.1 we see that v stays outside Q' until v'' and that v'' stays outside Q' until v'. We can thus apply the induction hypothesis on $\omega^{(v,v'')}$ (the third equality) and on $\omega^{(v'',v')}$ (the second equality) to infer that for all D we have

$$\sum \omega^{(v,v')}(D) = \sum \omega^{(v'',v')}(\omega^{(v,v'')}(D)) = \sum \omega^{(v,v'')}(D) = \sum D.$$

Thus we can now assume that $v' = 1PPD(v)$. If v is labeled **Skip** the claim is trivial; if v is labeled $x := E$ (or $x := \mathbf{Random}(\psi)$) then the claim follows from Lemma 4.13 (or Lemma 4.16). Note that $v \notin Q'$ (as v stays outside Q' until v'), so our assumptions entail that v cannot be an **Observe** node, and that if v is cycle-inducing then $\omega^{(v,v')}$ is sum-preserving and thus the claim.

We are thus left with the case that v is a branching node which is not cycle-inducing. With v_1 the *true*-successor and v_2 the *false*-successor of v, we thus have $\mathsf{LAP}(v_1,v') < \mathsf{LAP}(v,v')$ and $\mathsf{LAP}(v_2,v') < \mathsf{LAP}(v,v')$, and by Lemma B.2 also that v_1 and v_2 both stay outside Q' until v'. Hence we can apply the induction hypothesis to $\omega^{(v_1,v')}$ and $\omega^{(v_2,v')}$, to get the desired result:

$$\begin{aligned}\sum \omega^{(v,v')}(D) &= \sum(\omega^{(v_1,v')}(\mathsf{select}_B(D)) + \omega^{(v_2,v')}(\mathsf{select}_{\neg B}(D)))\\ &= \sum \omega^{(v_1,v')}(\mathsf{select}_B(D)) + \sum \omega^{(v_2,v')}(\mathsf{select}_{\neg B}(D))\\ &= \sum \mathsf{select}_B(D) + \sum \mathsf{select}_{\neg B}(D) \;= \sum D.\end{aligned}$$

□

B.4 Proofs for Section 6

Lemma B.11 *For each $(v,v') \in \mathsf{PD}$, and each $k \geq 0$, ${\gamma_k}^{(v,v')}$ is additive, multiplicative and non-increasing,*
Proof: We know from Lemma 4.26 that ω is additive, multiplicative and non-increasing (and so is the function 0); the result thus follows from Lemma B.8. □

Similarly, we have (with Φ_k defined in Def. 6.11):
Lemma B.12 *For each $(v,v') \in \mathsf{PD}$, and each $k \geq 0$, ${\Phi_k}^{(v,v')}$ is additive, multiplicative and non-increasing,*
Lemma 6.2 Assume that v stays outside $Q \cup Q_0$ until v'. Then ${\gamma_k}^{(v,v')} = \omega^{(v,v')}$ holds for all $k \geq 0$.

Proof: We do induction in k, where the base case $k = 0$ follows from the definition of γ_0.

For the inductive case, where $\gamma_k = \mathscr{H}_{\mathscr{V}}(\gamma_{k-1})$ with $k > 0$, we do induction in $\mathsf{LAP}(v,v')$, with a case analysis on the definition of $\mathscr{H}_{\mathscr{V}}$:

- If $v' = v$ then ${\gamma_k}^{(v,v')}(D) = D = \omega^{(v,v')}(D)$;
- If with $v'' = 1PPD(v)$ we have $v' \neq v''$ then we know from Lemma B.1 that v stays outside $Q \cup Q_0$ until v'', and that v'' stays outside $Q \cup Q_0$ until v'; by Lemma 3.5 we know that $\mathsf{LAP}(v,v'') < \mathsf{LAP}(v,v')$ and $\mathsf{LAP}(v'',v') < \mathsf{LAP}(v,v')$. Hence we can apply the inner induction hypothesis to infer that ${\gamma_k}^{(v,v'')} = \omega^{(v,v'')}$ and ${\gamma_k}^{(v'',v')} = \omega^{(v'',v')}$. But then we get the desired

$$ {\gamma_k}^{(v,v')} = {\gamma_k}^{(v,v'')} \;;\; {\gamma_k}^{(v'',v')} = \omega^{(v,v'')} \;;\; \omega^{(v'',v')} = \omega^{(v,v')}. $$

- Otherwise, when $v' = 1PPD(v)$, the claim is trivial except when v is a branching node. So consider such a v, and let B be its condition, v_1 its *true*-successor, and v_2 its *false*-successor. Our goal is to prove, for a given D, that ${\gamma_k}^{(v,v')}(D) = \omega^{(v,v')}(D)$ which amounts to

$$ \mathscr{H}_{\mathscr{V}}(\gamma_{k-1})^{(v,v')}(D) = \mathscr{H}_{\mathscr{V}}(\omega)^{(v,v')}(D). $$

  With $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, examining the definition of $\mathscr{H}_{\mathscr{V}}$ shows that it suffices if for $i \in \{1,2\}$ we can prove:
  – if $\mathsf{LAP}(v_i,v') < \mathsf{LAP}(v,v')$ then

$$ {\gamma_k}^{(v_i,v')}(D_i) = \omega^{(v_i,v')}(D_i) $$

  which follows by the inner induction hypothesis;

– if $\mathsf{LAP}(v_i,v') \geq \mathsf{LAP}(v,v')$ then

$$\gamma_{k-1}{}^{(v_i,v')}(D_i) = \omega^{(v_i,v')}(D_i)$$

which follows by the outer induction hypothesis.

□

To prepare for the proof of Lemma 6.5, we state a result about branching nodes:

Lemma B.13 *Assume that v is a branching node, with B its condition, and v_1 its true-successor and v_2 its false-successor. Let $D \in \mathscr{D}$ with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$. Assume that R, R_0, R_1, R_2 are such that $fv(B) \cup R_1 \cup R_2 \subseteq R$ and $R \cap R_0 = \emptyset$. Finally assume that R and R_0 are independent in D. Then for $i = 1,2$ we have*

1. *R_i and R_0 are independent in D_i.*
2. *$\forall s_0 \in \mathscr{S}(R_0) : D(s_0)\sum D_i = D_i(s_0)\sum D$, and*

