A Theory of Slicing for Imperative Probabilistic Programs

TORBEN AMTOFT, Kansas State University, USA
ANINDYA BANERJEE, IMDEA Software Institute, Spain

Dedicated to the memory of Sebastian Danicic.

We present a theory for slicing imperative probabilistic programs, containing random assignments and "observe" statements for conditioning. We represent such programs as probabilistic control-flow graphs (pCFGs) whose nodes modify probability distributions. This allows direct adaptation of standard machinery such as data dependence, postdominators, relevant variables, *etc.* to the probabilistic setting. We separate the specification of slicing from its implementation:

(1) first we develop syntactic conditions that a slice must satisfy (they involve the existence of another disjoint slice such that the variables of the two slices are *probabilistically independent* of each other);
(2) next we prove that any such slice is semantically correct;
(3) 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.

Our results carry over to the slicing of *structured* imperative probabilistic programs, as handled in recent work by Hur et al. For such a program we can define its slice which has the same "normalized" semantics as the original program; the proof of this property is based on a result proving the adequacy of the semantics of pCFGs wrt. the standard semantics of structured imperative probabilistic programs.

CCS Concepts: • **Theory of computation → Probabilistic computation**; **Program semantics**; • **Software and its engineering → Correctness**; **Automated static analysis**;

Additional Key Words and Phrases: probabilistic programming, program slicing, probabilistic control-flow graphs

ACM Reference Format:
Torben Amtoft and Anindya Banerjee. 2019. A Theory of Slicing for Imperative Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 1, 1, Article 1 (January 2019), 72 pages. https://doi.org/10.1145/3372895

1 INTRODUCTION

The task of program slicing [Tip 1995; Weiser 1984] is to remove the parts of a program that are irrelevant in a given context. This article addresses slicing of imperative probabilistic 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. [2014] 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

Authors' addresses: Torben Amtoft, Kansas State University, Manhattan, Kansas, USA, tamtoft@ksu.edu; Anindya Banerjee, IMDEA Software Institute, Pozuelo de Alarcon, Madrid, Spain, anindya.banerjee@imdea.org.

0164-0925/2019/1-ART1 $15.00
https://doi.org/10.1145/3372895

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 [Amtoft 2008; Ball and Horwitz 1993; Danicic et al. 2011; Podgurski and Clarke 1990; Ranganath et al. 2007]. 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. [2014] recently demonstrated that in the presence of random assignments and conditionings, standard notions of data and control dependence no longer suffice for semantically correct (backward) slicing. They develop a denotational framework in which they prove correct an algorithm for program slicing. In contrast, this article shows how *classical notions of dependence can be extended to give a semantic foundation for the (backward) slicing of imperative probabilistic programs.*

The article'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.
  To allow us to state and prove the correctness of slicing, we provide a novel 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' (much as the "Semantics 2" presented for a structured language in the seminal work by Kozen [1981]) so as to model what happens when "control" moves from v to v' in the control-flow graph. In Section 8 we discuss some choices involved in the design of the semantics.
- Syntactic conditions for correctness (Section 5) that in a non-trivial way extend classical work on program slicing [Danicic et al. 2011] 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 6.6 in Section 6).
  (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.)
- An algorithm (Section 9), with running time at most cubic in the size of the program, that (given an approximation of which loops terminate with probability 1) computes a slice which is optimal in that it is contained in any other syntactic slice of the program.

Our approach *separates* the specification of slicing from algorithms to compute (optimal) slices. The former is concerned with defining syntactic conditions for when a slice is correct, in that the behavior of the sliced program is equivalent to that of the original. The latter is concerned with how to compute a slice that satisfies the syntactic conditions and which is as small as possible; this slice is automatically a semantically correct slice —no separate proof is necessary. (It is obviously undecidable to compute the least *semantically* correct slice.)

This separation of concerns distinguishes our approach from the approach of Hur et al. [2014] which (to the best of our knowledge) is the main prior work on slicing of imperative probabilistic programs. That article does not state whether their algorithm computes slices that are in some sense the least possible; neither does it address the complexity of the algorithm. Their work incorporates powerful optimizations; in Section 10, we show that simple syntactic pre-processing may in some cases allow our approach to produce similar small slices.

Compared to the conference version of this article [Amtoft and Banerjee 2016], the additional contributions are:

- We allow to slice away certain loops if they are known (through some analysis, or an oracle, or deductive methods [McIver et al. 2018]) to terminate with probability 1.
- We show (Section 7) that our results apply to the slicing of *structured* imperative probabilistic programs; this involves establishing the adequacy of the semantics of pCFGs with respect to the "classical" semantics (based on expectation functions) of structured programs [Gordon et al. 2014; Hur et al. 2014]. In particular, we show that slicing based on our syntactic conditions (for the corresponding pCFG) will preserve the *normalized* semantics of a structured program.

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 Imperative Probabilistic Programs

Whereas in deterministic languages, a variable has only one value at a given time, we consider a simple imperative 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 [Panangaden 2009]).

Similarly to [Gordon et al. 2014], 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 [1981] in a denotational setting. As also in [Gordon et al. 2014], we shall use the construct $\mathbf{observe}(B)$ 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.

Slicing may be viewed as picking a set Q of "program points" and then removing from the program the program points not in Q. In order for Q to be a "correct" slice, it must satisfy certain conditions as we shall soon discuss.

The examples in this section all use a uniform 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 $x := \mathbf{random}(\psi_4)$ and $\mathbf{return}(x)$, and thus slice into a program P_x containing exactly these two instructions. The semantics of P_x is straightforward: 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 $\Delta'(\{x \mapsto i\}) = 0$.

2.2 Initial Examples

Example 2.1. Consider the program P_1 given by

$$\begin{array}{ll} 1: & x := \mathbf{random}(\psi_4) \\ 2: & y := \mathbf{random}(\psi_4) \\ 3: & \mathbf{observe}(y \geq 2) \\ 4: & \mathbf{return}(x) \end{array}$$

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 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.

Example 2.2. Consider the program P_2 given by

```
1 :  x := random(ψ4)
2 :  y := random(ψ4)
3 :  observe(x + y ≥ 5)
4 :  return(x)
```

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.

Example 2.3. Consider the program P_3 given by

```
1 :  x := random(ψ4)
2 :  if x ≥ 2
3 :     z := random(ψ4)
4 :     observe(z ≥ 3)
5 :  return(x)
```

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.

2.3 Why Standard Approaches Do Not Work

We shall now explain why slicing is particularly challenging when done in a probabilistic setting.

First note that the effect of an **observe** statement, which is to filter out undesired values, might also be achieved by means of a loop that does not terminate for those values. In particular, **observe**(B) may be thought of as equivalent to **while** $\neg B$ **do skip**, and we shall show (Example 4.30) that our semantics (Section 4) will indeed not distinguish between these two constructs (like the semantics in [Gordon et al. 2014; Hur et al. 2014] but unlike the semantics in [Bichsel et al. 2018]; see Section 8 for a further discussion). In principle, we could thus do our development without the **observe** construct, but we shall keep it as it is an important special case with a simple semantics.

Now let us consider our previous examples, with **observe** replaced by loops, and argue that it will not suffice to use standard techniques for slicing programs with loops. Such techniques [Ball and Horwitz 1993; Podgurski and Clarke 1990] come in two flavors: those that are "non-termination

sensitive" in that the sliced program must terminate exactly when the original does, and those that allow "irrelevant" loops to be sliced away which may cause the sliced program to terminate even when the original does not. In both cases, we need a slice to be closed under data dependence and control dependence; to achieve non-termination sensitive slicing, we also need the slice to be closed under *weak* control dependence [Podgurski and Clarke 1990]. (These concepts are informally described below.)

First assume that we go for slicing that is non-termination sensitive, and let us consider the program P_1 in Example 2.1 with **observe** replaced by a loop. But then line 4 is weakly control dependent on line 3, since from line 3 it is possible to avoid line 4 by looping forever. As we want the slice to contain line 4, it must thus also contain line 3, even though we earlier argued that line 3 can safely be sliced away. Hence non-termination sensitive slicing cannot readily be adapted for our purposes.

Next assume that we go for slicing that is non-termination *in*sensitive, and let us consider the program P_2 in Example 2.2 with **observe** replaced by a loop. Line 4 is data dependent on line 1 (since line 4 uses x which is defined in line 1), but not on line 2 or line 3. Also, line 4 is *not* control dependent on line 3, since there is no way to terminate from line 3 without reaching line 4. This shows that by using data dependence and (non-weak) control dependence, as done in non-termination insensitive slicing, lines 2 and 3 can be eliminated. But we argued that this will *not* be semantically correct, since line 3 in effect *changes* the distribution of x. It may appear there is an easy fix: modify the notion of data dependence so that an observe statement **observe**(B) is considered an implicit (random) assignment to the variables occurring in B. This will work for P_2 in that then line 3 will have to be in the slice. But the fix does *not* work for P_3 in Example 2.3 where line 5 is data dependent on line 1 but not dependent on any other lines, in particular line 5 does not depend on line 4 even if line 4 is considered an implicit (random) assignment to z. Yet it is not semantically correct to remove lines 2–4. We conclude that non-termination insensitive slicing cannot readily be adapted for our purposes.

The above considerations strongly suggest, as already observed by Hur et al. [2014], that to slice imperative probabilistic programs we need a new perspective on dependences.

2.4 Our Approach

A basic intuition behind our approach is that an **observe** statement can be removed if it does not depend on something on which the returned variable x also depends. This is the case in Example 2.1, whereas in Example 2.2, the **observe** statement is data dependent on the assignment to x, and in Example 2.3, the **observe** statement is control dependent on line 2 which is data dependent on the assignment to x.

More formally, this 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 (Figure 4) which contains a further requirement, motivated by the next example which addresses potentially non-terminating loops.

2.5 Examples with Various Degrees of Loop Termination

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 at most 3 iterations. In the resulting distribution D', for $i \in \{0, 1\}$ we have $D'(\{x \mapsto i\}) = D'(\{x \mapsto i,\ y \mapsto 0\}) = \frac{1}{4}$, and for $i \in \{2, 3\}$ we have $D'(\{x \mapsto i\}) = D'(\{x \mapsto i,\ y \mapsto 3\}) = \frac{1}{4}$. For all i we thus have $D'(\{x \mapsto i\}) = \Delta'(\{x \mapsto i\})$, and we see that 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 **if** construct 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 "$y :=$ **random**(ψ_4)". Then the loop may iterate arbitrarily many times, but will yet terminate with probability 1. Again, it is correct to slice P_4 into P_x.

3 PROBABILISTIC CONTROL-FLOW GRAPHS

To reason about the slicing of imperative probabilistic programs, we shall assume they are represented as 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 needed to define the semantics (Section 4) and conditions for slicing (Section 5). These concepts are mostly standard (see, *e.g.*, [Ball and Horwitz 1993; Podgurski and Clarke 1990]) and while defined for non-probabilistic programs, still apply in a probabilistic setting. Some of the concepts involve flow of data, such as data dependence (Definition 3.9) and relevant variables (Definition 3.10), while others involve only control aspects, such as postdomination (Definition 3.1 and Lemma 3.2). We omit (standard) control dependence since in Section 5 we shall take a recent alternative approach, but we do introduce a notion (Definition 3.14) which we have found useful for reasoning about control flow graphs.

Just as in the non-probabilistic case, a control-flow graph consists of a set V of nodes. There is a unique `end` node with no outgoing edges; from all nodes there must be a path to the `end` node. Each node $v \in \mathsf{V} \setminus \{\mathsf{end}\}$ has a label $Lab(v)$; also `end` may have a label which will then be of the form **return**(E) with E an arithmetic expression.

A *branching* node has a label containing a boolean expression and has *two* outgoing edges, one to its *true*-successor and one to its *false*-successor.

All other nodes (apart from branching nodes and the `end` node) have exactly one outgoing edge, to its *successor*. In the non-probabilistic case, a node with exactly one outgoing edge is labeled with either $x := E$ (x a program variable) or with **skip**. But in the probabilistic case, for a pCFG, such a node may also be labeled by

- **observe**(B) with B a boolean expression, or
- $x :=$ **random**(ψ) with ψ a random distribution, where for simplicity we shall assume, unlike say [Hur et al. 2014], that the random distribution ψ contains no program variables. We do not believe this to be a severe restriction since a random assignment that uses a random distribution employing program variables can often be encoded using variable-free random distributions.

We assume that one of the nodes is designated `start` (that node is always numbered 1 in our examples), and require that from `start` there are paths to all other nodes.

Figure 1 depicts pCFGs that represent the programs P_3 and P_4 from Examples 2.3 and 2.4.

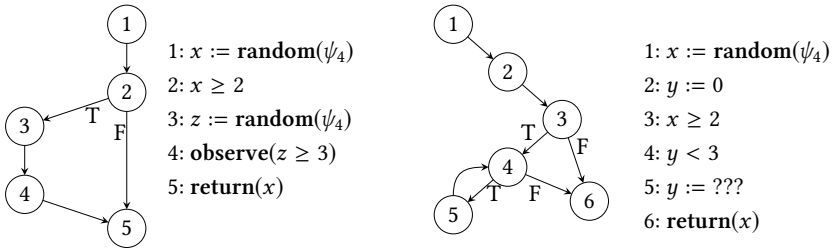

Fig. 1. The pCFGs for P_3 (left) and P_4 (right) from Examples 2.3 and 2.4.

We write $fv(E)$ for the free variables of E which (as there is no notion of bound variables) includes all variables that occur in E, similarly for $fv(B)$. 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**(E) then $fv(E)$. We demand that all variables be defined before they are used: for all nodes $v \in \mathsf{V}$, 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 (since all nodes have a path to the end node) 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.*

See Appendix B for the proof of this lemma, and of subsequent results not proved in main text.

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

*Definition 3.3 (*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 3.4. *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 3.5 (Cycle-inducing). A node v is *cycle-inducing* if with $v' = FPPD(v)$ 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 3.6. 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 3.7. *If v is cycle-inducing then there exists a cycle that contains v but not $FPPD(v)$.*

Proof. With $v' = FPPD(v)$, 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 3.8. *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 $\mathsf{LAP}(v, \mathsf{end})$. There must exist $v_0, v_1 \in \pi$ such that v_1 is a successor of v_0 and $f(v_1) \geq f(v_0)$ (since otherwise we can infer $f(v) < \ldots < f(v)$ for all $v \in \pi$). With $v' = FPPD(v_0)$ we then have (by Lemma 3.4)

$$\mathsf{LAP}(v_1, v') = f(v_1) - f(v') \geq f(v_0) - f(v') = \mathsf{LAP}(v_0, v')$$

which shows that v_0 is cycle inducing. □

Definition 3.9 (Data dependence). We say that v_2 is ***data dependent*** on v_1, written $v_1 \overset{dd}{\rightarrow} v_2$, 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), $2 \overset{dd}{\rightarrow} 4$ and $5 \overset{dd}{\rightarrow} 4$. A set of nodes Q is ***closed under data dependence*** if whenever $v_2 \in Q$ and $v_1 \overset{dd}{\rightarrow} v_2$ then also $v_1 \in Q$.

Definition 3.10 (Relevant variable). We say that x is ***relevant*** to Q before v, written $x \in rv_Q(v)$, if there exists $v' \in Q$ such that $x \in Use(v')$, and a 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 3.11. *Assume v is an assignment, of the form $x := E$, with successor v'. If $v \in Q$ then*

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

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 3.12. *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 3.13. *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* $Q_1 \cap Q_2 = \emptyset$ *when Q_1 and Q_2 are both closed under data dependence.*

Proof. The first claim is obvious. For the second claim, assume that Q_1 and Q_2 are closed under data dependence, and let us prove the contrapositive: we shall show $Q_1 \cap Q_2 \neq \emptyset$, assuming that there exists $x \in rv_{Q_1}(v) \cap rv_{Q_2}(v)$, 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, and that from `start` there are paths to all nodes, we infer that there is a path π from `start` to v where 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 $Q_1 \cap Q_2 \neq \emptyset$. □

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

Definition 3.14 (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 for all paths π from v to v' the following property holds: if v' occurs in π only at the end, then no node in $Q \setminus \{v'\}$ will occur in π.

Trivially, v stays outside Q until v for all Q and v. In Figure 1(right), node 4 stays outside $\{1, 6\}$ until 6, but does not stay outside $\{1, 5, 6\}$ until 6. Observe that $rv_{\{1,6\}}(4) = \{x\} = rv_{\{1,6\}}(6)$ which is an instance of a general property:

Lemma 3.15. *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 a 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 distribution transformers 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. We conclude by worked out examples (Section 4.5) that show how to use the semantics to reason about the termination of loops.

Our semantics is different in flavor to the semantics of "structured" imperative probabilistic programs given in [Gordon et al. 2014] (and [Hur et al. 2014]) which is a variation of one of the semantics proposed in [Kozen 1985]. In Section 7.3, however, we show that for a pCFG that is the translation of a structured imperative probabilistic program, the two semantics are adequately related.

4.1 Stores and Distributions

Let U be the universe of variables. A store s is a partial mapping from 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 $\mathsf{S}(R)$ for the set of stores with domain R, (and write S_U for $\mathsf{S}(\mathsf{U})$). If $s_1 \in \mathsf{S}(R_1)$ and $s_2 \in \mathsf{S}(R_2)$ 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 $s \in \mathsf{S}(R')$ and $R \subseteq R'$ we define $s|_R$ as the restriction of s to R. With R a subset of U, we say that s_1 agrees with s_2 on R, written $s_1 \overset{R}{=} s_2$, 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 $D \in \mathsf{D}$ (we shall later also use the letter Δ) is a mapping from $\mathsf{S_U}$ to $\mathbb{R}^{\infty}_{\geq 0}$, the non-negative reals augmented with ∞. We define $\sum D$ as $\sum_{s \in \mathsf{S_U}} D(s)$ which is well-defined even without measure theory, since $\mathsf{S_U}$ is a countable set due to our assumption that values are integers (as U can be assumed finite). Recall that for a random distribution ψ, used in random assignments, we require $\sum_{z \in \mathbb{Z}} \psi(z) = 1$; similarly, we *might* expect D to be a *probability* distribution in that $\sum D = 1$. But we also need to consider distributions D with $\sum D < 1$, since a branching node "splits" a distribution, and since we saw in Section 2 that parts of a distribution may "disappear" due to **observe** nodes or non-terminating loops. On the other hand, we never need to consider distributions D with $\sum D > 1$. We may therefore implicitly assume that we consider only *subprobability distributions*, that is, distributions D with $\sum D \leq 1$ (and thus $D(s) \leq 1$ for all $s \in \mathsf{S_U}$). This assumption is justified by, *e.g.*, Lemma 4.29 which ensures that the semantics transforms a subprobability distribution into a subprobability distribution.

We write $D = 0$ when $D(s) = 0$ for all s. We define $D_1 + D_2$ by stipulating $(D_1 + D_2)(s) = D_1(s) + D_2(s)$, and for $c \geq 0$ we define cD by stipulating $(cD)(s) = cD(s)$. Note that $D_1 + D_2$ may not be a subprobability distribution even if D_1 and D_2 both are (similarly for cD) unless we put some extra restrictions on D_1 and D_2. We find it convenient to develop our theory for general distributions, rather than only for subprobability distributions, so as not to clutter the presentation with checks that say $D_1 + D_2$ is well-defined.

Similarly, it is convenient to include ∞ and consider $\mathbb{R}^{\infty}_{\geq 0}$ since it forms a *a pointed cpo* with the usual ordering, as 0 is the bottom element and the supremum operator yields the least upper bound (which could be ∞) of a chain. (We refer to Appendix A for the basics of domain theory.) Hence also the set D of distributions forms 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. With $\mathsf{D_{fin}}$ the distributions with finite sum, that is $D \in \mathsf{D_{fin}}$ iff $\sum D < \infty$ (and hence $D(s) < \infty$ for all $s \in \mathsf{S_U}$), we shall state many of our results only for $D \in \mathsf{D_{fin}}$ (then our proofs do not have to worry about divisions by ∞).

The following result is often convenient; in particular, it shows that the least upper bound of a chain $\{D_k \mid k\}$ of subprobability distributions is a subprobability distribution.

