Linking Theories of Probabilistic Programming

He Jifeng(✉)

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
jifeng@sei.ecnu.edu.cn

Abstract. Formal methods advocate the critical role played by the algebraic approach in specification and implementation of programs. Traditionally, a top-down approach (with denotational model as its origin) links the algebra of programs with the denotational representation by establishment of the *soundness* and *completeness* of the algebra against the given model, while a bottom-up approach (a journey started from operational model) introduces a variety of bisimulations to establish the equivalence relation among programs. This paper follows up a new way presented in [1] to handle probabilistic programming. Our approach takes an algebra of probabilistic programs as its foundation, and then generates both denotational model and transition system, and explores the consistency among three types of representations.

1 Introduction

Formal methods [4,5,9] advocate the critical role played by the algebra of programs in specification and implementation of programs [6,8]. Study leads to the conclusion that both the top-down approach (with denotational model as its origin) [2,3,11] and the bottom-up approach (a journey started from operational model) [10] can meet in the middle.

This paper proposes a new roadmap for linking theories of probabilistic programming. Our new journey consists of the following steps:

Step 1: First we present an algebraic framework for a probabilistic programming language, which provides a set of algebraic laws for probabilistic programs, and introduces the concept of finite normal form. This paper then defines the refinement relation $\sqsubseteq_A$, and demonstrates how to reduce finite programs into finite normal form, and to transform an infinite program into an ascending chain of finite normal forms.

Step 2: Within the given program algebra we discuss the algebraic properties of the test operator $\mathcal{T}$ which composes test case tc and testing program P in

$$\mathcal{T}(tc,\, P) \;=_{df}\; (tc; P)$$

C. Jones et al. (Eds.): Zhou-Festschrift, LNCS 11180, pp. 186–210, 2018.
https://doi.org/10.1007/978-3-030-01461-2_10

where tc is represented by a total constant assignment

$$x, y, .., z := a,\ b,\ .., c$$

Based on the algebra of test, this paper identifies a probabilistic program P as a binary relation $[P]$ which relates the test case with the final observation

$$[P] =_{df} \{(tc,\ obs) \mid \mathcal{T}(tc,\ P) \sqsubseteq_A obs\}$$

and selects the set inclusion as the refinement relation $\sqsubseteq_{rel}$

$$P \sqsubseteq_{rel} Q =_{df} ([P] \supseteq [Q])$$

We establish the consistency of the denotational model against the algebraic framework by proof of

$$\sqsubseteq_{rel} = \sqsubseteq_A$$

Step 3: We propose an algebraic definition of the *consistency* of step relation of the transition system of programs such that any consistent transition system $(O,\ \sqsubseteq_O)$ satisfies

$$\sqsubseteq_O = \sqsubseteq_A$$

The paper is organised in the following way:

Section 2 is devoted to the algebraic framework of a probabilistic programming language with a collection of algebraic laws. Section 3 shows the normal forms of the finite and infinite probabilistic programs and proves that any probabilistic program can be converted into normal form with algebraic laws. Section 4 presents a test-based model, where each program is identified as a binary relation between test case and visible observation recorded during the execution of the test. It is shown that the refinement relation $\sqsubseteq_{rel}$ in the test model is equivalent to the algebraic refinement $\sqsubseteq_A$. Section 5 proposes a formal definition for the consistency of step relation of transition system against the algebra of programs. Moreover, it provides a transition system for the probabilistic programming language, and establishes its correctness. The paper ends with a short summary.

2 Probabilistic Programming Language

This section is going to construct an algebraic framework for the probabilistic programming language introduced in [12]

$$\begin{aligned} P ::= &\ \bot \mid \mathbf{skip} \mid var := exp \\ &\mid P \lhd bexp \rhd P \\ &\mid P\,;\,P \\ &\mid \oplus\{\underline{G}\} \\ &\mid \mu X \bullet P(X) \end{aligned}$$

$\oplus(\underline{G})$ denotes the probabilistic choice with a list of weighted alternatives $\underline{G}$ as its argument

$$\underline{G} ::= <> \mid \alpha(v) : P, \underline{G}$$

where the expression $\alpha(v)$ maps any given value of program variable v to a non-negative real number.

2.1 Probabilistic Choice

This section presents the algebraic properties of the probabilistic choice, which plays a crucial role in construction of normal form for the probabilistic language, and provides an elegant representation for finite observation. Later we are also going to use these algebraic laws to show that any finite program can be converted into a probabilistic choice.

The probabilistic choice is commutative.

($\oplus$-1) $\oplus\{\beta_1 : P_1, ..., \beta_m : P_m\} \;=_A \oplus\{\beta_{\rho(1)} : P_1, ..., \beta_{\rho(m)} : P_m\}$

where ρ is an arbitrary permutation of the list $< 1, ..., m >$.

The alternative $(1 - \beta) : \perp$ can be added to the probabilistic choice construct where $\beta \;=_{df}\; \Sigma_i \beta_i$.

($\oplus$-2) $\oplus\{\beta_1 : P_1, ..., \beta_m : P_m\} \;=_A \oplus\{\beta_1 : P_1, ..., \beta_k : P_k,\, (1 - \beta) : \perp\}$

The probabilistic choice operator becomes void whenever it contains an alternative with the probability 1.

($\oplus$-3) $\oplus\{1 : Q\} \;=_A\; Q$

Corollary. $\oplus\{\} \;=_A\; \perp$

Proof. From $\oplus$-2 and $\oplus$-3.

The next law shows how to eliminate the nested choices.

($\oplus$-4) Let $P \;=\; \oplus\{\beta_1 : P_1, ..., \beta_m : P_m\}$, then

$$\begin{aligned}&\oplus\{\alpha : P, \underline{G}\}\\ &\;=_A\; \oplus\{(\alpha\cdot\beta_1) : P_1, ..., (\alpha\cdot\beta_k) : P_k, \underline{G}\}\end{aligned}$$

The probabilistic choice operator distributes over sequential composition.

(⊕-5) $\oplus\{\beta_1 : P_1, ..., \beta_m : P_m\}\,;\, Q$
$=_A \quad \oplus\{\beta_1 : (P_1; Q), ..., \beta_k : (P_k; Q)\}$

Assignment distributes through the probabilistic choice.

(⊕-6) $(v := e)\,;\, \oplus\{\beta_1 : P_1, ..., \beta_m : P_m\}$

$=_A \quad \oplus\{\beta_1[e/v] : (v := e; P_1), ..., \beta_k[e/v] : (v := e; P_k)\}$

Two alternatives with the same guarded program can be merged.

(⊕-7) $\oplus\{\alpha : Q,\, \beta : Q,\, \underline{G}\} \;=_A\; \oplus\{(\alpha+\beta) : Q,\, \underline{G}\}$

Any alternative with zero probability can be removed.

(⊕-8) $\oplus\{0 : Q,\, \underline{G}\} \;=_A\; \oplus\{\underline{G}\}$

2.2 Conditional Choice

Conditional choice can be seen as a special form of probabilistic choice.

cond-1 $(P \lhd b \rhd Q) \;=_A\; \oplus\{(1 \lhd b \rhd 0) : P,\, (0 \lhd b \rhd 1) : Q\}$

From Law **cond**-1 and the laws of probabilistic choice presented in the previous section, we can derive the following set of well-known properties of conditional choice:

Theorem 2.1.
(1) $P \lhd b \rhd P \;=_A\; P$
(2) $P \lhd b \rhd Q \;=_A\; Q \lhd \neg b \rhd P$
(3) $(P \lhd b \rhd Q) \lhd c \rhd R \;=_A\; P \lhd b \wedge c \rhd (Q \lhd c \rhd R)$
(4) $P \lhd b \rhd (Q \lhd c \rhd R) \;=_A\; (P \lhd b \rhd Q) \lhd c \rhd (P \lhd b \rhd R)$
(5) $P \lhd true \rhd Q \;=_A\; P \;=_A\; Q \lhd false \rhd P$
(6) $(P \lhd b \rhd Q); R \;=_A\; (P; R) \lhd b \rhd (Q; R)$
(7) $(v := e); (P \lhd b \rhd Q) \;=_A\; ((v := e); P) \lhd b[e/v] \rhd ((v := e); Q)$

Proof.
For any finite program P:

(1)	$P \lhd b \rhd P$	$\{\mathbf{cond}-1\}$
$=_A$	$\oplus\{(1 \lhd b \rhd 0) : P,\; (0 \lhd b \rhd 1) : P\}$	$\{\oplus - 7\}$
$=_A$	$\oplus\{1 : P\}$	$\{\oplus - 3\}$
$=_A$	P	
(2)	$P \lhd b \rhd Q$	$\{\mathbf{cond}-1\}$
$=_A$	$\oplus\{(1 \lhd b \rhd 0) : P,\, (0 \lhd b \rhd 1) : Q\}$	$\{\oplus - 1\}$
$=_A$	$\oplus\{(1 \lhd \neg b \rhd 0) : Q,\, (0 \lhd \neg b \rhd 1) : P\}$	$\{\mathbf{cond}-1\}$
$=_A$	$Q \lhd \neg b \rhd P$	

$$\begin{array}{lll}
(3) & (P \lhd b \rhd Q) \lhd c \rhd R & \{\mathbf{cond}-1\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd c \rhd 0) : \oplus\{(1 \lhd b \rhd 0) : P,\ (0 \lhd b \rhd 1) : Q),\\ (0 \lhd c \rhd 1) : R\end{array}\right\} & \{\oplus - 4\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd b \wedge c \rhd 0) : P,\\ Let(1 \lhd \neg b \wedge c \rhd 0) : Q,\\ (1 \lhd \neg c \rhd 0) : R\end{array}\right\} & \{\oplus - 4\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd b \wedge c \rhd 0) : P,\\ (0 \lhd b \wedge c \rhd 1) : \oplus\{(1 \lhd c \rhd 0) : Q,\ (0 \lhd c \rhd 1) : R\}\end{array}\right\} & \{\mathbf{cond}-1\} \\
=_A & P \lhd b \wedge c \rhd (Q \lhd c \rhd R) &
\end{array}$$