Proof: We shall consider only the case $i = 1$ (as the case $i = 2$ is symmetric). For (2), we have the calculation (where the 3rd equality follows from R and R_0 being independent in D)

$$\begin{aligned}
D(s_0)\sum D_1 &= D(s_0) \sum_{s\in\mathscr{S}(R) \mid [\![B]\!]s} D(s) = \sum_{s\in\mathscr{S}(R) \mid [\![B]\!]s} D(s_0)D(s) \\
&= \sum_{s\in\mathscr{S}(R) \mid [\![B]\!]s} \big(D(s_0 \oplus s)\sum D\big) = \left(\sum_{s\in\mathscr{S}(R) \mid [\![B]\!]s} D(s_0 \oplus s)\right)\sum D \\
&= \left(\sum_{s'\in\mathscr{S}(R\cup R_0) \mid s' \overset{R_0}{=} s_0,\ [\![B]\!]s'} D(s')\right)\sum D \\
&= \left(\sum_{s\in\mathscr{F} \mid s \overset{R_0}{=} s_0,\ [\![B]\!]s} D(s)\right)\sum D = \left(\sum_{s\in\mathscr{F} \mid s \overset{R_0}{=} s_0} D_1(s)\right)\sum D = D_1(s_0)\sum D
\end{aligned}$$

For (1) we have with $s_1 \in \mathscr{S}(R_1)$ and $s_0 \in \mathscr{S}(R_0)$ the calculation (which uses (2) and the fact that if $D = 0$ then the claim is trivial)

$$\begin{aligned}
D_1(s_1 \oplus s_0)\sum D_1 &= \left(\sum_{s\in\mathscr{S}(R) \mid s \overset{R_1}{=} s_1,\ [\![B]\!]s} D(s \oplus s_0)\right)\sum D_1 \\
&= \left(\sum_{s\in\mathscr{S}(R) \mid s \overset{R_1}{=} s_1,\ [\![B]\!]s} \frac{D(s)D(s_0)}{\sum D}\right)\sum D_1 \\
&= \left(\sum_{s\in\mathscr{S}(R) \mid s \overset{R_1}{=} s_1,\ [\![B]\!]s} D(s)\right)\frac{D(s_0)\sum D_1}{\sum D} \\
&= D_1(s_1)D_1(s_0).
\end{aligned}$$

□

To facilitate the proof of Lemma 6.5, we introduce some notation:

Definition B.14 *We say that $h \in \mathsf{PD} \to \mathscr{D} \to_c \mathscr{D}$* **preserves probabilistic independence** *iff for all slicing pairs (Q, Q_0), the following holds for all $(v, v') \in \mathsf{PD}$, with $R = rv_Q(v)$, $R' = rv_Q(v')$, $R_0 = rv_{Q_0}(v)$, and $R'_0 = rv_{Q_0}(v')$: for all D such that R and R_0 are independent in D, with $D' = h^{(v,v')}(D)$ it is the case that*

1. *R' and R'_0 are independent in D'*
2. *if v stays outside Q until v' (and thus $R' = R$) then for all $s \in \mathscr{S}(R)$ we have*
$$D(s)\sum D' = D'(s)\sum D$$
3. *if v stays outside Q_0 until v' (and thus $R'_0 = R_0$) then for all $s_0 \in \mathscr{S}(R_0)$ we have*
$$D(s_0)\sum D' = D'(s_0)\sum D.$$

Lemma B.15 *For each $k \geq 0$, γ_k preserves probabilistic independence.*

Proof: We shall proceed by induction in k. We shall first consider the base case $k = 0$. For a given $(v, v') \in \mathsf{PD}$, the claims are trivial if $\gamma_0{}^{(v,v')} = 0$, so assume that v stays outside $Q \cup Q_0$ until v' (implying $R' = R$ and $R'_0 = R_0$) and thus $\gamma_0{}^{(v,v')} = \omega^{(v,v')}$. We can apply Lemma 5.8 (and part 1 of Lemma 3.14) to infer that for all D, with $D' = \gamma_0{}^{(v,v')}(D)$ we have $D' \overset{R \cup R_0}{=} D$, and by Lemma 4.6 thus also $D' \overset{R}{=} D$ and $D' \overset{R_0}{=} D$ and $D' \overset{\emptyset}{=} D$. That is, for $s \in \mathscr{S}(R)$ and $s_0 \in \mathscr{S}(R_0)$ we have $D'(s \oplus s_0) = D(s \oplus s_0)$ and $D'(s) = D(s)$ and $D'(s_0) = D(s_0)$, and also $D'(\emptyset) = D(\emptyset)$ which amounts to $\sum D' = \sum D$. This clearly implies claims 2 and 3 in Definition B.14, and also claim 1 since for $s \in \mathscr{S}(R)$ and $s_0 \in \mathscr{S}(R_0)$ we have, by our assumption that R and R_0 are independent in D:

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

We shall next consider the case $k > 0$, where we assume that γ_{k-1} preserves probabilistic independence and with $\gamma_k = \mathscr{H}_V(\gamma_{k-1})$ we must then prove that γ_k preserves probabilistic independence, that is: given $(v, v') \in \mathsf{PD}$ with $R = rv_Q(v)$, $R' = rv_Q(v')$, $R_0 = rv_{Q_0}(v)$, and $R'_0 = rv_{Q_0}(v')$, and given $D \in \mathscr{D}$ such that R and R_0 are independent in D, with $D' = \gamma_k{}^{(v,v')}(D)$ we must show that:

1. R' and R'_0 are independent in D'
2. if v stays outside Q until v' (and thus $R' = R$) then for all $s \in \mathscr{S}(R)$ we have
$$D(s)\sum D' = D'(s)\sum D$$
3. if v stays outside Q_0 until v' (and thus $R'_0 = R_0$) then for all $s_0 \in \mathscr{S}(R_0)$ we have
$$D(s_0)\sum D' = D'(s_0)\sum D.$$

We shall establish the required claims by induction in $\mathsf{LAP}(v, v')$. First observe that if $D' = 0$ the claims are trivial. We can thus assume that $\sum D' > 0$, which by Lemma B.11 entails that $\sum D > 0$.

If $v' = v$, then $D' = D$ and $R' = R$ and $R'_0 = R_0$ and again the claims are trivial.

Otherwise, let $v'' = 1PPD(v)$, and first assume that $v' \neq v''$ in which case the situation is that there exists D'' such that $\gamma_k{}^{(v,v'')}(D) = D''$ and $\gamma_k{}^{(v'',v')}(D'') = D'$; by Lemma B.11 we can assume that $D'' \neq 0$ (as otherwise $D' = 0$ which we have already considered). By Lemma 3.5 we have $\mathsf{LAP}(v, v'') < \mathsf{LAP}(v, v')$ and $\mathsf{LAP}(v'', v') < \mathsf{LAP}(v, v')$, so we can apply the induction hypothesis on $\gamma_k{}^{(v,v'')}$ and on $\gamma_k{}^{(v'',v')}$. With $R'' = rv_Q(v'')$ and $R''_0 = rv_{Q_0}(v'')$, the induction hypothesis now first gives us that R'' and R''_0 are independent in D'', and next that R' and R'_0 are independent in D'.

