MSc in Computer Science 2019-20

Project Dissertation

Title: On Termination of Higher-Order Recursive Programs with Continuous Distributions

Author: Raven Beutner

Term and year of submission: Trinity Term 2020

Title of Degree: MSc in Computer Science

Abstract

We study almost-sure termination (AST) of higher-order probabilistic functional programs with recursion, stochastic conditioning and sampling from continuous distributions.

In the first part of this work, we present a new *interval-based operational semantics* that we show sound and complete with respect to the standard sampling-based semantics à la Borgström et al. [2016]. In contrast to the sampling-based semantics, in our semantics enumeration of terminating computations provides arbitrarily tight lower bounds on the probability of termination, making it attractive for program analysis. As a consequence we obtain, to the best of our knowledge, the first proof that deciding AST for programs with continuous distributions is Π^0_2-complete. Extending the work of [Breuvart and Lago 2018] we give a compositional characterisation of our semantics in the form of an intersection type system that provides a local interpretation of AST.

In the second part, we propose a framework to prove almost-sure termination for *non-affine programs*, i.e., recursive programs that can make any number of recursive calls (of a first-order function) from a finite number of lexically different call-sites. While the number of recursion call-sites is irrelevant to qualitative questions about termination, we show that it is very much relevant in the quantitative setting of AST. Our framework over-approximates recursion by counting the number of recursive calls, reducing a program to a random walk for which we establish AST decidability in linear time. As a simple corollary we obtain a functional generalization of the *zero-one law of termination* by [McIver and Morgan 2005]. We show that our framework can be turned into a proof system that is easy to automate, and can verify AST for programs that are beyond the scope of existing methods.

Contents

Chapter 1

Introduction

Probabilistic (or randomised) programs are programs that employ a degree of randomness in their control flow. For many algorithmic problems, such as primality testing, approximation of high-dimensional integrals, and optimisation of complex objective functions, the best known deterministic algorithms perform poorly whereas probabilistic (or randomised) algorithms are often very efficient, and accurate either with high probability or in the limit. Recently, probabilistic programs have also become attractive in machine learning. In this area, known as (statistical) *probabilistic programming* [Gordon et al. 2014], programs are used as a means to express probabilistic models whose posterior can be inferred via generic inference methods.

One of the most studied and most prominent liveness-properties for non-probabilistic programs is the question of termination. In a non-probabilistic setting, termination is understood as a qualitative property where a program is said to terminate if *all* computation paths are terminating. In a probabilistic setting, we attribute probability to computation paths. The question of termination is therefore often no longer seen *qualitatively* but becomes *quantitative*. A program is said to be *almost-surely terminating (AST)* if the probability of termination is 1, which, in general, is not the same as termination on every path.

AST is a desirable property for programs. From the classical viewpoint, where a probabilistic program is viewed as an algorithm, i.e., a solution to an algorithmic problem, one requires programs that terminate with high probability, usually 1. When viewing probabilistic programs as a formal representation of some probabilistic model, AST is equally important. A common assumption of probabilistic programming systems is that programs terminate with probability 1 (see e.g. [Rainforth 2017, §4.3.2] and [Goodman et al. 2008]). In fact, it is standard for designers and implementors of probabilistic programming systems to regard any

program that does not satisfy this assumption as not defining a valid model, and so inadmissible [Goodman et al. 2008]. However, to our knowledge (and not surprisingly), no probabilistic programming system has provided any facilities for checking input programs for AST. There is another reason why AST is important for probabilistic programming. Mak et al. [2021] have recently shown that programs that are AST have density (a.k.a. weight) functions that are differentiable almost everywhere (a.e.). This is significant, because a.e. differentiability of the density function (of the probabilistic program) is a precondition for the correctness of some of the most (if not the most) important (and scalable) inference algorithms (namely, reparameterised gradient estimator [Lee et al. 2018], and Hamiltonian Monte Carlo [Zhou et al. 2019; Nishimura et al. 2020]) in each of the two main categories of Bayesian inference algorithms, stochastic variational inference and Monte Carlo methods.

This Work As AST is desirable and often required, a lot of effort is put into understanding AST and devising methods to show a program AST. This work falls within this active area of research (see e.g. [Kobayashi et al. 2019; McIver et al. 2018; Chen and He 2020]). Especially when modelling real-world processes as programs, one requires *continuous languages*, i.e., languages that can sample from continuous distributions, making AST analysis more intricate. In this work, we study a continuous, functional language with stochastic conditioning that can be seen as the foundation of many existing languages (see e.g. Church [Goodman et al. 2008] or Anglican [Tolpin et al. 2015]).

The contributions of this work are split into two parts. In the first part, we present a convenient operational semantics for continuous languages. Our soundness and completeness proof yields a, to our knowledge first, proof that deciding AST in a continuous language with mild restrictions is contained in the same level of the arithmetic hierarchy as for the discrete case. We also give a novel local representation of our semantics in the form of an intersection type system. In the second part, we focus on the number of recursive calls made by a program. We show that in the quantitative setting of AST the number of calls plays an intricate role and complicates AST analysis. We present a counting-based method to reason about termination of recursive programs. We show that our framework can be turned into an easy-to-automate proof system.

Part I - Interval-based Reasoning

In a language with samples from *discrete* probability distributions, program execution can be viewed as a countable tree where branching corresponds to a probabilistic sample. An operational semantics can thus be given as a probability mass on terms (see e.g. [Lago and Grellois 2019; Ehrhard et al. 2014; Kaminski et al. 2018]). If we want to verify AST, we check

that for every $\epsilon > 0$ we can find a finite set of terminating execution whose measure is at least $1 - \epsilon$; the sum over all (countably many) terminating executions is 1. Deciding AST is thus contained in Π^0_2, the second level of the arithmetic hierarchy (see [Kleene 1955]). Actually the problem is also Π^0_2-hard [Kaminski and Katoen 2015] and therefore much harder than deciding termination in the non-probabilistic case.

Checking AST is not so easy in the continuous case: Giving a formal semantics to a continuous language is an active area of research. In a continuous language execution can be seen as a tree with *uncountable* branching factor, so we cannot assign probability mass to terms directly. A first operational semantics for continuous languages was given in [Borgström et al. 2016] in a sampling-based style. The idea of a sampling-based semantics (going back to [Kozen 1981]) is to fix the probabilistic outcomes beforehand as a *program trace*, making the execution itself deterministic. The probability of termination is then given as the measure of all program traces on which the program terminates. Inferring lower bounds on the probability of termination is, however, not easy, as the measure of any countable set of traces is 0. The naïve idea of incrementally looking for terminating executions, working well in the discrete case, does not work. This leaves important questions such as the exact complexity of deciding AST in the presence of continuous distributions unanswered.

Interval-based Semantics The starting point of this work is an alternative sampling-based semantics. Instead of parametrizing execution on traces of real number, we parametrize executions on traces of intervals. With this new semantics, even a finite number of interval-traces can give positive bounds on the probability of termination. We show soundness and completeness with respect to the old semantics: instead of analysing a program by identifying uncountably many real-valued traces we can work in the interval setting where only countable many interval traces are required. This allows us to give a, to our knowledge first, proof that deciding AST is actually contained in Π^0_2.

As an example consider the program in Fig. 1.1a written in the functional language SPCF used in this work. μ^{φ}_{x} defines a recursive binding where φ is the recursive abstraction and x the argument. `sample`-evaluates to a uniform distribution on $[0, 1]$. We can derive the biased binary choice $M \oplus_p N$ in the style of McIver and Morgan as $M \oplus_p N \triangleq$ `if sample` $\leq p$ `then` M `else` N. The program describes a random walk of a particle that starts in location start and at each step travels a distance d sampled from a uniform distribution. With probability p it travels towards 0 and with $1 - p$ away from 0. The (probabilistic) output of the program is the cumulative distance travelled. In Fig. 1.1d a histogram of sampled outputs is given for different values of p, each starting at location start $= 12$. In the sampling-based style of Borgström et al. a program trace is a sequence on which the program terminates.