Lemma 4.1. *Assume that $\{D_k \mid k\}$ is a chain of (not necessarily subprobability) distributions. 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 can extend D from a function from stores in $\mathsf{S_U}$ to a function on arbitrary stores:

Definition 4.2. For $s \in \mathsf{S}(R)$, let $D(s) = \sum_{s_0 \in \mathsf{S_U} \mid s_0 \overset{R}{=} s} D(s_0)$.

Observe that $D(\emptyset) = \sum D$. The equation in Definition 4.2 holds also when s_0 does not range over $\mathsf{S_U}$:

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

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

If $R' = R$ this is trivial (as the right hand side is then the sum of the singleton set $\{D(s)\}$); if $R = \emptyset$ we get $D(\emptyset) = \sum_{s' \in \mathsf{S}(R')} D(s')$ and thus (by renaming)

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

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

For example, $D_1 \overset{\{x\}}{=} D_2$ holds when $D_1(\{x \mapsto 7,\ y \mapsto 5\}) = 1$ and $D_2(\{x \mapsto 7,\ y \mapsto 8\}) = 1$ but D_1, D_2 are 0 otherwise. Agreement on a set implies agreement on a subset:

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

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

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

□

Definition 4.7 (concentrated). We say that a distribution D is ***concentrated*** if there exists $s_0 \in \mathrm{S}_\mathrm{U}$ such that $D(s) = 0$ for all $s \in \mathrm{S}_\mathrm{U}$ 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.)

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 4.8 (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 $s_1 \in \mathrm{S}(R_1)$ and $s_2 \in \mathrm{S}(R_2)$, we have

$$D(s_1 \oplus s_2) \sum D = D(s_1) D(s_2). \tag{1}$$

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 $0 < \sum D < \infty$ it still amounts to that definition but for "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 $R_2 = \emptyset$.

Lemma 4.9. *If R_1 and R_2 are independent in D, so are $\{x_1\}$ and $\{x_2\}$ for all $x_1 \in R_1, x_2 \in R_2$.*

Example 4.10. 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 $\sum D_1 = \sum_{s \in \mathrm{S}_\mathrm{U}} D_1(s) = 8 \cdot \frac{1}{16} = \frac{1}{2}$, 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 4.11. 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 $D_2(\{x \mapsto 3\}) D_2(\{y \mapsto 3\}) = \frac{4}{256}$.

Lemma 4.12. *If D is concentrated then R_1 and R_2 are independent in D for all disjoint R_1, R_2.*

4.3 Distribution Transformers

To deal with traversing a single edge in the pCFG, we shall define a number of functions with functionality $\mathsf{D} \to \mathsf{D}$. Each such ***distribution transformer*** 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 with $0 \leq c < \infty$;
- *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 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

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

Distribution transformers for **observe** nodes are not lossless (unless the condition is always true).

To show that a function is lossless, it suffices to consider concentrated distributions:

Lemma 4.13. *Let $f \in \mathsf{D} \to \mathsf{D}$ be continuous and additive. Assume that for all D that are concentrated, f is lossless for D. Then f is lossless.*

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

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

Lemma 4.14. *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 $s' \in \mathsf{S}_\cup$,

$$D'(s') = \sum_{s\in\mathsf{S}_\cup \mid s'=s[x\mapsto[\![E]\!]s]} D(s)$$

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

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

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

Lemma 4.17. *$\mathsf{assign}_{x:=E}$ 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 $s' \in \mathsf{S}_\cup$,

$$D'(s') = \sum_{s\in\mathsf{S}_\cup \mid s' \overset{\cup\setminus\{x\}}{=} s} \psi(s'(x))D(s)$$

Lemma 4.18. *Assume that $\mathsf{rassign}_{x:=E}(D) = D'$ and $x \notin R$. Then $D \overset{R}{=} D'$.*

Lemma 4.19. *Each* $\mathsf{rassign}_{x:=E}$ *is additive, multiplicative, non-increasing, and lossless.*

Proof. That $\mathsf{rassign}_{x:=E}$ is lossless, and hence non-increasing, follows from Lemma 4.18, with $R = \emptyset$. Additivity and multiplicativity are trivial. □

Lemma 4.20. $\mathsf{rassign}_{x:=E}$ *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 3.5). 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 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 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.

H_X operates on $\mathsf{PD} \to \mathsf{D} \to \mathsf{D}$ and we shall show (Lemma 4.23) that it even operates on $\mathsf{PD} \to \mathsf{D} \to_c \mathsf{D}$ (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 4.21. $\mathsf{PD} \to (\mathsf{D} \to_c \mathsf{D})$ *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 4.22 (H_X*).* The functionality of H_X is given by

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

where, given

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

we define

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

by letting $h(v, v')$, written $h^{(v,v')}$, be stipulated by the rules in Figure 2 that are inductive in $\mathsf{LAP}(v, v')$.

We shall now briefly explain a couple of the clauses from Figure 2. Clause 3a expresses that only nodes in X are taken into account whereas other nodes are treated as if labeled **skip**.

Clause 3e says that for a branching node v, the distribution is split into a *true*-part which is given as input to the *true*-successor v_1, and a *false*-part given as input to the *false*-successor v_2; the results are eventually combined. We must make sure it is well-defined to apply the semantic function to a successor v_i and since $h^{(v,v')}$ is defined inductively in $\mathsf{LAP}(v, v')$ we cannot call $h^{(v_i,v')}$ if $\mathsf{LAP}(v_i, v') \geq \mathsf{LAP}(v, v')$; in that case, we will have to use the function h_0 given as input to the functional H_X.

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

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

(this is well-defined by Lemma 3.4);
(3) otherwise, that is if $v' = FPPD(v)$:
(a) if $v \notin X$ or $Lab(v) = \textbf{skip}$ 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 $x := \textbf{random}(\psi)$ then $h^{(v,v')}(D) = \mathsf{rassign}_{x:=\psi}(D)$;
(d) if $v \in X$ with $Lab(v)$ of the form $\textbf{observe}(B)$ 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 $D'_i = {h_0}^{(v_i,v')}(D_i)$.

Fig. 2. The rules for defining $\mathsf{H}_X(h_0)(v, v') : \mathsf{D} \to \mathsf{D}$, written $h^{(v,v')}$.

Lemma 4.23. *Assume that* ${h_0}^{(v,v')}$ *is continuous for all* $(v, v') \in \mathsf{PD}$ *and let* $h = \mathsf{H}_X(h_0)$. *Then* $h^{(v,v')}$ *is continuous for all* $(v, v') \in \mathsf{PD}$.

Proof. This follows by an easy induction in $\mathsf{LAP}(v, v')$, using Lemmas 4.14, 4.17 and 4.20, and the fact that the composition of two continuous functions is continuous. □

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

Lemma 4.24. *The functional* H_X *is continuous on* $\mathsf{PD} \to (\mathsf{D} \to_c \mathsf{D})$.

Lemmas 4.21 and 4.24, together with Lemma A.2 in Appendix A, give

Proposition 4.25. *The functional* H_X *has a least fixed point (belonging to* $\mathsf{PD} \to (\mathsf{D} \to_c \mathsf{D})$*), called fix*$(\mathsf{H}_X)$*, and given as* $lim_{k\to\infty}\, \mathsf{H}_X^k(0)$*, that is the limit of the chain* $\{\mathsf{H}_X^k(0) \mid k\}$ *(where* $\mathsf{H}_X^k(0)$ *denotes* k *applications of* H_X *to the modification function that maps all distributions to 0).*

We can now define the meaning of the original program:

Definition 4.26 (Meaning of Original Program). Given a pCFG, with V the set of its nodes, we define its meaning ω as $\omega = fix(\mathsf{H}_V)$. Thus $\omega = lim_{k\to\infty}\, \omega_k$ where $\omega_k = \mathsf{H}_V^k(0)$ (thus $\omega_0 = 0$).

Thus for all $k > 0$ we have $\omega_k = \mathsf{H}_V(\omega_{k-1})$. 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. (This is somewhat similar to the work by Barraclough et al. [2010] who to reason about slicing in the presence of non-termination present a semantics where while-loops may iterate at most k times; they define slicing to be correct if it preserves that semantics for arbitrarily large k, but not necessarily in the "limit" as we require.)

Recall that we view a *slice* as a set Q of nodes to be included in the sliced program:

Definition 4.27 (Meaning of Sliced Program). Given a pCFG, and given a slice Q, we define the meaning of the sliced program as $\phi = fix(\mathsf{H}_Q)$. Thus $\phi = lim_{k\to\infty}\, \phi_k$ where $\phi_k = \mathsf{H}_Q^k(0)$.

Example 4.28. 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 $\phi^{(1,4)}(D) = \mathsf{rassign}_{x:=\psi_4}(D)$.

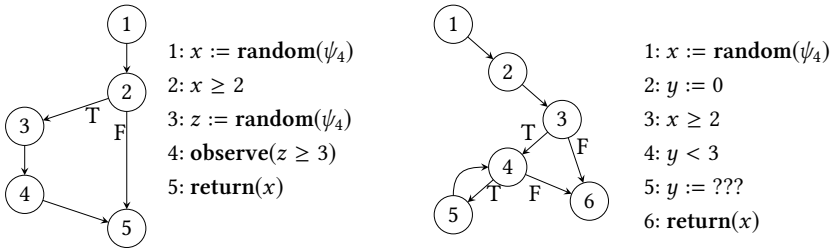

Fig. 3. The pCFGs for P_3 (left) and P_4 (right) from Examples 2.3 and 2.4 (copied from Figure 1).

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

Lemma 4.29. *Given a set X of nodes, let $h = fix(\mathsf{H}_X)$, and for each $k \geq 0$ let $h_k = \mathsf{H}_X^k(0)$. Then for each $(v, v') \in \mathsf{PD}$, $h^{(v,v')}$ is additive, multiplicative and non-increasing, as is each ${h_k}^{(v,v')}$, in particular (taking $X = \mathsf{V}$) each ${\omega_k}^{(v,v')}$.*

On the other hand, $h^{(v,v')}$ may fail to be lossless, due to **observe** nodes, or due to loops with non-zero probability of non-termination. Recall that a primary contribution of this article is to show that if a loop is lossless it may be safe to slice it away, even if it occurs in a branch (cf. Example 2.4).

Example 4.30. Consider a pCFG where a branching node v has condition B, *true*-successor v', and *false*-successor v (itself). Then for all D we have

$$\mathsf{H}_\mathsf{V}(h_0)^{(v,v')}(D) = \mathsf{select}_B(D) + {h_0}^{(v,v')}(\mathsf{select}_{\neg B}(D))$$

We shall now prove by induction in k that for all $k > 0$ and all D we have ${\omega_k}^{(v,v')}(D) = \mathsf{select}_B(D)$. This is obvious if $k = 1$ (as $\omega_1 = \mathsf{H}_\mathsf{V}(0)$), and for $k > 1$ we inductively have (as $\omega_k = \mathsf{H}_\mathsf{V}(\omega_{k-1})$)

${\omega_k}^{(v,v')}(D) = \mathsf{select}_B(D) + {\omega_{k-1}}^{(v,v')}(\mathsf{select}_{\neg B}(D)) = \mathsf{select}_B(D) + \mathsf{select}_B(\mathsf{select}_{\neg B}(D)) = \mathsf{select}_B D.$

We infer that for all D we have $\omega^{(v,v')}(D) = \mathsf{select}_B(D)$. This demonstrates (as mentioned in the beginning of Section 2.3) that in our semantics, **observe**(B) is equivalent to **while** $\neg B$ **do skip**.

The next two lemmas provide justification that Definition 3.14 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 lossless.

Lemma 4.31. *Given a pCFG, a slice Q, and $(v, v') \in \mathsf{PD}$ such that v stays outside Q until v'. We then have $\mathsf{H}_Q(h)^{(v,v')}(D) = D$ for all $D \in \mathsf{D}$ and all modification functions h.*

Lemma 4.32. *Let $(v, v') \in \mathsf{PD}$. Assume that Q is closed under data dependence and that v stays outside Q until v' (by Lemma 3.15 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 $\omega^{(v,v')}(D) \overset{R}{=} D$.

4.5 Formalizing Various Degrees of Loop Termination

We shall now again consider Example 2.4, with pCFG depicted in the right of Figure 3 (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 lossless when $Lab(5)$ is an assignment $y := y + 1$ or a random assignment $y :=$ **random**(ψ_4), but not when it is $y := 1$.

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 (as $\mathsf{U} = \{x, y\}$)

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

For all initial distributions D_0 with $\sum D_0 = 1$, it will be the case that with $D_3 = \omega^{(1,3)}(D_0)$ we have

$$D_3 = \sum_{i \in \{0,1,2,3\}} D_{i,0}^{0.25}$$

since y becomes zero in line 2. This shows that $\omega^{(1,6)}(D_0)$ does not depend on D_0 (as long as D_0 is a probability distribution) and may hence be considered the "meaning" of the program; to compute it, we need to compute $\omega^{(3,6)}(D_3)$ and for that purpose (cf. clause (3e) in Figure 2) we need to compute $\omega^{(4,6)}(D_4)$ where $D_4 = \mathsf{select}_{x \geq 2}(D_3)$ is given by

$$D_4 = \sum_{i=2,3} D_{i,0}^{0.25}.$$

For slicing purposes, our main interest is to investigate, for the various instantiations mentioned in Example 2.4, whether $\omega^{(4,6)}$ is lossless; by Lemma 4.13, that will hold iff $\omega^{(4,6)}$ is lossless for each $D^r_{i,j}$. Our first goal is to find an equation for $\omega^{(4,6)}(D^r_{i,j})$, with i and $r \geq 0$ given; we shall use that $\mathsf{LAP}(5,6) > \mathsf{LAP}(4,6)$ (cf. Example 3.6) and our calculations will depend on j.

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 Figure 2 (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

$$\forall j \geq 3 : \ \omega^{(4,6)}(D^r_{i,j}) = D^r_{i,j}. \tag{2}$$

Similarly, clause (3e) in Figure 2 also gives us

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

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

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

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 line (4) 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 line (3) 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 lossless.

$y := y + 1$ 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 line (4) 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 line (3) 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 line (2) confirms that $\omega^{(4,6)}$ is lossless, as we would expect since any loop from node 4 eventually (and soon) terminates.

$y := \mathbf{random}(\psi_4)$ 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 line (4), together with the fact (Lemma 4.29) 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 line (3) we get that

$$\forall k \geq 1, j < 3 : \omega_k{}^{(4,6)}(D^r_{i,j}) = 0.25 \cdot \left(\sum_{q=0,1,2,3} \omega_{k-1}{}^{(4,6)}(D^r_{i,q}) \right). \tag{5}$$

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

$$lim_{k\to\infty} D_k = D^r_{i,3} \tag{6}$$

which together with line (2) will demonstrate that $\omega^{(4,6)}$ is lossless, which tells us that any loop from node 4 will terminate with probability 1.
To show equation (6), observe that line (5) 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

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

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 4.33. *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 $lim_{i\to\infty} x_i = r$. □

5 CONDITIONS FOR SLICING

With Q a slice, we now develop conditions for Q that ensure semantic correctness. It is standard to require Q to be closed under data dependence, cf. Def. 3.9, and additionally also under some kind of "control dependence", a concept on which we elaborate in this section, 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 Q_0 containing all **observe** nodes to be sliced away. As stated in Proposition 6.4, 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 6.6.

5.1 Weak Slice Sets

Danicic et al. [2011] 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, and v a node, a node v' is a ***next visible*** in Q of v iff $v' \in Q \cup \{\mathsf{end}\}$, and v' occurs on all paths from v to a node in $Q \cup \{\mathsf{end}\}$.

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. If $v \in Q \cup \{\mathsf{end}\}$ then $next_Q(v) = v$, and if $v' = next_Q(v)$ then v' is a postdominator of v. We say that Q ***provides next visibles*** iff $next_Q(v)$ exists for all nodes v.

In the pCFG for P_3 (Figure 3(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 a suitable slice: node 1 is not a branching node and hence 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 [Amtoft 2008; Ranganath et al. 2007], 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"). In this article we have found it convenient to employ definitions slightly different from theirs, in particular we always consider end as "visible", and we demand $next_Q(v)$ to exist also when v is not reachable from Q.

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 $\mathsf{end} = next_\emptyset(v)$; also the set V of all nodes is a weak slice set since it is trivially closed under data dependence, and since for all v we have $v = next_{\mathsf{V}}(v)$. 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 $v \notin Q \cup \{\mathsf{end}\}$. Then v stays outside Q until $FPPD(v)$.*

Proof. Let $v' = FPPD(v)$ 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 $v' = FPPD(v)$) 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 3.5):

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 $v' = FPPD(v)$ and $v'' = next_Q(v)$. By Lemma 3.7 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 $v \in Q$. □

5.2 Adapting to the Probabilistic Setting

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

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

In applications, we shall also demand that Q contains the "slicing criterion", that is the nodes of interest. For imperative probabilistic programs, the slicing criterion will often be the singleton set $\{\mathsf{end}\}$ where end is labeled **return**(x), and in our subsequent development we shall indeed require $\mathsf{end} \in Q$ (even though this requirement is not needed for the statement of correctness, Theorem 6.6, in Section 6). In particular, the algorithm BSP for computing an optimal slice (Figure 14) incorporates this requirement (but could be easily modified to accommodate another slicing criterion).

We shall now see how these conditions work out for our example programs (represented as pCFGs).

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 Definitions 4.27 and 4.22, 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 3) 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 $6 = \mathit{next}_Q(v)$.

Definition 5.6 (slicing pair). Let Q, Q_0 be sets of nodes in a given pCFG. (Q, Q_0) is a ***slicing pair*** iff
(1) Q, Q_0 are both weak slice sets with 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) $\omega^{(v,FPPD(v))}$ is lossless (in which case we shall say that v is lossless).

Fig. 4. The definition of a slicing pair.

From Definitions 4.27 and 4.22 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 non-terminating 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 [Amtoft and Banerjee 2016].

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 3.8.

We have motivated adding condition 4 in the (final) definition of what is a correct slice, listed in Figure 4 and a main technical contribution of this article.

For Example 2.4, we saw in Example 3.6 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 lossless. Recalling the findings in Section 4.5, 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.

Let us next look at the pCFG in Figure 11(right) (on page 32) and see if we can remove nodes 4 and 6. As they are not lossless, we need $4, 6 \in Q_0$; as node 3 must have a next visible in Q_0, also $3 \in Q_0$. As $8 = \text{end} \in Q$, by data dependence also $5, 7 \in Q$; as node 3 must have a next visible in Q, also $3 \in Q$. But this conflicts with $Q \cap Q_0 = \emptyset$. We thus cannot remove the loops in Figure 11(right), even though that would be possible by standard slicing techniques if nodes 1,2 were deterministic assignments (say reading from input). Thus our approach is *not* a conservative extension of standard slicing; this may appear somewhat disconcerting but is not unexpected given that in Section 2.3 we argued that the probabilistic setting requires a radically different approach to slicing.

As one may intuitively expect, a distribution transformer is lossless if the corresponding path stays clear of **observe**-nodes and of non-lossless cycles:

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 $\omega^{(v_0, FPPD(v_0))}$ is lossless. 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 3.15), we have $\omega^{(v,v')}(D) \overset{R}{=} D$ 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 4.32. □

6 SLICING AND ITS CORRECTNESS

In this section, we shall embark on proving the semantic correctness of the slicing conditions developed in Section 5. Doing so involved at least two major challenges: rephrasing the fixed point semantics so as to facilitate proofs by induction, and stating a result about probabilistic independence strong enough to justify the removal of (for example) non-terminating loops. We shall address these challenges in the next two subsections, and then in Section 6.3 present the correctness result.

6.1 Alternative Convergence Towards Fixed Point

A proof of semantic correctness will involve reasoning about the behavior of $\omega^{(v,v')}$ for $(v, v') \in \mathrm{PD}$, but which reasoning principle should we employ? Just doing induction in $\mathrm{LAP}(v, v')$ 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 $\mathrm{LAP}(v, v')$ (possible since $\omega_{k+1} = \mathsf{H_V}(\omega_k)$ where $\mathsf{H_V}(\omega_k)^{(v,v')}$ is defined inductively in $\mathrm{LAP}(v, v')$);
(2) lift the results about ${\omega_k}^{(v,v')}$ to results about the limit $\omega^{(v,v')}$.