Concerning claim 2 (claim 3 is symmetric), assume that v stays outside Q until v'; by Lemma B.1 we see that v stays outside Q until v'' and v'' stays outside Q until v'. Inductively, we can thus assume that for $s \in \mathscr{S}(R)$ we have

$$D(s)\sum D'' = D''(s)\sum D \text{ and } D''(s)\sum D' = D'(s)\sum D''.$$

which since $\sum D'' > 0$ gives us the desired

$$\begin{aligned} D(s)\sum D' &= \frac{D''(s)\sum D}{\sum D''}\sum D' = D''(s)\sum D'\frac{\sum D}{\sum D''} \\ &= D'(s)\sum D''\frac{\sum D}{\sum D''} = D'(s)\sum D. \end{aligned}$$

We are left with the case $v' = 1PPD(v)$, and split into several cases, depending on $Lab(v)$, where the case for **Skip** is trivial.

Case 1: v is an observe node. With B the condition, for s' with $fv(B) \subseteq dom(s')$ we thus have $D'(s') = D(s')$ if $[\![B]\!]s'$, and $D'(s') = 0$ otherwise. As (Q, Q_0) is a slicing pair, either $v \in Q$ or $v \in Q_0$. Let us assume that $v \in Q$; the other case is symmetric. Thus $fv(B) \subseteq R$, and also $R' \subseteq R$ as $Def(v) = \emptyset$; as v stays outside Q_0 until v', by Lemma 3.16 we also have $R'_0 = R_0$.

The following calculation, where the 3rd equality is due to the assumption that R and R_0 are independent in D, shows that for $s_0 \in \mathscr{S}(R_0)$ we have $D'(s_0)\sum D = D(s_0)\sum D'$:

$$\begin{aligned} D'(s_0)\sum D &= \left(\sum_{s\in\mathscr{S}(R)} D'(s \oplus s_0)\right)\sum D = \left(\sum_{s\in\mathscr{S}(R)\ |\ [\![B]\!]s} D(s \oplus s_0)\right)\sum D \\ &= \sum_{s\in\mathscr{S}(R)\ |\ [\![B]\!]s} (D(s)D(s_0)) = D(s_0)\left(\sum_{s\in\mathscr{S}(R)\ |\ [\![B]\!]s} D(s)\right) \\ &= D(s_0)\left(\sum_{s\in\mathscr{S}(R)} D'(s)\right) = D(s_0)\sum D'. \end{aligned}$$

This yields claim 3 (while claim 2 vacuously holds) and also gives the last equality in the following derivation that establishes (again using the assumption that R and R_0 are independent in D) claim 1 by considering $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$:

$$\begin{aligned} D'(s' \oplus s_0)\sum D' &= \left(\sum_{s\in\mathscr{S}(R)\ |\ s\overset{R'}{=}s'} D'(s \oplus s_0)\right)\sum D' \\ &= \left(\left(\sum_{s\in\mathscr{S}(R)\ |\ s\overset{R'}{=}s',\ [\![B]\!]s} D(s \oplus s_0)\right)\sum D\right)\frac{\sum D'}{\sum D} \\ &= \left(\sum_{s\in\mathscr{S}(R)\ |\ s\overset{R'}{=}s',\ [\![B]\!]s} D(s)D(s_0)\right)\frac{\sum D'}{\sum D} \\ &= \left(\sum_{s\in\mathscr{S}(R)\ |\ s\overset{R'}{=}s'} D'(s)\right)\frac{D(s_0)\sum D'}{\sum D} \\ &= D'(s')D'(s_0). \end{aligned}$$

Case 2: v has exactly one successor and is not an Observe node. Then v is labeled with an assignment or with a random assignment; in both cases, the transfer function is sum-preserving (Lemmas 4.13 and 4.16) so we get $\sum D' = \sum D$. We further infer by Lemma 4.30 that if $v \notin Q$ then $R' = R$ and $D' \stackrel{R}{=} D$ (as then v stays outside Q until v'); similarly, if $v \notin Q_0$ then $R'_0 = R_0$ and $D'_0 \stackrel{R_0}{=} D_0$.

The above observations obviously establish the claims 2 and 3. We shall now address claim 1, that is show that $D'(s \oplus s_0)\sum D' = D'(s)D'(s_0)$ for $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$. To do so, we shall do a case analysis on whether $v \in Q \cup Q_0$.

First consider the case where $v \notin Q \cup Q_0$. Then (by Lemma 4.30) we get $R' = R$ and $R'_0 = R_0$ and $D' \stackrel{R\cup R_0}{=} D$. This establishes claim 1 since for $s \in \mathscr{S}(R)$ and $s_0 \in \mathscr{S}(R_0)$ we have (using the assumption that R and R_0 are independent in D) $D'(s \oplus s_0)\sum D' = D(s \oplus s_0)\sum D = D(s)D(s_0) = D'(s)D'(s_0)$.

Next consider the case where $v \in Q \cup Q_0$. Without loss of generality, we may assume that $v \in Q$. Thus $v \notin Q_0$ so $R'_0 = R_0$ and $R_0 \cap Def(v) = \emptyset$ and $D'(s_0) = D(s_0)$ for all $s_0 \in \mathscr{S}(R_0)$. We split into two cases, depending on whether $R' \cap Def(v)$ is empty or not.

First assume that $R' \cap Def(v) = \emptyset$. Then $R' \subseteq R$, and by Lemmas 4.12 and 4.15 (as $(R' \cup R_0) \cap Def(v) = \emptyset$) we get $D \stackrel{R'\cup R_0}{=} D'$. For $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$ we thus have $D'(s' \oplus s_o) = D(s' \oplus s_0)$ and $D'(s') = D(s')$ and $D'(s_0) = D(s_0)$ and $\sum D' = \sum D$, which (using the assumption that R and R_0 are independent in D) gives us the desired result

$$\begin{aligned} D'(s' \oplus s_0)\sum D' &= D(s' \oplus s_0)\sum D \\ &= \left(\sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s'} D(s \oplus s_0)\right)\sum D \\ &= \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s'} D(s)D(s_0) = D(s')D(s_0) = D'(s')D'(s_0). \end{aligned}$$

Next assume that $R' \cap Def(v) \neq \emptyset$. We now (finally) need to do a case analysis on the kind of assignment.