$$\begin{array}{lll}
(4) & P \lhd b \rhd (Q \lhd c \rhd R) & \{\mathbf{cond}-1\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd b \rhd 0) : P,\\ (0 \lhd b \rhd 1) : \oplus\{(1 \lhd c \rhd 0) : Q,\ (0 \lhd c \rhd 1) : R\}\end{array}\right\} & \{\oplus - 4\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd b \rhd 0) : P,\\ (1 \lhd \neg b \wedge c \rhd 0) : Q,\\ (1 \lhd \neg b \wedge \neg c \rhd 0) : R\end{array}\right\} & \{\oplus - 7\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd b \wedge c \rhd 0) : P,\\ (1 \lhd b \wedge \neg c \rhd 0) : P,\\ (1 \lhd \neg b \wedge c \rhd 0) : Q,\\ (1 \lhd \neg b \wedge \neg c \rhd 0) : R\end{array}\right\} & \{\oplus - 1 \text{ and cond} - 1\} \\
=_A & \oplus\left\{\begin{array}{l}(1 \lhd c \rhd 0) : (P \lhd b \rhd Q),\\ (0 \lhd \neg c \rhd 1) : (P \lhd b \rhd R)\end{array}\right\} & \{\mathbf{cond}-1\} \\
=_A & (P \lhd b \rhd Q) \lhd c \rhd (P \lhd b \rhd R) &
\end{array}$$

$$\begin{array}{lll}
(5) & P \lhd true \rhd Q & \{\mathbf{cond}-1\} \\
=_A & \oplus\{(1 \lhd true \rhd 0) : P,\ (1 \lhd true \rhd 1) : Q\} & \{\oplus - 8\} \\
=_A & \oplus\{1 : P\} & \{\oplus - 3\} \\
=_A & P & \{\oplus - 1, 3 \text{ and } 8\} \\
=_A & \oplus\{0 : Q,\ 1 : P\} & \{\text{calculation}\} \\
=_A & \oplus\{(1 \lhd false \rhd 0) : Q,\ (0 \lhd false \rhd 1) : P\} & \{\mathbf{cond}-1\} \\
=_A & Q \lhd false \rhd P &
\end{array}$$

(6) From **cond**-1 and $\oplus$-5.

(7) From **cond**-1 and $\oplus$-6.

The probabilistic choice operator distributes over conditional.

Theorem 2.2.
Let $P = \oplus\{\beta_1 : P_1, ..., \beta_m : P_m\}$, then
$(P \lhd b \rhd Q) =_A \oplus\{\beta_1 : (P_1 \lhd b \rhd Q), ..., \beta_k : (P_k \lhd b \rhd Q)\}$
provided that $\Sigma_i \beta_i = 1$

Proof.

$$\begin{array}{lll}
 & P \lhd b \rhd Q & \{\mathbf{cond}-1\} \\
=_A & \oplus\{(1 \lhd b \rhd 0) : P,\ (0 \lhd b \rhd 1); Q\} & \{\oplus - 4\} \\
=_A & \oplus\left\{\begin{array}{l}(\beta_1 \lhd b \rhd 0) : P_1,\\ ....,\\ (\beta_m \lhd b \rhd 0) : P_m,\\ (0 \lhd b \rhd 1) : Q\end{array}\right\} & \{(\oplus - 7) \text{ and assumption:} \sigma_i\beta_i = 1\}
\end{array}$$

$$=_A \ \oplus \left\{ \begin{array}{l} (\beta_1 \lhd b \rhd 0) : P_1, \\ ...., \\ (\beta_m \lhd b \rhd 0) : P_m, \\ (0 \lhd \neg b \rhd \beta_1) : Q, \\ ....., \\ (0 \lhd b \rhd \beta_m) : Q \end{array} \right\} \qquad \{\mathbf{cond} - 1\}$$

$$=_A \ \oplus \left\{ \begin{array}{l} \beta_1 : (P_1 \lhd b \rhd Q), \\ ...., \\ \beta_m : (P_m \lhd b \rhd Q) \end{array} \right\}$$

2.3 Sequential Composition

Sequential composition in the probabilistic programming language inherits the algebraic laws of its counterpart in the conventional programming language. It is associative, and has $\bot$ as its zero, and *skip* as its unit.
seq-1. $P;(Q;R) \ =_A \ (P;Q);R$
seq-2. $\bot;Q \ =_A \ \bot \ =_A \ P;\bot$
seq-3. $\mathbf{skip};Q \ =_A \ Q \ =_A \ Q;\mathbf{skip}$

2.4 Total Assignment

An assignment is a total one if all the variables of the program appear on the left hand side in some standard order

$$x,\, y,\, ..,z \ := \ e,\, f,\, ...,\, g$$

A non-total assignment $x := e$ can be transformed to a total assignment by addition of identity assignments

asgn-1. $(x := e) \ =_A \ (x,\, y,\, ..,z := e,\, y, ..., z)$

For the notational simplicity we will use v to stand for the list $x, y,\, .., z$ of program variables and $v := e$ for a total assignment.
The list of variables may be sorted into any desired order, provided that the right hand side is subject tote same permutation.

asgn-2. $(x,\, y,\, .. := e,\, f, ..) \ =_A \ (y,\, x, .. := f,\, e, ..)$

The following law enables us to eliminate sequential composition between total assignments

asgn-3. $(v := e\, ;\, v := f(v)) \ =_A \ (v := f(e))$

where the expression $f(e)$ is easily calculated by substituting the expression in the list e for the corresponding variables in the list v.

The following law deals with the conditional of total assignments

asgn-4. $((v := e) \lhd b \rhd (v := f)) \ =_A \ (v := (e \lhd b \rhd f))$

where the conditional expression $e \lhd b \rhd f$ is defined mathematically:

$$e \lhd b \rhd f \begin{cases} =_{df} e & \text{if } b \\ =_{df} f & \text{if } \neg b \end{cases}$$

Finally, we need a law that determines when two total assignments are equal.

asgn-5. $(v := f) \ =_A \ (v := g)$ iff $\forall v \bullet f(v) = g(v)$

3 Normal Form Reduction

This section is devoted to the concept of normal form. It will deal with the following issues:

- Transform a finite program into a finite normal form based on the algebraic laws of the previous section.
- Introduce the least upper bound operator for an ascending chain of finite normal forms.
- Establish the continuity of programming combinators.
- Verify the continuity of the recursion operator.
- Convert an infinite program into an ascending chain of finite normal forms.

First we introduce the concept of finite normal form.

3.1 Finite Normal Form

Definition 3.1 (finite normal form).
A finite normal form is a probabilistic choice with total assignments as its alternatives:

$$\oplus\{\beta_1 : (v := e_1), \ldots, \beta_k : (v := e_k)\}$$

Theorem 3.2. Let $S_1 \ = \ \oplus \{\beta_1 : (v := e_1), .., \beta_m : (v := e_m)\}$ and $S_2 \ = \ \oplus \{\alpha_1 : (v := f_1), .., \alpha_n : (v := f_n)\}$.
Then

$$(1)\ S_1; S_2 \ =_A \oplus \left\{ \begin{array}{l} (\beta_1 \cdot \alpha_1) : (v := f_1(e_1)), \\ \ldots, \\ (\beta_1 \cdot \alpha_n) : (v := f_n(e_1)), \\ \ldots \\ (\beta_m \cdot \alpha_1) : (v := f_1(e_m)), \\ \ldots, \\ (\beta_m \cdot \alpha_n) : (v := f_n(e_m)) \end{array} \right\}$$

$$(2)\ S_1 \lhd b \rhd S_2 \ =_A \oplus \left\{ \begin{array}{l} (\beta_1 \lhd b \rhd 0) : (v := e_1), \\ \ldots\ldots, \\ (\beta_m \lhd b \rhd 0) : (v := e_m), \\ (\alpha_1 \lhd \neg b \rhd 0) : (v := f_1), \\ \ldots\ldots, \\ (\alpha_n \lhd \neg b \rhd 0) : (v := f_n) \end{array} \right\}$$

Proof.

$$\begin{array}{lll}
(1) & S_1; S_2 & \{\oplus - 5\} \\
=_A & \oplus\{\beta_1 : (v := e_1; S_2), .., \beta_m : (v := e_m; S_2)\} & \{\oplus - 6\} \\
=_A & \oplus \left\{ \begin{array}{l} \beta_1 : \oplus \left\{ \begin{array}{l} \alpha_1[e_1/v] : (v := e_1; v := f_1), \\ ..., \\ \alpha_n[e_1/v] : (v := e_1; v := f_n) \end{array} \right\}, \\ \ldots\ldots, \\ \beta_m : \oplus \left\{ \begin{array}{l} \alpha_1[e_m/v] : (v := e_m; v := f_1), \\ ..., \\ \alpha_n[e_m/v] : (v := e_m; v := f_n) \end{array} \right\} \end{array} \right\} & \{\mathbf{asgn} - 3\} \\
=_A & \oplus \left\{ \begin{array}{l} \beta_1 : \oplus \left\{ \begin{array}{l} \alpha_1[e_1/v] : (v := f_1(e_1)), \\ ..., \alpha_n[e_1/v] : (v := f_n(e_1)) \end{array} \right\}, \\ \ldots\ldots, \\ \beta_m : \oplus \left\{ \begin{array}{l} \alpha_1[e_m/v] : (v := f_1(e_m)), \\ ..., \\ \alpha_n[e_m/v] : (v := f_n(e_m)) \end{array} \right\} \end{array} \right\} & \{\oplus - 4\} \\
=_A & \oplus \left\{ \begin{array}{l} (\beta_1 \cdot \alpha_1) : (v := f_1(e_1)), \\ ..., \\ (\beta_1 \cdot \alpha_n) : (v := f_n(e_1)), \\ ... \\ (\beta_m \cdot \alpha_1) : (v := f_1(e_m)), \\ ..., \\ (\beta_m \cdot \alpha_n) : (v := f_n(e_m)) \end{array} \right\} & \\
(2) & S_1 \lhd b \rhd S_2 & \{\mathbf{cond} - 1\} \\
=_A & \oplus\{(1 \lhd b \rhd 0) : S_1,\ (1 \lhd \neg b \rhd 0) : S_2\} & \{\oplus - 4\} \\
=_A & \oplus \left\{ \begin{array}{l} (\beta_1 \lhd b \rhd 0) : (v := e_1), \\ \ldots\ldots, \\ (\beta_m \lhd b \rhd 0) : (v := e_m), \\ (\alpha_1 \lhd \neg b \rhd 0) : (v := f_1), \\ \ldots\ldots, \\ (\alpha_n \lhd \neg b \rhd 0) : (v := f_n) \end{array} \right\} &
\end{array}$$

