Multi-Dimensional Probabilistic Ranking (Provisional Title)

ANONYMOUS AUTHOR(S)

ACM Reference format:

Anonymous Author(s). 2017. Multi-Dimensional Probabilistic Ranking (Provisional Title). 1, 1, Article 1 (May 2017), 6 pages.

https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PRELIMINARIES

1.1 Basic Notions, Linear Predicates, Valuations

For a set A we denote by $|A|$ the cardinality of A. We denote by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of all positive integers, non-negative integers, integers, and real numbers, respectively. We assume basic knowledge of matrix calculus. We use boldface notation for vectors, e.g. $\mathbf{x}$, $\mathbf{y}$, etc., and we denote an i-th component of a vector $\mathbf{x}$ by $\mathbf{x}[i]$. For the purpose of matrix calculations we assume that (non-transposed) vectors are row vectors. If $\mathbf{v}, \mathbf{v}'$ are n and m dimensional vectors, respectively, then $(\mathbf{v}, \mathbf{v}')$ is an $(n+m)$-dimensional vector obtained by "concatenation" of $\mathbf{v}$ and $\mathbf{v}'$. We identify 1-dimensional vectors with numbers. For an n-dimensional vector $\mathbf{x}$, index $1 \leq i \leq n$, and number a we denote by $\mathbf{x}(i \leftarrow a)$ a vector $\mathbf{y}$ such that $\mathbf{y}[i] = a$ and $\mathbf{y}[j] = \mathbf{x}[j]$ for all $1 \leq j \leq n$, $j \neq i$. For comparison of vectors (e.g. as in $\mathbf{x} \leq \mathbf{y}$), we consider componentwise comparison. For comparing functions f, g with the same domains, we write $f \leq g$ if $f(x) \leq g(x)$ for all x in the domain.

Variables and valuations. Throughout the paper we fix a countable set of variables $\mathcal{V}$. We consider some arbitrary but fixed linear order on the set of all variables. Hence, given some set of variables V we can enumerate its members in ascending order (w.r.t. the fixed ordering) and write $V = \{x_1, x_2, x_3, \dots\}$.

Affine expressions. An *affine expression* over the set of variables $\{x_1, \dots, x_n\}$ is an expression of the form $d + \sum_{i=1}^{n} a_i x_i$, where $d, a_1, \dots, a_n$ are real-valued constants. Each affine expression E over $\{x_1, \dots, x_n\}$ determines a function which for each m-dimensional vector $\mathbf{x}$, where $m \geq n$, returns a number resulting from substituting each x_i in E by $\mathbf{x}[i]$. Slightly abusing our notation, we denote this function also by E and the value of this function on argument $\mathbf{x}$ by $E(\mathbf{x})$. A function of the form $E(\mathbf{x})$ for some affine expression E is called affine.

XXXX-XXXX/2017/5-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Linear constraint, assertion, predicates. We use the following nomenclature:

- *Linear Constraint.* A *linear constraint* is a formula of the form ψ or $\neg\psi$, where ψ is a non-strict inequality between affine expressions.
- *Linear Assertion.* A *linear assertion* is a finite conjunction of linear constraints.
- *Propositionally Linear Predicate.* A *propositionally linear predicate* (PLP) is a finite disjunction of linear assertions.

Arity and satisfaction of PLP. For a PLP φ we denote by $\mathcal{V}(\varphi)$ the set of all variables that appear in φ. As noted above, we stipulate that $\mathcal{V}(\varphi) = \{x_1, \ldots, x_{n(\varphi)}\}$ for some $n(\varphi) \in \mathbb{N}$. A vector $\mathbf{x}$ of dimension $m \geq n(\varphi)$ *satisfies* φ, we write $\mathbf{x} \models \varphi$, if the arithmetic formula obtained by substituting each occurrence of a variable x_i in φ by $\mathbf{x}[i]$ is valid. We denote $[\![\varphi]\!] = \{\mathbf{x} \in \mathbb{R}^m \mid m \geq n(\varphi) \wedge \mathbf{x} \models \varphi\}$ and $[\![\varphi]\!]^{\triangleleft} = [\![\varphi]\!] \cap \mathbb{R}^{n(\varphi)}$.