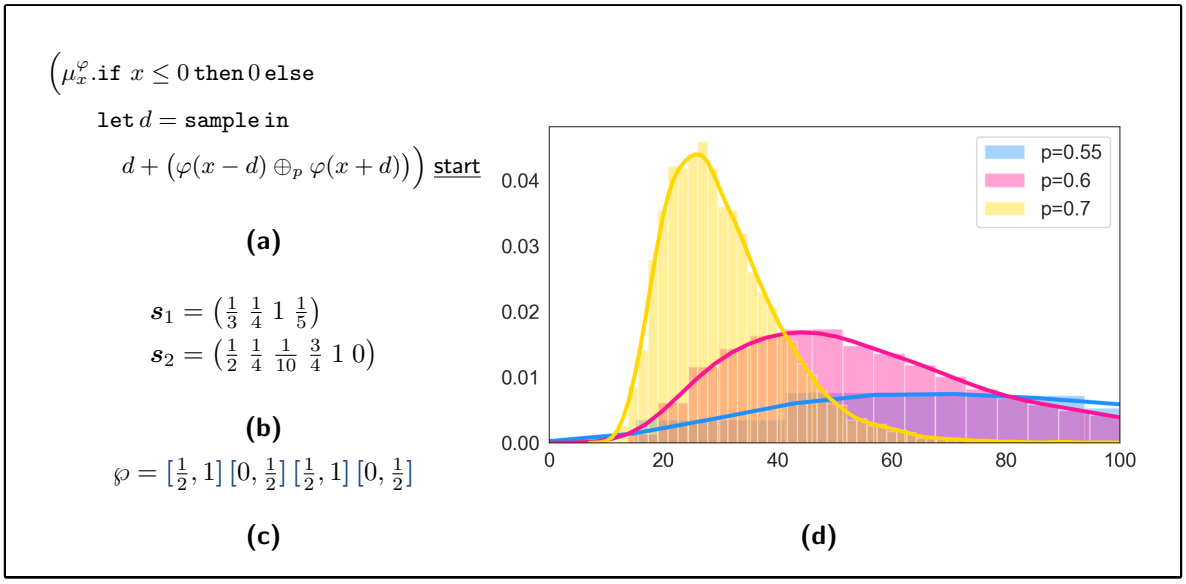


Figure 1.1: In **(a)** we depict an example SPCF program modelling a continuous random walk of a particle. **(d)** gives a histogram of the outcomes of executions at starting position $\mathsf{start} = 12$ and different values of p. **(b)** and **(c)** give terminating standard and interval traces respectively for $\mathsf{start} = 1$ and $p = \frac{1}{2}$.

Two example terminating traces for $p = \frac{1}{2}$ and $\mathsf{start} = 1$ are given in Fig. 1.1b. Assuming $p \geq \frac{1}{2}$, this program is AST but we cannot show this by enumerating terminating traces (even if we were allowed to, in the limit, enumerate countably infinite traces) as the measure of any countable set of such is 0. In Fig. 1.1c we depicted a terminating *interval* trace for $p = \frac{1}{2}$ and $\mathsf{start} = 1$. We can assign a positive measure to this interval trace as the product of the interval length (in this case $\frac{1}{16}$) giving us an immediate, non-zero lower bound.

Intersection Type System Recently, [Breuvart and Lago 2018] showed that intersection type systems are applicable for a probabilistic version of the λ-calculus whilst achieving similar completeness results as for deterministic languages. We build on those observations and use intersection types as a more compositional representation of the interval-based semantics. This generalizes the work by Breuvart and Lago by both moving to a continuous setting and also being able to reason about the expected termination time. In our system, both the probability of termination and the expected time to termination can be obtained as the least upper bound of all derivations. This gives us an alternative, recursion-theoretically optimal, characterisation of AST. It is somewhat surprising, that in our type system, *finite* derivations can give arbitrarily tight lower bounds on the probability of termination, which is a measure defined on an *uncountable* measurable set (of traces).

Part II - Non-affine Recursion

In the second part of this work, we provide a framework that gives a sound method to prove AST. Our system joins a long line of existing work on methods for AST verification ([Lago and Grellois 2019; Chakarov and Sankaranarayanan 2013; Fioriti and Hermanns 2015; McIver et al. 2018], to name only a few).

A particularly interesting feature of recursive programs is the ability to make more than one recursive call. In a probabilistic language this has direct implications on the (quantitative) termination behavior. As an example we consider the geometric distribution with parameter $p \in (0,1)$, i.e., the distribution $n \mapsto (1-p)^n p$. In an imperative setting, we can "compute" this distribution via the program in Fig. 1.2a, where $\mathsf{prob}(p)$ evaluates to 0 with probability p and to 1 with probability $1-p$. This program can easily be proved AST in most of the existing frameworks. The functional equivalent is the term in Fig. 1.2b which is also AST. We say this program has *affine recursion*, as it makes recursive calls from at most one call-site. But what happens if we change the term slightly and instead consider the term M in Fig. 1.2c which can make two recursive calls? The natural question is: is this program AST? While this is an easy question for the affine example in Fig. 1.2b (which terminates for all $p \in (0,1)$), the additional recursive call complicates things. Actually the answer is that the term is AST if and only if $p \geq \frac{1}{2}$.

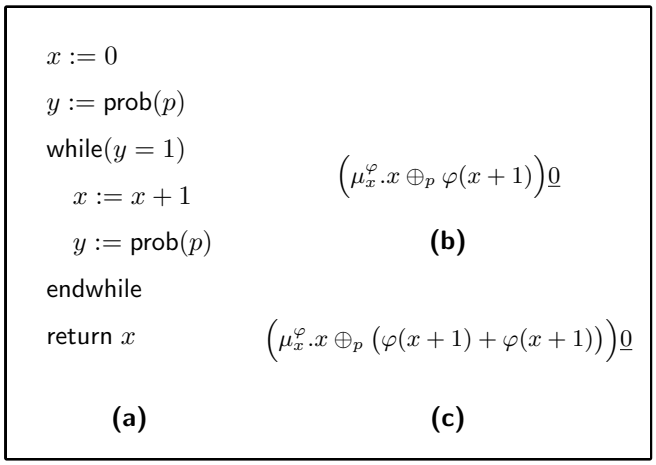


Figure 1.2: Variations of geometric distributions in imperative and functional languages.

For termination as a qualitative property the number of recursive calls is irrelevant, as a program terminates if all recursive calls terminate, no matter how many there are[1]. This is in stark contrast to a probabilistic world, where the number of recursive calls matters as termination is itself a quantitative property. This is well illustrated in the example in Fig. 1.2c. The severe consequences on the AST behavior when making more than one recursive call have already been identified as a cause of problems. As an example, the size-type system in [Lago

[1] Take any non-probabilistic fixpoint that terminates on every input. Then adding more recursive calls in the body does not change the termination behavior: the program is still terminating.

and Grellois 2019] requires affine recursion and is otherwise unsound.

In the second part we present a framework to reason about programs with not-necessarily-affine recursion, i.e., programs that can make more than one recursive calls[2]. By abuse of language, we write *non-affine* to mean not-necessarily-affine. This emphasises that the results and methods are specifically geared towards programs with more than one recursive call but are, of course, also applicable for affine recursion. We call our methods *counting-based*, as we over-approximate the recursive behavior by analysing how many calls are made. This reduces AST analysis to analysis of a random walk for which we show linear decidability (by generalizing the work by Lago and Grellois). Our method can be turned into an AST proof system that can verify programs beyond the reach of existing methods. As a simple corollary we obtain a functional generalization of a known result for imperative languages, known as the zero-one law for termination due to [McIver and Morgan 2005][3]. This result is implicitly used to design languages that are AST by construction (see e.g. [Lew et al. 2020]).

Contributions Our contributions include the following:

- We propose a new sampling-based operational semantics that is parametrized by intervals. We show that this new semantics is sound and complete with respect to the standard one by Borgström et al.. As a consequence we obtain a proof that deciding AST if in Π^0_2 and deciding positive almost-sure termination[4] is in Σ^0_2, matching the bounds known for discrete-probabilistic programming languages [Kaminski and Katoen 2015]. Our approach is also applicable to imperative languages (as e.g. used in [Fioriti and Hermanns 2015]) giving a definitive answer to the complexity of deciding AST.

- We devise an intersection type system supporting continuous distributions by extending [Breuvart and Lago 2018] and [Ehrhard et al. 2014]. Our type system relies on a novel interval-based semantics allowing (finite) typing derivations to give non-trivial lower bounds. We show that type derivations directly correspond to lower bounds on the probability of termination. Our type system can also reason about expected termination time, giving a further generalization of [Breuvart and Lago 2018].

- We provide a sound method for reducing programs with non-affine recursion to random walks on natural numbers. Our method can verify interesting recursive programs that

[2] When talking about the *number of recursive calls* we always refer to the number of recursive call-sites and *not* to the total number of recursive abstractions resolved during execution.

[3] The zero-one law states that if a loop is left with a probability that is lower bounded away from 0, then the loop is left eventually almost surely.

[4] A program is said to be positively almost-surely terminating (PAST) if the expected number of steps to termination are finite. PAST is a strictly stronger property than AST.

existing methods cannot. We show how to turn the framework into a proof system that can automatically verify AST.

1.1 Outline

In the next section, we summarize related work in the complexity analysis of AST, intersection type discipline and methods for AST verification. In Ch. 2 we give a brief overview of the necessary foundations in probability theory. Afterwards, in Ch. 3 we formally introduce SPCF, our language of study, and introduce both Call by Value and Call by Name semantics in the style of Borgström et al. [2016]. Part I of this work is then comprised of Chapters 4 and 5. In the first, we present the interval-based semantics and show it sound and complete. In Ch. 5 we give a compositional system as an intersection type system. In the second part of this work we first present counting-based methods for non-affine recursion (Ch. 6) and present a sound reduction to random walks. In Ch. 7, we turn our method into a proof system that we show correct.

1.2 Related Work

In this section, we study related approaches in the literature and outline how our contributions constitute an extension. We discuss the work done to study the complexity of AST **(1)** and discuss intersection type systems in a probabilistic setting **(2)**. We discuss the most active area of research: the pursuit of powerful proof systems to verify AST **(3)**. Lastly, we look at related work in the area of recursive programs and probabilistic pushdown systems **(4)**.