If $Lab(v) = x := E$, we have (by Lemma 3.12) $R = (R' \setminus \{x\}) \cup fv(E)$ and from our case assumptions also $x \in R'$ and $x \notin R_0$. Let us now consider $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$; the claim follows from the below calculation where the third equality uses the assumption that R and R_0 are independent in D, and the first and last equality both uses Lemma B.3:

$$\begin{aligned} D'(s' \oplus s_0)\sum D' &= \left(\sum_{s_1 \in \mathscr{S}(R\cup R_0) \mid s_1 \stackrel{R'\setminus\{x\}\cup R_0}{=} s' \oplus s_0,\ (s'\oplus s_0)(x) = [\![E]\!]s_1} D(s_1)\right)\sum D \\ &= \left(\sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'\setminus\{x\}}{=} s',\ s'(x) = [\![E]\!]s} D(s \oplus s_0)\right)\sum D \\ &= \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'\setminus\{x\}}{=} s',\ s'(x) = [\![E]\!]s} D(s)D(s_0) \\ &= D'(s')D'(s_0). \end{aligned}$$

Finally, if $Lab(v) = x := \mathbf{Random}(\psi)$, we have $R = (R' \setminus \{x\})$ and from our case assumptions also $x \in R'$ and $x \notin R_0$. Let us now consider $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$: the claim follows from the below calculation

where the third equality uses the assumption that R and R_0 are independent in D, and the first and last equality both uses Lemma B.4:

$$\begin{aligned}
D'(s' \oplus s_0) \sum D' &= \psi((s' \oplus s_0)(x)) \left(\sum_{s_1 \in \mathscr{S}(R \cup R_0) \mid s_1 \overset{R' \setminus \{x\} \cup R_0}{=} s' \oplus s_0} D(s_1) \right) \sum D \\
&= \psi(s'(x)) \left(\sum_{s \in \mathscr{S}(R) \mid s \overset{R' \setminus \{x\}}{=} s'} D(s \oplus s_0) \right) \sum D \\
&= \psi(s'(x)) \left(\sum_{s \in \mathscr{S}(R) \mid s \overset{R' \setminus \{x\}}{=} s'} D(s) D(s_0) \right) \\
&= D'(s') D'(s_0).
\end{aligned}$$

Case 3: v is a branching node. First assume that $v \notin Q \cup Q_0$. Here $Q \cup Q_0$ is a weak slice set (Lemma 5.3), so from Lemma 5.4 we see that v stays outside $Q \cup Q_0$ until v'; thus $R' = R$ and $R'_0 = R_0$ (by Lemma 3.16). By Lemma 6.2 we see that $D' = \gamma_k{}^{(v,v')}(D) = \omega^{(v,v')}(D)$, and Lemma 5.8 thus tells us that $D' \overset{R \cup R_0}{=} D$. In particular for all $s \in \mathscr{S}(R)$ and $s_0 \in \mathscr{S}(R_0)$ we have $D'(s \oplus s_0) = D(s \oplus s_0)$, $D'(s) = D(s)$, $D'(s_0) = D(s_0)$ and $\sum D' = \sum D$. But this clearly entails all the 3 claims.

In the following, we can thus assume that $v \in Q \cup Q_0$, and shall only look at the case $v \in Q$ as the case $v \in Q_0$ is symmetric. Claim 2 thus holds vacuously; we shall embark on the other two claims. As $v \in Q$ we have (by Lemma 3.13) $fv(B) \subseteq R = rv_Q(v)$, and as $v \notin Q_0$ we see (by Lemma 5.4) that v stays outside Q_0 until v' so that (by Lemma 3.16) $R'_0 = R_0$. With v_1 the *true*-successor of v and v_2 the *false*-successor of v, and with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, the situation is that $D' = D'_1 + D'_2$ where for each $i \in \{1,2\}$, D'_i is computed as

- if $\mathsf{LAP}(v_i, v') < \mathsf{LAP}(v, v')$ then $D'_i = \gamma_k{}^{(v_i,v')}(D_i)$;
- if $\mathsf{LAP}(v_i, v') \geq \mathsf{LAP}(v, v')$ then $D'_i = \gamma_{k-1}{}^{(v_i,v')}(D_i)$.

Let $R_1 = rv_Q(v_1)$ and $R_2 = rv_Q(v_2)$; thus (by Lemma 3.13) $R_1 \subseteq R$ and $R_2 \subseteq R$.

By Lemma B.13, we see that

$$\forall i \in \{1,2\} : R_i \text{ and } R_0 \text{ are independent in } D_i \tag{6}$$

$$\forall i \in \{1,2\} : D(s_0) \sum D_i = D_i(s_0) \sum D \text{ for all } s_0 \in \mathscr{S}(R_0). \tag{7}$$

Given (6), and the fact that v_i stays outside Q_0 until v', we can infer that

$$\forall i \in \{1,2\} : R' \text{ and } R_0 \text{ are independent in } D'_i \tag{8}$$

$$\forall i \in \{1,2\} : D_i(s_0) \sum D'_i = D'_i(s_0) \sum D_i \text{ for all } s_0 \in \mathscr{S}(R_0) \tag{9}$$

since when $\mathsf{LAP}(v_i, v') < \mathsf{LAP}(v, v')$ this follows from the (inner) induction hypothesis, and otherwise it follows from the assumption (the outer induction) about γ_{k-1}. We also have

$$D(s_0) \sum D' = D'(s_0) \sum D \text{ for all } s_0 \in \mathscr{S}(R_0) \tag{10}$$

since for $s_0 \in \mathscr{S}(R_0)$ we have

$$\begin{aligned} D'(s_0)\sum D &= (D'_1(s_0)+D'_2(s_0))\sum D \\ \text{by (9)} &= \left(D_1(s_0)\frac{\sum D'_1}{\sum D_1}+D_2(s_0)\frac{\sum D'_2}{\sum D_2}\right)\sum D \\ \text{by (7)} &= D(s_0)\sum D'_1+D(s_0)\sum D'_2 = D(s_0)\sum D' \end{aligned}$$

where we have assumed that $D_1 \neq 0$ and $D_2 \neq 0$; if say $D_1 = 0$ then $D'_1 = 0$ (as each γ_k is non-increasing by Lemma B.11) and $D = D_2$ and $D' = D'_2$ in which case the claim follows directly from (9).

From (10) we get claim 3, and are thus left with showing claim 1 which is that R' and R_0 are independent in D'. If $D'_1 = 0$ then $D' = D'_2$ and it follows from (8); similarly if $D'_2 = 0$. Otherwise, in which case also $\sum D_1 > 0$ and $\sum D_2 > 0$, for $s' \in \mathscr{S}(R')$ and $s_0 \in \mathscr{S}(R_0)$ we have

$$\begin{aligned} D'(s' \oplus s_0)\sum D' &= (D'_1(s' \oplus s_0)+D'_2(s' \oplus s_0))\sum D' \\ \text{by (8)} &= D'_1(s')D'_1(s_0)\frac{\sum D'}{\sum D'_1}+D'_2(s')D'_2(s_0)\frac{\sum D'}{\sum D'_2} \\ \text{by (9)} &= D'_1(s')D_1(s_0)\frac{\sum D'}{\sum D_1}+D'_2(s')D_2(s_0)\frac{\sum D'}{\sum D_2} \\ \text{by (7)} &= D'_1(s')D(s_0)\frac{\sum D'}{\sum D}+D'_2(s')D(s_0)\frac{\sum D'}{\sum D} \\ &= D'(s')D(s_0)\frac{\sum D'}{\sum D} \\ \text{by (10)} &= D'(s')D'(s_0) \end{aligned}$$

□

Lemma 6.5 [rephrased using Definition B.14]: ω preserves probabilistic independence.