1.2 Syntax of Affine Probabilistic Programs (Apps)

The Syntax. We consider the standard syntax for affine probabilistic programs, which encompasses basic programming mechanisms such as assignment statement (indicated by ':='), while-loop, if-branch. We also consider basic probabilistic mechanisms such as probabilistic branch (indicated by 'prob') and random sampling (e.g. $x :=$ **sample(**Uniform$[-2, 1]$**)** assigns to x a random number uniformly sampled from interval $[-2, 1]$). We also allow constructs for (demonic) non-determinism, in particular non-deterministic branching indicated by '**if** $\star$ **then...**' construct and non-deterministic assignment. Variables (or identifiers) of a probabilistic program are of *real* type, i.e., values of the variables are real numbers. We allow only affine expressions in test statements and in the right-hand sides of assignments. We also assume that assume that each App $\mathcal{P}$ is preceded by an initialization preamble in which each variable appearing in $\mathcal{P}$ is assigned some concrete number. Due to space restrictions, details (such as grammar) are relegated to the Appendix. For an example see Figure 1. We refer to this class of affine probabilistic programs as Apps.

1.3 Semantics of Affine Probabilistic Programs

We now formally define the semantics of App's. In order to do this, we first recall some fundamental concepts from probability theory.

Basics of Probability Theory. The crucial notion is the one of a probability space. A probability space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a non-empty set (so called *sample space*), $\mathcal{F}$ is a *sigma-algebra* over Ω, i.e. a collection of subsets of Ω that contains the empty set $\emptyset$, and that is closed under complementation and countable unions, and $\mathbb{P}$ is a *probability measure* on $\mathcal{F}$, i.e., a function $\mathbb{P} \colon \mathcal{F} \longrightarrow [0, 1]$ such that

- $\mathbb{P}(\emptyset) = 0$,
- for all $A \in \mathcal{F}$ it holds $\mathbb{P}(\Omega \setminus A) = 1 - \mathbb{P}(A)$, and
- for all pairwise disjoint countable set sequences $A_1, A_2, \dots \in \mathcal{F}$ (i.e., $A_i \cap A_j = \emptyset$ for all $i \neq j$) we have $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}(\bigcup_{i=1}^{\infty} A_i)$.

Random variables and filtrations. A *random variable* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $R \colon \Omega \longrightarrow \mathbb{R} \cup \{\infty\}$, i.e., a function such that for every $a \in \mathbb{R} \cup \{\infty\}$ the set $\{\omega \in \Omega \mid R(\omega) \leq a\}$ belongs to $\mathcal{F}$. We denote by $\mathbb{E}[R]$ the *expected value* of a random variable X (see [1, Chapter 5] for a formal definition). A *random vector* in $(\Omega, \mathcal{F}, \mathbb{P})$ is a vector whose every component is a random variable in this probability space. A *stochastic process* in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an infinite sequence of random vectors in this space. We will also use random variables of the form $R\colon \Omega \longrightarrow S$ for some finite set S, which is easily translated to the variables above. A *filtration* of a sigma-algebra $\mathcal{F}$ is a sequence $\{\mathcal{F}_i\}_{i=0}^{\infty}$ of σ-algebras such that $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n \subseteq \cdots \subseteq \mathcal{F}$.

Distributions. We assume the standard definition of a probability distribution specified by a cumulative distribution function [1]. We denote by $\mathcal{D}$ be a set of probability distributions on real numbers, both discrete and continuous.

Probabilistic Control Flow Graphs. The semantics can be defined as the semantics of an uncountable state-space Markov decision process (MDP) (uncountable due to real-valued variables). We take an operational approach to define the semantics, and associate to each program a certain stochastic process [2, 5, 6]. To define this process, we first define so called *probabilistic control flow graphs* [3].