Theorem 3.3. Assume that $S_i \ = \ \oplus\{\alpha_{i,1} : (v := e_{i,1}), ..., \alpha_{i,k_i} : (v := e_{i,k_i})\}$ for $1 \leq i \leq n$. Then

$$\oplus\{\beta_1 : S_1, ..., \beta_n : S_n\} \; =_A \; \oplus \left\{ \begin{array}{l} (\beta_1 \cdot \alpha_{1,1}) : (v := e_{1,\,1}), \\ ......, \\ (\beta_1 \cdot \alpha_{1,\,k_1}) : (v := e_{1,\,k_1}), \\ ......, \\ (\beta_n \cdot \alpha_{n,\,1}) : (v := e_{n,\,1}), \\ ......, \\ (\beta_n \cdot \alpha_{n,\,k_n}) : (v := e_{n,\,k_n}) \end{array} \right\}$$

Proof: Similar to Theorem 3.2(1).

Theorem 3.4 (finite normal reduction).
Any finite program can be converted into a finite normal form.

Proof.
Basic case:
(1) From $\oplus$-3, we have $(v := e) \;=_A\; \oplus\{1 : (v := e)\}$
(2) From Corollary of $\oplus$-3, it follows that $\bot \;=_A\; \oplus\{\}$

induction: The conclusion follows from Theorems 3.2 and 3.3.
The following law permits comparison of finite normal forms
norm-1. Let $S_1 \;=\; \oplus\{\beta_1(v) : (v := e_1(v)),\; ..,\; \beta_m(v) : (v := e_m(v))\}$
and $S_2 \;=\; \oplus\{\alpha_1(v) : (v := f_1(v)),\; ...,\; \alpha_n(v) : (v := f_n(v))\}$.
Then $S_1 \sqsubseteq_A S_2$ **iff**
$\forall c,\, d \bullet (\Sigma_i\{\beta_i(c) \mid e_i(c) = d\} \leq \Sigma_j\{\alpha_j(c) \mid f_j(c) = d\})$

Theorem 3.5. $S_1 \sqsubseteq_A S_2$ iff for all constants c

$$((v := c); S_1) \sqsubseteq_A ((v := c); S_2)$$

Proof. From $\oplus$-6 and **norm**-1.

Corollary. Assume that

$$T_1 = \oplus\{\beta_1 : (v := c_1), .., \beta_n : (v := c_m)\}$$
$$T_2 = \oplus\{\alpha_1 : (v := d_1), .., \alpha_n : (v := d_n)\}$$

where all β_i and α_j are constants, and furthermore both $\{c_1, .., c_m\}$ and $\{d_1, .., d_n\}$ are lists of distinct constants.
Then $T_1 \sqsubseteq_A T_2$ iff there exists an injective mapping ϕ from $\{1, .., m\}$ to $\{1, .., n\}$ such that

$$\forall i \bullet (c_i = d_{\phi(i)}) \;\wedge\; (\beta_i \leq \alpha_{\phi(i)})$$

The next theorem shows that all programming combinators F satisfy

$$F(S_1) \sqsubseteq_A F(S_2)$$

whenever both S_1 and S_2 are finite normal forms, and $S_1 \sqsubseteq_A S_2$

Theorem 3.6.
If S_1 and S_2 are finite normal forms satisfying $S_1 \sqsubseteq_A S_2$, then
(1) $(S_1 \lhd b \rhd R) \sqsubseteq_A (S_2 \lhd b \rhd R)$
provided that R is a finite program.
(2) $\oplus\{\gamma : S_1, \underline{G}\} \sqsubseteq_A \oplus\{\gamma : S_2, \underline{G}\}$
where $\underline{G} = \xi_1 : R_1, .., \xi_n : R_n$ and all R_i are finite.
(3) $(S_1; R) \sqsubseteq_A (S_2; R)$ provided that R is a finite program.
(4) $(R; S_1) \sqsubseteq_A (R; S_2)$ provided that R is a finite program.

Proof. From $\oplus$-6 we can transform $(v := c); S_1$ and $(v := c); S_2$ into the following form

$$\begin{aligned}(v := c); S_1 &=_A \oplus\{\beta_1 : (v := c_1), .., \beta_m : (v := c_m)\}\\ (v := c); S_2 &=_A \oplus\{\alpha_1 : (v := d_1), ..., \alpha_n : (v := d_n)\}\end{aligned}$$

where all β_i and α_j are constants, and furthermore both $\{c_1, .., c_m\}$ and $\{d_1, .., d_n\}$ are lists of distinct constants. From Corollary of Theorem 3.5 it follows that there exists an injective mapping ϕ from $\{1, .., m\}$ to $\{1, .., n\}$ satisfying

$$\forall i \bullet (c_i = d_{\phi(i)}) \;\wedge\; (\beta_i \leq \alpha_{\phi(i)})$$

Proof of (1).

Case 1:$b[c/v] = true$

$$\begin{array}{lll} (1) & (v := c); (S_1 \lhd b \rhd R) & \{\text{Theorem 2.1(5)}\}\\ \equiv_A & (v := c); S_1 & \{\text{Theorem 3.5}\}\\ \sqsubseteq_A & (v := c); S_2 & \{\text{Theorem 2.1(5)}\}\\ =_A & (v := c); (S_2 \lhd b \rhd R) & \end{array}$$

Case 2: $b[c/v] = false$. From Theorem 2.1 it follows that

$$(v := c); (S_1 \lhd b \rhd R) \;=_A\; (v := c); R \;=_A\; (v := c); (S_2 \lhd b \rhd R)$$

The conclusion follows from Theorem 3.5.

Proof of (2). From Theorem 3.4 it follows that all R_i can be transformed into finite normal forms:

$$R_i \;=_A\; \oplus\{\rho_{i,1}(v) : (v := f_{i,1}), ..., \rho_{i,k_i}(v) : (v := f_{i,k_i})\}$$

for $i \in \{1, .., n\}$.
From Theorem 3.3 it follows that
$(v := c); \oplus\{\gamma : S_1, \underline{G}\} \;=_A$

$$\oplus\left\{\begin{array}{l}(\gamma(c)\cdot\beta_1) : (v := c_1), ..., (\gamma(c)\cdot\beta_m) : (v := c_m),\\ (\xi_1(c)\cdot\rho_{1,1}(c)) : (v := f_{1,1}(c)), ..., (\xi_1(c)\cdot\rho_{1,k_1}(c)) : (v := f_{1,k_1}(c)),\\ .....,\\ (\xi_n(c)\cdot\rho_{n,1}(c)) : (v := f_{n,1}(c)), ..., (\xi_n(c)\cdot\rho_{n,k_n}(c)) : (v := f_{n,k_n}(c))\end{array}\right\}$$

$(v := c); \oplus\{\gamma : S_2, \underline{G}\} \;=_A$

$$\oplus\left\{\begin{array}{l}(\gamma(c)\cdot\alpha_1):\ (v:=d_1),....,(\gamma(c)\cdot\alpha_n):\ (v:=d_n),\\ (\xi_1(c)\rho_{1,1}(c)):\ (v:=f_{1,1}(c)),....,(\xi_1(c)\cdot\rho_{1,k_1}(c)):\ (v:=f_{1,k_1}(c)),\\ .....,\\ (\xi_n(c)\cdot\rho_{n,1}(c)):\ (v:=f_{n,1}(c)),.....,(\xi_n(c)\cdot\rho_{n,k_n}(c)):\ (v:=f_{n,k_n}(c))\end{array}\right\}$$

Then for any constant r

$$\begin{array}{lll} & \Sigma\begin{pmatrix}\{(\gamma(c)\cdot\beta_i)\mid(c_i=r)\}\\ \cup\{(\xi_i(c)\cdot\rho_{i,j}(c))\mid(f_{i,j}=r)\}\end{pmatrix} & \{\forall i\bullet(c_i=d_{\phi(i)})\}\\ = & \Sigma\begin{pmatrix}\{(\gamma(c)\cdot\beta_i)\mid(d_{\phi(i)}=r)\}\\ \cup\{(\xi_i(c)\cdot\rho_{i,j}(c))\mid(f_{i,j}=r\}\end{pmatrix} & \{\forall i\bullet(\beta_i\le\alpha_{\phi(i)})\}\\ \le & \Sigma\begin{pmatrix}\{(\gamma(c)\cdot\alpha_{\phi(i)})\mid(d_{\phi(i)}=r)\}\\ \cup\{(\xi_i(c)\cdot\rho_{i,j}(c))\mid(f_{i,j}=r)\}\end{pmatrix} & \{(\Sigma X)\ \le\ (\Sigma(X\cup Y))\}\\ \le & \Sigma\begin{pmatrix}\{(\gamma(c)\cdot\alpha_l)\mid(d_l=r)\}\\ \cup\{(\xi_i(c)\cdot\rho_{i,j}(c))\mid(f_{i,j}=r)\}\end{pmatrix} & \end{array}$$

which leads to the conclusion

$$(v:=c);\oplus\{\gamma:S_1,\underline{G}\}\ \sqsubseteq_A\ (v:=c);\oplus\{\gamma:S_2,\underline{G}\}$$