Proof: Let a slicing pair (Q, Q_0) be given, and let $\{\gamma_k \mid k\}$ be defined as in Definition 6.1. By Lemma B.15, each element in the chain $\{\gamma_k \mid k\}$ preserves probabilistic independence; by Proposition 6.3, it is sufficient to prove that also $lim_{k\to\infty}\gamma_k$ preserves probabilistic independence.

With (v, v') and D given, let $D_k = \gamma_k^{(v,v')}(D)$ for each $k \geq 0$, and let $D' = (lim_{k\to\infty}\gamma_k)^{(v,v')}(D)$. Then we can establish each of the 3 claims about D'; claim 1 follows from the calculation

$$\begin{aligned} D'(s_1 \oplus s_2)\sum D' &= (lim_{k\to\infty}D_k(s_1 \oplus s_2))\sum lim_{k\to\infty}D_k \\ \text{(Lemma 4.1)} &= (lim_{k\to\infty}D_k(s_1 \oplus s_2))\,(lim_{k\to\infty}\sum D_k) = lim_{k\to\infty}(D_k(s_1 \oplus s_2)\sum D_k) \\ \text{(Lemma B.15)} &= lim_{k\to\infty}(D_k(s_1)D_k(s_2)) = D'(s_1)D'(s_2) \end{aligned}$$

whereas claim 2 (claim 3 is symmetric) follows from the calculation

$$\begin{aligned} D(s)\sum D' &= D(s)\sum lim_{k\to\infty}D_k \\ \text{(Lemma 4.1)} &= D(s)lim_{k\to\infty}\sum D_k = lim_{k\to\infty}(D(s)\sum D_k) \\ \text{(Lemma B.15)} &= lim_{k\to\infty}(D_k(s)\sum D) = D'(s)\sum D. \end{aligned}$$

□

Lemma 6.6 For a given pCFG, let (Q, Q_0) be a slicing pair. For all $k \geq 0$, all $(v, v') \in \mathsf{PD}$ with $R = rv_Q(v)$ and $R' = rv_Q(v')$ and $R_0 = rv_{Q_0}(v)$, all $D \in \mathscr{D}$ such that R and R_0 are independent in D, and all $\Delta \in \mathscr{D}$ such that $D \overset{R}{=} \Delta$, we have

$$\gamma_k{}^{(v,v')}(D) \overset{R'}{=} c_{k,D}^{v,v'} \cdot \Phi_k{}^{(v,v')}(\Delta).$$

Proof: The proof is by induction in k, with an inner induction on $\mathsf{LAP}(v, v')$.

Let us first (for all k) consider the case where $\mathbf{v} \notin \mathbf{Q} \cup \mathbf{Q_0}$. Thus, by Lemma 5.4, v stays outside $Q \cup Q_0$ until v'. By Lemma 6.2 we see that $\gamma_k{}^{(v,v')} = \omega^{(v,v')}$, and we also see that $\Phi_k{}^{(v,v')}(\Delta) = \Delta$ (by Definition 6.11 if $k = 0$, and by Lemma 4.29 otherwise).

Our proof obligation is thus, since $R' = R$, that

$$\omega^{(v,v')}(D) \overset{R}{=} 1 \cdot \Delta.$$

But from Lemma 5.8 we get $\omega^{(v,v')}(D) \overset{R}{=} D$, and by assumption we have $D \overset{R}{=} \Delta$, so the claim follows since $\overset{R}{=}$ is obviously transitive.

If $\mathbf{k = 0}$ **but** $\mathbf{v}$ **does not stay outside** $\mathbf{Q \cup Q_0}$ **until** $\mathbf{v'}$ then our proof obligation is

$$0 \overset{R'}{=} c_{k,D}^{v,v'} \cdot 0$$

which obviously holds (no matter what $c_{k,D}^{v,v'}$ is).

We now consider $\mathbf{k > 0}$, in which case $\gamma_k = \mathscr{H}_V(\gamma_{k-1})$ and $\Phi_k = \mathscr{H}_Q(\Phi_{k-1})$, and again consider several cases.

First assume that $\mathbf{v'} = \mathbf{v}$, and thus $R' = R$. Then our obligation is

$$D \overset{R'}{=} 1 \cdot \Delta$$

which follows directly from our assumptions.

Next assume that $\mathbf{v'} \neq \mathbf{v''}$ **with** $\mathbf{v''} = \mathbf{1PPD}(\mathbf{v})$. Then, by Definition 4.19, $\gamma_k{}^{(v,v')} = \gamma_k{}^{(v,v'')}; \gamma_k{}^{(v'',v')}$ and with $D'' = \gamma_k{}^{(v,v'')}(D)$ we thus have $\gamma_k{}^{(v,v')}(D) = \gamma_k{}^{(v'',v')}(D'')$; similarly, with $\Delta'' = \Phi_k{}^{(v,v'')}(\Delta)$ we have $\Phi_k{}^{(v,v')}(\Delta) = \Phi_k{}^{(v'',v')}(\Delta'')$. Since $\mathsf{LAP}(v, v'') < \mathsf{LAP}(v, v')$ we can apply the inner induction hypothesis to (v, v'') and get

$$D'' \overset{R''}{=} c_{k,D}^{v,v''} \cdot \Delta''$$

where $R'' = rv_Q(v'')$. With $R_0'' = rv_{Q_0}(v'')$, by Lemma B.15 we moreover see that R'' and R_0'' are independent in D''. Hence we can apply the inner induction hypothesis to (v'', v') to get

$$\gamma_k{}^{(v'',v')}(D'') \overset{R'}{=} c_{k,D''}^{v'',v'} \cdot \Phi_k{}^{(v'',v')}(c_{k,D}^{v,v''} \cdot \Delta'')$$

which since Φ_k is multiplicative (Lemma B.12) amounts to

$$\gamma_k{}^{(v,v')}(D) \overset{R'}{=} c_{k,D''}^{v'',v'} \cdot c_{k,D}^{v,v''} \cdot \Phi_k{}^{(v,v')}(\Delta)$$

which is as desired since $c_{k,D}^{v,v'} = c_{k,D}^{v,v''} \cdot c_{k,\gamma_k{}^{(v,v'')}(D)}^{v'',v'}$.

We are left with the situation that $v' = 1PPD(v)$ (and $k > 0$), with either $v \in Q$ or $v \in Q_0$; we now consider each of these possibilities.