Our language of study and especially its operational semantics builds upon prior work in [Borgström et al. 2016].

(1) - Complexity of AST Termination in a non-probabilistic setting is based on the termination on every computation path and is therefore contained in Σ_1^0, as in the case of termination there exists a common bound on the maximal number of computation steps that can be recursively enumerated. In a probabilistic setting, on the other hand, the problem of termination becomes a quantitative problem and thus harder than standard termination. A first formal proof of this fact was given in [Kaminski and Katoen 2015], where they showed that AST is Π_2^0-hard and thus provably harder than deciding qualitative (non-probabilistic) termination. The proof by Kaminski and Katoen, reduces the Π_2^0-hard universal halting problem, i.e., the problem if a program terminates on all inputs, to AST by sampling the input from a geometric distribution. As we saw in the introduction containment in Π_2^0 for a discrete language can easily be established by enumerating terminating paths. While the

hardness trivially carries over to a continuous language, the containment does not. The simple containment proof (e.g. [Kaminski and Katoen 2015, Theorem 6]) does not work as the measure of any countable set of paths is always 0. We construct a new semantics, sound and complete w.r.t. [Borgström et al. 2016], and can thereby lift the results from Kaminski and Katoen [2015] to a continuous setting. As we construct a new operational semantics for SPCF we also mention a line of ongoing work dedicated to finding denotational models for higher-order probabilistic languages (see e.g. [Ścibior et al. 2018; Vákár et al. 2019]).

(2) - Intersection Type System The literature on intersection types in a non-probabilistic system is rich. (For a gentle introduction we refer to [Bucciarelli et al. 2017]). In a probabilistic setting, a first comprehensive study of intersection type systems was carried out by [Breuvart and Lago 2018]. Our work is closely related to theirs but differs in a few key aspects: their work focuses on a discrete untyped λ-calculus and considers the probability of termination, whereas we work in the continuous domain and explicitly support expected termination time. Another interesting read is [Ehrhard et al. 2014], where they used intersection types to give a compositional, local semantics to a probabilistic version of PCF. They also considered a discrete language.

(3) - Proof Systems for Almost-sure Termination The list of existing approaches on AST proof systems is rich enough to warrant a work on its own. We therefore restrict our discussion to the most influential results. Most of the existing approaches focus on imperative languages. A standard approach to termination (in a *non-probabilistic* setting) is to find a variant, i.e., a function from the state space to a well-ordered set (typically the natural numbers) that decreases in each step. Many approaches in a probabilistic setting build on this idea. In [Chakarov and Sankaranarayanan 2013], they introduced the idea of an ϵ-ranking function which is a function from the state space that decreases *in expectation* by at least ϵ in each step. If the value of V is bounded from below this already implies AST. From a mathematical standpoint, a variant describes a supermartingale, i.e., a real-valued stochastic process that does not increase in expectation [Ash and Doleans-Dade 1999, §6]. The approach of using ϵ-ranking functions was later extended by [Fioriti and Hermanns 2015] who added support for demonic non-determinism. It was shown in [Fu and Chatterjee 2019] that the ϵ-ranking based approach is complete for programs that are assumed to terminate in finite expected time. While this result does guarantee the existence of a ranking function, constructing one is still challenging. One of the most recent and most powerful rules to ensure AST was given in [McIver et al. 2018]. They relaxed the requirement of having a *fixed* expectation-decrease in each step. Instead, the ranking function in state s must decrease by at least d with probability at least p where d, p are antitone functions. This new rule can

verify many interesting examples including the fair random walk, a program that is hard to analyse as it neither terminates in finite expected time nor does it admit an ϵ-ranking function. [Agrawal et al. 2018] introduced lexicographic ranking functions: instead of giving a single ranking function on the entire program, the user can provide a number of such and needs to show that they decrease in lexicographic order. This is particularly useful for compositional reasoning. Apart from giving conditions on ranking functions that imply AST, it is interesting to automate the construction of the functions as much as possible. As an example in [Chatterjee et al. 2018] they showed that a special class of linear ranking functions can be synthesised in polynomial time by using Farkas Lemma.

(4) - Reasoning about Recursive Programs Most of the approaches to AST are given for imperative languages which is contrary to the fact that many of the probabilistic languages used in practice are functional ([Goodman et al. 2008; Tolpin et al. 2015]). It is not obvious at all whether those methods are applicable to functional languages. One of the few approaches in a functional setting is made in [Lago and Grellois 2019]. They designed a decidable size-type system where the change of size is modelled as a random walk. Their system is limited to affine-recursion and unsound for programs that make more than one recursive call.

In [Olmedo et al. 2016], they considered an imperative language with recursive calls. Their programs can exhibit interesting non-affine recursive behavior, making their approach relevant for ours. Their result lies in a line of work of inferring bounds on the expected number of computation steps ([Kaminski et al. 2018; Olmedo et al. 2016]) and is not geared towards AST analysis. We discuss the relation to their work in more detail in Ch. 6.

Recursive programs can naturally be modelled as pushdown systems. In the presence of probability, programs can be modelled as probabilistic pushdown systems (see [Olmedo et al. 2016] for an example in the discrete case). For finite state probabilistic pushdown systems many properties, including termination, can be expressed in the first-order existential fragment over the reals (Tarski algebra) and are thus decidable (see [Brázdil et al. 2013] for an excellent overview). We argue that recursion in continuous languages denotes a continuous state-space pushdown system, for which we develop counting-based methods.

Chapter 2

Preliminaries

In this chapter, we introduce basic mathematical notation and introduce the basics of probability theory needed to apprehend this work. A more detailed introduction can e.g. be found in [Ash and Doleans-Dade 1999].

Basic Notation With $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ we denote the set of natural numbers (including 0), integers, rationals and reals. $\mathbb{R}_+ \triangleq \{r \in \mathbb{R} \mid r \geq 0\}$ denotes the set of non-negative real numbers and $\mathbb{R}_{[0,1]} \triangleq \{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$ the set of real numbers within $[0, 1]$. We write $\overline{\mathbb{N}}$ and $\overline{\mathbb{R}_+}$ for $\mathbb{N}$ or $\mathbb{R}_+$ extended with ∞. With $[n]$ we denote the set $\{0, \cdots, n-1\}$. For a set A we denote the set of finite sequences over A by A^*. For $w \in A^*$, $|w|$ denotes the length and for $i \in [|w|]$, $w(i) \in A$ the ith position.

Basic Probability Theory A *σ-algebra* on a set Ω is a set of subsets $\Sigma_\Omega \subseteq 2^\Omega$ with $\Omega \in \Sigma_\Omega$ that is closed under complements and countable unions (and thus countable intersections). A *measurable space* is a tuple (Ω, Σ_Ω) where Ω is a set of outcomes and Σ_Ω is a σ-algebra on Ω. A set of outcomes $A \subseteq \Sigma$ is called *measurable* if $A \in \Sigma_\Omega$. A function $f : \Omega_1 \to \Omega_2$ between two measurable spaces $(\Omega_1, \Sigma_{\Omega_1})$, $(\Omega_2, \Sigma_{\Omega_2})$ is called *measurable* if for every $A \in \Sigma_{\Omega_2}$, $f^{-1}(A) \in \Sigma_{\Omega_1}$. A *measure* on (Ω, Σ_Ω) is a function $\mu : \Sigma_\Omega \to \overline{\mathbb{R}_+}$ that satisfies $\mu(\emptyset) = 0$ and is σ-additive: if $\{A_i\}_{i \in \mathbb{N}}$ is a countable family of sets from Σ_Ω that are pairwise disjoint then $\mu(\cup_i A_i) = \sum_i \mu(A_i)$. If $\mu(\Omega) < \infty$ we call μ finite, if $\mu(\Omega) \leq 1$ we call μ a subprobability measure and if $\mu(\Omega) = 1$ we call it a probability measure (or distribution). A measurable set A with $\mu(A) = 0$ is called a *null-set*. Assume A, B are measurable sets with $\mu(B) > 0$ for some finite measure μ. Then the conditional measure of A given B is defined by $\mu(A \mid B) \triangleq \frac{\mu(A \cap B)}{\mu(B)}$. We say A and B are independent if $\mu(A \cap B) = \mu(A)\mu(B)$. A and B are conditionally independent on a non null-set C, $\mu(A \cap B \mid C) = \mu(A \mid C)\mu(B \mid C)$. For every $x \in \Omega$ we denote with δ_x the *Dirac distribution* on x. It is defined by $\delta_x(A) \triangleq 1$ if $x \in A$ and $\delta_x(A) \triangleq 0$

otherwise. A *measure space* is a triple $(\Omega, \Sigma_\Omega, \mu)$ where (Ω, Σ_Ω) is a measurable space and μ a measure on Σ_Ω. A probability space and subprobability space are defined analogously. For the n-dimensional euclidean space $\mathbb{R}^n$ we denote with $\Sigma_{\mathbb{R}^n}$ the Borel σ-algebra over $\mathbb{R}^n$ which is the smallest σ-algebra that contains all open and closed n-dimensional boxes. So for every intervals $[a_1, b_1], \cdots, [a_n, b_n]$ the box $[a_1, b_1] \times \cdots [a_n, b_n]$ is contained in $\Sigma_{\mathbb{R}^n}$. In the special case of $n = 1$, this is the set generated by all open and closed intervals. The n-dimensional Lebesgue measure, denoted λ_n, is the unique measure on $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$ that satisfies $\lambda_n([a_1, b_1] \times \cdots \times [a_n, b_n]) = \prod_{i=1}^n (b_i - a_i)$.