Proof of (3). From Theorem 3.4 we can convert R into a finite normal form

$$R\ =_A\ \oplus\{\rho_1(v):(v:=f_1),...,\rho_n(v):(v:=f_n)\}$$

From $\oplus$-5 and 6 we obtain

$(v:=c);S_1;R\ =_A$

$$\oplus\left\{\begin{array}{l}\beta_1\cdot\rho_1(c_1):(v:=f_1(c_1)),....,\beta_1\cdot\rho_n(c_1):(v:=f_n(c_1)),\\ .....,\\ \beta_m\cdot\rho_1(c_m):(v:=f_1(c_m)),....,\beta_m\cdot\rho_n(c_m):(v:=f_n(c_m))\end{array}\right\}$$

$(v:=c);S_2;R\ =_A$

$$\oplus\left\{\begin{array}{l}(\alpha_1\cdot\rho_1(d_1)):\ (v:=f_1(d_1)),....,(\alpha_1\cdot\rho_n(d_1)):\ (v:=f_n(d_1)),\\ .....,\\ (\alpha_n\cdot\rho_1(d_n)):\ (v:=f_1(d_n)),....,(\alpha_n\cdot\rho_n(d_n)):(v:=f_n(d_n))\end{array}\right\}$$

Then for any constant r we have

$$\begin{array}{lll} & \Sigma_{i,j}\{\beta_i\cdot\rho_j(c_i)\mid f_j(c_i)=r\} & \{\forall i\bullet(c_i=d_{\phi(i)})\}\\ = & \Sigma_{i,j}\{\beta_i\cdot\rho_j(d_{\phi(i)})\mid f_j(d_{\phi(i)})=r\} & \{\forall i\bullet(\beta_i\le\alpha_{\phi(i)})\}\\ \le & \Sigma_{i,j}\{\alpha_{\phi(i)}\cdot\rho_j(d_{\phi(i)})\mid f_j(d_{\phi(i)})=r\} & \{\Sigma X\ \le\ \Sigma(X\cup Y)\}\\ \le & \Sigma_{i,j}\{\alpha_i\cdot\rho_j(d_i)\mid f_j(d_i)=r\} & \end{array}$$

which leads to the conclusion that

$$(v:=c);S_1;R\sqsubseteq_A\ (v:=c);S_2;R$$

3.2 Infinite Normal Form

Definition 3.2 (infinite normal form).
An infinite normal form is represented by an infinite sequence of finite normal forms

$$S\ =\ \{S_i\mid i\in Nat\}$$

where each S_{i+1} is a more accurate description than its predecessor

$$(S_{i+1}\ \sqsupseteq_A\ S_i)\qquad\text{for all } i\in Nat$$

This is called ascending chain condition. It is this type of chain that will be taken as the normal form for programs that contains recursion. The exact behaviour of the normal form is captured by the least upper bound of the whole sequence, written

$$\bigsqcup S$$

The least upper bound operator is characterised by two laws:

norm-2. $\bigsqcup S \sqsubseteq_A Q$ iff $\forall i \bullet (S_i \sqsubseteq_A Q)$
norm-3. If P is a finite normal form, then $P \sqsubseteq_A \bigsqcup T$ iff

$$\forall c,\ \exists j \bullet ((v := c); P) \sqsubseteq_A ((v := c); T_j)$$

The following theorem states that $\bigsqcup S$ is actually the least upper bound of the ascending chain with respect to the refinement order $\sqsubseteq_A$.

Theorem 3.7.
(1) $S_i \sqsubseteq_A \bigsqcup S$ for all $i \in Nat$.
(2) If $S_i \sqsubseteq_A Q$ for all $i \in Nat$ then $\bigsqcup S \sqsubseteq_A Q$

Proof. (1) From **norm**-3.
(2) From **norm**-2.

3.3 Continuity

This section deals with the continuity of programming combinators (including recursion) before we show how to transform a program into an ascending chain of finite normal form.

Definition 3.3 (continuity).
An operator is continuous if it distributes through least upper bound of descending chains.

The following laws explore the continuity of finite programming combinators.

norm-4. $(\bigsqcup S) \lhd b \rhd P =_A \bigsqcup_i (S_i \lhd b \rhd P)$
norm-5. $(\bigsqcup S); P =_A \bigsqcup_i (S_i; P)$
norm-6. $P; (\bigsqcup S) =_A \bigsqcup_i (P; S_i)$
provided that P is a finite normal form.
norm-7. $\oplus\{\alpha : (\bigsqcup S),\ \underline{G}\} =_A \bigsqcup_i \oplus\{\alpha : S_i,\ \underline{G}\}$
The next concern is how to eliminate the nested least upper bound operators.
norm-8. $\bigsqcup_k (\bigsqcup_l S_{k,l}) =_A \bigsqcup_i S_{i,i}$
provided that $(S_{k,i+1} \sqsubseteq_A S_{k,i})$ and $(S_{i+1,l} \sqsubseteq_A S_{i,l})$ for all i, k and l.

Law **norm**-8 lays down the foundation for computation of normal forms by eliminating programming operators.

Theorem 3.8 (Continuity of finite programming combinators).
(1) $(\bigsqcup S) \lhd b \rhd (\bigsqcup T) \;=_{df}\; \bigsqcup_i (S_i \lhd b \rhd T_i)$
(2) $(\bigsqcup S)\,;\,(\bigsqcup T) \;=_{df}\; \bigsqcup_i (S_i\,;\,T_i)$
(3) $\oplus\{\alpha : (\bigsqcup S), .., \beta : (\bigsqcup T)\} \;=_{df}\; \bigsqcup_i \oplus\{\alpha : S_i, ..., \beta : T_i\}$

Proof.

$$\begin{array}{lll}
(2) & (\bigsqcup S);(\bigsqcup T) & \{\mathbf{norm}-5\} \\
=_A & \bigsqcup_i (S_i;\bigsqcup T) & \{\mathbf{norm}-6\} \\
=_A & \bigsqcup_i \bigsqcup_j (S_i;T_j) & \{\mathbf{norm}-8\} \\
=_A & \bigsqcup_i (S_i;T_i) &
\end{array}$$

The continuity theorem ensures that ascending chains constitute a valid normal form for all the combinators of our probabilistic language, and the stage is set for treatment of recursion.

3.4 Recursion

Consider first an innermost recursive program

$$\mu X \bullet P(X)$$

where $P(X)$ contains X as its only free identifier. Because X is certainly not in normal form, it is impossible to express $P(X)$ in normal form. However, all other components of $P(X)$ are expressible in finite normal form, and all its combinators permit reduction to finite normal form. So if X were replaced by $\bot$, $P(\bot)$ can be reduced to finite normal form, and so on $P(\bot)$, $P^2(\bot)$,.., Furthermore from Theorem 3.6 it follows that P is monotonic, this constitutes an ascending chain of finite normal forms.

rec-1. $\mu X \bullet P(X) \;=_A\; \bigsqcup_n P^n(\bot)$ provided that P is continuous.
where $P^0(X) \;=_{df}\; \bot$ and $P^{n+1}(X) \;=_{df}\; P(P^n(\bot))$.
Finally we are going to show that the μ operator is also continuous.

Theorem 3.9 (Continuity of the recursion operator).
If $S_i(X)$ contains X as its only free recursive identifier for all i,
and that all S_i are continuous and they form an ascending chain for all finite normal forms X:

$$S_{i+1}(X) \;\sqsupseteq_A\; S_i(X) \qquad \text{for all } i \in Nat$$

then $\mu X \bullet \bigsqcup_i S_i(X) \;=_A\; \bigsqcup_i \mu X \bullet S_i(X)$

Proof. Let $P(X) \;=_{df}\; \bigsqcup_i S_i(X)$. By induction we are going to establish for all $n \in Nat$

$$P^n(\bot) \;=_A\; \bigsqcup_i S_i^n(\bot) \qquad (*)$$

Base case: $n = 0$

$$P^0(\bot) \;=_A\; \bot \;=_A \bigsqcup_i \bot \;=_A\; \bigsqcup_i S_i^0(\bot)$$

Induction:

$$\begin{array}{ll}
\quad P^{n+1}(\bot) & \{\text{Def of } P^{n+1}\} \\
=_A \bigsqcup_i S_i(P^n(\bot)) & \{\text{induction hypothesis}\} \\
=_A \bigsqcup_i S_i(\bigsqcup_j S_j^n(\bot)) & \{S_i \text{ is continuous}\} \\
=_A \bigsqcup_i \bigsqcup_j S_i(S_j^n(\bot)) & \{\mathbf{norm}-8\} \\
=_A \bigsqcup_i S_i(S_i^n(\bot)) & \{\text{Def of } S_i^{n+1}\} \\
=_A \bigsqcup_i S_i^{n+1}(\bot) &
\end{array}$$

which leads to the conclusion:

$$\begin{array}{ll}
\quad \mu X \bullet P(X) & \{\mathbf{rec}-1\} \\
=_A \bigsqcup_n P^n(\bot) & \{\text{Conclusion } (*)\} \\
=_A \bigsqcup_n (\bigsqcup_i S_i^n(\bot)) & \{\mathbf{norm}-8\} \\
=_A \bigsqcup_i (\bigsqcup_n S_i^n(\bot)) & \{\mathbf{rec}-1\} \\
=_A \bigsqcup_i \mu X \bullet S_i(X) &
\end{array}$$

Now we reach the stage to eliminate the recursion operator.

Theorem 3.10.

Any recursive program $\mu X \bullet F(X)$ can be converted into the least upper bound of an ascending chain.