Assume $\mathbf{v} \in \mathbf{Q_0}$ **(and** $\mathbf{k > 0}$ **and** $\mathbf{v' = 1PPD(v)}$**).** Thus $v \notin Q$, and hence (by Lemma 5.4) v stays outside Q until v' and thus $R' = R$. With $D' = {\gamma_k}^{(v,v')}(D)$, by Lemma B.15 we see that

$$D(s)\sum D' = D'(s)\sum D \text{ for all } s \in \mathscr{S}(R). \tag{11}$$

Since $v \notin Q$, and $\Phi_k = \mathscr{H}_Q(\Phi_{k-1})$ as $k > 0$, we see from clause 3a in Definition 4.19 that ${\Phi_k}^{(v,v')}(\Delta) = \Delta$. Thus our proof obligation is

$$D' \stackrel{R}{=} c_{k,D}^{v,v'} \cdot \Delta$$

which (as we assume $D \stackrel{R}{=} \Delta$) amounts to proving that for all $s \in \mathscr{S}(R)$:

$$D'(s) = c_{k,D}^{v,v'} \cdot D(s)$$

If $D = 0$ and hence $D' = 0$ then this is obvious. Otherwise, Definition 6.7 stipulates

$$c_{k,D}^{v,v'} = \frac{\sum D'}{\sum D}$$

and (11) yields the claim.

We shall finally consider the case $\mathbf{v} \in \mathbf{Q}$ **(and** $\mathbf{k > 0}$ **and** $\mathbf{v' = 1PPD(v)}$**).** Thus $v \notin Q_0$, and hence (by Lemma 5.4) v stays outside Q_0 until v'; thus $c_{k,D}^{v,v'} = 1$ so that our proof obligation, with $D' = {\gamma_k}^{(v,v')}(D)$ and $\Delta' = {\Phi_k}^{(v,v')}(\Delta)$, is to establish $D' \stackrel{R'}{=} \Delta'$, that is $D'(s') = \Delta'(s')$ for all $s' \in \mathscr{S}(R')$. We need a case analysis on the label of v, where the case **Skip** is trivial.

If $Lab(\mathbf{v}) = \mathbf{Observe}(\mathbf{B})$, we have $fv(B) \subseteq R = rv_Q(v)$ and as $Def(v) = \emptyset$ also $R' \subseteq R$; for $s' \in \mathscr{S}(R')$ this gives us the desired

$$\begin{aligned}\Delta'(s') &= \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s'} \Delta'(s) = \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s',\ [\![B]\!]s} \Delta(s) = \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s',\ [\![B]\!]s} D(s) \\ &= \sum_{s \in \mathscr{S}(R) \mid s \stackrel{R'}{=} s'} D'(s) = D'(s').\end{aligned}$$

If **v is a branching node**, with B its condition, and v_1 its *true*-successor and v_2 its *false*-successor, with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$ and $\Delta_1 = \mathsf{select}_B(\Delta)$ and $\Delta_2 = \mathsf{select}_{\neg B}(\Delta)$ the situation is that $D' = D'_1 + D'_2$ and $\Delta' = \Delta'_1 + \Delta'_2$ where for each $i \in \{1,2\}$, D'_i and Δ'_i is computed as

- if $\mathsf{LAP}(v_i, v') < \mathsf{LAP}(v, v')$ then $D'_i = {\gamma_k}^{(v_i,v')}(D_i)$ and $\Delta'_i = {\gamma_k}^{(v_i,v')}(\Delta_i)$;
- if $\mathsf{LAP}(v_i, v') \geq \mathsf{LAP}(v, v')$ then $D'_i = {\gamma_{k-1}}^{(v_i,v')}(D_i)$ and $\Delta'_i = {\gamma_{k-1}}^{(v_i,v')}(\Delta_i)$.

Let $R_1 = rv_Q(v_1)$ and $R_2 = rv_Q(v_2)$; thus (by Lemma 3.13) $R_1 \subseteq R$ and $R_2 \subseteq R$, and also $fv(B) \subseteq R$. For each $i = 1,2$, we see

- that R_i is independent of R_0 in D_i (by Lemma B.13),
- that $D_i \stackrel{R_i}{=} \Delta_i$ (by Lemma 4.6 from $D_i \stackrel{R}{=} \Delta_i$ which holds as for $s \in \mathscr{S}(R)$ we have $D_1(s) = \Delta_1(s)$ and $D_2(s) = \Delta_2(s)$ since if say $[\![B]\!]s$ is false then the first equation amounts to $0 = 0$ and the second to $D(s) = \Delta(s)$),
- that v_i stays outside Q_0 until v', and thus $c_{k,D_i}^{v_i,v'} = 1$.

We now infer that for each $i = 1,2$ we have $D'_i \stackrel{R'}{=} \Delta'_i$ (when $\mathsf{LAP}(v_i, v') < \mathsf{LAP}(v, v')$ this follows from the inner induction hypothesis, and otherwise it follows from the outer induction hypothesis on k). For $s \in \mathscr{S}(R')$ we

thus have $D'_1(s') = \Delta'_1(s')$ and $D'_2(s') = \Delta'_2(s')$, which implies $D'(s') = \Delta'(s')$. This amounts to the desired $D' \stackrel{R'}{=} \Delta$.

If **v is a (random) assignment**, let $x = Def(v)$ and first assume that $x \notin R'$. Thus $R' \subseteq R$ so that $D \stackrel{R'}{=} \Delta$, and by Lemma 4.12 (4.15) we get $D \stackrel{R'}{=} D'$ and $\Delta \stackrel{R'}{=} \Delta'$. But this implies the desired $D' \stackrel{R'}{=} \Delta'$ since for $s' \in \mathscr{S}(R')$ we have $D'(s') = D(s') = \Delta(s') = \Delta'(s')$.

We can thus assume $x \in R'$ and first consider when $Lab(v) = x := E$. Then (by Lemma 3.12) $R = R'' \cup fv(E)$ with $R'' = R' \setminus \{x\}$, so given $s' \in \mathscr{S}(R')$ we can use Lemma B.3 twice to give us the desired

$$D'(s') \;=\; \sum_{s \in \mathscr{S}(R) \;|\; s \stackrel{R''}{=} s',\; s'(x) = [\![E]\!]s} D(s) \;=\; \sum_{s \in \mathscr{S}(R) \;|\; s \stackrel{R''}{=} s',\; s'(x) = [\![E]\!]s} \Delta(s) \;=\; \Delta'(s').$$

We next consider the case when $Lab(v) = x := \mathbf{Random}(\psi)$. Then $R = R' \setminus \{x\}$, so given $s' \in \mathscr{S}(R')$ we can use Lemma B.4 twice to give us the desired

$$D'(s') = \psi(s'(x)) \left(\sum_{s \in \mathscr{S}(R) \;|\; s \stackrel{R}{=} s'} D(s) \right) = \psi(s'(x)) \left(\sum_{s \in \mathscr{S}(R) \;|\; s \stackrel{R}{=} s'} \Delta(s) \right) = \Delta'(s')$$

□

B.5 Proofs for Section 7

Lemma 7.5 There exists an algorithm $\mathtt{DD}^{\mathrm{close}}$ which given a node set Q that is closed under data dependence, and a node set Q_1, returns the least set containing Q and Q_1 that is closed under data dependence. Moreover, assuming $\mathtt{DD}^*$ is given, $\mathtt{DD}^{\mathrm{close}}$ runs in time $O(n \cdot |Q_1|)$.