For a measurable function $f : \Omega \to \mathbb{R}$ on a measure space $(\Omega, \Sigma_\Omega, \mu)$ we can define the integral $\int f d\mu$ following Lebesgue theory (see [Ash and Doleans-Dade 1999, §1.5] for details). We often write this integral as $\int \mu(dx) f(x)$. For a measurable set B we define the indicator function $\mathbf{1}_B(x)$ by $\mathbf{1}_B(x) \triangleq 1$ if $x \in B$ and otherwise $\mathbf{1}_B(x) \triangleq 0$. $\mathbf{1}_B$ is trivially measurable. We can then define the integral over a measurable set B by

$$\int_B \mu(dx) f(x) \triangleq \int \mu(dx) \big(f(x) \cdot \mathbf{1}_B(x)\big)$$

Discrete Sample Space In the case where Ω is a countable set, we can work with the powerset as the trivial σ-algebra. Every probability measure is then uniquely determined by a *probability mass function (pmf)*, a function $p : \Omega \to \mathbb{R}_{[0,1]}$ with $\sum_{x \in \Omega} p(x) = 1$. Every pmf p gives rise to a probability measure by defining $\mu(A) \triangleq \sum_{x \in A} p(x)$ and conversely for every probability measure μ on the powerset we can recover a generating pmf by inspecting μ on singletons, i.e., defining $p(x) \triangleq \mu(\{x\})$. A subprobability mass function is defined analogously.

Kernel Let $(\Omega_1, \Sigma_{\Omega_1})$ and $(\Omega_2, \Sigma_{\Omega_2})$ be two measurable spaces. A *subprobability kernel* is a function $\kappa : \Omega_1 \times \Sigma_{\Omega_2} \to \mathbb{R}_{[0,1]}$ that satisfies the following: **(1)** For every $x \in \Omega_1$, the function $\kappa(x, \cdot) : \Sigma_{\Omega_2} \to \mathbb{R}_{[0,1]}$ is a subprobability distribution, and **(2)** for every $A \in \Sigma_{\Omega_2}$, the function $\kappa(\cdot, A) : \Omega_1 \to \mathbb{R}_{[0,1]}$ is measurable with respect to Σ_{Ω_1} and $\Sigma_{\mathbb{R}_{[0,1]}}$. If for every $x \in X$, the function $\kappa(x, \cdot)$ is a proper distribution we call κ a proper kernel (or *Markov Kernel*). Given a kernel κ_1 between measurable spaces $(\Omega_1, \Sigma_{\Omega_1})$ and $(\Omega_2, \Sigma_{\Omega_2})$ and κ_2 between $(\Omega_2, \Sigma_{\Omega_2})$ and $(\Omega_3, \Sigma_{\Omega_3})$ we can define the composed kernel $\kappa_2 \circ \kappa_1$ between $(\Omega_1, \Sigma_{\Omega_1})$ and $(\Omega_3, \Sigma_{\Omega_3})$ by:

$$(\kappa_2 \circ \kappa_1)(x, A) \triangleq \int \kappa_1(x, dy) \kappa_2(y, A)$$

In particular, for any kernel between a measurable space and itself we can define the nth-fold self-composition. We later justify that we can view a Markov kernel $(\Omega_1, \Sigma_{\Omega_1})$ to $(\Omega_1, \Sigma_{\Omega_1})$ as the transition kernel of a continuous state space Markov chain.

Chapter 3

Statistical PCF (SPCF)

We begin introducing the main object of study: the higher-order functional language Statistical PCF (SPCF). SPCF is an extension of PCF [Plotkin 1977] with support to sample from a uniform distribution on $[0, 1]$. To justify the name "statistical" and make SPCF attractive to writing probabilistic models it also supports a scoring construct that is used to condition executions of the program (see e.g. [Gordon et al. 2014]). The language is a typed version with explicit recursion of the untyped λ-calculus used in [Borgström et al. 2016], making it arguably more intuitive to programmers than a pure λ-calculus.

Syntax of SPCF Terms in SPCF are implicitly parametrized over a set of measurable functions $\mathbb{R}^n \to \mathbb{R}$ that model primitive operations. Assume $\mathbb{F}$ is a fixed set of such primitive functions. Each function $f \in \mathbb{F}$ has an arity $|f| \geq 0$. We assume a denumerable set of variables. The sets of terms and values (*Term* and *Val*) are defined by the following grammar where $x \neq \varphi$ are variables, $r \in \mathbb{R}$ and $f \in \mathbb{F}$:

$$\begin{aligned} V &\triangleq x \mid \underline{r} \mid \lambda x.M \mid \mu_x^\varphi.M \\ M, N, P &\triangleq V \mid MN \mid \texttt{if}(M, N, P) \mid f(M_1, \cdots, M_{|f|}) \mid \texttt{sample} \mid \texttt{score}(M) \end{aligned}$$

We identify terms modulo α-conversion and adopt the variable convention from [Barendregt 1985]. All language constructs with exception of `sample` and `score` are standard. For the fixpoint combinator $\mu_x^\varphi.M$ the variable x is the variable bound by the abstraction and φ a recursive reference to itself. `sample` evaluates to a uniform distribution on $[0, 1]$ and `score` conditions an execution. We abbreviate $M \oplus_p N \triangleq \texttt{if}\ (\texttt{sample} - p, M, N)$ in the style of [McIver and Morgan 2005] and $\big(\texttt{let}\, x = M\, \texttt{in}\, N\big) \triangleq (\lambda x.N)M$. We write $M \oplus N$ for $M \oplus_{.5} N$.

$$\frac{x:\alpha\in\Gamma}{\Gamma \Vdash x:\alpha} \qquad \frac{\Gamma;x:\alpha \Vdash M:\beta}{\Gamma \Vdash \lambda x.M:\alpha\to\beta} \qquad \frac{\Gamma;\varphi:\alpha\to\beta;x:\alpha \Vdash M:\beta}{\Gamma \Vdash \mu_x^{\varphi}.M:\alpha\to\beta} \qquad \frac{}{\Gamma \Vdash \underline{r}:\mathbf{R}}$$

$$\frac{\Gamma \Vdash M:\beta\to\alpha \qquad \Gamma \Vdash N:\beta}{\Gamma \Vdash MN:\alpha} \qquad \frac{\Gamma \Vdash M:\mathbf{R} \qquad \Gamma \Vdash N:\alpha \qquad \Gamma \Vdash P:\alpha}{\Gamma \Vdash \mathtt{if}(M,N,P):\alpha}$$

$$\frac{}{\Gamma \Vdash \mathtt{sample}:\mathbf{R}} \qquad \frac{\Gamma \Vdash M_1:\mathbf{R} \qquad \cdots \qquad \Gamma \Vdash M_{|f|}:\mathbf{R}}{\Gamma \Vdash f(M_1,\cdots,M_{|f|}):\mathbf{R}} \qquad \frac{\Gamma \Vdash M:\mathbf{R}}{\Gamma \Vdash \mathtt{score}(M):\mathbf{R}}$$

Figure 3.1: Typing rules of the from $\Gamma \Vdash M:\alpha$ for the simple type system for SPCF.

Simple Type System Simple types are constructed from a base type $\mathbf{R}$ and the arrow constructor: $\alpha,\beta \triangleq \mathbf{R} \mid \alpha\to\beta$. Typing judgements are of the form $\Gamma \Vdash M:\alpha$ where Γ is a typing context, i.e., a partial mapping from variables to types. We write $x:\alpha\in\Gamma$ whenever $\Gamma(x)=\alpha$. We define $\Gamma;x:\alpha$ as the context that extends Γ with the mapping $x:\alpha$. Formally $\big(\Gamma;x:\alpha\big)(x) \triangleq \alpha$ and $\big(\Gamma;x:\alpha\big)(y) \triangleq \Gamma(y)$ for $y\neq x$. A typing judgment is of the from $\Gamma \Vdash M:\alpha$ where Γ is a context, M a term and α a type[1]. The typing judgments are defined by induction as the smallest relation closed under the rules in 3.1. We call a term typeable if there exist a Γ and α such that $\Gamma \Vdash M:\alpha$ holds. We only consider typeable terms. With Λ we denote the set of typeable SPCF terms and with Λ_0 the set of closed terms[2]. We remark that our fixpoint (μ) combinator differs from the ones used in other variants of a probabilistic (functional) language studied in the literature [Mak et al. 2021; Ehrhard et al. 2014]. We treat a fixpoint combinator as an abstraction that also bounds the recursive abstraction. This allows for an easier Call by Value reduction.