Proof.

$$\begin{array}{ll}
\quad \mu X \bullet F(X,\, \mu Y.G_1(Y), ..., \mu Y \bullet G_m(Y)) & \{\mathbf{rec}-1\} \\
=_A \mu X \bullet F(X,\, \bigsqcup_n G^n(\bot),\, ..,\, \bigsqcup_n G_m^n(\bot)) & \{\text{Theorem 3.8}\} \\
=_A \mu X \bullet \bigsqcup_n (F(X,\, G^n(\bot), .., G_m^n(\bot))) & \{\text{Theorem 3.9}\} \\
=_A \bigsqcup_n \mu X \bullet F(X,\, G^n, .., G_m^n(\bot)) & \left\{\begin{array}{l}\mathbf{rec}-1 \text{ and let} \\ F_n =_{df} F(X,\, G^n(\bot), .., G_m^n(\bot))\end{array}\right\} \\
=_A \bigsqcup_n \bigsqcup_m F_n^m(\bot) & \{\mathbf{norm}-8\} \\
=_A \bigsqcup_n F_n^n(\bot) &
\end{array}$$

Theorem 3.11.
(1) If $P \sqsubseteq_A Q$, then
(a) $(P \lhd b \rhd R) \sqsubseteq_A (Q \lhd b \rhd R)$
(b) $(P; R) \sqsubseteq_A (Q; R)$
(c) $\oplus\{\gamma : P, \xi_1 : U_1, ..., \xi_l : U_l\} \sqsubseteq_A \oplus\{\gamma : Q, \xi_l : U_l\}$
(d) $(R; P) \sqsubseteq_A (R; Q)$
(2) If $P(S) \sqsubseteq_A Q(S)$ for any finite normal form S, then
$\mu X \bullet P(X) \sqsubseteq_A \mu X \bullet Q(X)$

Proof: From Theorems 3.4 and 3.10 we can transform P, Q and R into descending chain of finite normal forms:

$$P =_A \bigsqcup_i P_i, \qquad Q =_A \bigsqcup_j Q_j \qquad R =_A \bigsqcup_k R_k$$

From Definition 3.2 it follows that for any constant c there exists a mapping ψ_c satisfying $\forall i \bullet (i \le \psi_c(i))$, and

$$\forall i \bullet ((v := c); P_i) \sqsubseteq_A ((v := c); Q_{\psi_c(i)}) \qquad (*)$$

Proof of 1.(a): From $(*)$ and Theorem 3.6(1) we reach the conclusion

$$\forall i \bullet ((v := c); (P_i \lhd b \rhd R_i)) \sqsubseteq_A ((v := c); (Q_{\psi_c(i)} \lhd b \rhd R_{\psi_c(i)}))$$

which implies

$$\begin{array}{lll} & (P \lhd b \rhd R) & \{\text{Theorem 3.8}\} \\ =_A & \bigsqcup_i (P_i \lhd b \rhd R_i) & \{\textbf{norm } 2 \text{ and } 3\} \\ \sqsubseteq_A & \bigsqcup_j (Q_j \lhd b \rhd R_j) & \{\text{Theorem 3.8}\} \\ =_A & (Q \lhd b \rhd R) & \end{array}$$

Proof of 1.(b): From $(*)$ and Theorem 3.6(3)(4) we reach the conclusion

$$\forall i \bullet ((v := c); (P_i; R_i)) \sqsubseteq_A ((v := c); (Q_{\psi_c(i)}; R_{\psi_c(i)}))$$

which implies

$$\begin{array}{lll} & (P; R) & \{\text{Theorem 3.8}\} \\ =_A & \bigsqcup_i (P_i; R_i) & \{\textbf{norm } 2 \text{ and } 3\} \\ \sqsubseteq_A & \bigsqcup_j (Q_j; R_j) & \{\text{Theorem 3.8}\} \\ =_A & (Q; R) & \end{array}$$

Proof of 1.(c): From Theorems 3.4 and 3.10 there exists a family $\{\{U_{i,n}\ n \in Nat\} \mid 1 \le i \le l\}$ of ascending chains such that for all i, $U_i =_A \bigsqcup_j U_{i,j}$. Then we have

$$\begin{array}{lll} & (v := c); \oplus \left\{ \begin{array}{l} \gamma : P_i, \\ \xi_1 : U_{1,i}, \\ .., \\ \xi_l : U_{l,i} \end{array} \right\} & \{\oplus - 6\} \\ =_A & \oplus \left\{ \begin{array}{l} \gamma(c) : ((v := c); P_i)), \\ \xi_1(c) : ((v := c); U_{1,i}), \\ ..., \\ \xi_l(c) : ((v := c); U_{l,i}) \end{array} \right\} & \{\text{Theorem 3.6}\} \\ \sqsubseteq_A & \oplus \left\{ \begin{array}{l} \gamma(c) : ((v := c); Q_{\psi_c(i)}), \\ \xi_1(c) : ((v := c); U_{1,i}), \\ ..., \\ \xi_l(c) : ((v := c); U_{l,i}) \end{array} \right\} & \{(i \le \psi_c(i)) \implies \forall j \bullet (U_{j,i} \sqsubseteq_A U_{j,\psi_c(i)}\} \\ \sqsubseteq_A & \oplus \left\{ \begin{array}{l} \gamma(c) : ((v := c); Q_{\psi_c(i)}), \\ \xi_1(c) : ((v := c); U_{1,\psi_c(i)}), \\ ..., \\ \xi_l(c) : ((v := c); U_{l,\psi_c(i)}) \end{array} \right\} & \{\oplus - 6\} \\ =_A & (v := c); \oplus \left\{ \begin{array}{l} \gamma : Q_{\psi_c(i)}, \\ \xi_1 : U_{1,\psi_c(i)}, \\ ..., \\ \xi_l : U_{l,\psi_c(i)} \end{array} \right\} & \end{array}$$

which leads to the conclusion.

Proof of 1.(d): Assume that

$$R_i =_A \oplus\{\xi_{i,1} : (v := e_{i,1}), ..., \xi_{i,n_i} : (v := e_{i,n_i} : (v := e_{i,n_i})\}$$

Define $\Phi_c(i) \;=_{df}\; \mathbf{max}(\psi_{e_{i,1}(c)}(i),\, ...,\, \psi_{e_{i,n_i}(c)}(i))$. Then we have

$$\begin{array}{lll}
 & (v := c); R_i; P_i & \{\oplus - 5 \text{ and } 6\} \\
=_A & \oplus\left\{\begin{array}{l} x_{i,1}(c) : (v := e_{i,1}(c)); P_i), \\ ...., \\ x_{i,n_i} : (v := e_{i,n_i}; P_i) \end{array}\right\} & \{\text{Conclusion } 1(c)\} \\
\sqsubseteq_A & \oplus\left\{\begin{array}{l} x_{i,1}(c) : (v := e_{i,1}(c)); Q_{\psi_{e_{i,1}(c)}(i)}, \\ ...., \\ x_{i,n_i} : (v := e_{i,n_i}; Q_{\psi_{e_{i,n_i}}(c)(i)}, \end{array}\right\} & \{\text{Def of } \Phi \text{ and Conclusion } 1(c)\} \\
\sqsubseteq_A & \oplus\left\{\begin{array}{l} x_{i,1}(c) : (v := e_{i,1}(c)); Q_{\Phi_c(i)}, \\ ...., \\ x_{i,n_i} : (v := e_{i,n_i}; Q_{\Phi_c(i)}, \end{array}\right\} & \{\oplus - 6\} \\
=_A & (v := c); R_i; Q_{\Phi_c(i)} & \{i \le \Phi_c(i)\} \\
\sqsubseteq_A & (v := c); R_{\Phi_c(i)}; Q_{\Phi_c(i)} &
\end{array}$$

which leads to the conclusion.

4 Testing Programs

An operational approach usually defines the relationship between a program and its possible execution by machine. In an abstract way, a computation consists of a sequence of individual steps with the following features:

– each step takes the machine from one state to a closely similar state;
– each step is drawn from a very limited repertoire.

In a stored program computer, the machine states are represented as pairs

$$(s,\ P)$$

where
(1) s is a text, defining the *data state* as an assignment of constant to all variables of the alphabet

$$x,\, y,\, ...,z \;:=\; a,\, b,\, ...,c$$

(2) P is a program text, representing the rest of the program that remains to be executed. When this becomes the empty text ϵ, there is no more program to be executed. The machine state

$$(t,\, \epsilon)$$

is the last state of any execution sequence that contains it, and t presents the final value of the variables in the end of execution.
The following lemma indicates that data states are the best programs.

Lemma 4.1.
$(s \;\sqsubseteq_A\; P)$ implies $(s \;=_A\; P)$.

Definition 4.1 (Probabilistic state).
Let $S_i\; =, \oplus\{\xi_{i,1} : (v := c_{i,1}),\, ..\xi_{i,m_i} : (v := c_{i,m_i})\}$ be a finite normal form for all $i \in Nat$ in which all $\xi_{i,j}$ and $c_{i,j}$ are constants,

and $c_{i,l}, \neq c_{i,m}$ for all $l \neq m$.
If $S_i \sqsubseteq_A S_{i+1}$ for all $i \in Nat$, then

$$\bigsqcup_i S_i$$

is called a probabilistic state.

The execution of program $(s; P)$ can be seen as a *test* on P with the test case s. The result of such a testing gives rise to a set of possible outcomes. We are then able to compare the behaviours of two programs based on testing.
Formally, the test operator for our probabilistic programming language is defined by

$$\mathcal{T}(s,\, P) \;=_{df}\; (s; P)$$

When $\perp$ is taken as the test case, we obtain

$$\mathcal{T}(\perp,\, P) \;=_A\; \perp$$

Execution of a test will deliver a probabilistic state.

Theorem 4.1. For any test $\mathcal{T}(s,\, P)$, there exists a probabilistic state t such that

$$\mathcal{T}(s,\, P) \;=_A\; t$$

Proof. From Theorem 3.4 it follows that any finite program P can be converted into a finite normal form:

$$P \;=_A \oplus\{\beta_1 : (v := e_1), ..., \,\beta_m : (v := e_m)\}$$

The conclusion is derived from $\oplus - 6$.
For any program P there exists an ascending chain $S \;=\; \{S_i \mid i \in Nat\}$ of finite normal form such that

$$P \;=_A\; \bigsqcup S$$

The conclusion follows from $\mathbf{norm} - 6$.