Definition 1.1. A *probabilistic control flow graph (pCFG)* is a tuple $C = (L, V, \ell_{init}, \mathbf{x}_{init}, \mapsto, Pr, G)$, where

- L is a finite set of *locations* partitioned into three pairwise disjoint subsets L_N, L_P, and L_D of non-deterministic, probabilistic, and deterministic locations;
- $V = \{x_1, \ldots, x_{|V|}\}$ is a finite set of *program variables* (note that $V \subseteq \mathcal{V}$) ;
- ℓ_{init} is an initial location and $\mathbf{x}_{init}$ is an initial *assignment vector*;
- $\mapsto$ is a transition relation, whose members are tuples of the form (ℓ, i, u, ℓ'), where ℓ and ℓ' are source and target program locations, respectively, $1 \leq i \leq |V|$ is a *target variable index*, and u is an *update element*, which can be one of the following mathematical objects: (a) an affine function $u\colon \mathbb{R}^{|V|} \longrightarrow \mathbb{R}$; (b) a distribution $d \in \mathcal{D}$; or (c) a set $R \subseteq \mathbb{R}$.
- $Pr = \{Pr_\ell\}_{\ell \in L_P}$ is a collection of probability distributions, where each Pr_ℓ is a discrete probability distribution on the set of all transitions outgoing from ℓ.
- G is a function assigning a propositionally linear predicate (a *guard*) over V to each transition outgoing from a deterministic location.

We assume that each location has at least one outgoing transition. Also, for every deterministic location ℓ we assume the following: if $\tau_1, \ldots, \tau_k$ are all transitions outgoing from ℓ, then $G(\tau_1) \vee \cdots \vee G(\tau_k) \equiv$ *true* and $G(\tau_i) \wedge G(\tau_j) \equiv$ *false* for each $1 \leq i < j \leq k$. Moreover, for each distribution d appearing in the pCFG we assume the following features are known: expected value $\mathbb{E}[d]$ of d and a single-variable PLP φ_d such that the *support* of d (i.e. the smallest closed set of real numbers whose complement has probability zero under d)[1] satisfies $supp(d) \subseteq [\![\varphi_d]\!]^{\triangleleft}$. Finally, we assume that for each transition (ℓ, j, u, ℓ') such that u is a set the location ℓ is deterministic. This is just a technical assumption yielding no loss of generality, and it somewhat simplifies notation.

Configurations. A *configuration* of a pCFG C is a tuple $(\ell, \mathbf{x})$, where ℓ is a location of C and $\mathbf{x}$ is an $|V|$-dimensional vector. We say that a transition τ is *enabled* in a configuration $(\ell, \mathbf{x})$ if ℓ is the source location of τ and in addition, $\mathbf{x} \models G(\tau)$ provided that ℓ is deterministic. A configuration $(\ell, \mathbf{x})$ is non-deterministic/probabilistic/deterministic if ℓ is non-deterministic/probabilistic/deterministic, respectively.

Executions and reachable configurations. A *finite path* (or *execution fragment*) in C is a finite sequence of configurations $(\ell_0, \mathbf{x}_0) \cdots (\ell_k, \mathbf{x}_k)$ such that for each $0 \leq i < k$ there is a transition $(\ell_i, j, u, \ell_{i+1})$ enabled in $(\ell_i, \mathbf{x}_i)$ such that $\mathbf{x}_{i+1} = \mathbf{x}_i(j \leftarrow a)$ where a satisfies one of the following:

- u is a function $f\colon \mathbb{R}^{|X|} \longrightarrow \mathbb{R}$ and $a = f(\mathbf{x}_i)$;

[1]In particular, a support of a *discrete* probability distribution d is simply the at most countable set of all points on a real line that have positive probability under d.

- u is an integrable[2] distribution d and $a \in \mathit{supp}(d)$; or
- u is a set and $a \in u$.

A *run* (or *execution*) in C is an infinite sequence of configurations whose every finite prefix is a finite path. A configuration $(\ell, \mathbf{x})$ is *reachable* from the initial configuration $(\ell_{init}, \mathbf{x}_{init})$ if there is a finite path starting in $(\ell_{init}, \mathbf{x}_{init})$ that ends in $(\ell, \mathbf{x})$.