3.1 Operational Semantics - Call by Value

We give a sampling-based semantics for SPCF. The idea of a sampling-based style is to evaluate a term M together with a sequence of probabilistic outcomes. In our case a sequence of real numbers from $\mathbb{R}_{[0,1]}$. Each `sample`-statement during execution is then resolved according to the trace of random samples. This idea of giving an operational semantics for probabilistic programs goes back to [Kozen 1981] and was first proposed in its current presentation in [Borgström et al. 2016].

[1] Note that we do not use the typical symbol $\vdash$, as we reserve this for later (more sophisticated) type systems.
[2] A term is closed if it does not contain any free variables. Using the type system we can a term being closed as being typeable in the empty context.

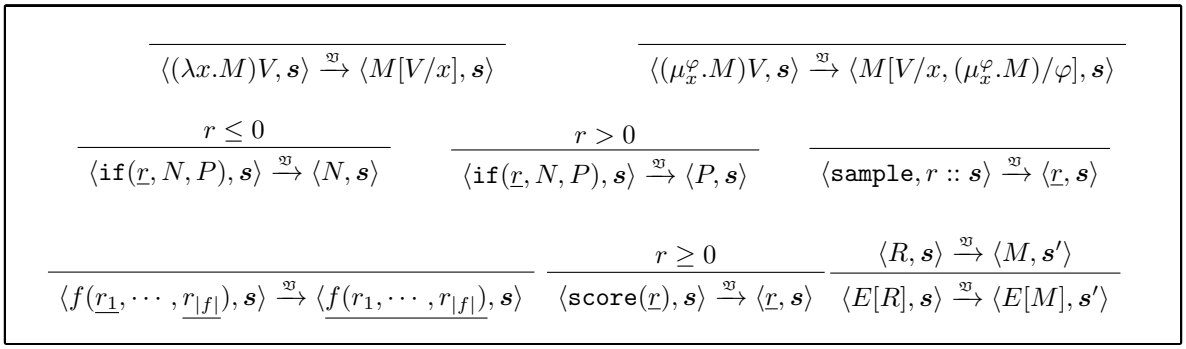


Figure 3.2: Call by Value small-step reduction for SPCF.

Evaluation Strategy In a probabilistic setting, the evaluation strategy plays an important role. In the untyped λ-calculus, it is well known that reduction is confluent: if a term M is reduced at two different position to two terms M_1, M_2 they can always reduce to common a term N in a finite number of steps. In a probabilistic setting, the reduction relation is no longer confluent (c.f. [Breuvart and Lago 2018]) and more importantly, the denotation of a program depends on the evaluation strategy. In this work, we consider both a Call by Value (CbV) and a Call by Name (CbN) semantics. Informally speaking in a CbV setting, the argument is first reduced to value before a β-step is done whereas in CbN the β-step is performed first. This gives rise to an interesting asymmetry between both strategies[3].

Traces We define the set of traces $\mathbb{S}$ as all finite sequences of real numbers from $\mathbb{R}_{[0,1]}$, i.e., $\mathbb{S} \triangleq \mathbb{R}_{[0,1]}^* = \bigcup_{n\in\mathbb{N}} \mathbb{R}_{[0,1]}^n$. We let $\boldsymbol{s}$ range over elements in $\mathbb{S}$. With ϵ we denote the empty sequence. For $r \in \mathbb{R}_{[0,1]}$, $r :: \boldsymbol{s}$ denotes the concatenation of r to $\boldsymbol{s}$. For two traces $\boldsymbol{s}_1, \boldsymbol{s}_2$ we write $\boldsymbol{s}_1\boldsymbol{s}_2$ for their concatenation. $\mathbb{S}^m$ denotes the set of traces of length m.

Call by Value Reduction The set of CbV redexes and evaluation contexts is defined by:

$$\begin{aligned} \mathit{Red}_{\mathscr{V}} \ni R &\triangleq (\lambda x.M)V \mid (\mu_x^{\varphi}.M)V \mid \mathtt{if}(\underline{r}, N, P) \mid f(\underline{r_1}, \cdots, \underline{r_{|f|}}) \mid \mathtt{sample} \mid \mathtt{score}(\underline{r}) \\ \mathit{ECont}_{\mathscr{V}} \ni E &\triangleq [\cdot] \mid EM \mid (\lambda y.M)E \mid (\mu_x^{\varphi}.M)E \mid \mathtt{if}(E, N, P) \\ &\quad \mid f(\underline{r_1}, \cdots, \underline{r_{k-1}}, E, M_{k+1}, \cdots, M_{|f|}) \mid \mathtt{score}(E) \end{aligned}$$

A redex is a term that can make an immediate reduction step by e.g. reducing a β-redex. An evaluation context gives the context, i.e., the position inside a term, at which a reduction step can take place. Given a context E and a term M the (capture-permitting) substitution

[3] As an example take the term $(\lambda x.x + x)(\underline{0} \oplus \underline{1})$. In a CbV reduction, we sample before doing the β-step so the output would be $\underline{0}$ ro $\underline{2}$ each with equal probability $\frac{1}{2}$. In a CbN reduction, we β-reduce before resolving the probabilistic outcome. The output is hence $\underline{0}$ or $\underline{2}$ each with probability $\frac{1}{4}$ and $\underline{1}$ with probability $\frac{1}{2}$.

$E[M]$ is defined in the obvious way. An easy induction establishes that every $M \in \Lambda_0$ is either a value or there are *unique* E and R, s.t., $M = E[R]$. As we work with a CbV strategy, we only perform β-reductions (on either a λ-or μ-abstraction) once the argument position has reduced to a value. In evaluation contexts reduction occurs inside the argument position of an application. The small-step reduction relation has the from $\langle M, \boldsymbol{s}\rangle \xrightarrow{\mathfrak{w}} \langle M', \boldsymbol{s}'\rangle$ where M, M' are terms and $\boldsymbol{s}, \boldsymbol{s}'$ are traces. It is defined inductively by the rules given in Fig. 3.2[4]. The evaluation context takes care of the contextual closure of the reduction. Note that the redex $\texttt{score}(\underline{r})$ for an $r < 0$ can not reduce. That means our reduction system does not enjoy progress, i.e., not every term is either a value or can make a reduction step[5].

Remark: We remark that our semantics differs substantially from the ones in [Borgström et al. 2016; Mak et al. 2021] as it is missing a weight parameter. If we were interested in the exact denotation of a program, a weight parameter would be crucial to model stochastic conditioning (c.f. [Gordon et al. 2014]). The weight of a derivation is, however, irrelevant for the probability of termination (c.f. the definition of AST in [Mak et al. 2021]). We, therefore, omit it and obtain a more succinct representation.

3.2 A Measure on Traces

The reduction relation itself is purely deterministic once we fix a trace $\boldsymbol{s} \in \mathbb{S}$ and does itself not represent probabilistic execution. The probabilistic behavior is instead obtained by giving a probability measure to the set of traces $\mathbb{S}$. If we were e.g. interested in the probability of terminating with a negative value we would take the measure over all traces where the program terminates with a negative value.

A Measure Space of Traces As the set of traces is continuous we cannot assign probability mass to individual traces directly. Instead, we use a measurable space of program traces. We follow the approach by Borgström et al.: let $\Sigma_{\mathbb{R}^n_{[0,1]}}$ be the standard Borel σ-algebra on $\mathbb{R}^n_{[0,1]}$, i.e., the smallest σ-algebra that contains all open and closed intervals within $\mathbb{R}^n_{[0,1]}$ (We set $\Sigma_{\mathbb{R}^0_{[0,1]}} \triangleq \{\emptyset, \{\epsilon\}\}$). Recall that λ_n denotes the Lebesgue measure on $(\mathbb{R}^n_{[0,1]}, \Sigma_{\mathbb{R}^n_{[0,1]}})$. We can define a σ-alegbra on traces $\Sigma_{\mathbb{R}_{[0,1]}}$ and a measure $\mu_{\mathbb{S}}$ by:

- $\Sigma_{\mathbb{S}} \triangleq \{ \biguplus_{n\in\mathbb{N}} B_n \mid B_n \in \Sigma_{\mathbb{R}^n_{[0,1]}}\}$
- $\mu_{\mathbb{S}}\big(\biguplus_{n\in\mathbb{N}} B_n\big) \triangleq \sum_{n\in\mathbb{N}} \lambda_n(B_n)$

As shown in [Borgström et al. 2016, Lemma 7 & 8] we get that $(\mathbb{S}, \Sigma_{\mathbb{S}})$ is a measurable space

[4] For a term M, a list of distinct variables $\{x_i\}_{i\in[n]}$ and a list of terms $\{N_i\}_{i\in[n]}$, the capture avoiding substitution $M[N_i/x_i]_{i\in[n]}$ is defined in the standard way. See e.g. [Barendregt 1985].

[5] In similar reduction systems, e.g. by Borgström et al., they introduced an explicit failure term. We allow terms to get stuck and therefore avoid a dedicated failure construct.

and $\mu_{\mathbb{S}}$ a (σ-finite) measure on $(\mathbb{S}, \Sigma_{\mathbb{S}})$.