Corollary.
$P \sqsubseteq_A Q$ iff for all test case s

$$\mathcal{T}(s,\, P) \sqsubseteq_A \mathcal{T}(s,\, Q)$$

Definition 4.2.
A program P can be identified as a binary relation $[P]$ between test case s and a final probabilistic data state t it may enter in the end of testing

$$[P] \;=_{df}\; \{(s,\, t) \mid \mathcal{T}(s,\, P) \sqsubseteq_A t\}$$

As usual we define the refinement relation $\sqsubseteq_{rel}$ on the relational model by the set inclusion

$$P \sqsubseteq_{rel} Q \;=_{df}\; ([P] \supseteq [Q])$$

Theorem 4.2.
$\sqsubseteq_{rel} = \sqsubseteq_A$

Proof.

$$\begin{array}{lll}
 & P \sqsubseteq_A Q & \{\text{Corollary of Theorem 4.1}\} \\
\equiv & \forall s \bullet \mathcal{T}(s, P) \sqsubseteq_A \mathcal{T}(s, Q) & \{\text{Theorem 4.1}\} \\
\equiv & \forall s, t \bullet (\mathcal{T}(s, Q) \sqsubseteq_A t) \implies (\mathcal{T}(s, P) \sqsubseteq_A t) & \{\text{Definition 4.2}\} \\
\equiv & [Q] \subseteq [P] & \{\text{Definition of } \sqsubseteq_{rel}\} \\
\equiv & P \sqsubseteq_{rel} Q &
\end{array}$$

Theorem 4.3.
(1) $[P \lhd b \rhd Q](v := c) \ = \ [P](v := c) \lhd b[c/v] \rhd [Q](v := c)$
(2) $[\oplus\{\beta_1 : P_1, .., \beta_m : P_m\}](v := c) \ = \ \left\{ \begin{array}{l} \oplus\{\beta_1[c/v] : t_1, .., \beta_m[c/v] : t_m\} \mid \\ \forall i \bullet t_i \in [P_i](v := c) \end{array} \right\}$
(3) $[P; Q] \ = \ [P] \circ [Q] \uparrow$
where $[Q] \uparrow$ is defined inductively:

$$[Q] \uparrow (v := c) =_{df} [Q](v := c)$$

$$[Q] \uparrow (\oplus\{\rho_1 : (v := c_1), .., \rho_m : (v := c_m)\}) =_{df} \left\{ \begin{array}{l} \oplus\{\rho_1 : t_1, .., \rho_m : t_m\} \mid \\ \forall i \bullet t_i \in [Q] \uparrow (v := c_i) \end{array} \right\}$$

$$[Q] \uparrow (\bigsqcup_i t_i) =_{df} \{\bigsqcup_i u_i \mid \forall i \bullet u_i \in [Q] \uparrow (t_i)\}$$

(4) $[\mu X \bullet P(X)] \ = \ \bigcap_n [P^n(\bot)]$

Proof of (3)

$$\begin{array}{lll}
 & (s, t) \in [P; Q] & \{\text{Definition 2.1}\} \\
\equiv & \mathcal{T}(s, (P; Q)) \sqsubseteq_A t & \{\text{Theorem 4.1}\} \\
\equiv & \exists u \bullet \mathcal{T}(s, P) =_A u \wedge (u; Q) \sqsubseteq_A t & \{\text{Def of } [Q] \uparrow\} \\
\equiv & \exists u \bullet \mathcal{T}(s, P) =_A u \wedge (u, t) \in [Q] \uparrow & \{\text{Theorem 3.11}\} \\
\equiv & \exists u \bullet (\mathcal{T}(s, P) \sqsubseteq_A u) \wedge (u, t) \in [Q] \uparrow & \{\text{Definition 4.2}\} \\
\equiv & \exists u \bullet ((s, u) \in [P] \wedge (u, t) \in [Q] \uparrow) & \{\text{Def of relational composition}\} \\
\equiv & (s, t) \in ([P] \circ [Q] \uparrow) &
\end{array}$$

5 Operational Approach

This section provides an operational semantics for our probabilistic programming language. We will introduce the concept of the *consistency* of an operational framework with respect to the algebra of programs, and present a transition system for the probabilistic language. This section also explores the link between the consistent transition system with the normal form representation of probabilistic programs.
There are two types of transitions for our language
(1) Transition $(s, P) \to (t, Q)$ means P transfers to Q with the data state s replaced by t.
We define the concept of *divergence*, being a machine state that can lead to an infinite execution

$$\textbf{divergence}(s, P) \ =_{df} \ \forall n \, \exists t, Q \bullet ((s, P) \to_n (t, Q))$$

where $\rightarrow_0 =_{df} \ id$, and $\rightarrow_{n+1} =_{df} \ (\rightarrow ; \rightarrow_n)$.
(2) Transition $(s,\, P) \xrightarrow{r} (s,\, Q)$ (where $0 < r \leq 1$ means Q is chosen by P to be executed with the probability r, whereas the data state remains unchanged.
We examine the concept of *finitary*, being a machine state that can only engage in finite number of probabilistic choices

$$\mathbf{finitary}(s,\, P) \ =_{df} \ \exists n \, \forall t, Q, r, m \bullet \begin{pmatrix} (s,\, P) \xrightarrow{r}_m (t,\, Q) \wedge m > n \\ \implies \mathbf{divergence}(t,\, Q) \end{pmatrix}$$

where $\xrightarrow{r}_1 =_{df} \xrightarrow{r}$
and $\xrightarrow{r}_{n+1} =_{df} \ (\rightarrow ; \xrightarrow{r}_n) \cup \{(\xrightarrow{r_1} ; \xrightarrow{r_2}_n) \mid r_1 \cdot r_2 \ = \ r\}$
and $\xrightarrow{r}_* =_{df} \bigcup_n \xrightarrow{r}_n$

Definition 5.1. A transition system is *consistent* with respect to the algebraic semantics if for all machine states $(s,\, P)$
(1) $\mathbf{divergence}(s,\, P)$ implies $\mathcal{T}(s,\, P) \ =_A \ \bot$, and
(2) $\mathbf{finitary}(s,\, P)$ if P does not contain μ operator.
(3) $\mathcal{T}(s,\, P) \ =_A \ \oplus \{r : \mathcal{T}(t,\, Q) \mid (s,\, P) \xrightarrow{r} (t,\, Q)\}$,
where we extend the definition of the test operator to deal with the empty program text ϵ by

$$\mathcal{T}(s,\, \epsilon) \ =_{df} \ s$$

Theorem 5.1. Let $\rightarrow$ be a consistent transition system.
If $\mathbf{finitary}(s,\, P)$, then there exists n such that

$$\mathcal{T}(s,\, P) \ =_A \ \oplus \, \{r \ : \ t \mid \exists m \bullet (m \leq n) \wedge (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)\}$$

Otherwise

$$\mathcal{T}(s,\, P) \ =_A \ \bigsqcup_n \oplus \{r \ : \ t \mid \exists m \bullet (m \leq n) \wedge (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)\}$$

Proof. (1) Assume that $\mathbf{finitary}(s,\, P)$. For $k > 0$ define

$$\mathbf{finitary}_k(s,\, P) \ =_{df} \ \forall (t,\, Q), \forall r,\, m \bullet \begin{pmatrix} (s,\, P) \xrightarrow{r}_m (t,\, Q) \wedge m > k \\ \implies \mathbf{divergence}(t,\, Q) \end{pmatrix}$$

The following inductive proof is based on the length of transition sequences
Basic case: $\mathbf{finitary}_1(s,\, P)$.
The conclusion directly follows from (1) and (3) of Definition 5.1.
Induction step: $\mathbf{finatary}_{k+1}(s,\, P)$.
From the definition of $\mathbf{finitary}_{n+1}$ it follows that

$$(s,\, P) \xrightarrow{r} (t,\, Q) \implies \mathbf{finitary}_k(t,\, Q) \qquad (*)$$

$$\begin{array}{lll} & \mathcal{T}(s,\, P) & \{\text{Def } 5.1(3)\} \\ =_A & \oplus\{r : \mathcal{T}(u,\, Q) \mid (s,\, P) \xrightarrow{r} (u,\, Q)\} & \{(*) \text{ and inductive hypothesis}\} \\ =_A & \oplus \left\{ \begin{array}{l} r : \oplus\{\lambda : t \mid \exists m \bullet (m \le n) \wedge \\ (s,\, P) \xrightarrow{r} (u,\, Q) \wedge (u,\, Q) \xrightarrow{\lambda}_m (t,\, \epsilon)\} \end{array} \right\} & \{\oplus - 4 \text{ Let } l = n+1\} \\ =_A & \oplus\{\beta : t \mid \exists m \le l \bullet (s,\, P) \xrightarrow{\beta} (t, \epsilon)\} & \end{array}$$

(2) Consider the case where $\neg$**finitary**$(s,\, P)$.
First we are going to establish the inequality

$$\mathcal{T}(s,\, P) \sqsupseteq_A \bigsqcup_n \oplus\{r\,:\,t \mid \exists m \le n \bullet (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)$$

By **norm**-2 we are required to prove for all n

$$\mathcal{T}(s,\, P) \sqsupseteq_A \oplus\{r\,:\,t \mid \exists m \bullet (m \le n) \wedge (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)\}$$

Basic case: $n = 1$.

$$\begin{array}{lll} & \mathcal{T}(s,\, P) & \{\text{Def } 5.1(3)\} \\ =_A & \oplus \left\{ \begin{array}{l} \{\lambda\,:\,\mathcal{T}(t,\, Q) \mid (s,\, P) \xrightarrow{\lambda} (t,\, Q)\} \cup \\ \{\beta\,:\,t \mid (s,\, P) \xrightarrow{\beta} (t,\, \epsilon)\} \end{array} \right\} & \{\text{Theorem } 3.11(c)\} \\ \sqsupseteq_A & \oplus \left\{ \begin{array}{l} \{\lambda\,:\,\bot \mid (s,\, P) \xrightarrow{\lambda} (t,\, Q)\} \cup \\ \{\beta\,:\,t \mid (s,\, P) \xrightarrow{\beta} (t,\, \epsilon)\} \end{array} \right\} & \{\oplus - 2\} \\ =_A & \oplus\{r\,:\,t \mid (s,\, P) \xrightarrow{r} (t,\, \epsilon)\} & \end{array}$$