Unfortunately, this approach does not work for a property such as losslessness as this property will hold only in the limit. To see why this is a problem for proving the correctness of slicing, again consider the pCFG for P_4 depicted in Figure 3(right), with $Lab(5)$ given by $y := \mathbf{random}(\psi_4)$. We saw in Section 4.5 that then $\omega^{(4,6)}$ is lossless, obviously implying that also $\omega^{(1,6)}$ is lossless, so for all D_0 with $\sum D_0 = 1$ we will have $\omega^{(1,6)}(D_0)(\{x \mapsto i\}) = 0.25$ for $i \in \{0, 1, 2, 3\}$. With ϕ and ϕ_k as in Definition 4.27, using $Q = \{1, 6\}$ since $(\{1, 6\}, \emptyset)$ is a slicing pair (cf. Section 5.2), we will also have $\phi^{(1,6)}(D_0)(\{x \mapsto i\}) = {\phi_k}^{(1,6)}(D_0)(\{x \mapsto i\}) = 0.25$ for all $i \in \{0, 1, 2, 3\}$ and all $k \geq 0$. Thus we have the expected correctness result $\omega^{(1,6)}(D_0) \overset{\{x\}}{=} \phi^{(1,6)}(D_0)$. To prove such a result in general, it would be helpful if one could prove a similar relation for ω_k and ϕ_k, in particular that for each $k \geq 0$ there exists c such that ${\omega_k}^{(1,6)}(D_0) \overset{\{x\}}{=} c \cdot {\phi_k}^{(1,6)}(D_0)$. But for the given example this *cannot* be the case, since for each $k \geq 0$ we have ${\omega_k}^{(1,6)}(D_0)(\{x \mapsto 1\}) = 0.25$ (forcing $c = 1$) but ${\omega_k}^{(1,6)}(D_0)(\{x \mapsto 3\}) < 0.25$ (forcing $c < 1$).

To fix this problem, we shall introduce a family ($k \geq 0$) of functions

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

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 certain properties (cf. Lemma 6.9) hold from the beginning of the fixed point iteration, not just at the limit. This is achieved by this inductive definition:

Definition 6.1. Given (as implicit parameter) a slicing pair (Q, Q_0), for $k \geq 0$ define γ_k by:

$$\begin{array}{rcl} {\gamma_0}^{(v,v')} & = & \omega^{(v,v')} \text{ if } v \text{ stays outside } Q \cup Q_0 \text{ until } v' \\ {\gamma_0}^{(v,v')} & = & 0 \quad \text{otherwise} \\ \gamma_k & = & \mathsf{H_V}(\gamma_{k-1}) \text{ for } k > 0 \end{array}$$

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

In the above example, since 3 stays outside $\{1, 6\}$ until 6, for all $k \geq 0$ we have ${\gamma_k}^{(3,6)} = \omega^{(3,6)}$ and thus for all D_0 also ${\gamma_k}^{(1,6)}(D_0) \stackrel{\{x\}}{=} {\phi_k}^{(1,6)}(D_0)$. We see that (for this example) we accomplish our goal, that the correctness result should hold not just in the limit but also for each iteration, by working with γ_k rather than with ω_k (recall that we did *not* have ${\omega_k}^{(1,6)}(D_0) \stackrel{\{x\}}{=} {\phi_k}^{(1,6)}(D_0)$).

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 $\mathsf{H_V}$ 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 $\mathsf{H_V}$) 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 $lim_{k\to\infty}\, \gamma_k = \omega$.*

6.2 Probabilistic Independence

A main contribution of this article is 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 4.8) of the Q_0-relevant variables, assuming they are at `start` (which will be the case if say no variables are relevant there). This follows from

Proposition 6.4 (Independence). *Let (Q, Q_0) be a slicing pair, and let $D \in \mathsf{D}_{\mathrm{fin}}$ be given with $D' = \omega^{(v,v')}(D)$ (thus also $D' \in \mathsf{D}_{\mathrm{fin}}$). 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'.*

To prove this proposition, and to justify that nodes in Q_0 are sliced away, it turned out that we need to prove a stronger result:

Lemma 6.5. *Let (Q, Q_0) be a slicing pair. Let $D \in \mathsf{D}_{\mathrm{fin}}$ be given with $D' = \omega^{(v,v')}(D)$. 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 3.15 thus $R' = R$) then for all $s \in \mathsf{S}(R)$ 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 $s_0 \in \mathsf{S}(R_0)$ we have

$$D(s_0) \sum D' = D'(s_0) \sum D$$

We shall now briefly sketch (details in Appendix B) how part 2 of the Lemma justifies that nodes in Q_0 are sliced away. For if $v \in Q_0$ then $v \notin Q$ which by Lemma 5.4 implies (assuming $v' = FPPD(v)$) that v stays outside Q until v'. But then part 2 ensures that for the variables R relevant for the sliced program, the distribution at v' equals the distribution at v except for a constant factor.

6.3 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 6.6. *For a given pCFG, let (Q, Q_0) be a slicing pair, and let $\phi = fix(\mathsf{H}_Q)$ (cf. Definition 4.27) be the meaning of the sliced program. For a given $(v, v') \in \mathsf{PD}$, and a given $D \in \mathsf{D}_{\mathrm{fin}}$ 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 $c = 1$.

It may happen that $c = 0$, since it is possible that the original program never terminates but the sliced program may terminate, for example if the original program starts with an **observe**(*false*) node as such a node can be sliced away.

In Theorem 6.6, 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 6.6 (as done at the end of this section), we need to define a counterpart to γ_k (Definition 6.1) for the sliced program:

Definition 6.7. Given a slicing pair (Q, Q_0), for $k \geq 0$ define Φ_k as follows:

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

Lemma 6.8. *$\{\Phi_k \mid k\}$ is a chain, with $lim_{k\to\infty} \Phi_k = lim_{k\to\infty} \phi_k = \phi$.*

Proof. By Lemma 4.31 we get $\phi_0 \leq \Phi_0 \leq \phi_1$ so by the monotonicity of H_Q we inductively get

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

which yields the claim. □

We can now express γ_k in terms of Φ_k, where we allow (so as to facilitate a proof by induction in $\mathsf{LAP}(v, v')$) 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 6.9. *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 $D \in \mathsf{D}_{\mathrm{fin}}$ such that R and R_0 are independent in D, and all $\Delta \in \mathsf{D}_{\mathrm{fin}}$ such that $D \overset{R}{=} \Delta$, we have*

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

Here the numbers $c_{k,D}^{v,v'}$ are given by

Definition 6.10. For $k \geq 0$, $(v, v') \in \mathsf{PD}$, and $D \in \mathsf{D}_{\mathrm{fin}}$, 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 $v'' = FPPD(v)$ 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 6.10, since it is easy to prove by induction in $\mathsf{LAP}(v, v')$ that

LEMMA 6.11. *If v stays outside Q_0 until v' then $c_{k,D}^{v,v'} = 1$ for all $k \geq 0$ and $D \in \mathsf{D}_{\mathrm{fin}}$.*

Since each γ_k is non-increasing (Lemma B.12), it is easy to prove by induction in $\mathsf{LAP}(v, v')$ that

LEMMA 6.12. *We have $0 \leq c_{k,D}^{v,v'} \leq 1$ for all $k \geq 0$, $(v, v') \in \mathsf{PD}$, $D \in \mathsf{D}_{\mathrm{fin}}$.*

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

LEMMA 6.13. *$\{c_{k,D}^{v,v'} \mid k\}$ is a chain for each $(v, v') \in \mathsf{PD}$ and $D \in \mathsf{D}_{\mathrm{fin}}$.*

Proof of Theorem 6.6. We are given $(v, v') \in \mathsf{PD}$, and $D \in \mathsf{D}_{\mathrm{fin}}$ such that $rv_Q(v)$ and $rv_{Q_o}(v)$ are independent in D; let $R' = rv_Q(v')$. For each $s' \in \mathsf{S}(R')$ we have the calculation

$$\begin{array}{rcl} & & \omega^{(v,v')}(D)(s') \\ (\textit{Proposition 6.3}) & = & lim_{k\to\infty}\, \gamma_k{}^{(v,v')}(D)(s') \\ (\textit{Lemma 6.9}) & = & lim_{k\to\infty}\, (c_{k,D}^{v,v'} \cdot \Phi_k{}^{(v,v')}(D)(s')) \\ (\textit{Lemma 6.8}) & = & (lim_{k\to\infty}\, c_{k,D}^{v,v'}) \cdot \phi^{(v,v')}(D)(s'). \end{array}$$

With $c = lim_{k\to\infty}\, c_{k,D}^{v,v'}$ (well-defined by Lemma 6.13) 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 6.11).

7 STRUCTURED PROGRAMS

Probabilistic programming is usually, as in [Gordon et al. 2014; Hur et al. 2014], expressed using *structured* programs, rather than control flow graphs. In this section we shall show that our results on slicing of pCFGs can be applied to the slicing of structured imperative probabilistic programs.

We first present (Section 7.1) the syntax and semantics of the structured imperative probabilistic language SL we are considering and next we present (Section 7.2) a translation from SL to pCFGs; we then show (Section 7.3) an adequacy result relating the semantics of a structured probabilistic program to the semantics of its translation into a pCFG. We next address slicing, first defining (Section 7.4) what it means to slice a structured imperative probabilistic program, and then stating (Section 7.5) a result expressing the correctness of slicing.

7.1 The Structured Imperative Probabilistic Language

We shall consider a statement-based language SL that is *structured* in that each non-atomic statement is built compositionally from constructs that combine sub-statements.

$$
\begin{array}{rcl}
S & ::= & S_1 \ ; \ S_2 \\
& | & l : \textbf{skip} \\
& | & l : x := E \\
& | & l : x := \textbf{random}(\psi) \\
& | & l : \textbf{observe}(B) \\
& | & l : \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \\
& | & l : \textbf{while } B \textbf{ do } S \\
P & ::= & S \ ; \ \textbf{return}(E)
\end{array}
$$

Fig. 5. The grammar defining structured imperative probabilistic statements/programs.

Syntax of SL. We mostly adopt the syntax used in, *e.g.*, [Gordon et al. 2014; Hur et al. 2014]; the main difference is that (primarily to facilitate the translation presented in Section 7.2) we augment each substatement (except for sequential composition) with a label l which is a natural number.

A program is a *statement* followed by the return of an expression, where a statement S is defined by the BNF in Figure 5. That is, a statement S is either a sequential composition $S_1 \ ; \ S_2$, or consists of a label and then either **skip**, an assignment $x := E$, a random assignment $x := \textbf{random}(\psi)$, a conditioning statement $\textbf{observe}(B)$, a conditional $\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2$, or a while loop $\textbf{while } B \textbf{ do } S$. We shall assume that no label occurs more than once within a statement.

Semantics. For the semantics of the structured imperative probabilistic language we shall follow Gordon et al. [2014] (and [Hur et al. 2014]) who modified (in particular to handle conditioning) one of the semantics proposed by Kozen [1985]. The semantics (which ignores the labels) manipulates "expectation functions" where an expectation function F maps stores to $\mathbb{R}^{\infty}_{\geq 0}$; we can think of $F(s)$ as the expected return value for store s. The semantics of a program will be given in Definition 7.2; we shall first present the semantics of a statement S, written $[\![S]\!]$, which is a transformation of expectation functions: with $[\![S]\!]F' = F$, one should think of F' as taking a store *after* S and giving its expected return value, and F as taking a store *before* S and giving its expected return value.

In Figure 6, we define $[\![S]\!]$ by a definition inductive in S. Let us explain a few cases:

- the expected return value for a store before an assignment $x := E$ equals the expected return value for the updated store;
- the expected return value for a store before a random assignment $x := \textbf{random}(\psi)$ can be found by taking the weighted average of the expected return values for the possible updated stores;
- the expected return value for a store before a conditioning statement $\textbf{observe}(B)$ is 0 if B is not true;
- the semantics of a while loop $\textbf{while } B \textbf{ do } S$ can be found as the limit of the semantics of the kth iteration, $\textbf{while } B \textbf{ do}_k \ S$, which is defined inductively in k as follows:

$$
\begin{array}{rcl}
\textbf{while } B \textbf{ do}_0 \ S & = & \textbf{observe}(\mathit{false}) \\
\textbf{while } B \textbf{ do}_{k+1} \ S & = & \textbf{if } B \textbf{ then } (S \ ; \ \textbf{while } B \textbf{ do}_k \ S) \textbf{ else skip}
\end{array}
$$

Example 7.1. Consider the statement S_2 given by (cf. Example 2.2)

$$
S_2 \stackrel{def}{=} 1 : x := \textbf{random}(\psi_4) \ ; \ 2 : y := \textbf{random}(\psi_4) \ ; \ 3 : \textbf{observe}(x + y \geq 5)
$$

$$
\begin{array}{rcll}
F = [\![l : \mathbf{skip}]\!]F' & \text{iff} & F = F' \\
F = [\![l : x := E]\!]F' & \text{iff} & F(s) = F'(s[x \mapsto [\![E]\!]s]) \text{ for all } s \\
F = [\![l : x := \mathbf{random}(\psi)]\!]F' & \text{iff} & F(s) = \sum_{z \in \mathbb{Z}} \psi(z) F'(s[x \mapsto z]) \text{ for all } s \\
F = [\![l : \mathbf{observe}(B)]\!]F' & \text{iff} & F(s) = F'(s) \text{ for all } s \text{ with } [\![B]\!]s \\
 & \text{and} & F(s) = 0 \text{ for all other } s \\
F = [\![S_1 \ ; \ S_2]\!]F' & \text{iff} & F = [\![S_1]\!]([\![S_2]\!]F') \\
F = [\![l : \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]F' & \text{iff} & F(s) = [\![S_1]\!](F')(s) \text{ for all } s \text{ with } [\![B]\!]s \\
 & \text{and} & F(s) = [\![S_2]\!](F')(s) \text{ for all other } s \\
F = [\![l : \mathbf{while}\ B\ \mathbf{do}\ S]\!]F' & \text{iff} & F(s) = lim_{k \to \infty} F_k(s) \text{ for all } s \\
 & \text{where} & F_0(s) = 0 \\
 & \text{and} & F_{k+1}(s) = [\![S]\!](F_k)(s) \text{ if } [\![B]\!]s \\
 & \text{and} & F_{k+1}(s) = F'(s) \text{ otherwise}
\end{array}
$$

Fig. 6. The semantics of a structured probabilistic statement.

For all F that map stores into $\mathbb{R}^{\infty}_{\geq 0}$, and all stores s, we have

$$
\begin{array}{rcl}
[\![S_2]\!]F\ s & = & [\![x := \mathbf{random}(\psi_4)]\!]([\![y := \mathbf{random}(\psi_4);\ \mathbf{observe}(x + y \geq 5)]\!]F)\ s \\
 & = & \displaystyle\sum_{q \in 0..3} \frac{1}{4} ([\![y := \mathbf{random}(\psi_4);\ \mathbf{observe}(x + y \geq 5)]\!]F\ s[x \mapsto q]) \\
 & = & \displaystyle\sum_{q \in 0..3} \frac{1}{4} (\sum_{q' \in 0..3} \frac{1}{4} ([\![\mathbf{observe}(x + y \geq 5)]\!]F\ s[x \mapsto q][y \mapsto q'])) \\
 & = & \displaystyle\frac{1}{16} \left(F(s[x \mapsto 2][y \mapsto 3]) + F(s[x \mapsto 3][y \mapsto 2]) + F(s[x \mapsto 3][y \mapsto 3])\right).
\end{array}
$$

For a program $P = S \ ; \ \mathbf{return}(E)$, the expectation function at the end will map s into $[\![E]\!]s$, and thus the expectation function at the beginning appears to be given as $[\![S]\!](\lambda s.[\![E]\!]s)$. But this assumes that runs that fail conditioning statements count as zero; such runs should rather not be taken into account at all. This motivates the following definition [Gordon et al. 2014] of the *normalized* semantics of a structured imperative probabilistic program:

Definition 7.2 (Normalized Semantics). Given a structured imperative probabilistic program $P \equiv S \ ; \ \mathbf{return}(E)$. With $\perp$ an "initial store", we define $[\![P]\!]$, the normalized semantics of P, as

$$
[\![P]\!] = \frac{[\![S]\!](\lambda s.[\![E]\!]s)(\perp)}{[\![S]\!](\lambda s.1)(\perp)} \tag{7}
$$

Note that this may not be well-defined, in case $[\![S]\!](\lambda s.1)(\perp) = 0$ or $[\![S]\!](\lambda s.1)(\perp) = \infty$, and that the choice of initial store $\perp$ is irrelevant since we demand that all variables are defined before they are used.

To illustrate Definition 7.2, observe that P_2 from Example 2.2 is essentially S_2 ; **return**(x) with S_2 defined as in Example 7.1 from which we see (since $[\![x]\!]s = s(x)$) that

$$[\![S_2]\!](\lambda s.[\![x]\!]s)\,\bot = \frac{1}{16}(2+3+3) = \frac{8}{16}$$
$$[\![S_2]\!](\lambda s.1)\,\bot = \frac{1}{16}(1+1+1) = \frac{3}{16}$$

and hence we see by Equation (7) that

$$[\![P_2]\!] = \frac{[\![S_2]\!](\lambda s.[\![x]\!]s)(\bot)}{[\![S_2]\!](\lambda s.1)(\bot)} = \frac{8}{3}.$$

This makes sense: if P_2 terminates then $x + y \geq 5$ which holds in 3 cases; in two cases, $x = 3$ whereas in one case, $x = 2$, for a weighted average of $\frac{8}{3}$.

7.2 Translating Structured Imperative Probabilistic Statements Into pCFGs

We shall present a translation T from structured imperative probabilistic programs (Section 7.1) to pCFGs (Section 3). To allow for a deterministic translation (that does not rely on "fresh" nodes), we shall assume functions l2v and l2v′ that map statement labels into nodes. Intuitively, if l2v(l) occurs in the translation of the statement labeled l then it is its `start` node, and if l2v′(l) occurs in the translation then it is its end node. We require uniqueness, in the sense that if $l_1 \neq l_2$ then l2v(l_1), l2v(l_2), l2v′(l_1), and l2v′(l_2) are 4 distinct nodes.

Definition 7.3 (Translation from Structured Imperative Probabilistic Statements to pCFGs). For a structured imperative probabilistic statement S we define a pCFG T(S) by structural induction in S, using the definition given in Figure 7 (and illustrated in Figure 8).

Definition 7.3 is such that for all structured imperative probabilistic statements S:

- T(S) has unlabeled end node;
- if v is a node in T(S) then v = l2v(l) or v = l2v′(l) for some l occurring in S (and thus T(S_1) and T(S_2) have disjoint node sets if S_1 and S_2 have disjoint labels);
- if S_1 is a substatement of S then T(S_1) is a sub-pCFG of T(S), in the sense that all nodes and edges in T(S_1) will also belong to T(S) (and if a node has a label in T(S_1) it will have the same label in T(S)).

It is easy to see by induction in S (relying on the assumption that no label occurs more than once in S) that the translation in Definition 7.3 does indeed always define a well-formed pCFG G with the listed properties; in particular, all nodes are reachable from `start`(G) (the `start` node), and can reach end(G) (the end node).

Lemma 7.4. *When S is of the form l : _ then in G = T(S) it is then the case that* end(G) = *FPPD*(`start`(G)) *where* `start`(G) = l2v(l) *and* end(G) = l2v′(l).

Definition 7.5 (Translation from Structured Imperative Probabilistic Programs to pCFGs). For a structured imperative probabilistic program $P = S$; **return**(E), we define a pCFG T(P) as follows: first construct the pCFG T(S); then label its end node (unlabeled so far) with **return**(E).

Example 7.6. Consider (cf. Example 2.2) the structured imperative probabilistic program P_2 given by S_2 ; **return**(x) with S_2 defined in Example 7.1. Then G_2 = T(P_2) will be as depicted in Figure 9(left), but can be simplified by compression of edges from nodes labeled **skip**.

Below we define $\mathsf{T}(S)$ by induction in S, doing a case analysis on S.