3.3 (Positive) Almost-Sure Termination

For now we drop the subscript of the CbV reduction relation $\xrightarrow{\mathfrak{V}}$ and just write it as $\to$. With $\to^n$ we denote the n-fold relation-composition and with $\to^*$ the reflexive-transitive closure. We define $\mathbb{T}_{M,\text{term}}$ as the set of all traces on which a term M terminates. Formally:

$$\mathbb{T}_{M,\text{term}} \triangleq \{\boldsymbol{s} \in \mathbb{S} \mid \exists V : \langle M, \boldsymbol{s}\rangle \to^* \langle V, \epsilon\rangle\}$$

As the probability of termination for a term we take the measure of $\mathbb{T}_{M,\text{term}}$. We can easily encode SPCF in the untyped λ-calculus and then use [Borgström et al. 2016, Lemma 9] to get that the set $\mathbb{T}_{M,\text{term}}$ is a measurable set of traces. As shown in [Mak et al. 2021, Lemma 7], $\mu_{\mathbb{S}}\big(\mathbb{T}_{M,\text{term}}\big) \leq 1$, which justifies the interpretation of $\mu_{\mathbb{S}}\big(\mathbb{T}_{M,\text{term}}\big)$ as a "probability".

Definition 3.1: *For a closed term M the* probability of termination*, denoted $\mathbb{P}_{term}(M)$, is defined by $\mathbb{P}_{term}(M) \triangleq \mu_{\mathbb{S}}(\mathbb{T}_{M,term})$. M is called* AST *if $\mathbb{P}_{term}(M) = 1$.*

Positive Almost-Sure Termination A term is AST if it terminates with maximal probability. An even stronger property we can ask for is not only a.s. termination but termination in expected finite time. A natural measure of computation time is the number of reduction steps. For any trace $\boldsymbol{s} \in \mathbb{T}_{M,\text{term}}$ we define $\#^{\boldsymbol{s}}_{\downarrow}(M) \in \mathbb{N}$ as the unique number such that $\langle M, \boldsymbol{s}\rangle \to^{\#^{\boldsymbol{s}}_{\downarrow}(M)} \langle V, \epsilon\rangle$ for some value V Note this is well defined as all traces in $\mathbb{T}_{M,\text{term}}$ eventually terminate. For any $n \in \mathbb{N}$ we can define

$$\mathbb{T}^n_{M,\text{term}} \triangleq \{\boldsymbol{s} \in \mathbb{T}_{M,\text{term}} \mid \#^{\boldsymbol{s}}_{\downarrow}(M) = n\}$$

That is all traces on which M terminates in exactly n-steps. It is easy to see that $\mathbb{T}^n_{M,\text{term}}$ is measurable. We also define the set of traces where termination occurs within the first n steps by $\mathbb{T}^{\leq n}_{M,\text{term}} \triangleq \bigcup_{i \leq n} \mathbb{T}^i_{M,\text{term}}$. As a countable (in fact finite) union this set is also measurable.

Definition 3.2: *For a term M we define the expected time to termination, $\mathbb{E}_{term}(M) \in \overline{\mathbb{R}_+}$, by*

$$\mathbb{E}_{term}(M) \triangleq \sum_{n=0}^{\infty} \Big(1 - \mu_{\mathbb{S}}\big(\mathbb{T}^{\leq n}_{M,term}\big)\Big)$$

We say M is positive almost-surely terminating *(PAST) if $\mathbb{E}_{term}(M) < \infty$.*

The definition of the expected time to termination does not seem very intuitive as it does not resemble the "standard" definition of an expected value[6]. We can show that, provided M is

[6] As remarked by Kaminski and Katoen, the expected time to termination can be phrased as

AST, the expected time to termination is in fact the expected value of the random variable that gives the number of reduction steps:

Lemma 3.3 (Expected Termination Time as an Expectation): *If M is AST then*

$$\mathbb{E}_{term}(M) = \sum_{n=0}^{\infty} \mu_{\mathbb{S}}\left(\mathbb{T}^{n}_{M,term}\right) \cdot n$$

Proof We have $\biguplus_n \mathbb{T}^n_{M,\text{term}} = \mathbb{T}_{M,\text{term}}$. Now define $\mathbb{T}^{>n}_{M,\text{term}} \triangleq \biguplus_{i>n} \mathbb{T}^i_{M,\text{term}}$. It is easy to see that $\mu_{\mathbb{S}}\left(\mathbb{T}^{>n}_{M,\text{term}}\right) + \mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right) = 1$ as by assumption $\mu_{\mathbb{S}}\left(\mathbb{T}_{M,\text{term}}\right) = 1$. Now by definition $\mathbb{E}_{\text{term}}(M) = \sum_{n=0}^{\infty}\left(1 - \mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right)\right)$ and we can combine

$$\sum_{n=0}^{\infty}\left(1 - \mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right)\right) \overset{(1)}{=} \sum_{n=0}^{\infty} \mu_{\mathbb{S}}\left(\mathbb{T}^{>n}_{M,\text{term}}\right) \overset{(2)}{=} \sum_{n=0}^{\infty}\sum_{j>n} \mu_{\mathbb{S}}\left(\mathbb{T}^{i}_{M,\text{term}}\right) \overset{(3)}{=} \sum_{n=0}^{\infty} \mu_{\mathbb{S}}\left(\mathbb{T}^{n}_{M,\text{term}}\right) \cdot n$$

where **(1)** holds as $\mu_{\mathbb{S}}\left(\mathbb{T}^{>n}_{M,\text{term}}\right) + \mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right) = 1$, **(2)** follows as the union in the definition of $\mathbb{T}^{>n}_{M,\text{term}}$ is disjoint and **(3)** is an easy combinatorial argument. ■

Note that the above really only holds for terms that are AST. As an easy example to see this, take any term that diverges deterministically (i.e., terminates with probability 0.) Clearly the right hand side is 0 even though the expected time to termination is clearly infinite.

We can easily show that PAST is a stronger criterion than AST:

Lemma 3.4 (PAST implies AST): *If M is PAST then M is AST*

Proof Assume M is PAST so by definition $\sum_{n=0}^{\infty}\left(1-\mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right)\right)$ is a finite sum. As this sum converges to a finite value the sequence $\left(\mu_{\mathbb{S}}\left(\mathbb{T}^{\leq n}_{M,\text{term}}\right)\right)_{n\in\mathbb{N}}$ must converge to 1. And as $\mathbb{T}^{\leq n}_{M,\text{term}} \subseteq \mathbb{T}_{M,\text{term}}$ for every n and $\mu_{\mathbb{S}}$ is a measure (in particular monotone w.r.t. to $\subseteq$ and $\leq$) we get $\mu_{\mathbb{S}}(\mathbb{T}_{M,\text{term}}) = 1$, so M is AST. ■

3.4 Operational Semantics - Call by Name

We define call by name redexes and evaluation contexts by:

$$\begin{aligned} \mathit{Red}_{\mathscr{N}} \ni R &\triangleq (\lambda x.M)N \mid (\mu^{\varphi}_{x}.M)N \mid \texttt{if}(\underline{r}, N, P) \mid f(\underline{r_1}, \cdots, \underline{r_{|f|}}) \mid \texttt{sample} \mid \texttt{score}(\underline{r}) \\ \mathit{ECont}_{\mathscr{N}} \ni E &\triangleq [\cdot] \mid EM \mid \texttt{if}(E, N, P) \mid f(\underline{r_1}, \cdots, \underline{r_{k-1}}, E, M_{k+1}, \cdots, M_{|f|}) \mid \texttt{score}(E) \end{aligned}$$

Note that the right hand side of a β-redex is no longer required to be a value. For the evaluation contexts, we do not permit reductions in the argument position of an application.

$\sum_{n=0}^{\infty}\mathbb{P}(\text{“M runs for more than } n \text{ steps”}) = \sum_{n=0}^{\infty}\left(1-\mathbb{P}(\text{“M terminates within } n \text{ steps”})\right)$ the latter of which is expressed in Definition 3.2.

$$\frac{}{\langle (\lambda x.M)N, \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle M[N/x], \boldsymbol{s}\rangle} \qquad \frac{}{\langle (\mu_x^{\varphi}.M)N, \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle M[N/x, (\mu_x^{\varphi}.M)/\varphi], \boldsymbol{s}\rangle}$$

$$\frac{r \leq 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle N, \boldsymbol{s}\rangle} \qquad \frac{r > 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle P, \boldsymbol{s}\rangle} \qquad \frac{}{\langle \mathtt{sample}, r :: \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle \underline{r}, \boldsymbol{s}\rangle}$$

$$\frac{}{\langle f(\underline{r_1}, \cdots, \underline{r_{|f|}}), \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle \underline{f(r_1, \cdots, r_{|f|})}, \boldsymbol{s}\rangle} \qquad \frac{r \geq 0}{\langle \mathtt{score}(\underline{r}), \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle \underline{r}, \boldsymbol{s}\rangle} \qquad \frac{\langle R, \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle M, \boldsymbol{s}'\rangle}{\langle E[R], \boldsymbol{s}\rangle \xrightarrow{\mathfrak{N}} \langle E[M], \boldsymbol{s}'\rangle}$$

Figure 3.3: Call by Name small-step reduction for SPCF.

We define the CbN reduction relation by the rules in Fig. 3.3. The definitions in Sec. 3.3 regarding AST and PAST extend naturally to CbN[7]. Throughout this work we always make clear what evaluation strategy we are using, so the notation never clashes.