Induction:

$$\begin{array}{lll} & \mathcal{T}(s,\, P) & \{\text{Def } 5.1(3)\} \\ =_A & \oplus \left\{ \begin{array}{l} \{\lambda\,:\,\mathcal{T}(t,\, Q) \mid (s,\, P) \xrightarrow{\lambda} (t,\, Q)\} \cup \\ \{\beta\,:\,t \mid (s,\, P) \xrightarrow{\beta} (t,\, \epsilon)\} \end{array} \right\} & \{\text{inductive hypothesis}\} \\ \sqsupseteq_A & \oplus \left\{ \begin{array}{l} \{\lambda\,:\,\oplus\{\,\gamma : u \mid (s,\, P) \xrightarrow{\lambda} (t,\, Q) \;\wedge \\ \exists m \le n \bullet (t,\, Q) \xrightarrow{\gamma}_m (u,\, \epsilon)\}\} \cup \\ \{\beta\,:\,t \mid (s,\, P) \xrightarrow{\beta} (t,\, \epsilon)\} \end{array} \right\} & \{\text{Def } \xrightarrow{r}_n\} \\ =_A & \oplus\{r\,:\,t \mid \exists m \le n+1 \bullet (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)\} & \end{array}$$

Now we are going to prove the inequality

$$\mathcal{T}(s,\, P) \sqsubseteq_A \bigsqcup_n \oplus\{r\,:\,t \mid \exists m \bullet (m \le n) \wedge (s,\, P) \xrightarrow{r}_m (t,\, \epsilon)\}$$

From Definition 5.1 (2) we conclude that P must contain μ operator. Let us begin with the simplest case:

$$P \;=\; \mu X \bullet F(X)$$

where $F(X)$ does not refer to μ operator. Clearly from Definition 5.1(2) we have for all n
(i) **finitary**$(F^n(\bot))$
By induction it can be shown that
(ii) $(s,\, F^n(\bot)) \xrightarrow{\lambda}_m (t,\, \epsilon) \implies \exists k \le n \bullet (s,\, \mu X \bullet F(X)) \xrightarrow{\lambda}_{m+k} (t,\, \epsilon)$
From (i) it follows that for all n there exists k_n such that

$$\begin{array}{lll}
 & \mathcal{T}(s,\, F^n(\bot)) & \{\textbf{finitary}(F^n(\bot))\} \\
=_A & \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le k_n) \wedge \\ (s,\, F^n(\bot)) \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} & \{(ii) \text{ and Corollary of } \textbf{norm}-1\} \\
\sqsubseteq_A & \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le (k_n+n)) \wedge \\ (s,\, \mu X \bullet F(X)) \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} & \{\textbf{norm}-3\} \\
\sqsubseteq_A & \bigsqcup_n \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le n) \wedge \\ (s,\, \mu X \bullet F) \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} &
\end{array}$$

which leads to the conclusion

$$\begin{array}{lll}
 & \mathcal{T}(s,\, \mu X \bullet F(X)) & \{\text{Theorem 3.8 and } \textbf{rec}-1\} \\
=_A & \bigsqcup_n \mathcal{T}(s,\, F^n(\bot)) & \{\text{previous conclusion}\} \\
\sqsubseteq_A & \bigsqcup_n \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le n) \wedge \\ (s,\, \mu X \bullet F) \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} &
\end{array}$$

Finally let us examine the case where

$$P \;=\; F(\mu X \bullet Q(X), .., \mu X \bullet R(X))$$

In a similar way we can prove

$$\begin{array}{lll}
 & \bullet\ \mathcal{T}(s,\, F(\mu X \bullet Q(X),\, .., \mu X \bullet R(X))) & \{\text{Theorem 3.8}\} \\
=_A & \bigsqcup_n \mathcal{T}(s,\, F(Q^n(\bot), .., R^n(\bot))) & \{\textbf{finitary}(F(Q^n(\bot), .., R^n(\bot)))\} \\
=_A & \bigsqcup_n \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le n_k) \wedge \\ (s,\, F(Q^n(\bot), ...)) \\ \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} & \{\text{proof for } (P \;=\; \mu X \bullet F(X))\} \\
\sqsubseteq_A & \bigsqcup_n \oplus\{r:t \mid \begin{pmatrix}\exists m \bullet (m \le n) \wedge \\ (s,\, P) \\ \xrightarrow{r}_m (t,\, \epsilon)\end{pmatrix}\} &
\end{array}$$

We propose the following transition system for our probabilistic programming language.

Definition 5.2.
(1) Assignment
$((v := c),\, v := e) \rightarrow ((v := e[c/v]),\, \epsilon)$.
(2) Probabilistic Choice
(a) $((v := c),\ \oplus\ \{r_1 : P_1, .., r_m; P_m\}) \xrightarrow{r_k[c/v]} ((v := c),\, P_k)$
provided that $r_k[c/v] > 0$
(b) $((v := c),\ \oplus\ \{r_1 : P_1, .., r_m; P_m\}) \xrightarrow{1-\sum_k r_k[c/v]} ((v := c),\, \bot)$
provided that $\sum_k r_k[c/v] < 1$.
(3) Conditional
(a) $((v := c),\ P \lhd b \rhd Q) \rightarrow ((v := c),\, P)$ if $b[c/v] = true$
(b) $((v := c),\ P \lhd b \rhd Q) \rightarrow ((v := c),\, Q)$ if $b[c/v] = false$
(4) Composition
(a) $(s,\, P; Q) \xrightarrow{r} (t,\, R; Q)$ if $(s,\, P) \xrightarrow{r} (t,\, R)$
(b) $(s,\, P; Q) \rightarrow (t,\, R; Q)$ if $(s,\, P) \rightarrow (t,\, R)$
(c) $(s,\, P; Q) \xrightarrow{r} (t,\, Q)$ if $(s,\, P) \xrightarrow{r} (t,\, \epsilon)$
(d) $(s,\, P; Q) \rightarrow (t,\, Q)$ if $(s,\, P) \rightarrow (t,\, \epsilon)$
(5) Recursion
$(s,\, \mu X \bullet P(X)) \rightarrow (s,\, P(\mu X \bullet P(X)))$
(6) Chaos
$(s,\, \bot) \rightarrow (s,\, \bot)$

We are going to show that Definition 5.2 gives a consistent transition system. First, we show that the given transition system satisfies Definition 5.1(3)

Lemma 5.2. $\mathcal{T}(s,\, P) =_A \oplus\{r : \mathcal{T}(t,\, Q) \mid (s,\, P) \xrightarrow{r} (t,\, Q)\}$

Proof. Direct from the following properties of the test operator $\mathcal{T}$:
(1) From $\oplus$-6 it follows that
$\mathcal{T}((v := c),\, \oplus\{r_1 : P_1, .., r_n : P_n\} \;=_A\; \oplus\{r_1[c/v] : \mathcal{T}((v := c),\, P_1), ..., r_n[c/v] : \mathcal{T}((v := c),\, P_n)\}$
(2) From Theorem 2.1(7) we obtain
$\mathcal{T}((v := c),\, (P \lhd b \rhd Q)) \;=_A\; \mathcal{T}((v := c),\, P) \lhd b[c/v] \rhd \mathcal{T}((v := c),\, Q)$
(3) From Theorem 4.1 we have
$\mathcal{T}((v := c),\, (P; Q)) \;=_A\; \oplus\{r_1 : \mathcal{T}((v := d_1),\, Q), ..., r_m : \mathcal{T}((v := d_m),\, Q)\}$
provided that $\mathcal{T}((v := c),\, P) \;=_A\; \oplus\{r_1 : (v := d_1), .., r_m : (v := d_m\}$
(4) $\mathcal{T}(s,\, \mu X \bullet P(X)) \;=_A\; \mathcal{T}(s,\, P(\mu X \bullet P(X)))$
Next we deal with the condition (1) of Definition 5.1.

Lemma 5.3. If P is a finite program, then

$$\mathbf{divergence}(s,\, P) \implies \mathcal{T}(s,\, P) =_A \perp$$

Proof. We give an induction proof based on the structure of program text P:
Base case: Clearly the conclusion holds for the case $P = v := e$ and $P = \perp$
Inductive step:

$$\begin{array}{lll}
& \mathbf{divergence}((v := c),\ (P \lhd b \rhd Q)) & \{\text{Rule (3) in Definition 5.2}\} \\
\implies & \begin{pmatrix} \mathbf{divergence}((v := c),\, P) \\ \lhd b[c/v] \rhd \\ \mathbf{divergence}((v := c),\, Q) \end{pmatrix} & \{\text{Induction hypothesis}\} \\
\implies & \begin{pmatrix} (\mathcal{T}((v := c),\, P) =_A \perp) \\ \lhd b[c/v] \rhd \\ (\mathcal{T}((v := c),\, Q) =_A \perp) \end{pmatrix} & \{\text{Theorem 2.1(7)}\} \\
\implies & \mathcal{T}((v := c),\ (P \lhd b \rhd Q)) =_A \perp & \\
& \mathbf{divergence}((v := c),\ \oplus\{r_1 : P_1, .., r_n : P_n\}) & \{\text{Rule (2) in Definition 5.2}\} \\
\implies & \Sigma\{(r_k[c/v]) | \mathbf{divergence}((v := c),\, P_k)\} = 1 & \{\text{Induction hypothesis}\} \\
\implies & \Sigma\{(r_k[c/v]) | \mathcal{T}((v := c),\, P_k) \;=_A\; \perp\} = 1 & \{\oplus - 3 \text{ and } 6\} \\
\implies & \mathcal{T}((v := c),\ \oplus\{r_1 : P_1, .., r_n : P_n\}) =_A \perp &
\end{array}$$

Finally we are going to tackle infinite programs.