- First assume S is of the form l : **skip**, or $l : x := E$, or $l : x :=$ **random**(ψ), or $l :$ **observe**(B). Then $\mathsf{T}(S)$ is a pCFG with 2 nodes, v given by $\mathsf{l2v}(l)$ and v' given by $\mathsf{l2v'}(l)$, and with one edge, from v to v'. Here v is the `start` node and is labeled according to the form of S, whereas v' is the `end` node with no label.
- Next assume that S is of the form S_1 ; S_2. Inductively, we construct a pCFG $G_1 = \mathsf{T}(S_1)$ with `start` node v_1 and unlabeled `end` node v_1', and a pCFG $G_2 = \mathsf{T}(S_2)$ with `start` node v_2 and unlabeled `end` node v_2'. As illustrated in Figure 8 (left), the pCFG $G = \mathsf{T}(S)$ is then constructed by taking the union of G_1 and G_2 (which by our assumptions have disjoint node sets), and augmenting the result as follows:
  - let G have `start` node v_1, and `end` node v_2';
  - let G contain an edge from v_1' to v_2, and let the label of v_1' be **skip**.
- Next assume that S is of the form l : **if** B **then** S_1 **else** S_2. Inductively, we construct a pCFG $G_1 = \mathsf{T}(S_1)$ with `start` node v_1 and unlabeled `end` node v_1', and a pCFG $G_2 = \mathsf{T}(S_2)$ with `start` node v_2 and unlabeled `end` node v_2'. As illustrated in Figure 8 (middle), the pCFG $G = \mathsf{T}(S)$ is then constructed by taking the union of G_1 and G_2 (which by our assumptions must have disjoint node sets), and augmenting the result as follows, where $v = \mathsf{l2v}(l)$ becomes the `start` node of G and $v' = \mathsf{l2v'}(l)$ becomes the (unlabeled) `end` node of G:
  - let v be a branching node with condition B, *true*-successor v_1, and *false*-successor v_2;
  - let G contain edges from v_1' to v' and from v_2' to v', and let the labels of v_1' and v_2' be **skip**.
- Finally, assume that S is of the form l : **while** B **do** S_1. Inductively, we construct a pCFG $G_1 = \mathsf{T}(S_1)$ with `start` node v_1 and unlabeled `end` node v_1'. As illustrated in Figure 8 (right), the pCFG $G = \mathsf{T}(S)$ is then constructed by augmenting G_1 as follows, where $v = \mathsf{l2v}(l)$ becomes the `start` node of G and $v' = \mathsf{l2v'}(l)$ becomes the (unlabeled) `end` node of G:
  - let v be a branching node with condition B, *true*-successor v_1, and *false*-successor v';
  - let G contain an edge from v_1' to v, and let the label of v_1' be **skip**.

Fig. 7. The rules for translating a structured imperative probabilistic statement S into a pCFG $\mathsf{T}(S)$.

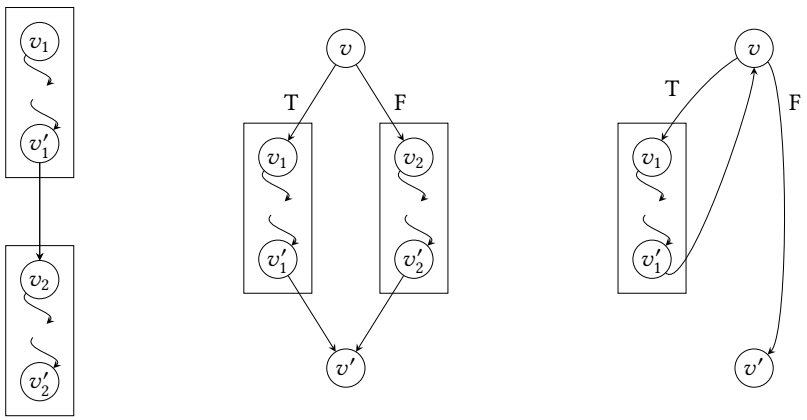

Fig. 8. Translating sequential composition (left), conditionals (middle), while loops (right).

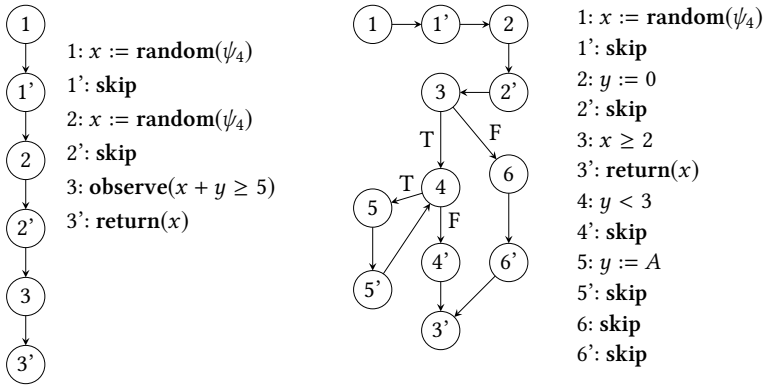

Fig. 9. Left: a pCFG G_2 that is $\mathsf{T}(P_2)$ (cf. Example 2.2); Right: a pCFG G_4 that is $\mathsf{T}(P_4)$ (cf. Example 2.4).

Example 7.7. Consider (cf. Example 2.4) the structured imperative probabilistic program $P_4 \stackrel{def}{=} S_4$; **return**(x) where

$$S_4 \stackrel{def}{=} 1 : x := \textbf{random}(\psi_4)\ ;\ 2 : y := 0\ ;\ 3 : \textbf{if}\ x \geq 2\ \textbf{then}\ 4 : \textbf{while}\ y < 3\ \textbf{do}\ 5 : y := A\ \textbf{else}\ 6 : \textbf{skip}$$

Then $G_4 = \mathsf{T}(P_4)$ will be as depicted in Figure 9(right), but can be simplified by compression of edges from nodes labeled **skip** which will result in a pCFG isomorphic to the one given in Figure 1(right).

For a given pCFG G, there may not exist a structured imperative probabilistic program P such that G is isomorphic to a simplification of $\mathsf{T}(P)$.

7.3 Adequacy Result for the Two Semantics

To motivate how the semantics in Section 7.1 relates to the semantics in Section 4, consider S_2 as defined in Example 7.1 where we saw that if $F = [\![S_2]\!]F'$ for some F' then for all stores s we have

$$F(s) = \frac{1}{16}\left(F'(s[x \mapsto 2][y \mapsto 3]) + F'(s[x \mapsto 3][y \mapsto 2]) + F'(s[x \mapsto 3][y \mapsto 3])\right).$$

We saw in Example 7.6 that $\mathsf{T}(S_2)$ is the pCFG G_2 depicted in Figure 9(left), except that node 3′ is unlabeled. For that pCFG, it is not hard to see (cf. Example 2.2) that if $D' = \omega^{(1,3')}(D)$ (with ω given by Definition 4.26) for some D with $\sum D = 1$ then

$$\begin{aligned} D'(s) &= \frac{1}{16} \text{ if } s \in \{\{x \mapsto 2,\ y \mapsto 3\},\ \{x \mapsto 3,\ y \mapsto 2\},\ \{x \mapsto 3,\ y \mapsto 3\}\} \\ D'(s) &= 0 \text{ otherwise.} \end{aligned}$$

We now observe that (with the first equality due to $F(s)$ not depending on s since $\mathsf{U} = \{x, y\}$)

$$\begin{aligned} \sum_{s \in \mathsf{S_U}} F(s)D(s) &= F(s) \sum_{s \in \mathsf{S_U}} D(s) = F(s) \\ &= \frac{1}{16}F'\{x \mapsto 2,\ y \mapsto 3\} + \frac{1}{16}F'\{x \mapsto 3,\ y \mapsto 2\} + \frac{1}{16}F'\{x \mapsto 3,\ y \mapsto 3\} \\ &= \sum_{s \in \mathsf{S_U}} F'(s)D'(s). \end{aligned}$$

And this is indeed an instance of the general result (stated on an abstract level by Kozen [1985]) relating the two semantics:

Theorem 7.8 (adequacy). *Let P be a structured imperative probabilistic program, let $G = \mathsf{T}(P)$, and let ω be the meaning of G.*

Let S be a structured imperative probabilistic statement that is part of P. Thus $\mathsf{T}(S)$ will be a sub-pCFG of G; let $v = \mathtt{start}(\mathsf{T}(S))$ and $v' = \mathsf{end}(\mathsf{T}(S))$.

For all distributions D, D' and expectation functions F, F', if $[\![S]\!]F' = F$ and $\omega^{(v,v')}(D) = D'$ then

$$\sum_{s \in \mathsf{S_U}} F(s)D(s) = \sum_{s \in \mathsf{S_U}} F'(s)D'(s). \tag{8}$$

This result shows that we do not lose any information by using the semantics in Section 4, in that for any structured probabilistic statement S, and any expectation function F', we can retrieve $F = [\![S]\!]F'$ from ω: for given $s_0 \in \mathsf{S_U}$, define D_0 such that $D_0(s_0) = 1$ but $D_0(s) = 0$ otherwise; with $D_0' = \omega^{(v,v')}(D_0)$ we then have

$$F(s_0) = \sum_{s \in \mathsf{S_U}} F(s)D_0(s) = \sum_{s \in \mathsf{S_U}} F'(s)D_0'(s).$$

Similarly, the new semantics can be retrieved from the old: for given $s_0 \in \mathsf{S_U}$, define F' such that $F'(s_0) = 1$ but $F'(s) = 0$ otherwise; with $F = [\![S]\!]F'$ we then have

$$\omega^{(v,v')}(D)(s_0) = \sum_{s \in \mathsf{S_U}} F'(s) \cdot \omega^{(v,v')}(D)(s) = \sum_{s \in \mathsf{S_U}} F(s) \cdot D(s).$$

Let us explore some special cases of Theorem 7.8:

- If $D' = 0$ and $D(s_0) > 0$ for some $s_0 \in \mathsf{S_U}$ then from Equation (8) we can infer $F(s_0) = 0$. This make sense, since for such s_0 the program (almost) never returns anything.
- If D is concentrated on s_0 and D' is concentrated on s_0', with $D(s_0) = D'(s_0') = 1$, then from Equation (8) we can infer $F(s_0) = F'(s_0')$. This makes sense, since for such s_0 the program will (almost) always end up in store s_0'.

7.4 Slicing

We shall now show how to slice structured probabilistic programs. The approach is to translate such a program P into a pCFG for which we then find a slice (which should satisfy certain conditions mentioned in Section 5) which we finally use to slice P. More precisely, we have

Definition 7.9 (Slicing a Structured Probabilistic Program). Given a structured probabilistic program $P \equiv S$; **return**(x), and a subset L of the labels in P. Then the slice of P wrt L, written $\mathsf{slc}_L(P)$, is the structured probabilistic program $\mathsf{slc}_L(S)$; **return**(x) where the slicing function slc_L, defined in Figure 10, transforms structured probabilistic statements.

Note that we do not put any requirements on L, but for the slicing to be "correct", as expressed in Theorem 7.13, there must exists Q, Q_0 such that Q "extends" L in that $l \in L$ iff $\mathsf{l2v}(l) \in Q$, and (Q, Q_0) is a slicing pair (cf. Definition 5.6).

Example 7.10. Let us slice the structured probabilistic program P_4 from Example 7.7, where $\mathsf{T}(P_4)$ was depicted in Figure 9(right). Let us assume that A is $y + 1$; then the cycle-inducing node 4 is lossless (as shown in Section 4.5) and hence $(Q, \emptyset)$ is a slicing pair for $Q = \{1, 3'\}$. With $L = \{1\}$, and thus $l \in L$ iff $\mathsf{l2v}(l) \in Q$, the structured probabilistic program $\mathsf{slc}_L(P_4)$ will be

$$1 : x := \mathbf{random}(\psi_4)\ ;\ 2 : \mathbf{skip}\ ;\ 3 : \mathbf{skip}\ ;\ \mathbf{return}(x)$$

which is equivalent to $1 : x := \mathbf{random}(\psi_4)$; **return**(x) which in Section 2.5 was mentioned as the slice of P_4 when A is $y + 1$.

$$
\begin{array}{rcll}
\mathsf{slc}_L(S_1\ ;\ S_2) & = & \mathsf{slc}_L(S_1)\ ;\ \mathsf{slc}_L(S_2) & \\
\mathsf{slc}_L(l : \_) & = & l : \mathbf{skip} & \text{if } l \notin L \\
\mathsf{slc}_L(l : \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2) & = & l : \mathbf{if}\ B\ \mathbf{then}\ \mathsf{slc}_L(S_1)\ \mathbf{else}\ \mathsf{slc}_L(S_2) & \text{if } l \in L \\
\mathsf{slc}_L(l : \mathbf{while}\ B\ \mathbf{do}\ S_1) & = & l : \mathbf{while}\ B\ \mathbf{do}\ \mathsf{slc}_L(S_1) & \text{if } l \in L \\
\mathsf{slc}_L(S) & = & S & \text{otherwise}
\end{array}
$$

Fig. 10. The function slc_L slices a structured probabilistic statement S wrt. labels L.

Lemma 7.11. *For all structured probabilistic statements S, and all label sets L:* $\mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S))) = \mathtt{start}(\mathsf{T}(S))$ *and* $\mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S))) = \mathsf{end}(\mathsf{T}(S))$

Proof. An easy induction in S, using Lemma 7.4. □

For a structured probabilistic program, translating its slice has the same meaning as slicing its translation:

Lemma 7.12. *Let a structured probabilistic program P be given, and let V be the nodes in $G = \mathsf{T}(P)$. For a given subset Q of V, let $L = \{l \mid \mathsf{l2v}(l) \in Q\}$, and let $P_L = \mathsf{slc}_L(P)$ and $G_L = \mathsf{T}(P_L)$.*

Let ϕ be the meaning of the slice Q in G, that is (cf. Definition 4.27) $\phi = lim_{k\to\infty}\, \phi_k$ where $\phi_k = \mathsf{H}^k_Q(0)$ with H defined as in Figure 2 for the graph G.

Let ω be the meaning of G_L, that is (cf. Definition 4.26) $\omega = lim_{k\to\infty}\, \omega_k$ where $\omega_k = \mathsf{H}^k_{\mathsf{V}_L}(0)$ with H defined as in Figure 2 for the graph G_L, and with V_L the nodes in G_L.

Then for all S that are substatements of P: with $v = \mathtt{start}(\mathsf{T}(S))$ and $v' = \mathsf{end}(\mathsf{T}(S))$, and by Lemma 7.11 thus also $v = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S)))$ and $v' = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S)))$, we have $\phi^{(v,v')} = \omega^{(v,v')}$.

7.5 Correctness of Slicing Structured Programs

For a structured probabilistic program, slicing as defined in Section 7.4 does preserve the *normalized* semantics (Definition 7.2):

Theorem 7.13. *Given a structured program $P \equiv S\ ;\ \mathbf{return}(x)$ such that $[\![P]\!]$ is well-defined. Assume that (Q, Q_0) is a slicing pair (cf. Definition 5.6) on $\mathsf{T}(P)$, and let $L = \{l \mid \mathsf{l2v}(l) \in Q\}$. Then $[\![\mathsf{slc}_L(P)]\!] = [\![P]\!]$.*

(In the case when $[\![P]\!]$ is not well-defined, it may happen that $[\![\mathsf{slc}_L(P)]\!]$ is well-defined, as when P is given by $1 : x := 1\ ;\ 2 : \mathbf{observe}(\mathit{false})\ ;\ \mathbf{return}(x)$ in which case there will exist a slicing pair (Q, Q_0) such that $L = \{1\}$ and thus $\mathsf{slc}_L(P)$ is $1 : x := 1\ ;\ 2 : \mathbf{skip}\ ;\ \mathbf{return}(x)$.)

Proof. Let $G = \mathsf{T}(P)$, $P_L = \mathsf{slc}_L(P) = S_L\ ;\ \mathbf{return}(x)$ where $S_L = \mathsf{slc}_L(S)$, and $G_L = \mathsf{T}(P_L)$. By Lemma 7.11, there exists v, v' such that $v = \mathtt{start}(G) = \mathtt{start}(G_L)$ and $v' = \mathsf{end}(G) = \mathsf{end}(G_L)$; here $v' \in Q$ is labeled $\mathbf{return}(x)$ and thus $rv_Q(v') = \{x\}$.

Let ω be the meaning of G, that is (cf. Definition 4.26) $\omega = \mathit{fix}(\mathsf{H}_{\mathsf{V}})$ with H defined as in Figure 2 for the graph G, and with V the nodes in G.

Let ϕ be the meaning of the slice Q in G, that is (cf. Definition 4.27) $\phi = \mathit{fix}(\mathsf{H}_Q)$ with H defined as in Figure 2 for the graph G.

Let ω_L be the meaning of G_L, that is (cf. Definition 4.26) $\omega = \mathit{fix}(\mathsf{H}_{\mathsf{V}_L})$ with H defined as in Figure 2 for the graph G_L, and with V_L the nodes in G_L.

By Lemma 7.12,

$$\phi^{(v,v')} = {\omega_L}^{(v,v')}. \tag{9}$$

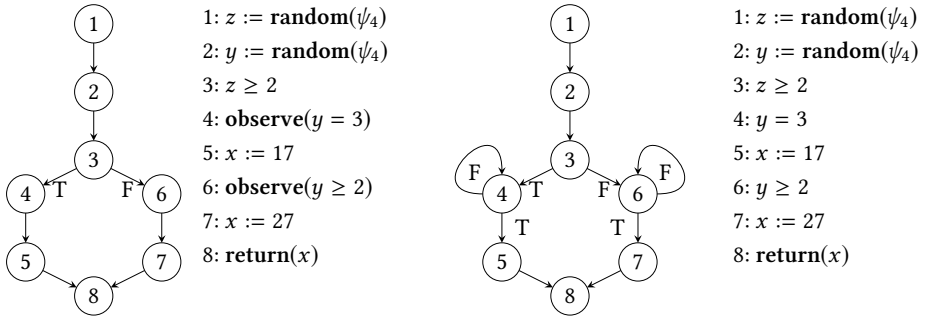

Fig. 11. Examples showing how (local) distribution may be lost due to conditioning and/or non-termination.

Let $\perp \in \mathsf{S_U}$ be the initial store mentioned in Definition 7.2, and let D_0 be such that $D_0(\perp) = 1$ but $D_0(s) = 0$ for all $s \in \mathsf{S_U} \setminus \{\perp\}$. Since (Q, Q_0) is a slicing pair we know that $Q \cap Q_0 = \emptyset$ and that Q and Q_0 are closed under data dependence; hence Lemma 3.13 tells us that $rv_{Q_1}(v) \cap rv_{Q_2}(v) = \emptyset$ which by Lemma 4.12 implies (since D_0 is concentrated) that $rv_{Q_1}(v)$ and $rv_{Q_2}(v)$ are independent in D_0. By Theorem 6.6, we therefore see that there exists a real number c with $0 \leq c \leq 1$ such that

$$\omega^{(v,v')}(D_0) \overset{rv_Q(v')}{=} c \cdot \phi^{(v,v')}(D_0).$$

Since $rv_Q(v') = \{x\}$, this amounts (cf. Definition 4.5) to

$$\forall q \in \mathbb{Z} : \ \omega^{(v,v')}(D_0)(\{x \mapsto q\}) = c \cdot \phi^{(v,v')}(D_0)(\{x \mapsto q\}). \tag{10}$$

Let F' be any expectation function satisfying $F'(s_1) = F'(s_2)$ whenever $s_1(x) = s_2(x)$; we can thus define $F'(q)$ such that $F'(q) = F'(s)$ whenever $s(x) = q$. The calculation in Table 1 shows that $([\![S]\!]F')(\perp) = c([\![S_L]\!]F')(\perp)$, in particular

$$\begin{aligned} ([\![S]\!]\lambda s.1)(\perp) &= c([\![S_L]\!]\lambda s.1)(\perp) \\ ([\![S]\!]\lambda s.s(x))(\perp) &= c([\![S_L]\!]\lambda s.s(x))(\perp) \end{aligned}$$

By the assumption that $[\![P]\!]$ is well-defined, we see from Equation (7) that $0 < [\![S]\!](\lambda s.1)(\perp) < \infty$ and then infer from the above that $c > 0$ and $0 < [\![S_L]\!](\lambda s.1)(\perp) < \infty$. Thus

$$[\![\mathsf{slc}_L(P)]\!] = \frac{[\![S_L]\!](\lambda s.s(x))(\perp)}{[\![S_L]\!](\lambda s.1)(\perp)} = \frac{c\,[\![S_L]\!](\lambda s.s(x))(\perp)}{c\,[\![S_L]\!](\lambda s.1)(\perp)} = \frac{[\![S]\!](\lambda s.s(x))(\perp)}{[\![S]\!](\lambda s.1)(\perp)} = [\![P]\!]$$

giving us the desired $[\![\mathsf{slc}_L(P)]\!] = [\![P]\!]$.