> **Remark:** Switching from CbV to CbN is not entirely straightforward as we need to check that the proofs in the literature also work in the CbN setting (especially since [Borgström et al. 2016; Mak et al. 2021] both work in a CbV setting). By careful inspection of their proof we can, however, verify that all results also hold for CbN as expected.

A natural question to ask is whether it is useful to explicitly consider CbN. After all, most functional languages do evaluate eager, i.e., follow a CbV or Call by Need reduction strategy. Especially in a probabilistic setting, CbN differs from CbV in that it does not allow for duplication of probabilistic outcomes. This interesting asymmetry motivates to study if and to what extend concepts are also applicable in CbN. Many concepts established in this work are applicable for *both* CbV and CbN and we discuss possible extensions at the end of each chapter. Most chapters in this work explicitly work in a CbV system. For the chapter on intersection types (Ch. 5) we do, however, switch to a CbN evaluation as this allows for an easier typing system (as already observed by Ehrhard et al. [2014] or the monadic type system in [Breuvart and Lago 2018, §6])

[7] While the concepts extend naturally, they obviously are not identical. E.g., the probability of a termination in CbN may very well differ from the one in CbV.

Part I

Interval-based Reasoning

Chapter 4

Interval-based Semantics

Reasoning about SPCF using the standard semantics is impractical, due to the continuous nature of program traces. Assume we are interested in the lower bound question, i.e., checking if a term terminates with probability strictly greater than p. In the discrete case, this problem is recursively enumerable (in Σ_1^0) as we can enumerate terminating paths until the sum of the weight of those paths exceeds p [Kaminski and Katoen 2015][1]. In our continuous semantics, this is not longer possible. A well known property of the Lebesgue measure on $\mathbb{R}^n$ (that the trace measure $\mu_{\mathbb{S}}$ inherits) is that every countable set of elements is a null-set. So even if we were to identify a countably infinite set of traces $A \subseteq \mathbb{T}_{M,\text{term}}$, we cannot obtain any possible lower bound on $\mathbb{P}_{\text{term}}(M)$. The semantics itself can thus not give a definite answer to questions like: is the lower bound problem for SPCF in Σ_1^0, the AST problem in Π_2^0 or the PAST problem in Σ_2^0? In this chapter, we introduce a novel operational semantics for SPCF by executing term parametrized by a trace of *intervals*. We demonstrate that this semantics is well suited to derive lower bound and establish completeness w.r.t. the standard one. This gives a, to our knowledge first, positive answer to the questions above.

Syntax of Interval Terms As we now work with interval traces we must also adjust the syntax of term slightly. Namely instead of having real-valued numerals, numerals in the interval-based semantics are now themself intervals. We define interval values and interval terms by the following where $a \leq b \in \mathbb{R}$ and again treat them modulo α-conversion.

$$\mathcal{V} \triangleq x \mid \underline{[a,b]} \mid \lambda x.\mathcal{M} \mid \mu_x^{\varphi}.\mathcal{M}$$
$$\mathcal{M},\mathcal{N},\mathcal{P} \triangleq \mathcal{V} \mid \mathcal{M}\mathcal{N} \mid \texttt{if}(\mathcal{M},\mathcal{N},\mathcal{P}) \mid f(\mathcal{M}_1,\cdots,\mathcal{M}_{|f|}) \mid \texttt{sample} \mid \texttt{score}(\mathcal{M})$$

[1] Note, that we phrase the lower bound problem with a strict inequality. While the set of programs M with $\mathbb{P}_{\text{term}}(M) > p$ is recursively enumerable (in a discrete language), the set of terms M with $\mathbb{P}_{\text{term}}(M) \geq p$ is not (See e.g. [Kobayashi et al. 2019; Kaminski and Katoen 2015]).

The simple type system (Fig. 3.1) naturally extends to interval terms. With $\Lambda_{\mathfrak{I}}$ we denote the set of well simply-typeable interval terms. We define the set of (closed) intervals as $\mathfrak{I} \triangleq \{[a,b] \mid a,b \in \mathbb{R}, a \leq b\}$ and define $\mathfrak{I}_{0,1} \triangleq \{[a,b] \mid a,b \in \mathbb{R}, 0 \leq a \leq b \leq 1\}$ as the set of intervals with endpoints within $[0,1]$. With $\mathfrak{I}^{\mathbb{Q}}$ and $\mathfrak{I}^{\mathbb{Q}}_{0,1}$ we denote the sets $\mathfrak{I}$ and $\mathfrak{I}_{0,1}$ respectively, restricted to *rational endpoints*.

> **Remark:** In mathematics, we usually write $[a,b]$ to refer to the set $\{r \in \mathbb{R} \mid a \leq r \leq b\}$. That is we treat $[a,b]$ as an abbreviation of a set ($[a,b] \subseteq \mathbb{R}$). From a different viewpoint we can also view $[a,b]$ as just a symbol that is fully characterized by two numbers $a,b \in \mathbb{R}$. Intervals are thus well suited to denote an uncountable set while at the same time being fully characterized by just two parameters.

Interval Preserving Functions Numerals in interval terms are $[a,b]$ and can informally be understood as any value within that interval (we formalize this in Fig. 4.2). When we present a reduction relation we, therefore, require that the primitive functions preserve intervals. This motivates the class of functions we call *interval preserving*, to which we restrict the set of primitive functions. For a function $f : \mathbb{R}^n \to \mathbb{R}$ and a subset $A \subseteq \mathbb{R}^n$ we write the image of f on A as $f(A) \triangleq \{f(a) \mid a \in A\}$.

> **Definition 4.1:** *We call* $f : \mathbb{R}^n \to \mathbb{R}$ interval preserving *if there is a function* $\hat{f} : \mathbb{R}^{2n} \to \mathfrak{I}$ *such that for every sequence of intervals* $[a_1,b_1], \cdots, [a_n,b_n] \in \mathfrak{I}$ *we have* $f\big([a_1,b_1] \times \cdots \times [a_n,b_n]\big) = \hat{f}(a_1,b_1,\cdots,a_n,b_n)$*, i.e., the image of every n-dimensional box is an interval.*

Many interesting primitive functions (including e.g. $+, \cdot, -, \exp, \cdots$) are interval preserving. If we consider addition, $+$, as an example we can derive $\hat{+}$ explicitly by $\hat{+}(a_1,b_1,a_2,b_2) \triangleq [a_1+b_1, a_2+b_2]$. The following gives us a broad class of interval preserving functions:

> **Lemma 4.2 :** *If* $f : \mathbb{R}^n \to \mathbb{R}$ *is continuous then* f *is interval preserving*

> **Proof** Let $A \triangleq [a_1,b_1] \times \cdots \times [a_n,b_n]$ as in the definition of interval perseverance. We need higher-dimensional version of the intermediate value theorem (IVT): if $\mathbf{x}, \mathbf{y} \in A$ and $c \in \mathbb{R}$ be such that $f(\mathbf{x}) \leq c \leq f(\mathbf{y})$ then there is a $\mathbf{z} \in A$ such that $f(\mathbf{z}) = c$ **(1)**. The IVT implies that $f(A)$ is a connected set. A standard property of continuous functions is that the images of compact sets are compact sets. Due to the Heine–Borel theorem, compact euclidean sets are exactly those that are bounded and closed. As A is obviously compact, we get that $f(A)$ is compact and thus bounded and closed. As $f(A)$ is also connected (by the IVT), it is a closed (bounded) interval as required.
>
> It remains to show **(1)**: Let $\gamma : [0,1] \to A$ be defined by $\gamma(t) \triangleq t\mathbf{x} + (1-t)\mathbf{y}$ which is obviously continuous. In particular, note that since A is a box (and thus convex) $\gamma(t) \in A$ for every $t \in [0,1]$. Now define $\phi : [0,1] \to \mathbb{R}$ by $\phi \triangleq f \circ \gamma$ which is the composition of

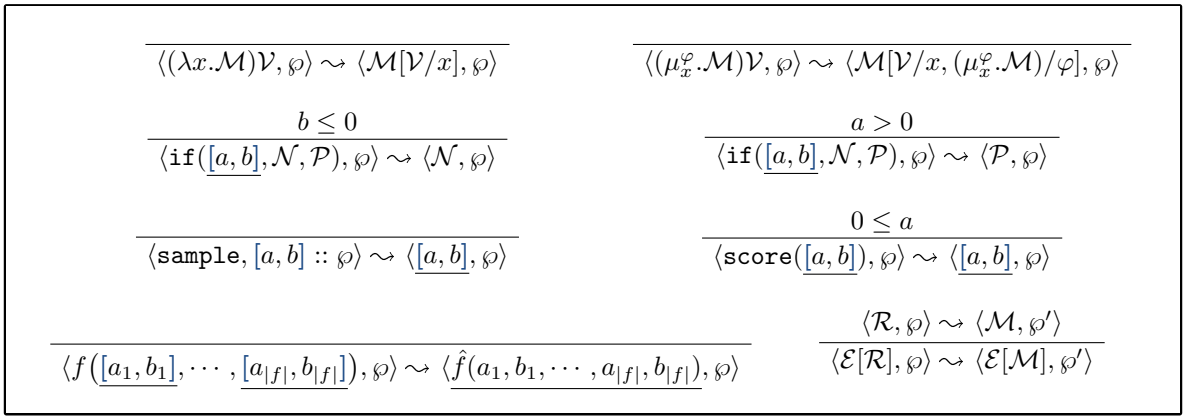


Figure 4.1: Internal-based (CbV) small-step reduction.