Proof: We incrementally augment Q as follows: for each $v_1 \in Q_1$, and each $v \notin Q$, we add v to Q iff $\mathtt{DD}^*(v, v_1)$ holds. This is necessary since any set containing Q_1 that is closed under data dependence must contain v; observe that Q will end up containing Q_1 since for all $v_1 \in Q_1$ we have $\mathtt{DD}^*(v_1, v_1)$.

Thus the only non-trivial claim is that the resulting Q will be closed under data dependence. With $v \in Q$ we must show that if $v' \stackrel{dd}{\to} v$ then $v' \in Q$. If v was in Q initially, this follows since Q was assumed to be closed under data dependence. Otherwise, assume that v was added to Q because for some $v_1 \in Q_1$ we have $\mathtt{DD}^*(v, v_1)$. By correctness of $\mathtt{DD}^*$ this means that $v \stackrel{dd}{\to}{}^* v_1$ which implies $v' \stackrel{dd}{\to}{}^* v_1$ and thus $\mathtt{DD}^*(v', v_1)$ holds. Hence also v' will be added to Q. □

Lemma 7.8 The function PNV? runs in time $O(n)$ and, given Q, returns C such that $C \cap Q = \emptyset$ and

- if C is empty then Q provides next visibles
- if C is non-empty then all supersets of Q that provide next visibles will contain C.

Proof: It is convenient to introduce some terminology: we say that $q \in Q \cup \{\mathtt{End}\}$ is m-next from v iff there exists a path $v = v_1 \ldots v_k = q$ with $k \leq m$ and $v_j \notin Q$ for all j with $1 \leq j < k$. Also, we use superscript m to denote the value of a variable when the guard of the while loop is evaluated for the m'th time; note that if $q = N^m(v)$ and $m' > m$ then also $q = N^{m'}(v)$. A key part of the proof is to establish 3 facts, for each $m \geq 1$:

1. if $q = N^m(v)$ then q is m-next from v
2. if $v \in F^m$ then $N^m(v) \neq \bot$ and no $q \in Q$ is $(m-1)$-next from v
3. if $C^m = \emptyset$ and q is m-next from v then

(a) $q = N^m(v)$, and
(b) if q is not $(m-1)$-next from v then $v \in F^m$.

We shall prove the above facts simultaneously, by induction in m. The base case is when $m = 1$ and the facts follow from inspecting the preamble of the while loop: for (1), if $q = N^1(v)$ then $q = v \in Q \cup \{\texttt{End}\}$ and the trivial path v shows that q is 1-next from v; for (2), if $v \in F^1$ then $N^1(v) \neq \bot$ and no q can be 0-next from v; for (3), if q is 1-next from v then $q = v \in Q \cup \{\texttt{End}\}$ in which case $q = N^1(v)$ and $v \in F^1$.

We now do the inductive case where $m > 1$. Note that $C^{m-1} = \emptyset$ (as otherwise the loop would have exited already). For (1), we assume that $q = N^m(v)$, and split into two cases: if $q = N^{m-1}(v)$ then we inductively infer that q is $(m-1)$-next of v and thus also m-next of v. Otherwise, $v \notin Q$ and there exists an edge from v to some $v' \in F^{m-1}$ with $q = N^{m-1}(v')$. Inductively, q is $(m-1)$-next from v'. But then, as $v \notin Q$, q is m-next from v.

For (2), we assume that $v \in F^m$ in which case code inspection yields that $N^{m-1}(v) = \bot$ and that $N^m(v) = N^{m-1}(v')$ for some $v' \in F^{m-1}$ so inductively $N^m(v) \neq \bot$. Also, no q can be $m-1$-next from v, for if so then we could inductively use (3a) to infer $N^{m-1}(v) = q$.

For (3), we assume that $C^m = \emptyset$ and q is m-next from v. We have two cases:

- if q is $(m-1)$-next from v then (3b) holds vacuously, and as $C^{m-1} = \emptyset$ we infer inductively that $q = N^{m-1}(v)$ and thus $q = N^m(v)$ which is (3a).
- if q is not $(m-1)$-next from v, we can inductively use (1) to get $q \neq N^{m-1}(v)$, and also infer that there is a path $vv' \ldots q$ where $v \notin Q$ and q is $(m-1)$-next from v' but not $(m-2)$-next from v'. As $C^{m-1} = \emptyset$ we inductively infer that $q = N^{m-1}(v')$ and $v' \in F^{m-1}$. Thus the edge from v to v' has been considered in the recent iteration, and since $C_m = \emptyset$ it must be the case that $N^{m-1}(v) = \bot$ so we get the desired $q = N^m(v)$ and $v \in F^m$.

We are now ready to address the claims in the lemma. From (2) we see that each node gets into F at most once and hence the running time is in $O(n)$. That $C \cap Q = \emptyset$ follows since only nodes not in Q get added to C. Next we shall prove that

if C is empty then Q provides next visibles

and thus consider the situation where for some m, $C^m = \emptyset$ and $F^m = \emptyset$.

We shall first prove that for all $v \in \mathscr{V}$, all $q \in Q \cup \{\texttt{End}\}$, and all $k \geq 1$ we have that if q is k-next from v then $N^m(v) = q$. To see this, we may wlog. assume that k is chosen as small as possible, that is, q is not $(k-1)$-next from v. It is impossible that $k > m$ since then there would be a path $v \ldots v' \ldots q$ where q is m-next from v' but not $(m-1)$-next from v' which by (3b) entails $v' \in F^m$ which is a contradiction. Thus $k \leq m$ and q is m-next from v so (3a) yields the claim.

Now let $v \in \mathscr{V}$ be given, to show that v has a next visible in Q. Since there is a path from v to $\texttt{End}$ there will be a node $q \in Q \cup \{\texttt{End}\}$ such that (for some k) q is k-next from v. By what we just proved, $N^m(v) = q$ and we shall show that q is a next visible in Q of v. Thus assume, to get a contradiction, that we have a path not containing q from v to a node in $Q \cup \{\texttt{End}\}$. Then there exists $q' \neq q$ and k' such that q' is k'-next from v. Again applying what we just proved, $N^m(v) = q'$ which is a contradiction.