□

8 CHOICE OF SEMANTICS

There is a variety of language models for probabilistic programming, and a variety of approaches to giving their semantics (some of which are presented or mentioned in say [Olmedo et al. 2018] which is partly inspired by the equivalences between certain formulations proved in [Gretz et al. 2014]). And indeed, while we have designed our pCFG semantics so as to facilitate a correctness proof for slicing, alternative definitions are possible as we shall now discuss. As an illustrating example, consider the pCFG listed in Figure 11(left).

Let us first show how it is handled by our semantics. With D_0 a distribution with $\sum D_0 = 1$, the distribution $D_3 = \omega^{(1,3)}(D_0)$ satisfies $D_3(\{y \mapsto i,\ z \mapsto j\}) = \frac{1}{16}$ for all $i, j \in \{0, 1, 2, 3\}$ (but 0

$$
\begin{aligned}
& (\llbracket S \rrbracket F')(\bot) \\
= \quad & \sum_{s \in \mathsf{S_U}} (\llbracket S \rrbracket F')(s) \cdot D_0(s) \\
\overset{\text{Thm. 7.8}}{=} \quad & \sum_{s \in \mathsf{S_U}} F'(s) \cdot \omega^{(v,v')}(D_0)(s) \\
= \quad & \sum_{q \in \mathbb{Z}} \sum_{s \in \mathsf{S_U} \mid s(x)=q} F'(s) \cdot \omega^{(v,v')}(D_0)(s) \\
= \quad & \sum_{q \in \mathbb{Z}} F'(q) \cdot \sum_{s \in \mathsf{S_U} \mid s(x)=q} \omega^{(v,v')}(D_0)(s) \\
\overset{\text{Def. 4.2}}{=} \quad & \sum_{q \in \mathbb{Z}} F'(q) \cdot \omega^{(v,v')}(D_0)(\{x \mapsto q\}) \\
\overset{(10)}{=} \quad & \sum_{q \in \mathbb{Z}} F'(q) \cdot c \cdot \phi^{(v,v')}(D_0)(\{x \mapsto q\}) \\
= \quad & c \sum_{q \in \mathbb{Z}} F'(q) \cdot \sum_{s \in \mathsf{S_U} \mid s(x)=q} \phi^{(v,v')}(D_0)(s) \\
= \quad & c \sum_{q \in \mathbb{Z}} \sum_{s \in \mathsf{S_U} \mid s(x)=q} F'(s) \cdot \phi^{(v,v')}(D_0)(s) \\
= \quad & c \sum_{s \in \mathsf{S_U}} F'(s) \cdot \phi^{(v,v')}(D_0)(s) \\
\overset{(9)}{=} \quad & c \sum_{s \in \mathsf{S_U}} F'(s) \cdot {\omega_L}^{(v,v')}(D_0)(s) \\
\overset{\text{Thm. 7.8}}{=} \quad & c \sum_{s \in \mathsf{S_U}} (\llbracket S_L \rrbracket F')(s) \cdot D_0(s) \\
= \quad & c(\llbracket S_L \rrbracket F')(\bot)
\end{aligned}
$$

Table 1. Proving $(\llbracket S \rrbracket F')(\bot) = c(\llbracket S_L \rrbracket F')(\bot)$ for all F' such that $F'(s_1) = F'(s_2)$ when $s_1(x) = s_2(x)$.

otherwise). The distribution $D_5 = \omega^{(4,5)}(\mathsf{select}_{z \geq 2}(D_3))$ satisfies $D_5(\{y \mapsto i,\ z \mapsto j,\ x \mapsto k\}) = \frac{1}{16}$ when $i = 3$ and $j \in \{2, 3\}$ and $k = 17$ (but is 0 otherwise) and thus

$$D_5(\{x \mapsto 17\}) = \frac{2}{16},\ D_5(\{x \mapsto k\}) = 0 \text{ for } k \neq 17.$$

The distribution $D_7 = \omega^{(6,7)}(\mathsf{select}_{z<2}(D_3))$ satisfies $D_7(\{y \mapsto i,\ z \mapsto j,\ x \mapsto k\}) = \frac{1}{16}$ when $i \in \{2, 3\}$ and $j \in \{0, 1\}$ and $k = 27$ (but is 0 otherwise) and thus

$$D_7(\{x \mapsto 27\}) = \frac{4}{16},\ D_7(\{x \mapsto k\}) = 0 \text{ for } k \neq 27.$$

We conclude that with $D_8 = \omega^{(3,8)}(D_3) = \omega^{(1,8)}(D_0)$ we have

$$D_8(\{x \mapsto 17\}) = \frac{1}{8},\ D_8(\{x \mapsto 27\}) = \frac{1}{4},\ D_8(\{x \mapsto k\}) = 0 \text{ for } k \notin \{17, 27\}.$$

That is, the final distribution D_8 is a subprobability distribution, with $x = 27$ twice as likely as $x = 17$, but *not* a probability distribution.

It may seem more informative to "normalize" D_8 into a probability distribution D' with $D'(\{x \mapsto 17\}) = \frac{1}{3}$ and $D'(\{x \mapsto 27\}) = \frac{2}{3}$. (Recall that normalization is implicit in Definition 4.8 of probabilistic independence.) This would be perfectly valid (and allow us to use $c = 1$ when applying Theorem 6.6 to the whole pCFG) as long as we do it "globally". We cannot do it "locally" in each branch, since a conditioning is indeed supposed to make the branch in which it occurs less likely, as we mentioned for Example 2.3. Similarly, [Bichsel et al. 2018, p.148,paragraph 2] remarks that a given example "illustrates that it is not possible to condition parts of the program on there being no observation failure" in that "conditioning the two branches in isolation yields" an undesired result.

We have decided, however, to refrain from normalization, as it would add an extra layer (and thus extra proof obligations) to our semantics. On the other hand, in Section 7 we showed how our results apply to the slicing of structured imperative probabilistic programs and proved that slicing preserves the *normalized* semantics given by, *e.g.*, Gordon et al. [2014]; Hur et al. [2014] (who do indeed normalize only on top-level and not locally).

Now consider Figure 11(right) where the conditioning statements on the left have been replaced by a possibly non-terminating loop with the same effect (cf. Example 4.30). In this case, one *could* argue that $x = 17$ and $x = 27$ should be equally likely, something which might be achieved by doing *local* normalization.

To keep our semantic development reasonably simple, however, we have decided to consider non-termination as equivalent to failure of a conditioning, just as done by Gordon et al. [2014] and Hur et al. [2014]. This is in contrast to several approaches that do distinguish between non-termination and observation failure, such as the semantics presented by Olmedo et al. [2018], and by Bichsel et al. [2018] which considers ill-defined operations as a third kind of exception (also the slicer presented by Léchenet et al. [2016] distinguishes, in a non-probabilistic setting, between assertion failure and non-termination).

9 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 (whereas the size of Q_0 is irrelevant):

Definition 9.1. A slicing pair (Q, Q_0) is an *optimal* slicing pair iff whenever (Q', Q_0') is also a slicing pair then $Q \subseteq Q'$.

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 lossless is undecidable, as it is easy to see that the halting problem can be reduced to it (see [Kaminski and Katoen 2015] for more results about the decidability of termination in a probabilistic setting).
(2) While cycles in a graph can be detected in low polynomial time, it *may* be harder to detect the specific nodes that are "cycle-inducing" since this involves (cf. Definition 3.5) finding longest acyclic paths which is in general an NP-hard problem (as the Hamiltonian path problem can be reduced to it). However, 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 9.2 (essential nodes). A node v is essential iff
(1) v is an **observe** node, or
(2) v is cycle-inducing but not lossless.

We can now provide a computable version of Definition 5.6:

Definition 9.3. 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 end $\in Q$;
(2) Q, Q_0 are disjoint;
(3) $\mathsf{ESS} \subseteq Q \cup Q_0$.

If we find (Q, Q_0) satisfying Definition 9.3 then (Q, Q_0) will also satisfy Definition 5.6 (and hence Theorem 6.6, etc, will apply):

Proposition 9.4. *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 $\mathsf{ESS} \not\subseteq Q \cup Q_0$ 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 lossless, we may have $\mathsf{ESS} = \{4\}$, 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 [Chakarov and Sankaranarayanan 2013; Fioriti and Hermanns 2015; Monniaux 2001] for detecting that loops terminate with probability one, or techniques from [Kaminski et al. [n. d.]] for detecting a stronger property: that the expected run-time is finite.

If the pCFG in question is a translation of a structured imperative probabilistic program, cf. Section 7.2, 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 9.3. We shall measure its running time in terms of |V|, the number of nodes in the pCFG; we shall often write n instead of |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 9.1) the data dependences (in time $O(n^3)$);
(2) to construct an algorithm PNV? (Section 9.2) that (in linear time) checks if 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 9.3) 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. 9.4) an optimal slicing pair wrt. the given ESS.

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

9.1 Computing Data Dependences

Our algorithms use a boolean table DD^* such that $\mathsf{DD}^*(v, v')$ is true iff $v \overset{dd^*}{\rightarrow} v'$ where $\overset{dd^*}{\rightarrow}$ is the reflexive and transitive closure of $\overset{dd}{\rightarrow}$ defined in Definition 3.9.

```
PNV?(Q)
    F ← Q ∪ {end}
    C ← ∅
    for each v ∈ V \ F
        N[v] ← ⊥
    for each v ∈ F
        N[v] ← v
    while F ≠ ∅ ∧ C = ∅
        F' ← ∅
        for each edge v → v' with the source v ∉ Q and the target 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
```

Fig. 12. An algorithm to check if Q provides next visibles.

Lemma 9.5. *There exists an algorithm that computes* DD^* *in time* $O(n^3)$.

Proof. First, for each node v with $Def(v) \neq \emptyset$, we find the nodes v' with $v \xrightarrow{dd} v'$ 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 $\mathsf{DD}(v, v')$ is true iff $v \xrightarrow{dd} v'$. To compute 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 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 9.6. *There exists an algorithm* $\mathsf{DD}^{\mathrm{close}}$ *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* DD^* *is given,* $\mathsf{DD}^{\mathrm{close}}$ *runs in time* $O(n \cdot |Q_1|)$.

9.2 Checking for Next Visibles

A key ingredient in our approach is the function PNV?, presented in Figure 12, 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 $Q \cup \{\mathsf{end}\}$ to find (using the table N that approximates "next visible") the first node(s), if any, from which two nodes in $Q \cup \{\mathsf{end}\}$ are reachable without going through Q; such "conflict" nodes are stored in C and must be included in any superset providing next visibles.

Example 9.7. 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 $\{2, 3, 4\}$ and next $\{1\}$ and finally $\emptyset$.

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

Fig. 13. An algorithm that finds the least weak slice set containing $\hat{Q}$.

Example 9.8. Consider the program P_4 from Example 2.4, with pCFG depicted in Figure 3(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 9.9. *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.*

9.3 Computing Least Weak Slice Set

We are now ready to define, in Figure 13, 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 9.10. We shall continue Example 9.7 (which considers the program P_1 from Example 2.1). First observe that the non-trivial true entries of 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 $\emptyset$.

Example 9.11. We shall continue Example 9.8 (which considers the program P_4 from Example 2.4, with pCFG depicted in Figure 3(right)).

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

- 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 9.8) returns $\{3\}$. Since $1 \xrightarrow{dd} 3$ 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 9.12. *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 $Q \subseteq Q'$.*

```
BSP(ESS)
    W ← ESS
    for each 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 ← ∅
        for each v ∈ W
            if Q_v ∩ Q ≠ ∅
                W ← W \ {v}
                F ← F ∪ Q_v
    Q_0 ← ⋃_{v∈W} Q_v
    return (Q, Q_0)
```

Fig. 14. Finding an optimal slicing pair (BSP) wrt. given ESS.

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

9.4 Computing an Optimal Slicing Pair

We are now ready to define, in Figure 14, an algorithm BSP which given a set ESS that contains all essential nodes (for an implicitly given pCFG) returns an optimal slicing pair (Q, Q_0) wrt. that ESS. 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 9.13. We shall continue Examples 9.7 and 9.10 (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 9.10 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 9.14. We shall continue Examples 9.8 and 9.11 (which consider the program P_4 from Example 2.4, with pCFG depicted in Figure 3(right)). We know from Example 3.6 that node 4 is cycle-inducing but node 3 is not; in Section 4.5 we showed that node 4 is essential when $Lab(5)$ is an assignment $y := 1$ (as then $\omega^{(4,6)}$ is not lossless) 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 lossless).

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 9.11 we see that 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 ESS = {4}. BSP thus needs to run LWS on {4} and {6}, and from Example 9.11 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 an *optimal slicing pair* wrt. a given ESS is captured by the following result:

Theorem 9.15. *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).*

10 IMPROVING PRECISION

Section 9 presented an algorithm for computing the least slice satisfying Definition 9.3; such a slice will also satisfy Definition 5.6 and hence be semantically correct (as phrased in Theorem 6.6). 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.)

10.1 Improvement by Semantic Analysis

Already in Section 9 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 15, 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 pCFG on the right for which 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, 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".

10.2 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 [Hur et al. 2014], a simple syntactic transformation often suffices to get the benefits of that information, as we illustrate on the program from [Hur et al. 2014, Figure 4] whose pCFG (in slightly modified form) is depicted in Figure 16. In our setting, if (Q, Q_0)

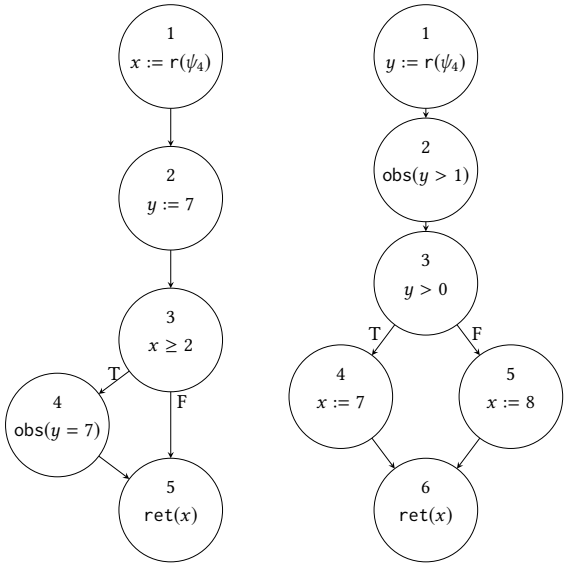

Fig. 15. A redundant **observe** node (left) and a potentially redundant branch (right).

with $18 \in Q$ is an optimal slicing pair, then Q will contain 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 [Hur et al. 2014, 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.

11 CONCLUSION AND RELATED WORK

We have developed a theory for the slicing of imperative probabilistic programs. We have used and extended techniques from the literature [Amtoft 2008; Ball and Horwitz 1993; Podgurski and Clarke 1990; Ranganath et al. 2007] on the slicing of deterministic imperative programs, represented as control-flow graphs. These frameworks, some of which have been partly verified by mechanical proof assistants [Blazy et al. 2015; Wasserrab 2010], were recently coalesced by Danicic et al. [2011] 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* slices rather than one. The technical foundations of our work rest on a novel semantics of probabilistic control-flow graphs (pCFGs).

We establish an adequacy result that shows that for pCFGs that are translations of programs in a structured imperative probabilistic language, our semantics is suitably related to that language's denotational semantics as formulated by first Kozen [1981] and later augmented by Gordon et al. [2014] (in particular to handle conditioning). As a consequence, our results on slicing of pCFGs

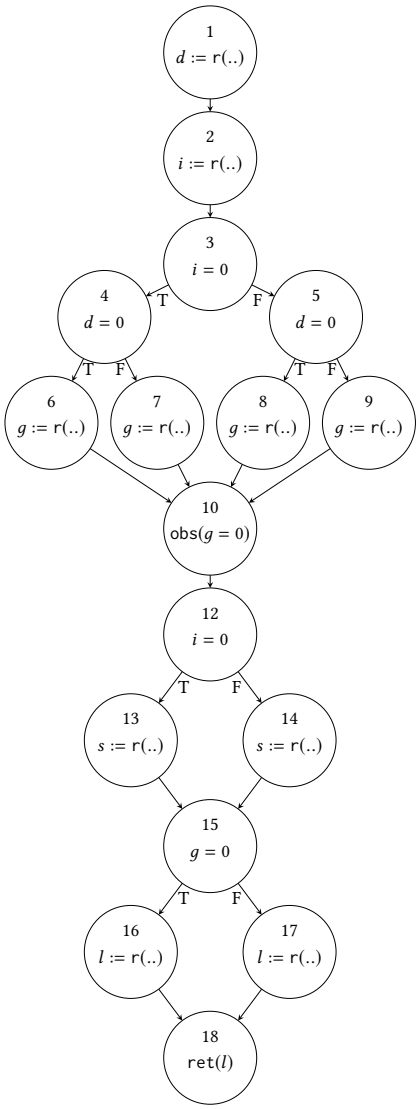

Fig. 16. The program from Figure 4 of Hur et al. (modified).

allow us to prove a result stating the correctness of slicing structured imperative probabilistic programs.

We were directly inspired by Hur et al. [2014] who point out the challenges involved in the slicing of probabilistic programs, and present an algorithm which constructs a semantically correct slice. That article 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 semantics presented by Gordon et al. [2014] which is based on expectation functions, 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 10 they incorporate powerful optimizations (which in some cases we can also obtain by means of simple syntactic transformations).

Much work remains. In future we plan to:

- investigate how our techniques can be used to statically analyze which sets of variables in a given probabilistic program are probabilistically independent of each other (a topic explored in, for example, [Bouissou et al. 2016]);
- investigate how to adapt our techniques to a semantics (such as [Bichsel et al. 2018; Olmedo et al. 2018]) that distinguishes between observation failure and non-termination;
- allow variables to contain reals rather than just integers (which will require us to employ measure theory, cf. the remark in Section 2.1);
- generalize the semantics to a trace semantics and use it to verify/calculate properties of probabilistic programs.

ACKNOWLEDGMENTS

We thank the anonymous TOPLAS reviewers as well as the FoSSaCS 2016 reviewers for their thorough and insightful feedback which was instrumental in improving the quality of the final manuscript. For encouragement, discussions, and comments on earlier versions we thank Sasa Misailovic, Aditya Nori, Sriram Rajamani, Daniel Ritchie, Sriram Sankaranarayanan, Gordon Stewart, and Hongseok Yang.