Schedulers. Due to the presence of non-determinism and probabilistic choices, a pCFG C may represent a multitude of possible behaviours. The probabilistic behaviour of C can be captured by constructing a suitable probability measure over the set of all its runs. Before this can be done, non-determinism in C needs to be resolved. This is done using the standard notion of a *scheduler.*

Definition 1.2 (Schedulers). A scheduler in an pCFG C is a tuple $\sigma = (\sigma_t, \sigma_a)$, where

- σ_t (here 't' stands for 'transition') is a function assigning to every finite path that ends in a non-deterministic configuration $(\ell, \mathbf{x})$ a probability distribution on transitions outgoing from ℓ; and
- σ_a (here 'a' stands for 'assignment') is a function which takes as an argument a finite path ending in a deterministic configuration in which some transition (ℓ, j, u, ℓ') with u being a set is enabled, and for such a path it returns a probability distribution on u.

Stochastic process. A pCFG C together with a scheduler σ can be seen as a stochastic process which produces a random run $(\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)(\ell_2, \mathbf{x}_2)\cdots$. The evolution of this process can be informally described as follows: we start in the initial configuration, i.e. $(\ell_0, \mathbf{x}_0) = (\ell_{init}, \mathbf{x}_{init})$. Now assume that i steps have elapsed, i.e. a finite path $(\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\cdots(\ell_i, \mathbf{x}_i)$ has already been produced. Then

- A transition $\tau = (\ell, j, u, \ell')$ enabled in $(\ell_i, \mathbf{x}_i)$ is chosen as follows:
  - If ℓ_i is non-deterministic then τ is chosen randomly according to the distribution specified by scheduler σ, i.e. according to the distribution $\sigma_t((\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\cdots(\ell_i, \mathbf{x}_i))$.
  - If ℓ_i is probabilistic, then τ is chosen randomly according to the distribution Pr_{ℓ_i}.
  - If ℓ_i is deterministic, then by the definition of a pCFG there is exactly one enabled transition outgoing from ℓ_i, and this transition is chosen as τ.
- Once τ is chosen as above, we put $\ell_{i+1} = \ell'$. Next, we put $\mathbf{x}_{i+1} = \mathbf{x}_i(j \leftarrow a)$, where a chosen as follows:
  - If u is a function $u \colon \mathbb{R}^{|V|} \to \mathbb{R}$, then $a = f(\mathbf{x}_i)$.
  - If u is a distribution d, then a is sampled from d.
  - If u is a set, then a is sampled from a distribution $\sigma_a((\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\cdots(\ell_i, \mathbf{x}_i))$.

The above intuitive explanation can be formalized by showing that each pCFG C together with a scheduler σ uniquely determines a certain probabilistic space $(\Omega, \mathcal{R}, \mathbb{P}^\sigma)$ in which Ω is a set of all runs in C, and a stochastic process $C^\sigma = \{\mathrm{C}_i^\sigma\}_{i=0}^\infty$ in this space such that for each $\varrho \in \Omega$ we have that $\mathrm{C}_i^\sigma(\varrho)$ is the i-th configuration on run ϱ (i.e., C_i^σ is a random vector $(\ell_i^\sigma, \mathbf{x}_i^\sigma)$ with ℓ_i^σ taking values in L and $\mathbf{x}_i^\sigma$ being a random vector of dimension $|V|$ consisting of real-valued random variables). The sigma-algebra $\mathcal{R}$ is the smallest (w.r.t. inclusion) sigma algebra under which all the functions C_i^σ, for all $i \geq 0$ and all schedulers σ, are $\mathcal{R}$-measurable (a function f returning vectors is $\mathcal{R}$-measurable if for all real-valued vectors $\mathbf{y}$ of appropriate dimension the set $\{\omega \in \Omega \mid f(\omega) \leq \mathbf{y}\}$ belongs to $\mathcal{R}$). The probability measure $\mathbb{P}^\sigma$ is such that for each i, the distribution of C_i^σ reflects the aforementioned way in which runs are randomly generated. The formal construction of $\mathcal{R}$ and $\mathbb{P}^\sigma$ is

[2]A distribution on some numerical domain is integrable if its expected value exists and is finite. In particular, each Dirac distribution is integrable.

standard [1] and somewhat technical, hence we omit it. We denote by $\mathbb{E}^\sigma$ the expectation operator in probability space $(\Omega, \mathcal{R}, \mathbb{P}^\sigma)$. The translation from probabilistic programs to the corresponding pCFG is standard [4]. We point out that the construction produces pCFGs with a property that only transitions outgoing from a deterministic state can update program variables. All other transitions are assumed to be of the form $(\ell, 1, id_1, \ell')$ for some locations ℓ, ℓ', where $id_1(\mathbf{x}) = \mathbf{x}[1]$ for all $\mathbf{x}$. We use this to simplify notation. An illustration of a pCFG is given in Figure 1.