Finally, we shall prove that

if C is non-empty
then all supersets of Q that provide next visibles will contain C

and thus consider the situation where for some m, $C^m \neq \emptyset$. It is sufficient to consider $v \in C^m$ (and thus $v \notin Q$) and prove that if $Q \subseteq Q_1$ where Q_1 provides next visibles then $v \in Q_1$.

Since $v \in C^m$, the situation is that there is an edge from v to some $v' \in F^{m-1}$ with $q \neq q'$ where $q = N^m(v)$ and $q' = N^{m-1}(v')$. From (1) we see that q is m-next from v, and that q' is $(m-1)$-next from v'. That is, there exists a path π from v to q and a path π' from v to q'. Since $q, q' \in Q_1 \cup \{\texttt{End}\}$ and Q_1 provides next

visibles, there exists $v_0 \in Q_1$ that occurs on both paths. If $v_0 \neq v$ then q and q' are both $(m-1)$-next from v_0 which since $C^{m-1} = \emptyset$ implies $q = N^{m-1}(v_0) = q'$ which is a contradiction. Hence $v_0 = v$ which amounts to the desired $v \in Q_1$. □

Lemma 7.11 The function LWS, given $\hat{Q}$, returns Q such that
- Q is a weak slice set
- $\hat{Q} \subseteq Q$
- if Q' is a weak slice set with $\hat{Q} \subseteq Q'$ then $Q \subseteq Q'$.

Moreover, assuming $\mathtt{DD}^*$ is given, LWS runs in time $O(n^2)$.

Proof: We shall establish the following loop invariant:
- Q is closed under data dependence;
- Q includes $\hat{Q}$ and is a subset of any weak slice set that includes $\hat{Q}$.

This holds before the first iteration, by the properties of $\mathtt{DD}^{\mathrm{close}}$.

We shall now argue that each iteration preserves the invariant. This is obvious for the part about Q being closed under data dependence and including $\hat{Q}$. Now assume that Q' is a weak slice set that includes $\hat{Q}$; we must prove that $Q \subseteq Q'$ holds after the iteration. We know that before the iteration we have $Q \subseteq Q'$, and also $C = \mathtt{PNV?}(Q) \neq \emptyset$ so we know from Lemma 7.8 (since Q' provides next visibles) that $C \subseteq Q'$; hence we can apply Lemma 7.5 to infer (since Q' is closed under data dependence) $\mathtt{DD}^{\mathrm{close}}(Q,C) \subseteq Q'$ which yields the claim.

When the loop exits, with $C = \emptyset$, Lemma 7.8 tells us that Q provides next visibles. Together with the invariant, this yields the desired correctness property.

The loop will terminate, as Q cannot keep increasing; the total number of calls to PNV? is in $O(n)$. By Lemma 7.8 we see that the total time spent in PNV? is in $O(n^2)$. And by Lemma 7.5 we infer that the total time spent in $\mathtt{DD}^{\mathrm{close}}$ is in $O(n^2)$. Hence the running time of LWS is in $O(n^2)$. □

Theorem 2 The algorithm BSP returns, given a pCFG and a set of nodes ESS, sets Q and Q_0 such that
- (Q, Q_0) is a slicing pair wrt. ESS
- if (Q', Q_0') is a slicing pair wrt.ESS then $Q \subseteq Q'$.

Moreover, BSP runs in time $O(n^3)$ (where n is the number of nodes in the pCFG).

Proof: As stated in Figure 5, we shall use the following invariants for the while loop:
1. Q is a weak slice set
2. F is a weak slice set
3. $\mathtt{End} \in Q \cup F$
4. $W \subseteq \mathtt{ESS}$
5. if $v \in W$ then $Q_v \cap Q = \emptyset$
6. if $v \in \mathtt{ESS}$ but $v \notin W$ then $v \in Q \cup F$
7. if (Q', Q_0') is a slicing pair wrt. ESS then $Q \cup F \subseteq Q'$.

We shall first show that the invariants are established by the loop preamble, which is mostly trivial; for (2,3,7) we use Lemma 7.11.

Let us next show that the invariants are preserved by each iteration of the while loop. For (1) this follows from Lemma 5.3, the repeated application of which and Lemma 7.11 gives (2). Code inspection easily gives (3,4,5). To show (6) we do a case analysis: either v was not in W before the iteration so that (by the invariant) v belonged to $Q \cup F$ and thus $v \in Q$ by the end of the iteration, or v was removed from W during the iteration in which case $Q_v \subseteq F$ and thus (Lemma 7.11) $v \in F$.

For (7), let (Q', Q'_0) be a slicing pair wrt. ESS; we know that $Q \cup F \subseteq Q'$ holds before the iteration and thus $Q \subseteq Q'$ holds before the members of W are processed. It is sufficient to prove that if $v \in W$ with $Q_v \cap Q \neq \emptyset$ then $Q_v \subseteq Q'$. The invariant tells us that $v \in \mathtt{ESS}$, so as (Q', Q'_0) is a slicing pair wrt. ESS we infer that $v \in Q'$ or $v \in Q'_0$; by Lemma 7.11 this shows $Q_v \subseteq Q'$ (as desired) or $Q_v \subseteq Q'_0$ which we can rule out: for then we would have $Q_v \cap Q' = \emptyset$ and thus $Q_v \cap Q = \emptyset$ before W is processed which contradicts our assumption.

The while loop will terminate since W keeps getting smaller which cannot go on infinitely, and if an iteration does not make W smaller then it will have $F = \emptyset$ at the end and the loop exits.

When the loop exits, with $F = \emptyset$, we have:

- Q is a weak slice set with $\mathtt{End} \in Q$, by invariants (1) and (3);
- Q_0 is a weak slice set, by Lemmas 7.11 and 5.3;
- $Q \cap Q_0 = \emptyset$, by invariant (5);
- if $v \in \mathtt{ESS}$ then $v \in Q \cup Q_0$ since
  - if $v \notin W$ then $v \in Q$ by invariant (6) (and $F = \emptyset$),
  - if $v \in W$ then $v \in Q_0$ by construction of Q_0.

Thus (Q, Q_0) is a slicing pair. If (Q', Q'_0) is another slicing pair we see from invariant (7) that $Q \cup F \subseteq Q'$ and thus $Q \subseteq Q'$.

Finally, we can address the running time. By Lemma 7.4, we can compute $\mathtt{DD}^*$ in time $O(n^3)$. Then Lemma 7.11 tells us that each call to LWS takes time in $O(n^2)$. As there are $O(n)$ such calls, this shows the the code in BSP before the while loop runs in time $O(n^3)$. The while loop iterates $O(n)$ times, with each iteration processing $O(n)$ members of W; as each such processing (taking intersection and union) can be done in time $O(n)$ this shows that the while loop runs in time $O(n^3)$. The total running time is thus in $O(n^3)$. □