Banerjee's research was based on work supported by the US National Science Foundation (NSF), while working at the Foundation. Any opinion, finding, and conclusion or recommendation expressed in this article are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

Torben Amtoft. 2008. Slicing for Modern Program Structures: A Theory for Eliminating Irrelevant Loops. *Inform. Process. Lett.* 106, 2 (April 2008), 45–51. https://doi.org/10.1016/j.ipl.2007.10.002

Torben Amtoft and Anindya Banerjee. 2016. A Theory of Slicing for Probabilistic Control Flow Graphs. In *Foundations of Software Science and Computation Structures - 19th International Conference (FoSSaCS) (Lecture Notes in Computer Science)*, Bart Jacobs and Christof Löding (Eds.), Vol. 9634. Springer-Verlag, 180–196. https://doi.org/10.1007/978-3-662-49630-5_11

Thomas Ball and Susan Horwitz. 1993. Slicing Programs with Arbitrary Control Flow. In *Proceedings of the First International Workshop on Automated and Algorithmic Debugging (AADEBUG '93) (LNCS)*, Peter Fritzon (Ed.), Vol. 749. Springer-Verlag, London, UK, 206–222. https://doi.org/10.1007/BFb0019410

Richard W. Barraclough, David Binkley, Sebastian Danicic, Mark Harman, Robert M. Hierons, Ákos Kiss, Mike Laurence, and Lahcen Ouarbya. 2010. A trajectory-based strict semantics for program slicing. *Theoretical Computer Science* 411, 11 (2010), 1372 – 1386. https://doi.org/10.1016/j.tcs.2009.10.025

Benjamin Bichsel, Timon Gehr, and Martin Vechev. 2018. Fine-Grained Semantics for Probabilistic Programs. In *Programming Languages and Systems (ESOP 2018) (LNCS)*, Amal Ahmed (Ed.), Vol. 10801. Springer International Publishing, Cham, 145–185. https://doi.org/10.1007/978-3-319-89884-1_6

Sandrine Blazy, Andre Maroneze, and David Pichardie. 2015. Verified Validation of Program Slicing. In *Proceedings of the 2015 Conference on Certified Programs and Proofs (CPP '15)*. ACM, New York, NY, USA, 109–117. https://doi.org/10.1145/2676724.2693169

Olivier Bouissou, Eric Goubault, Sylvie Putot, Aleksandar Chakarov, and Sriram Sankaranarayanan. 2016. Uncertainty Propagation Using Probabilistic Affine Forms and Concentration of Measure Inequalities. In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016 (LNCS)*, Marsha Chechik and Jean-François Raskin (Eds.), Vol. 9636. Springer Berlin Heidelberg, 225–243. https://doi.org/10.1007/978-3-662-49674-9_13

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Lecture Notes in Computer Science, Vol. 8044. Springer Berlin Heidelberg, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Sebastian Danicic, Richard W. Barraclough, Mark Harman, John D. Howroyd, Ákos Kiss, and Michael R. Laurence. 2011. A Unifying Theory of Control Dependence and Its Application to Arbitrary Program Structures. *Theoretical Computer Science* 412, 49 (Nov. 2011), 6809–6842. https://doi.org/10.1016/j.tcs.2011.08.033

Luis María Ferrer Fioriti and Holger Hermanns. 2015. 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*. 489–501. https://doi.org/10.1145/2676726.2677001

Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic Programming. In *ICSE, Future of Software Engineering track (FOSE 2014)*, Matthew B. Dwyer and James Herbsleb (Eds.). ACM, New York, NY, USA, 167–181. https://doi.org/10.1145/2593882.2593900

Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2014. Operational Versus Weakest Pre-expectation Semantics for the Probabilistic Guarded Command Language. *Performance Evaluation* 73 (March 2014), 110–132. https://doi.org/10.1016/j.peva.2013.11.004

Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. 2014. Slicing Probabilistic Programs. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*, Keshav Pingali (Ed.). ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/2594291.2594303

Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost–Sure Termination. In *Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, Giuseppe F Italiano, Giovanni Pighizzini, and Donald T Sannella (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 307–318. https://doi.org/10.1007/978-3-662-48057-1_24

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and booktitle = ESOP'16 volume = 9632 series = LNCS year = 2016 pages = 364–389 doi = 10.1007/978-3-662-49498-1_15 publisher = Springer Verlag Federico Olmedo, title = Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. [n. d.].

Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. System Sci.* 22 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

Dexter Kozen. 1985. A probabilistic *PDL. J. Comput. Syst. Sci.* 30, 2 (April 1985), 162–178. https://doi.org/10.1016/0022-0000(85)90012-1

Jean-Christophe Léchenet, Nikolai Kosmatov, and Pascale Le Gall. 2016. Cut Branches Before Looking for Bugs: Sound Verification on Relaxed Slices. In *FASE'16 (LNCS)*, Vol. 9633. Springer Verlag, 179–196. https://doi.org/10.1007/978-3-662-49665-7_11

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A new proof rule for almost-sure termination. *PACMPL* 2, POPL (2018), 33:1–33:28. https://doi.org/10.1145/3158121

David Monniaux. 2001. An Abstract Analysis of the Probabilistic Termination of Programs. In *8th International Static Analysis Symposium (SAS'01) (Lecture Notes in Computer Science)*, Vol. 2126. Springer Verlag, 111–126. https://doi.org/10.1007/3-540-47764-0_7

Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. 2018. Conditioning in Probabilistic Programming. *ACM Trans. Program. Lang. Syst.* 40, 1, Article 4 (Jan. 2018), 50 pages. https://doi.org/10.1145/3156018

Prakash Panangaden. 2009. *Labelled Markov Processes.* Imperial College Press.

Andy Podgurski and Lori A. Clarke. 1990. A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance. *IEEE Transactions on Software Engineering* 16, 9 (Sept. 1990), 965–979. https://doi.org/10.1109/32.58784

Venkatesh Prasad Ranganath, Torben Amtoft, Anindya Banerjee, John Hatcliff, and Matthew B. Dwyer. 2007. A New Foundation for Control Dependence and Slicing for Modern Program Structures. *ACM Trans. Program. Lang. Syst. (TOPLAS)* 29, 5, Article 27 (Aug. 2007). https://doi.org/10.1145/1275497.1275502 A special issue with extended versions of selected papers from the 14th European Symposium on Programming (ESOP'05).

David A. Schmidt. 1986. *Denotational Semantics, a Methodology for Language Development.* Allyn and Bacon, Boston.

Frank Tip. 1995. A Survey of Program Slicing Techniques. *Journal of Programming Languages* 3 (1995), 121–189.

Daniel Wasserrab. 2010. *From Formal Semantics to Verified Slicing.* Ph.D. Dissertation. Karlsruher Institut für Technologie.

Mark Weiser. 1984. Program Slicing. *IEEE Transactions on Software Engineering* 10, 4 (July 1984), 352–357. https://doi.org/10.1109/TSE.1984.5010248

Glynn Winskel. 1993. *The Formal Semantics of Programming Languages.* MIT Press.

A DOMAIN THEORY

This section summarizes key aspects of domain theory, as presented in, *e.g.*, [Schmidt 1986; Winskel 1993].

A domain is a set D equipped with a partial order $\sqsubseteq$, that is $\sqsubseteq$ is reflexive, transitive, and antisymmetric. 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 $x_1, x_2, x_2, \ldots$ is a chain and by continuity thus also $f(x_1), f(x_2), f(x_2), \ldots$ is a chain).

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 $y \sqsubseteq z$. □

B MISCELLANEOUS PROOFS

We shall often let ";" denote function composition: $(f; g)(x) = g(f(x))$.

B.1 Proofs for Section 3

Lemma B.1. *For given v, let $\prec_v$ be an ordering among proper postdominators of v, by stipulating that $v_1 \prec_v v_2$ iff in all acyclic paths from v to* end*, v_1 occurs strictly before v_2. Then $\prec_v$ is transitive, antisymmetric, and total. Also, if $v_1 \prec_v v_2$ 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 $\prec_v$ 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

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

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 $v_1 \prec_v v_2$, 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 $v_1 \prec_v v_2$ 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 3.2: For any v with $v \neq \mathsf{end}$, there is a unique first proper postdominator of v.

Proof. It is obvious that v can have at most one first proper postdominator; we shall now argue that v does have one.

Since Lemma B.1 says that $\prec_v$ is a linear order among the proper postdominators of v, and since v has at least one proper postdominator (we can use end since $v \neq \mathsf{end}$), we infer that v has a proper postdominator v_1 that is least wrt. the $\prec_v$ order.

To establish that v_1 is indeed the first proper postdominator of v, let v' be another proper postdominator of v and let π be a path from v to v'; our task is to show that v_1 occurs in π. As $v_1 \prec_v v'$, and π can be extended into a path π' from v to end, we infer that in π' the first occurrence of v_1 is before the first occurrence of v', in particular that v_1 occurs in π. □

Lemma 3.4: 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 B.1 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.2. *Assume that v' is a proper postdominator of v, that with $v'' = FPPD(v)$ 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.3. *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 (not necessarily subprobability) distributions 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 $\epsilon > 0$ such that $lim_{k\to\infty} \sum_{s\in S} D_k(s) + \epsilon < \sum_{s\in S} D'(s)$. We infer that there exists a finite set S_0 with $S_0 \subseteq S$ such that $lim_{k\to\infty} \sum_{s\in S} D_k(s) + \epsilon < \sum_{s\in S_0} D'(s)$. For each $s \in S_0$ there exists K_s such that $D_k(s) > D'(s) - \epsilon/|S_0|$ 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 $k \geq K$, $D_k(s) + \epsilon/|S_0| > D'(s)$. But then we get the desired contradiction:

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

□

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

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

Proof. We have the calculation

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

where the third equality is justified as follows: for a given $s_0 \in \mathrm{S_U}$, exactly one $s' \in \mathrm{S}(R')$ 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 4.9: If R_1 and R_2 are independent in D, so are $\{x_1\}$ and $\{x_2\}$ for all $x_1 \in R_1, x_2 \in R_2$.

Proof. This follows since for all values v_1, v_2 we have the calculation (due to Lemma 4.3):

$$\begin{aligned}
D(\{x_1 \mapsto v_1\} \cdot D(\{x_2 \mapsto v_2\}) &= \sum_{s_1 \in \mathrm{S}(R_1) \mid s_1(x_1)=v_1} D(s_1) \sum_{s_2 \in \mathrm{S}(R_2) \mid s_2(x_2)=v_2} D(s_2) \\
&= \sum_{s_1 \in \mathrm{S}(R_1),\ s_2 \in \mathrm{S}(R_2) \mid s_1(x_1)=v_1,\ s_2(x_2)=v_2} D(s_1) D(s_2) \\
&= \sum_{s_1 \in \mathrm{S}(R_1),\ s_2 \in \mathrm{S}(R_2) \mid s_1(x_1)=v_1,\ s_2(x_2)=v_2} D(s_1 \oplus s_2) \sum D \\
&= \sum_{s \in \mathrm{S}(R_1 \cup R_2) \mid s(x_1)=v_1,\ s(x_2)=v_2} D(s) \sum D \\
&= D(\{x_1 \mapsto v_1,\ x_2 \mapsto v_2\}) \sum D
\end{aligned}$$

□

Lemma 4.12: If D is concentrated then R_1 and R_2 are independent in D for all disjoint R_1, R_2.

Proof. Let $s_0 \in \mathrm{S_U}$ be such that $D(s) = 0$ for all $s \in \mathrm{S_U}$ with $s \neq s_0$. Let $s_1 \in \mathrm{S}(R_1)$ and $s_2 \in R(s_2)$ be given. We split into two cases:

- First assume that $s_1 \overset{R_1}{=} s_0$ and $s_2 \overset{R_2}{=} s_0$. Then $D(s_1)$, $D(s_2)$, $D(s_1 \oplus s_2)$ and $\sum D$ all equal $D(s_0)$.
- Otherwise, for some $i \in \{1, 2\}$ we do not have $s_i \overset{R_i}{=} s_0$. Then $D(s_i)$ and $D(s_1 \oplus s_2)$ both equal 0.

In both cases, it is obvious that Equation (1) has been established. □

Lemma 4.13: Let $f \in \mathrm{D} \to \mathrm{D}$ be continuous and additive. Assume that for all D that are concentrated, f is lossless for D. Then f is lossless.

Proof. Let $s_1, s_2, \ldots$ be an enumeration of stores in $\mathrm{S_U}$. For given $D \in \mathrm{D}$, 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 lossless for each D_k)

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

□

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

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

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

□

Lemma 4.17 $\text{assign}_{x:=E}$ is continuous.

Proof. Obviously, $\text{assign}_{x:=E}$ is monotone. To show continuity, let $\{D_k \mid k\}$ be a chain. With $D'_k = \text{assign}_{x:=E}(D_k)$, 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 $D' = \text{assign}_{x:=E}(D)$. But this follows since by Lemma 4.1 for each s' we have the calculation

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

□

Lemma 4.18: Assume that $\text{rassign}_{x:=E}(D) = D'$ and that $x \notin R$. Then $D \overset{R}{=} D'$.

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

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

□

Lemma 4.20 $\text{rassign}_{x:=E}$ 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

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

□

Lemma 4.24 The functional H_X is continuous on $\mathsf{PD} \to (\mathsf{D} \to_c \mathsf{D})$.

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

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

and shall do so by induction in $\mathsf{LAP}(v, v')$, with a case analysis in Figure 2. 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 $v'' = FPPD(v)$ then we have the calculation

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

  where the second equality follows from the induction hypothesis, and the third equality from continuity of $\mathsf{H}_X(g_k)(v'', v')$ (Lemma 4.23).
- 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):

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

□

The following two lemmas are often convenient.

Lemma B.4. *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* $s' \in \mathsf{S}(R')$ *we then have*

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

Proof. This follows from the calculation

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

□

Lemma B.5. *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* $s' \in \mathsf{S}(R')$ *we then have*

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

Proof. This follows from the calculation

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

□

Next some results that prepare for the proof of Lemma 4.29.

Lemma B.6. *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.7. *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

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

□

Lemma B.8. *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.9. *With H_X as defined in Def. 4.22, we have:*

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

Proof. An easy induction in $\mathsf{LAP}(v, v')$, using Lemmas 4.14, 4.16 and 4.19. □

Lemma 4.29 Given a set X of nodes, let $h = fix(\mathsf{H}_X)$, and for each $k \geq 0$ let $h_k = \mathsf{H}_X^k(0)$. Then for each $(v, v') \in \mathsf{PD}$, $h^{(v,v')}$ is additive, multiplicative and non-increasing, as is each ${h_k}^{(v,v')}$, in particular (taking $X = \mathsf{V}$) each ${\omega_k}^{(v,v')}$.

Proof. The function 0 is obviously additive, multiplicative and non-increasing, so by Lemma B.9 we infer that for each $k \geq 0$, and for each $(v, v') \in \mathsf{PD}$, ${h_k}^{(v,v')}$ is additive, multiplicative and non-increasing. The claim about h now follows from Lemmas B.6, B.7, and B.8. □

Lemma 4.31 Given a pCFG, a slice Q, and $(v, v') \in \mathsf{PD}$ such that v stays outside Q until v'. We then have $\mathsf{H}_Q(h)^{(v,v')}(D) = D$ for all $D \in \mathsf{D}$ and all modification functions h.

Proof. We do induction in $\mathsf{LAP}(v, v')$. The claim is obvious if $v = v'$. If with $v'' = FPPD(v)$ we have $v' \neq v''$, we can (by Lemmas B.2 and 3.4) apply the induction hypothesis to (v, v'') and (v'', v'), to get the desired $\mathsf{H}_Q(h)^{(v,v')}(D) = \mathsf{H}_Q(h)^{(v'',v')}(\mathsf{H}_Q(h)^{(v,v'')}(D)) = \mathsf{H}_Q(h)^{(v'',v')}(D) = D$.
We are left with the case when $v' = FPPD(v)$, and since v stays outside Q until v' we see that $v \notin Q$ and thus clause (3a) in Figure 2 gives the desired $\mathsf{H}_Q(h)^{(v,v')}(D) = D$. □

We now prepare for the proof of Lemma 4.32.

Definition B.10. A function $f : \mathsf{D} \to \mathsf{D}$ is non-increasing wrt. R, a set of variables, if $f(D)(s) \leq D(s)$ holds for all $D \in \mathsf{D}$ and $s \in \mathsf{S}(R)$.

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

Lemma B.11. *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 3.15), and let $\omega_k = \mathsf{H}_{\mathsf{V}}^k(0)$ 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 $\omega_k = \mathsf{H}_V(\omega_{k-1})$. If $v' = v$, then ${\omega_k}^{(v,v')}(D) = D$ and the claim is trivial. Otherwise, let $v_0 = FPPD(v)$; if $v_0 \neq v'$ then Lemma B.2 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 $s \in \mathsf{S}(R)$ 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 $v' = FPPD(v)$. If $Lab(v)$ is of the form **observe**(B) the claim is trivial.

If $Lab(v)$ is of the form $x := E$ or of the form $x :=$ **random**(ψ) 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 4.15 and 4.18.

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 $D \in \mathsf{D}$, 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.3), either the outer induction hypothesis, or the inner induction hypothesis, to infer that for all $s \in \mathsf{S}(R)$, $D'_i(s) \leq D_i(s)$. For each $s \in \mathsf{S}(R)$ 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 4.32 With $(v, v') \in \mathsf{PD}$, assume that Q is closed under data dependence and that v stays outside Q until v' (by Lemma 3.15 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 $D \in \mathsf{D}$ with $\sum \omega^{(v,v')}(D) = \sum D$, for all $s \in \mathsf{S}(R)$, Lemma B.11 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)

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

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

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

which since $\omega^{(v,v')}(D)(s) \leq D(s)$ for all $s \in \mathsf{S}(R)$ is possible only if $\omega^{(v,v')}(D)(s) = D(s)$ for all $s \in \mathsf{S}(R)$, 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 $v \stackrel{dd}{\rightarrow} v'$ 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 $v \in Q \cup \{\mathsf{end}\}$ 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 B.1 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 $v_1 \in Q \cup \{\mathsf{end}\}$. Now consider a path π from v to $Q \cup \{\mathsf{end}\}$. We must show that π contains v_1, which is obvious if the path π is to $Q_1 \cup \{\mathsf{end}\}$. 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 $\omega^{(v_0, FPPD(v_0))}$ is lossless. 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 $v'' = FPPD(v)$, let us first assume that $v' \neq v''$. By Lemma 3.4 we have $\mathsf{LAP}(v, v'') < \mathsf{LAP}(v, v')$ and $\mathsf{LAP}(v'', v') < \mathsf{LAP}(v, v')$, and by Lemma B.2 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 $v' = FPPD(v)$. If v is labeled **skip** the claim is trivial; if v is labeled $x := E$ (or $x :=$ **random**(ψ)) then the claim follows from Lemma 4.16 (or Lemma 4.19). 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 lossless 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.3 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.12. *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 4.29 that ω is additive, multiplicative and non-increasing (and so is the function 0); the result thus follows from Lemma B.9. □

Similarly, we have (with Φ_k defined in Def. 6.7):

Lemma B.13. *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 $\gamma_k = \mathsf{H_V}(\gamma_{k-1})$ with $k > 0$, we do induction in $\mathsf{LAP}(v, v')$, with a case analysis on the definition of $\mathsf{H_V}$:

- If $v' = v$ then ${\gamma_k}^{(v,v')}(D) = D = \omega^{(v,v')}(D)$;
- If with $v'' = FPPD(v)$ we have $v' \neq v''$ then we know from Lemma B.2 that v stays outside $Q \cup Q_0$ until v'', and that v'' stays outside $Q \cup Q_0$ until v'; by Lemma 3.4 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 $v' = FPPD(v)$, 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

$$ \mathsf{H_V}(\gamma_{k-1})^{(v,v')}(D) = \mathsf{H_V}(\omega)^{(v,v')}(D). $$

  With $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, examining the definition of $\mathsf{H_V}$ 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.14. *Assume that v is a branching node, with B its condition, and v_1 its true-successor and v_2 its false-successor. Let $D \in \mathsf{D_{fin}}$ 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, and

(2) $\forall s_0 \in \mathsf{S}(R_0) : D(s_0) \sum D_i = D_i(s_0) \sum D$.

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

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

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

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

□

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

Definition B.15. We say that $h \in \mathsf{PD}\to\mathsf{D}\to_c \mathsf{D}$ ***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 $D \in \mathsf{D}_{\mathrm{fin}}$ 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 $s \in \mathsf{S}(R)$ 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 $s_0 \in \mathsf{S}(R_0)$ we have

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

Lemma B.16. *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 Lemma 3.13) to infer that for all D, with $D' = {\gamma_0}^{(v,v')}(D)$ we have $D' \stackrel{R \cup R_0}{=} D$, and by Lemma 4.6 thus also $D' \stackrel{R}{=} D$ and $D' \stackrel{R_0}{=} D$ and $D' \stackrel{\emptyset}{=} D$. That is, for $s \in \mathsf{S}(R)$ and $s_0 \in \mathsf{S}(R_0)$ 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.15, and also claim 1 since for $s \in \mathsf{S}(R)$ and $s_0 \in \mathsf{S}(R_0)$ 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 $\gamma_k = \mathsf{H}_V(\gamma_{k-1})$ 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 $D \in \mathsf{D}_{\mathrm{fin}}$ 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 $s \in \mathsf{S}(R)$ 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 $s_0 \in \mathsf{S}(R_0)$ 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.12 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 $v'' = FPPD(v)$, 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.12 we can assume that $D'' \neq 0$ (as otherwise $D' = 0$ which we have already considered). By Lemma 3.4 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.2 we see that v stays outside Q until v'' and v'' stays outside Q until v'. Inductively, we can thus assume that for $s \in \mathsf{S}(R)$ 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 $v' = FPPD(v)$, 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 3.15 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 $s_0 \in \mathsf{S}(R_0)$ we have $D'(s_0) \sum D = D(s_0) \sum D'$:

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

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 $s' \in \mathsf{S}(R')$ and $s_0 \in \mathsf{S}(R_0)$:

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

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 distribution transformer is lossless (Lemmas 4.16 and 4.19) so we get $\sum D' = \sum D$. We further infer by Lemma 4.32 that if $v \notin Q$ then $R' = R$ and $D' \overset{R}{=} D$ (as then v stays outside Q until v'); similarly, if $v \notin Q_0$ then $R'_0 = R_0$ and $D' \overset{R_0}{=} D$.