1.4 Almost-Sure and Probabilistic Termination

We consider computational problems related to the basic liveness properties of Apps, namely *termination* and its generalization, *reachability*.

Termination, reachability, and termination time. In the following, consider an App P and its associated pCFG C_P. We say that a run ϱ of C_P *reaches* a set of configurations C if it contains a configuration from C. A run *terminates* if it reaches a configuration whose first component (i.e. a location of C_P) is the location ℓ_P^{out} corresponding to the value of the program counter after executing P. To each set of configurations C we can assign a random variable T^C such that for each run ϱ the value $T^C(\varrho)$ represents the first point in time when the current configuration on ϱ is in C. If a run ϱ does *not* reach a set C, then $T^C(\varrho) = \infty$. We call T^C the *reachability time* of C. In particular, if C is the set of all configurations $(\ell, \mathbf{x})$ such that $\ell = \ell_P^{out}$ (the terminal location of C_P), then T^C is called a *termination time*, as it returns the number of steps after which ϱ terminates. Since termination time is an important concept on its own, we use a special notation *Term* for it. Since a probabilistic program may exhibit more than one run, we are interested in probabilities of runs that terminate or reach some set of configurations. This gives rise to the following fundamental computational problems regarding termination:

(1) *Almost-sure termination:* A probabilistic program P is almost-surely (a.s.) terminating if under each scheduler σ it holds that $\mathbb{P}^\sigma(\{\varrho \mid \varrho \text{ terminates}\}) = 1$, or equivalently, if for each σ it holds $\mathbb{P}^\sigma(\mathit{Term} < \infty) = 1$. In almost-sure termination question for P we aim to prove that P is almost-surely terminating.
(2) *Probabilistic termination:* In probabilistic termination question for P we aim to compute a *lower bound on the probability of termination*, i.e. a bound $b \in [0, 1]$ such that for each scheduler σ it holds $\mathbb{P}^\sigma(\{\varrho \mid \varrho \text{ terminates}\}) \geq b$ (or equivalently $\mathbb{P}^\sigma(\mathit{Term} < \infty) \geq b$).

We also define corresponding questions for the more general reachability concept.

(1) *Almost-sure reachability:* For a set C of configurations of a probabilistic program P, prove (if possible) that under each scheduler σ it holds that $\mathbb{P}^\sigma(T^C < \infty) = 1$.
(2) *Probabilistic reachability:* For a set C of configurations of a probabilistic program P, compute a bound $b \in [0, 1]$ such that for each scheduler σ it holds $\mathbb{P}^\sigma(T^C < \infty) \geq b$.

Since termination is a special case of reachability, each solution to the almost-sure or probabilistic reachability questions provides solution for the corresponding termination questions.

REFERENCES

[1] P. Billingsley. 1995. *Probability and Measure* (3rd ed.). Wiley.
[2] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

```
x := 10
while  x ≥ 1  do
    if  prob(0.75)  then  x := x − 1 else  x := x + 1
    fi
od
```

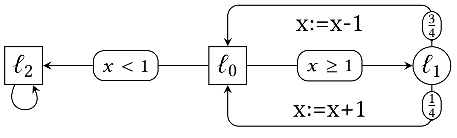

Fig. 1. An App modelling an asymmetric 1-D random walk and the associated pCFG. Probabilistic locations are depicted by circles, with probabilities given on outgoing transitions. Transitions are labelled by their effects. Location ℓ_0 is initial and ℓ_2 is terminal.

[3] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs through Positivstellensatz's. *CoRR* abs/1604.07169 (2016). http://arxiv.org/abs/1604.07169

[4] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 327–342. https://doi.org/10.1145/2837614.2837639

[5] 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*, Sriram K. Rajamani and David Walker (Eds.). ACM, 489–501. https://doi.org/10.1145/2676726.2677001

[6] Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2