Lemma 5.4. If $(s,\, G(Q)) \rightarrow_* (t,\, \epsilon)$,
then either $\mathbf{divergence}(s,\, G(\perp))$ or $(s,\, G(\perp)) \rightarrow_* (t,\, \epsilon)$.

Proof. Induction on the structure of G
Base case. $G(Q) = Q$ From Rule (6)

$$(s,\, \perp) \rightarrow (s,\, \perp)$$

in Definition 5.2.
Inductive step:
(1) $G(Q) = G_1(Q) \lhd b \rhd G_2(X)$

$$\begin{array}{ll}
\quad (s,\ G(Q)) \to_* (t,\ \epsilon) & \{\text{Rule (3) in Def 5.2}\} \\
\Longrightarrow \begin{pmatrix} (s,\ G_1(Q)) \to_* (t,\ \epsilon) \\ \lhd(s;b)\rhd \\ (s,\ G_2(Q)) \to_* (t,\ \epsilon) \end{pmatrix} & \{\text{Induction hypothesis}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ G_1(\bot))\ \vee\ (s,\ G_1(\bot)) \to_* (t,\ \epsilon) \\ \lhd(s;b)\rhd \\ \mathbf{divergence}(s,\ G_2(\bot))\ \vee\ (s,\ G_2(\bot)) \to_* (t,\ \epsilon) \end{pmatrix} & \{\text{Rule (3) in Def 5.2}\} \\
\Longrightarrow \mathbf{divergence}(s.\, G(\bot)) \vee (s,\ G(\bot)) \to_* (t,\ \epsilon) &
\end{array}$$

(2) $G(Q) = \oplus\{\alpha_1 : G_1(Q),\ ...\ \alpha_k : G_k(Q)\}$. Similar to Case (1).

(3) $G(Q) = G_1(Q); G_2(Q)$

$$\begin{array}{ll}
\quad (s,\ G(Q)) \to_* (t,\ \epsilon) & \{\text{Rule (4) in Def 5.2}\} \\
\Longrightarrow \exists u \bullet \begin{pmatrix} (s,\ G_1(Q)) \to_* (u,\ \epsilon) \vee \\ (u,\ G_2(Q)) \to_* (t,\ \epsilon) \end{pmatrix} & \{\text{Induction hypothesis}\} \\
\Longrightarrow \exists u \bullet \begin{pmatrix} \mathbf{divergence}(s,\ G_1(\bot))\ \vee \\ (s,\ G_1(\bot)) \to_* (u,\ \epsilon)\ \vee \\ \mathbf{divergence}(u,\ G_2(\bot))\ \vee \\ (u,\ G_2(\bot)) \to_* (t,\ \epsilon) \end{pmatrix} & \{\text{Rule (4) in Def 5.2}\} \\
\Longrightarrow \mathbf{divergence}(s.\, G(\bot)) \vee (s,\ G(\bot)) \to_* (t,\ \epsilon) &
\end{array}$$

(4) $G(Q) = \mu X \bullet P(Q,\ X)$

$$\begin{array}{ll}
\quad (s,\ \mu X \bullet P(Q,\ X)) \to_* (t,\ \epsilon) & \{\text{Rule (5) in Def 5.2}\} \\
\Longrightarrow (s,\ P(Q,\ \mu X \bullet P(Q,\ X))) \to_* (t,\ \epsilon) & \{\text{Induction hypothesis}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ P(\bot,\ \mu X \bullet P(\bot,\ X))) \vee \\ (s,\ P(\bot,\ \mu X \bullet P(\bot,\ X))) \to_* (t,\ \epsilon) \end{pmatrix} & \{\text{Rule (5) in Def 5.2}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ \mu X \bullet P(\bot,\ X)) \vee \\ (s,\ \mu X \bullet P(\bot,\ X)) \to_* (t,\ \epsilon) \end{pmatrix} &
\end{array}$$

Lemma 5.5.
(1) $\mathbf{divergence}(s,\ F(P)) \implies \mathbf{divergence}(s,\ \mathcal{F}(\bot))$
(2) $\mathbf{divergence}(s,\ F(\mu X \bullet P(X)) \implies \mathbf{divergence}(s,\ F(P(\mu X \bullet P(X))))$

Proof. (1) Based on induction on the structure of F.
Base case: $F(X) = X$. The conclusion follows from the Rule (6) in Definition 5.2.
Inductive Step:

$$\begin{array}{ll}
\quad \mathbf{divergence}(s,\ F_1(Q) \lhd b \rhd F_2(Q)) & \{\text{Rule (3) in Def 5.2}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ F_1(Q)) \\ \lhd(s;b)\rhd \\ \mathbf{divergence}(s,\ F_2(Q)) \end{pmatrix} & \{\text{induction hypothesis}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ F_1(\bot)) \\ \lhd(s;b)\rhd \\ \mathbf{divergence}(s,\ F_2(\bot)) \end{pmatrix} & \{\text{Rule (3) in Def 5.2}\} \\
\Longrightarrow \mathbf{divergence}((s,\ (F_1(\bot) \lhd b \rhd F_2(\bot)))) &
\end{array}$$

$$\begin{array}{ll}
\quad \mathbf{divergence}(s,\ F_1(Q); F_2(Q)) & \{\text{Rule (4) in Def 5.2}\} \\
\Longrightarrow \mathbf{divergence}(s,\ F_1(Q)) \vee & \\
\quad \exists t \bullet (s,\ F_1(Q)) \to_* (t,\ \epsilon)\ \wedge\ \mathbf{divergence}(t,\ F_2(Q)) & \{\text{Lemma 5.4}\} \\
\Longrightarrow \begin{pmatrix} \mathbf{divergence}(s,\ F_1(\bot)) \vee \\ (s,\ F_1(\bot)) \to_* (t,\ \epsilon)\ \wedge\ \mathbf{divergence}(t,\ F_2(\bot)) \end{pmatrix} & \{\text{Rule (4) in Def 5.2}\} \\
\Longrightarrow \mathbf{divergence}(s,\ F_1(\bot); F_2(\bot)) &
\end{array}$$

Lemma 5.6.
$\mathbf{divergence}(s,\ F(\mu X \bullet P(X)) \implies \mathcal{T}(s,\ F(\mu X \bullet P(X))) =_A \bot$

Proof.

$$\begin{array}{lr}
\quad \mathbf{divergence}(s,\ F(\mu X \bullet P(X))) & \{\text{Lemma 5.5(2)}\} \\
\Longrightarrow \forall n \bullet \mathbf{divergence}(s,\ F(P^n(\mu X \bullet P(X)))) & \{\text{Lemma 5.5(1)}\} \\
\Longrightarrow \forall n \bullet \mathbf{divergence}(s,\ F(P^n(\bot))) & \{\text{Lemma 5.3}\} \\
\Longrightarrow \forall n \bullet \mathcal{T}(s,\ F(P^n(\bot))) =_A \bot & \{\text{Theorem 3.8}\} \\
\Longrightarrow \mathcal{T}(s,\ F(\mu X \bullet P(X))) =_A \bot &
\end{array}$$

Lemma 5.7.
If P is finite, then **fnitary**$(s,\ P)$ holds for all states s

Proof. On structural induction

Combining Lemmas 5.2, 5.6 and 5.7 we conclude

Theorem 5.8.
The transition system defined in Definition 5.2 is consistent.

6 Conclusions

This paper begins with an algebraic framework for our probabilistic programming language, and then shows how to deliver the corresponding denotational and operational representations consistently. The main contributions include:

- Clarify the type of observations we are able to record during the execution of a probabilistic programs:
  - The behaviour of a program cannot simply be modelled as a relation between the initial data state and a finite distribution on the possible final data states.
  - The normal approach permits us to distinguish a program which can terminate and deliver a final distribution function from a program which can only generate an approximate distribution function during its everlasting execution.
- The test algebra lays down the foundation for construction of a denotational framework for our probabilistic programming language.
- The consistency of an operational approach against the algebra of programs can be formalised and validated within the algebra of programs.

The language we put forward in this paper has not included the nondeterministic choice operator given in the traditional programming languages. As a result, we lose the case where the probabilistic choice can be identified as a refinement of the nondeterministic choice. Moreover, the refinement order in the conventional languages was directly induced from the choice operator, whereas we were forced to adopt an inductive definition in Sects. 1 and 2 based on finite and infinite normal forms. Consequently, it makes the proof of monotonicity of programming combinators in this paper look cumbersome.
In future, we will investigate a language armed with both probabilistic and nondeterministic choice operators, and follow up the algebraic approach advocated in this paper to explore the links among various programming presentations for the probabilistic languages.

References

1. Jifeng, H., Qin, L.: A new roadmap for linking theories of programming and its applications on GCL and CSP. Sci. Comput. Program. Elsevier **162**, 3–34 (2018)
2. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge Press, Cambridge (1996)
3. Abrial, J.-R.: Modelling in Event-B: System and Software Engineering. Cambridge Press, Cambridge (2010)
4. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
5. Hehner, E.C.R.: A more complete model of communicating processes. Theor. Comput. Sci. **26**, 105–120 (1983)
6. Hehner, E.C.R.: Predicative programming, part 1 and 2. Commun. ACM **27**(2), 134–151 (1984)
7. Hennessy, M.C.: Algebraic Theory of Process. The MIT Press, Cambridge (1988)
8. Hoare, C.A.R., et al.: Laws of programming. Commun. ACM **30**(8), 672–686 (1987)
9. Jones, C.B.: Systematic Software Development Using VDM. Prentice Hall, New York (1986)
10. Plotkin, G.D.: A structural approach to operational semantics. Technical Report, DAIMI-FN-19, Aarhus University, Denmark (1981)
11. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice Hall, New York (1992)
12. Jifeng, H., Seidel, K., McIver, A.: Probabilistic models for the guarded command language. Sci. Comput. Program. **28**(2–3), 171–192 (1997)