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 $s' \in \mathsf{S}(R')$ and $s_0 \in \mathsf{S}(R_0)$. 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 4.32) we get $R' = R$ and $R'_0 = R_0$ and $D' \overset{R \cup R_0}{=} D$. This establishes claim 1 since for $s \in \mathsf{S}(R)$ and $s_0 \in \mathsf{S}(R_0)$ 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 $s_0 \in \mathrm{S}(R_0)$. 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 4.15 and 4.18 (as $(R' \cup R_0) \cap Def(v) = \emptyset$) we get $D \overset{R' \cup R_0}{=} D'$. For $s' \in \mathrm{S}(R')$ and $s_0 \in \mathrm{S}(R_0)$ 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

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

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 3.11) $R = (R' \setminus \{x\}) \cup fv(E)$ and from our case assumptions also $x \in R'$ and $x \notin R_0$. Let us now consider $s' \in \mathrm{S}(R')$ and $s_0 \in \mathrm{S}(R_0)$; 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:

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

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 $s' \in \mathrm{S}(R')$ and $s_0 \in \mathrm{S}(R_0)$: 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.5:

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

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 3.15). 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 $s \in \mathsf{S}(R)$ and $s_0 \in \mathsf{S}(R_0)$ 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 3.12) $fv(B) \subseteq R = rv_Q(v)$, and as $v \notin Q_0$ we see (by Lemma 5.4) that v stays outside Q_0 until v' so that (by Lemma 3.15) $R'_0 = R_0$. 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 3.12) $R_1 \subseteq R$ and $R_2 \subseteq R$.

By Lemma B.14, we see that

$$\forall i \in \{1, 2\} : R_i \text{ and } R_0 \text{ are independent in } D_i \tag{11}$$

$$\forall i \in \{1, 2\} : D(s_0) \sum D_i = D_i(s_0) \sum D \text{ for all } s_0 \in \mathsf{S}(R_0). \tag{12}$$

Given line (11), and the fact that v_i stays outside Q_0 until v', we can infer that

$$\forall i \in \{1, 2\} : R' \text{ and } R_0 \text{ are independent in } D'_i \tag{13}$$

$$\forall i \in \{1, 2\} : D_i(s_0) \sum D'_i = D'_i(s_0) \sum D_i \text{ for all } s_0 \in \mathsf{S}(R_0) \tag{14}$$

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

$$D(s_0) \sum D' = D'(s_0) \sum D \text{ for all } s_0 \in \mathsf{S}(R_0) \tag{15}$$

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

$$\begin{array}{rcl} D'(s_0)\sum D & = & (D'_1(s_0)+D'_2(s_0))\sum D \\ \text{by (14)} & = & \left(D_1(s_0)\frac{\sum D'_1}{\sum D_1}+D_2(s_0)\frac{\sum D'_2}{\sum D_2}\right)\sum D \\ \text{by (12)} & = & D(s_0)\sum D'_1+D(s_0)\sum D'_2 \;=D(s_0)\sum D' \end{array}$$

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.12) and $D = D_2$ and $D' = D'_2$ in which case the claim follows directly from line (14).

From line (15) 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 line (13); similarly if $D'_2 = 0$. Otherwise, in which case also $\sum D_1 > 0$ and $\sum D_2 > 0$, for $s' \in \mathsf{S}(R')$ and $s_0 \in \mathsf{S}(R_0)$ we have

$$\begin{array}{rcl} D'(s'\oplus s_0)\sum D' & = & (D'_1(s'\oplus s_0)+D'_2(s'\oplus s_0))\sum D' \\ \text{by (13)} & = & D'_1(s')D'_1(s_0)\frac{\sum D'}{\sum D'_1}+D'_2(s')D'_2(s_0)\frac{\sum D'}{\sum D'_2} \\ \text{by (14)} & = & D'_1(s')D_1(s_0)\frac{\sum D'}{\sum D_1}+D'_2(s')D_2(s_0)\frac{\sum D'}{\sum D_2} \\ \text{by (12)} & = & D'_1(s')D(s_0)\frac{\sum D'}{\sum D}+D'_2(s')D(s_0)\frac{\sum D'}{\sum D} \\ & = & D'(s')D(s_0)\frac{\sum D'}{\sum D} \\ \text{by (15)} & = & D'(s')D'(s_0) \end{array}$$

□

Lemma 6.5 [rephrased using Definition B.15]: 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.16, 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

$$\begin{array}{rcl} D'(s_1\oplus s_2)\sum D' & = & (lim_{k\to\infty}\,D_k(s_1\oplus s_2))\sum lim_{k\to\infty}\,D_k \\ \text{(Lemma 4.1)} & = & (lim_{k\to\infty}\,D_k(s_1\oplus s_2))\,(lim_{k\to\infty}\sum D_k) = lim_{k\to\infty}\,(D_k(s_1\oplus s_2)\sum D_k) \\ \text{(Lemma B.16)} & = & lim_{k\to\infty}\,(D_k(s_1)D_k(s_2)) = D'(s_1)D'(s_2) \end{array}$$

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

$$\begin{array}{rcl} D(s)\sum D' & = & D(s)\sum lim_{k\to\infty}\,D_k \\ \text{(Lemma 4.1)} & = & D(s)lim_{k\to\infty}\sum D_k = lim_{k\to\infty}\,(D(s)\sum D_k) \\ \text{(Lemma B.16)} & = & lim_{k\to\infty}\,(D_k(s)\sum D) = D'(s)\sum D. \end{array}$$

□

Lemma 6.9 For a given pCFG, let (Q, Q_0) be a slicing pair. For all $k \geq 0$, all $(v, v') \in \mathrm{PD}$ with $R = rv_Q(v)$ and $R' = rv_Q(v')$ and $R_0 = rv_{Q_0}(v)$, all $D \in \mathsf{D}_{\mathrm{fin}}$ such that R and R_0 are independent in D, and all $\Delta \in \mathsf{D}_{\mathrm{fin}}$ 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 $\mathrm{LAP}(v, v')$.

Let us first (for all k) consider the case where $\mathbf{v}$ **stays outside** $\mathbf{Q \cup Q_0}$ **until** $\mathbf{v'}$. 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 6.7 if $k = 0$, and by Lemma 4.31 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 $\gamma_k = \mathsf{H}_V(\gamma_{k-1})$ and $\Phi_k = \mathsf{H}_Q(\Phi_{k-1})$, 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** $\mathbf{v''} = \mathbf{FPPD(v)}$. Then, by Figure 2, $\gamma_k{}^{(v,v')} = \gamma_k{}^{(v,v'')} ; \gamma_k{}^{(v'',v')}$ 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 $\mathrm{LAP}(v, v'') < \mathrm{LAP}(v, v')$ 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.16 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.13) 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 $v' = FPPD(v)$ (and $k > 0$) and v does not stay outside $Q \cup Q_0$ until v'. By Lemma 5.4, we infer $v \in Q$ or $v \in Q_0$; we now consider each of these possibilities.

Assume $\mathbf{v \in Q_0}$ **(and** $\mathbf{k > 0}$ **and** $\mathbf{v' = FPPD(v)}$**).** 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.16 we see that

$$D(s) \sum D' = D'(s) \sum D \text{ for all } s \in \mathsf{S}(R). \tag{16}$$

Since $v \notin Q$, and $\Phi_k = \mathsf{H}_Q(\Phi_{k-1})$ as $k > 0$, we see from clause (3a) in Figure 2 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 $s \in \mathsf{S}(R)$:

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

If $D = 0$ and hence $D' = 0$ then this is obvious. Otherwise, Definition 6.10 stipulates

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

and equation (16) yields the claim.

We shall finally consider the case $\mathbf{v} \in \mathbf{Q}$ **(and** $\mathbf{k} > \mathbf{0}$ **and** $\mathbf{v}' = \mathbf{FPPD(v)}$**).** 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 $s' \in \mathsf{S}(R')$. We need a case analysis on the label of v, where the case **skip** is trivial.

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

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

If $\mathbf{v}$ **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 3.12) $R_1 \subseteq R$ 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.14),
- that $D_i \stackrel{R_i}{=} \Delta_i$ (by Lemma 4.6 from $D_i \stackrel{R}{=} \Delta_i$ which holds as for $s \in \mathsf{S}(R)$ 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 $s \in \mathsf{S}(R')$ 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 $\mathbf{v}$ **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 4.15 (4.18) we get $D \stackrel{R'}{=} D'$ and $\Delta \stackrel{R'}{=} \Delta'$. But this implies the desired $D' \stackrel{R'}{=} \Delta'$ since for $s' \in \mathsf{S}(R')$ 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 3.11) $R = R'' \cup fv(E)$ with $R'' = R' \setminus \{x\}$, so given $s' \in \mathsf{S}(R')$ we can use Lemma B.4 twice to give us the

desired

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

We next consider the case when $Lab(v) = x := \mathbf{random}(\psi)$. Then $R = R' \setminus \{x\}$, so given $s' \in \mathsf{S}(R')$ we can use Lemma B.5 twice to give us the desired

$$D'(s') = \psi(s'(x))\left(\sum_{s\in\mathsf{S}(R) \mid s \overset{R}{=} s'} D(s)\right) = \psi(s'(x))\left(\sum_{s\in\mathsf{S}(R) \mid s \overset{R}{=} s'} \Delta(s)\right) = \Delta'(s')$$

□

B.5 Proofs for Section 7

We know from clause (2) in Figure 2 that when $v'' = FPPD(v)$ we have the equation $\omega^{(v,v')} = \omega^{(v,v'')};\omega^{(v'',v')}$ (and similarly for ω_k). But to reason about the translation of structured programs, we need that equation to hold for all $(v, v''), (v'', v') \in \mathsf{PD}$.

Lemma B.17. *Given a pCFG, let* H *be given as in Definition 4.22. For all* $(v, v_1), (v_1, v_2) \in \mathsf{PD}$*, and for all* $h_0 \in \mathsf{PD} \to \mathsf{D} \to \mathsf{D}$*, with* $h = \mathsf{H}(h_0)$ *we have*

$$h^{(v,v_2)} = h^{(v,v_1)}; h^{(v_1,v_2)}.$$

Proof. The claim is by induction in $\mathsf{LAP}(v, v_1)$. If $v_1 = v$, the claim is obvious (as then $h^{(v,v_1)}$ is the identity); if $v_1 = FPPD(v)$, the equality follows from the definition of H.

So assume that with $v_0 = FPPD(v)$ we have $v_1 \neq v$ and $v_1 \neq v_0$. By Lemma 3.4, $\mathsf{LAP}(v_0, v_1) < \mathsf{LAP}(v, v_1)$. Inductively, we thus have $h^{(v_0,v_2)} = h^{(v_0,v_1)}; h^{(v_1,v_2)}$ which gives us the desired result:

$$h^{(v,v_2)} = h^{(v,v_0)}; h^{(v_0,v_2)} = h^{(v,v_0)}; (h^{(v_0,v_1)}; h^{(v_1,v_2)}) = h^{(v,v_1)}; h^{(v_1,v_2)}.$$

□

Lemma B.18. *Given a pCFG, let* ω *be as in Definition 4.26. For all* $(v, v_1), (v_1, v_2) \in \mathsf{PD}$*:*

$$\omega^{(v,v_2)} = \omega^{(v,v_1)}; \omega^{(v_1,v_2)}.$$

Proof. This follows from Lemma B.17 since $\omega = \mathsf{H}(\omega)$. □

Theorem 7.8 Let P be a structured probabilistic program, let $G = \mathsf{T}(P)$, and let ω be the meaning of G (cf. Definition 4.26).

Let S be a structured probabilistic statement that is part of P. Thus $\mathsf{T}(S)$ will be a sub-pCFG of G; let $v = \mathtt{start}(\mathsf{T}(S))$ and $v' = \mathsf{end}(\mathsf{T}(S))$.

For all distributions D, D' and expectation functions F, F', if $[\![S]\!]F' = F$ and $\omega^{(v,v')}(D) = D'$ then

$$\sum_{s\in\mathsf{S}_\mathsf{U}} F(s)D(s) = \sum_{s\in\mathsf{S}_\mathsf{U}} F'(s)D'(s).$$

Proof. We do structural induction in S, with a case analysis.

- The case with $S = \mathbf{skip}$ is trivial, as then $F = F'$ and $D' = D$.
- For the case $S = \mathbf{observe}(B)$, the claim follows from the calculation

$$\sum_{s\in\mathsf{S}_\mathsf{U}} F'(s)D'(s) = \sum_{s\in\mathsf{S}_\mathsf{U}} F'(s)\mathsf{select}_B(D)(s) = \sum_{s\in\mathsf{S}_\mathsf{U} \mid [\![B]\!]s} F'(s)D(s)$$
$$= \sum_{s\in\mathsf{S}_\mathsf{U}} F(s)D(s).$$

- For the case $S = x := E$, the claim follows from the calculation

$$\sum_{s' \in \mathsf{S_U}} F'(s')D'(s') = \sum_{s' \in \mathsf{S_U}} F'(s') \left(\sum_{s \in \mathsf{S_U} \mid s' = s[x \mapsto [\![E]\!]s]} D(s) \right) = \sum_{s' \in \mathsf{S_U},\ s \in \mathsf{S_U} \mid s' = s[x \mapsto [\![E]\!]s]} F'(s')D(s)$$

$$= \sum_{s \in \mathsf{S_U}} F'(s[x \mapsto [\![E]\!]s])D(s) = \sum_{s \in \mathsf{S_U}} F(s)D(s).$$

- For the case $S = x := \mathbf{random}(\psi)$, the claim follows from the calculation

$$\sum_{s' \in \mathsf{S_U}} F'(s')D'(s') = \sum_{s' \in \mathsf{S_U}} F'(s') \sum_{s \in \mathsf{S_U} \mid s' \stackrel{\mathsf{U} \setminus \{x\}}{=} s} \psi(s'(x))D(s)$$

$$= \sum_{s, s' \in \mathsf{S_U} \mid s' = s[x \mapsto s'(x)]} F'(s')\psi(s'(x))D(s) = \sum_{s \in \mathsf{S_U}} \sum_{z \in \mathbb{Z}} \sum_{s' \in \mathsf{S_U} \mid s' = s[x \mapsto z]} F'(s')\psi(s'(x))D(s)$$

$$= \sum_{s \in \mathsf{S_U}} \sum_{z \in \mathbb{Z}} F'(s[x \mapsto z])\psi(z)D(s) = \sum_{s \in \mathsf{S_U}} D(s) \sum_{z \in \mathbb{Z}} F'(s[x \mapsto z])\psi(z) = \sum_{s \in \mathsf{S_U}} D(s)F(s).$$

- For the case $S = S_1 \ ; \ S_2$, let $G_1 = \mathsf{T}(S_1)$ and $G_2 = \mathsf{T}(S_2)$, and let $v_1 = \mathtt{start}(G_1)$, $v_1' = \mathtt{end}(G_1)$, $v_2 = \mathtt{start}(G_2)$, and $v_2' = \mathtt{end}(G_2)$. By construction (Figure 7), with $G = \mathsf{T}(S)$ we have $v_1 = \mathtt{start}(G)$ and $v_2' = \mathtt{end}(G)$, and in G it is the case that v_1' postdominates v_1, that $v_2 = FPPD(v_1')$, and that v_2' postdominates v_2. Hence we infer, by Lemma B.18, that

$$\omega^{(v_1, v_2')} = \omega^{(v_1, v_1')}; \omega^{(v_1', v_2)}; \omega^{(v_2, v_2')}$$

  and since $\omega^{(v_1', v_2)}$ is the identity there exists D'' such that

$$D'' = \omega^{(v_1, v_1')}(D) \text{ and } D' = \omega^{(v_2, v_2')}(D'').$$

  From Figure 6 we see that that there exists F'' such that $F'' = [\![S_2]\!]F'$ and $F = [\![S_1]\!]F''$. By applying the induction hypothesis to first S_2 and next S_1, we get the desired

$$\sum_{s \in \mathsf{S_U}} F'(s)D'(s) = \sum_{s \in \mathsf{S_U}} F''(s)D''(s) = \sum_{s \in \mathsf{S_U}} F(s)D(s).$$

- For the case $S = l : \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2$, let $G_1 = \mathsf{T}(S_1)$ and $G_2 = \mathsf{T}(S_2)$, and let $v_1 = \mathtt{start}(G_1)$, $v_1' = \mathtt{end}(G_1)$, $v_2 = \mathtt{start}(G_2)$, and $v_2' = \mathtt{end}(G_2)$. Also, let $v = \mathsf{l2v}(l)$ and $v' = \mathsf{l2v}(l')$. By construction (Figure 7), with $G = \mathsf{T}(S)$ we see that $v = \mathtt{start}(G)$ which is a branching node with condition B and *true*-successor v_1 and *false*-successor v_2, and that $v' = \mathtt{end}(G)$ to which there are edges from v_1' and v_2' which both have label **skip**. We further see that in G it is the case that $v' = FPPD(v)$, that v_1' postdominates v_1, and that v_2' postdominates v_2.
  Hence, with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, and with $D_1' = \omega^{(v_1, v_1')}(D_1)$ and $D_2' = \omega^{(v_2, v_2')}(D_2)$, by Lemma B.18 we have the calculation

$$D' = \omega^{(v, v')}(D) = \omega^{(v_1, v')}(D_1) + \omega^{(v_2, v')}(D_2) = \omega^{(v_1, v_1')}(D_1) + \omega^{(v_2, v_2')}(D_2) = D_1' + D_2'.$$

  From Figure 6 we see that $F(s) = [\![S_1]\!](F')(s)$ if $[\![B]\!]s$ holds, and $F(s) = [\![S_2]\!](F')(s)$ otherwise.

By applying the induction hypothesis to S_1 and S_2, the claim now follows from the calculation

$$\begin{aligned}\sum_{s\in S_U} F'(s)D'(s) &= \sum_{s\in S_U} F'(s)D'_1(s) + \sum_{s\in S_U} F'(s)D'_2(s)\\ &= \sum_{s\in S_U} ([\![S_1]\!](F')(s)\cdot D_1(s)) + \sum_{s\in S_U} ([\![S_2]\!](F')(s)\cdot D_2(s))\\ &= \sum_{s\in S_U \mid [\![B]\!]s} F(s)D(s) + \sum_{s\in S_U \mid [\![\neg B]\!]s} F(s)D(s) = \sum_{s\in S_U} F(s)D(s).\end{aligned}$$

- For the case $S = l : \textbf{while}\ B\ \textbf{do}\ S_1$, let $G_1 = \mathsf{T}(S_1)$ and let $v_1 = \mathtt{start}(G_1)$ and $v'_1 = \mathsf{end}(G_1)$; also, let $v = \mathsf{l2v}(l)$ and $v' = \mathsf{l2v}(l')$. By construction (Figure 7), with $G = \mathsf{T}(S)$ we see that $v = \mathtt{start}(G)$ which is a branching node with condition B and *true*-successor v_1 and *false*-successor v' where $v' = \mathsf{end}(G)$, and that v'_1 is labeled **skip** and has an edge to v.
  We further see that in G it is the case that $v' = FPPD(v)$, and that v postdominates v'_1 which postdominates v_1; moreover, $\mathsf{LAP}(v, v') = 1 < \mathsf{LAP}(v_1, v')$. For each $k \geq 0$ we thus have the calculation

$$\begin{aligned}&{\omega_{k+1}}^{(v,v')}(D)\\ =\ &{\omega_{k+1}}^{(v',v')}(\mathsf{select}_{\neg B}(D)) + {\omega_k}^{(v_1,v')}(\mathsf{select}_B(D))\\ =\ &\mathsf{select}_{\neg B}(D) + {\omega_k}^{(v,v')}({\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D))).\end{aligned}$$

  where the first equality follows by clause (3e) in Figure 2, and where the second equality is obvious if $k = 0$ and otherwise follows by Lemma B.17 (since ${\omega_k}^{(v'_1,v)}$ is the identity).
  For our proof, it is convenient to define a chain $\{g_k \mid k\}$ of functions in $D \to_c D$ by stipulating

$$\begin{aligned}g_0(D) &= 0\\ g_{k+1}(D) &= \mathsf{select}_{\neg B}(D) + g_k(\omega^{(v_1,v'_1)}(\mathsf{select}_B(D))).\end{aligned}$$

  Observe that for all k, and all D, we have

$${\omega_k}^{(v,v')}(D) \leq g_k(D). \tag{17}$$

  This is trivial for $k = 0$, and for the inductive step we have

$$\begin{aligned}{\omega_{k+1}}^{(v,v')}(D) &= \mathsf{select}_{\neg B}(D) + {\omega_k}^{(v,v')}({\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D)))\\ &\leq \mathsf{select}_{\neg B}(D) + g_k(\omega^{(v_1,v'_1)}(\mathsf{select}_B(D)))\\ &= g_{k+1}(D).\end{aligned}$$

  For all k, and all D, we also have

$$g_k(D) \leq lim_{k\to\infty}\, {\omega_k}^{(v,v')}(D). \tag{18}$$

  For $k = 0$ this is obvious, and for the inductive step we have

$$\begin{aligned}g_{k+1}(D) &= \mathsf{select}_{\neg B}(D) + g_k(\omega^{(v_1,v'_1)}(\mathsf{select}_B(D)))\\ &= \mathsf{select}_{\neg B}(D) + g_k(lim_{k\to\infty}\, {\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D)))\\ &\leq \mathsf{select}_{\neg B}(D) + lim_{k\to\infty}\, {\omega_k}^{(v,v')}(lim_{k\to\infty}\, {\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D)))\\ &= \mathsf{select}_{\neg B}(D) + lim_{k\to\infty}\, ({\omega_k}^{(v,v')}({\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D))))\\ &= lim_{k\to\infty}\, (\mathsf{select}_{\neg B}(D) + {\omega_k}^{(v,v')}({\omega_k}^{(v_1,v'_1)}(\mathsf{select}_B(D))))\\ &= lim_{k\to\infty}\, {\omega_{k+1}}^{(v,v')}(D).\end{aligned}$$

From lines (17) and (18) we see that for all k, and all D, we have

$$\omega_k{}^{(v,v')}(D) \leq g_k(D) \leq lim_{k\to\infty}\, \omega_k{}^{(v,v')}(D)$$

from which we infer that

$$\text{for all } D : lim_{k\to\infty}\, g_k(D) = lim_{k\to\infty}\, \omega_k{}^{(v,v')}(D). \tag{19}$$

The virtue of working on g_k rather than on ω_k is that we can then prove the following result:

$$\text{for all } k \text{ and } D,\ \sum_{s\in S_U} F_k(s)\cdot D(s) = \sum_{s\in S_U} F'(s)\cdot g_k(D)(s). \tag{20}$$

where in Figure 6 we defined F_k as follows:

$$\begin{array}{rcl} F_0(s) & = & 0 \\ F_{k+1}(s) & = & [\![S_1]\!](F_k)(s) \text{ if } [\![B]\!]s \\ F_{k+1}(s) & = & F'(s) \text{ otherwise.} \end{array}$$

The proof of equation (20) is by induction in k, where the base case $k = 0$ is obvious as $F_0 = 0 = g_0(D)$. For the inductive step, we have the calculation

$$\begin{array}{rl} & \displaystyle\sum_{s\in S_U} F_{k+1}(s)\cdot D(s) \\ = & \displaystyle\sum_{s\in S_U \mid [\![B]\!]s} [\![S_1]\!](F_k)(s)\cdot D(s) + \sum_{s\in S_U \mid [\![\neg B]\!]s} F'(s)\cdot D(s) \\ = & \displaystyle\sum_{s\in S_U} [\![S_1]\!](F_k)(s)\cdot \mathsf{select}_B(D)(s) + \sum_{s\in S_U} F'(s)\cdot \mathsf{select}_{\neg B}(D)(s) \\ = & \displaystyle\sum_{s\in S_U} F_k(s)\cdot \omega^{(v_1,v_1')}(\mathsf{select}_B(D))(s) + \sum_{s\in S_U} F'(s)\cdot \mathsf{select}_{\neg B}(D)(s) \\ = & \displaystyle\sum_{s\in S_U} F'(s)\cdot g_k(\omega^{(v_1,v_1')}(\mathsf{select}_B(D)))(s) + \sum_{s\in S_U} F'(s)\cdot \mathsf{select}_{\neg B}(D)(s) \\ = & \displaystyle\sum_{s\in S_U} F'(s)\cdot \left(g_k(\omega^{(v_1,v_1')}(\mathsf{select}_B(D))) + \mathsf{select}_{\neg B}(D)\right)(s) \\ = & \displaystyle\sum_{s\in S_U} F'(s)\cdot g_{k+1}(D)(s) \end{array}$$

where the 3rd equality comes from applying the outer (structural) induction hypothesis to S_1 while the 4th equality comes from the inner induction hypothesis (in k).

Since $F(s) = lim_{k\to\infty} F_k(s)$ (by Figure 6), the desired claim now follows from

$$\begin{aligned}
\sum_{s\in S_U} F(s)D(s) &= \sum_{s\in S_U} lim_{k\to\infty} F_k(s)D(s)\\
\text{(Lemma 4.1)} &= lim_{k\to\infty} \sum_{s\in S_U} F_k(s)D(s)\\
\text{(by (20))} &= lim_{k\to\infty} \sum_{s\in S_U} F'(s)g_k(D)(s)\\
\text{(Lemma 4.1)} &= \sum_{s\in S_U} lim_{k\to\infty} F'(s)g_k(D)(s)\\
\text{(by (19))} &= \sum_{s\in S_U} F'(s) lim_{k\to\infty}\, {\omega_k}^{(v,v')}(D)\\
&= \sum_{s\in S_U} F'(s)D'(s).
\end{aligned}$$

This concludes the proof of Theorem 7.8.

□

Lemma 7.12 Let a structured probabilistic program P be given, and let V be the nodes in $G = \mathsf{T}(P)$. For a given subset Q of V, let $L = \{l \mid \mathsf{l2v}(l) \in Q\}$, and let $P_L = \mathsf{slc}_L(P)$ and $G_L = \mathsf{T}(P_L)$.

Let ϕ be the meaning of the slice Q in G, that is (cf. Definition 4.27) $\phi = lim_{k\to\infty} \phi_k$ where $\phi_k = \mathsf{H}^k_Q(0)$ with H defined as in Figure 2 for the graph G.

Let ω be the meaning of G_L, that is (cf. Definition 4.26) $\omega = lim_{k\to\infty} \omega_k$ where $\omega_k = \mathsf{H}^k_{V_L}(0)$ with H defined as in Figure 2 for the graph G_L, and with V_L the nodes in G_L.

Then for all S that are substatements of P: with $v = \mathtt{start}(\mathsf{T}(S))$ and $v' = \mathsf{end}(\mathsf{T}(S))$, and by Lemma 7.11 thus also $v = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S)))$ and $v' = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S)))$, we have ${\phi}^{(v,v')} = {\omega}^{(v,v')}$.

Proof. It is sufficient to prove ${\phi_k}^{(v,v')} = {\omega_k}^{(v,v')}$ for all $k \geq 0$, which we shall do by induction in k where the case $k = 0$ is obvious (as both sides are zero).

For $k > 0$, we do an inner structural induction on S, looking at the various cases in Figure 10.

- First assume $S = S_1 \ ; \ S_2$. Then $\mathsf{slc}_L(S_1 \ ; \ S_2) = \mathsf{slc}_L(S_1) \ ; \ \mathsf{slc}_L(S_2)$. By Lemma 7.11, there exists v_1, v_1', v_2, v_2' such that $v_1 = \mathtt{start}(\mathsf{T}(S_1)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S_1)))$, $v_1' = \mathsf{end}(\mathsf{T}(S_1)) = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S_1)))$, $v_2 = \mathtt{start}(\mathsf{T}(S_2)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S_2)))$, and $v_2' = \mathsf{end}(\mathsf{T}(S_2)) = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S_2)))$. Then, cf. the translation rules in Figure 7, $v_1 = \mathtt{start}(\mathsf{T}(S)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S))) = v$ and $v_2' = \mathsf{end}(\mathsf{T}(S)) = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S))) = v'$; also, in $\mathsf{T}(S)$ as well as in $\mathsf{T}(\mathsf{slc}_L(S))$, v_1' is labeled **skip** with an edge to v_2. By Lemma B.17, we have (since ${\phi_k}^{(v_1',v_2)}$ is the identity as is ${\omega_k}^{(v_1',v_2)}$)

$$\begin{aligned}
{\phi_k}^{(v,v')} &= {\phi_k}^{(v_1,v_1')}; {\phi_k}^{(v_2,v_2')}\\
{\omega_k}^{(v,v')} &= {\omega_k}^{(v_1,v_1')}; {\omega_k}^{(v_2,v_2')}.
\end{aligned}$$

Inductively on S_1 and S_2, we have

$${\phi_k}^{(v_1,v_1')} = {\omega_k}^{(v_1,v_1')} \text{ and } {\phi_k}^{(v_2,v_2')} = {\omega_k}^{(v_2,v_2')}.$$

But this shows the desired equality:

$${\phi_k}^{(v,v')} = {\phi_k}^{(v_1,v_1')}; {\phi_k}^{(v_2,v_2')} = {\omega_k}^{(v_1,v_1')}; {\omega_k}^{(v_2,v_2')} = {\omega_k}^{(v,v')}.$$

- Next assume S is of the form $l : \_$ with $l \notin L$, that is $\mathsf{l2v}(l) \notin Q$. Then $\mathsf{slc}_L(S) = l : \mathbf{skip}$. We know from Lemma 7.4 that $v = \mathtt{start}(\mathsf{T}(S)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S))) = \mathsf{l2v}(l)$ and that $v' = \mathsf{end}(\mathsf{T}(S)) = \mathsf{end}(\mathsf{T}(\mathsf{slc}_L(S))) = \mathsf{l2v}'(l)$.

In $\mathsf{T}(S)$, Lemma 7.4 also tells us that $v' = FPPD(v)$. Since $v \notin Q$, we see from clause (3a) in Figure 2 that ${\phi_k}^{(v,v')}$ is the identity.
In $\mathsf{T}(\mathsf{slc}_L(S))$, v is labeled **skip** and has an edge to v', so also ${\omega_k}^{(v,v')}$ is the identity.

- Next assume $S = l : \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2$ with $l \in L$, that is $\mathsf{l2v}(l) \in Q$. Then $\mathsf{slc}_L(l : \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2) = l : \textbf{if } B \textbf{ then } \mathsf{slc}_L(S_1) \textbf{ else } \mathsf{slc}_L(S_2)$. By Lemma 7.11, there exists v_1, v_1', v_2, v_2' such that $v_1 = \mathtt{start}(\mathsf{T}(S_1)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S_1)))$, $v_1' = \mathtt{end}(\mathsf{T}(S_1)) = \mathtt{end}(\mathsf{T}(\mathsf{slc}_L(S_1)))$, $v_2 = \mathtt{start}(\mathsf{T}(S_2)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S_2)))$, and $v_2' = \mathtt{end}(\mathsf{T}(S_2)) = \mathtt{end}(\mathsf{T}(\mathsf{slc}_L(S_2)))$. Then, cf. the translation rules in Figure 7, the following holds for $\mathsf{T}(S)$ and for $\mathsf{T}(\mathsf{slc}_L(S))$: the `start` node is $v = \mathsf{l2v}(l)$ which is a branching node with condition B and *true*-successor v_1 and *false*-successor v_2; the `end` node is $v' = \mathsf{l2v}'(l)$ to which there are edges from v_1' and from v_2' which are both labeled **skip**.
For a given D, with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, we now have

$$\begin{aligned} {\phi_k}^{(v,v')}(D) &= {\phi_k}^{(v_1,v')}(D_1) + {\phi_k}^{(v_2,v')}(D_2) = {\phi_k}^{(v_1,v_1')}(D_1) + {\phi_k}^{(v_2,v_2')}(D_2) \\ {\omega_k}^{(v,v')}(D) &= {\omega_k}^{(v_1,v')}(D_1) + {\omega_k}^{(v_2,v')}(D_2) = {\omega_k}^{(v_1,v_1')}(D_1) + {\omega_k}^{(v_2,v_2')}(D_2) \end{aligned}$$

where we have used that $\mathsf{LAP}(v_i, v') < \mathsf{LAP}(v, v')$ for $i = 1, 2$ holds in $\mathsf{T}(S)$ and in $\mathsf{T}(\mathsf{slc}_L(S))$, and also used Lemma B.17 together with the fact that ${\phi_k}^{(v_1',v')}$ is the identity as is ${\omega_k}^{(v_1',v')}$. The desired equality ${\phi_k}^{(v,v')}(D) = {\omega_k}^{(v,v')}(D)$ now follows since inductively on S_1, S_2 we have ${\phi_k}^{(v_1,v_1')} = {\omega_k}^{(v_1,v_1')}$ and ${\phi_k}^{(v_2,v_2')} = {\omega_k}^{(v_2,v_2')}$.
- Next assume $S = l : \textbf{while } B \textbf{ do } S_1$ with $l \in Q$, that is $\mathsf{l2v}(l) \in Q$. Then $\mathsf{slc}_L(l : \textbf{while } B \textbf{ do } S_1) = l : \textbf{while } B \textbf{ do } \mathsf{slc}_L(S_1)$. By Lemma 7.11, there exists v_1, v_1' such that $v_1 = \mathtt{start}(\mathsf{T}(S_1)) = \mathtt{start}(\mathsf{T}(\mathsf{slc}_L(S_1)))$ and $v_1' = \mathtt{end}(\mathsf{T}(S_1)) = \mathtt{end}(\mathsf{T}(\mathsf{slc}_L(S_1)))$. Then, cf. the translation rules in Figure 7, the following holds for $\mathsf{T}(S)$ and for $\mathsf{T}(\mathsf{slc}_L(S))$: the `start` node is $v = \mathsf{l2v}(l)$ and the `end` node is $v' = \mathsf{l2v}'(l)$, where v is a branching node with condition B and *true*-successor v_1 and *false*-successor v', and v_1' is labeled **skip** and has an edge to v; thus v postdominates v_1' which postdominates v_1, and $\mathsf{LAP}(v', v') < \mathsf{LAP}(v, v') = 1 < \mathsf{LAP}(v_1, v')$.
For a given D, with $D_1 = \mathsf{select}_B(D)$ and $D_2 = \mathsf{select}_{\neg B}(D)$, we now have (using Lemma B.17)

$$\begin{aligned} {\phi_k}^{(v,v')}(D) &= {\phi_{k-1}}^{(v_1,v')}(D_1) + {\phi_k}^{(v',v')}(D_2) \\ &= {\phi_{k-1}}^{(v,v')}({\phi_{k-1}}^{(v_1',v)}({\phi_{k-1}}^{(v_1,v_1')}(D_1))) + D_2 \\ {\omega_k}^{(v,v')}(D) &= {\omega_{k-1}}^{(v_1,v')}(D_1) + {\omega_k}^{(v',v')}(D_2) \\ &= {\omega_{k-1}}^{(v,v')}({\omega_{k-1}}^{(v_1',v)}({\omega_{k-1}}^{(v_1,v_1')}(D_1))) + D_2 \end{aligned}$$

The desired equality ${\phi_k}^{(v,v')}(D) = {\omega_k}^{(v,v')}(D)$ now follows since we have the equations

$$\begin{aligned} {\phi_{k-1}}^{(v_1,v_1')} &= {\omega_{k-1}}^{(v_1,v_1')} \\ {\phi_{k-1}}^{(v_1',v)} &= {\omega_{k-1}}^{(v_1',v)} \\ {\phi_{k-1}}^{(v,v')} &= {\omega_{k-1}}^{(v,v')} \end{aligned}$$

where the first is due to the outer induction hypothesis (applied to S_1), the second holds because each side is the identity when $k > 1$ and is 0 when $k = 1$, and the third is due to the outer induction hypothesis (applied to S).
- If none of the above cases hold then $\mathsf{slc}_L(S) = S$, and there exists l with $l \in Q$, that is $\mathsf{l2v}(l) \in Q$, such that S is of the form $l : \textbf{skip}$ or $l : x := E$ or $l : x := \textbf{random}(\psi)$ or $l : \textbf{observe}(B)$. In $\mathsf{T}(S) = \mathsf{T}(\mathsf{slc}_L(S))$, we see from Figure 7 that the `start` node v is $\mathsf{l2v}(l)$ (with $\mathsf{l2v}'(l)$ the `end` node v') and that there is only one edge, from v to v'. Since $v \in Q$, obviously ${\phi_k}^{(v,v')} = {\omega_k}^{(v,v')}$.

□

B.6 Proofs for Section 9

Lemma 9.6 There exists an algorithm $\mathsf{DD}^{\mathrm{close}}$ 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 DD^* is given, $\mathsf{DD}^{\mathrm{close}}$ 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 $\mathsf{DD}^*(v, v_1)$ 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 $\mathsf{DD}^*(v_1, v_1)$.

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 $v' \xrightarrow{dd} v$ 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 $\mathsf{DD}^*(v, v_1)$. By correctness of DD^* this means that $v \xrightarrow{dd^*} v_1$ which implies $v' \xrightarrow{dd^*} v_1$ and thus $\mathsf{DD}^*(v', v_1)$ holds. Hence also v' will be added to Q. □

Lemma 9.9 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 $q \in Q \cup \{\mathsf{end}\}$ is m-next from v iff there exists a path $v = v_1 \dots v_k = q$ 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 fact (1), if $q = N^1(v)$ then $q = v \in Q \cup \{\mathsf{end}\}$ and the trivial path v shows that q is 1-next from v; for fact (2), if $v \in F^1$ then $N^1(v) \neq \bot$ and no q can be 0-next from v; for fact (3), if q is 1-next from v then $q = v \in Q \cup \{\mathsf{end}\}$ 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 fact (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 fact (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 fact (3a) to infer $N^{m-1}(v) = q$.

For fact (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 fact (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 fact (3a).
- if q is not $(m-1)$-next from v, we can inductively use fact (1) to get $q \neq N^{m-1}(v)$, and also infer that there is a path $vv' \dots q$ 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 fact (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 $v \in \mathsf{V}$, all $q \in Q \cup \{\mathsf{end}\}$, 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 fact (3b) entails $v' \in F^m$ which is a contradiction. Thus $k \leq m$ and q is m-next from v so fact (3a) yields the claim.

Now let $v \in \mathsf{V}$ be given, to show that v has a next visible in Q. Since there is a path from v to end there will be a node $q \in Q \cup \{\mathsf{end}\}$ 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 $Q \cup \{\mathsf{end}\}$. 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 fact (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 $q, q' \in Q_1 \cup \{\mathsf{end}\}$ 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 9.12 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 $O(n^2)$.

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 $\mathsf{DD}^{\mathsf{close}}$.

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 $C = \mathsf{PNV?}(Q) \neq \emptyset$ so we know from Lemma 9.9 (since Q' provides next visibles) that $C \subseteq Q'$; hence we can apply Lemma 9.6 to infer (since Q' is closed under data dependence) $\mathsf{DD}^{\mathsf{close}}(Q, C) \subseteq Q'$ which yields the claim.

When the loop exits, with $C = \emptyset$, Lemma 9.9 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 9.9 we see that the total time spent in PNV? is in $O(n^2)$. And by Lemma 9.6 we infer that the total time spent in $\mathtt{DD}^{\mathtt{close}}$ is in $O(n^2)$. Hence the running time of LWS is in $O(n^2)$. □

Theorem 9.15 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 14, 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) $\mathsf{end} \in Q \cup F$
(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 invariants (2), (3), (7) we use Lemma 9.12.

Let us next show that the invariants are preserved by each iteration of the while loop. For invariant (1) this follows from Lemma 5.3, the repeated application of which and Lemma 9.12 gives invariant (2). Code inspection easily gives invariants (3), (4), (5). To show invariant (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 9.12) $v \in F$.

For invariant (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 9.12 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 $\mathsf{end} \in Q$, by invariants (1) and (3);
- Q_0 is a weak slice set, by Lemmas 9.12 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 9.5, we can compute $\mathtt{DD}^*$ in time $O(n^3)$. Then Lemma 9.12 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 $O(n^3)$. □

Received October 2017; revised July 2019; accepted October 2019