> continuous functions and thus also continuous. Now $\phi(0) = f(\mathbf{x}) \le c \le f(\mathbf{y}) = \phi(1)$ so by the intermediate value theorem in the 1d case there exists a $t \in [0,1]$ with $\phi(t) = c$. We can define $\mathbf{z} \triangleq \gamma(t)$ which satisfies the requirement by definition of ϕ. ■

To allow for a restriction of intervals with rational endpoints we also define the concept of $f : \mathbb{R}^n \to \mathbb{R}$ being $\mathbb{Q}$-*interval preserving* by restricting the above to interval with rational endpoints. I.e., we require a function $\hat{f} : \mathbb{Q}^{2n} \to \mathfrak{I}^{\mathbb{Q}}$ such that for every $[a_1,b_1],\cdots,[a_n,b_n] \subseteq \mathfrak{I}^{\mathbb{Q}}$, $f\big([a_1,b_1]\times\cdots\times[a_n,b_n]\big) = \hat{f}(a_1,b_1,\cdots,a_n,b_n)$. Informally the image of every n-dimensional box with rational endpoints is an interval with rational endpoints.

4.1 Interval-based Syntax and Semantics

We define the set of interval traces by $\mathbb{S}_{\mathfrak{I}} \triangleq \mathfrak{I}^*_{0,1} = \bigcup_{n\in\mathbb{N}} \mathfrak{I}^n_{0,1}$, i.e., finite sequences of intervals with endpoints between 0 and 1 (inclusive). We let $\wp$ range over elements in $\mathbb{S}_{\mathfrak{I}}$. To avoid confusion, we shall refer to elements of $\mathbb{S}_{\mathfrak{I}}$ as *interval traces*, and elements of $\mathbb{S}$ as *standard traces*. We define redexes and evaluation contexts by:

$$\mathcal{R} \triangleq (\lambda x.\mathcal{M})\mathcal{V} \mid (\mu^{\varphi}_{x}.\mathcal{M})\mathcal{V} \mid \mathtt{if}([a,b],\mathcal{N},\mathcal{P}) \mid f([a_1,b_1],\cdots,[a_{|f|},b_{|f|}]) \mid \mathtt{sample} \mid \mathtt{score}([a,b])$$
$$\mathcal{E} \triangleq [\cdot] \mid \mathcal{E}\mathcal{M} \mid (\lambda x.\mathcal{M})\mathcal{E} \mid (\mu^{\varphi}_{x}.\mathcal{M})\mathcal{E} \mid \mathtt{if}(\mathcal{E},\mathcal{N},\mathcal{P})$$
$$\mid f([a_1,b_1],\cdots,[a_{k-1},b_{k-1}],\mathcal{E},\mathcal{M}_{k+1},\cdots,\mathcal{M}_{|f|}) \mid \mathtt{score}(\mathcal{E})$$

We can state the small-step semantics, based on interval traces, as a relation $\rightsquigarrow \subseteq (\Lambda_{\mathfrak{I}} \times \mathbb{S}_{\mathfrak{I}})^2$. The relation is defined by the rules in Fig. 4.1. The reduction rules for interval terms look almost identical to the CbV reduction for standard SPCF. The only difference is that real values (standard) numerals are now replaced by interval numerals. An interval numeral $[a,b]$ can be understood as any value within that interval. If we e.g. focus on the rule for reducing

$$\frac{}{x \lhd x} \qquad \frac{}{\texttt{sample} \lhd \texttt{sample}} \qquad \frac{r \in [a,b]}{\underline{r} \lhd \underline{[a,b]}} \qquad \frac{M \lhd \mathcal{M}}{\lambda x.M \lhd \lambda x.\mathcal{M}}$$

$$\frac{M \lhd \mathcal{M} \qquad N \lhd \mathcal{N}}{MN \lhd \mathcal{M}\mathcal{N}} \qquad \frac{M \lhd \mathcal{M}}{\mu^{\varphi}_{x}.M \lhd \mu^{\varphi}_{x}.\mathcal{M}} \qquad \frac{M \lhd \mathcal{M}}{\texttt{score}(M) \lhd \texttt{score}(\mathcal{M})}$$

$$\frac{M \lhd \mathcal{M} \qquad N \lhd \mathcal{N} \qquad P \lhd \mathcal{P}}{\texttt{if}(M,N,P) \lhd \texttt{if}(\mathcal{M},\mathcal{N},\mathcal{P})} \qquad \frac{M_1 \lhd \mathcal{M}_1 \qquad \cdots \qquad M_{|f|} \lhd \mathcal{M}_{|f|}}{f(M_1,\cdots,M_{|f|}) \lhd f(\mathcal{M}_1,\cdots,\mathcal{M}_{|f|})}$$

Figure 4.2: Inductive definition of the refinement relation $\lhd$ between Λ and $\Lambda_{\mathfrak{I}}$.

conditional redexes we thus only reduce if the entire interval (i.e., every possible value) is either at most 0 or greater than 0. For example, a term of the form $\texttt{if}(\underline{[-1,1]},\mathcal{N},\mathcal{P})$ cannot reduce further. As in the standard semantics, we are interested in all interval traces that lead to a normal form. We hence define

$$\mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}} \triangleq \{\wp \in \mathbb{S}_{\mathfrak{I}} \mid \exists \mathcal{V} : \langle \mathcal{M}, \wp\rangle \rightsquigarrow^{*} \langle \mathcal{V}, \epsilon\rangle\}$$

For any $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$, we define $\#^{\wp}_{\downarrow}(\mathcal{M})$ as the unique number such that $\langle \mathcal{M}, \wp\rangle \rightsquigarrow^{\#^{\wp}_{\downarrow}(\mathcal{M})} \langle \mathcal{V}, \epsilon\rangle$. Note that this is well defined as $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$, i.e., the reduction eventually terminates in a value.

Refinement between Standard and Interval Terms It is very intuitive to see an interval numeral $[a,b]$ as any value between a and b. An interval term can thus be seen as a summary of many standard terms. We can formalize this as a relation $\lhd$ between ordinary terms Λ and interval terms $\Lambda_{\mathfrak{I}}$. Loosely speaking, $M \lhd \mathcal{M}$ if both have the same structure and every numeral position in M is a number within the interval numeral at the respective position in $\mathcal{M}$. This relation is formalized in Fig. 4.2. We can also define the refinement between standard traces and interval traces as a different relation $\lhd \subseteq \mathbb{S} \times \mathbb{S}_{\mathfrak{I}}$[2]. $\lhd$ for traces is defined as expected by: $r_1 \cdots r_n \lhd [a_1,b_1] \cdots [a_n,b_n]$ if and only if $r_i \in [a_i,b_i]$ for all $i \in [n]$. Note that if $\boldsymbol{s} \lhd \wp$, $|\boldsymbol{s}| = |\wp|$. We can then state the following result that formally justifies to reason in the interval-based semantics:

> **Lemma 4.3 :** *If $\langle \mathcal{M}, \wp\rangle \rightsquigarrow^{*} \langle \mathcal{N}, \wp'\rangle$ and $M \lhd \mathcal{M}$ and $\boldsymbol{s} \lhd \wp$ then there exists a $N \lhd \mathcal{N}$ and $\boldsymbol{s}' \lhd \wp'$ such that $\langle M, \boldsymbol{s}\rangle \rightarrow^{*} \langle N, \boldsymbol{s}'\rangle$.*

> **Proof** We first show that $\lhd$ is stable under substitution and evaluation contexts. We then do a simple induction on $\mathcal{M}$. A full proof can be found in the appendix: A ■

[2] We use the same symbol to avoid cluttered notation and as it is always clear from the context if we refer to the refinement between terms or traces.

Canonical Embedding As we work with closed intervals we can embed SPCF terms into interval terms in the obvious way by mapping a numeral $\underline{r}$ to the interval numeral $\underline{[r,r]}$. We denote this canonical interval term by $M^{2\mathfrak{J}}$. Obviously $M \triangleleft M^{2\mathfrak{J}}$.

4.2 Soundness

We show that reasoning in the interval-based semantics is sound for the standard semantics. In our context, we show that terminating interval traces give lower bounds on the probability of termination and excepted time to termination in the standard semantics. For an interval trace $\wp \in \mathbb{S}_{\mathfrak{J}}$, we define the set of standard traces that refine $\wp$ by $(\!|\wp|\!) \triangleq \{\boldsymbol{s} \in \mathbb{S} \mid \boldsymbol{s} \triangleleft \wp\}$. Using Lem. 4.3 it is easy to see the following:

Lemma 4.4: *If $\wp \in \mathbb{T}^{\mathfrak{J}}_{\mathcal{M},term}$ and $M \triangleleft \mathcal{M}$ then $(\!|\wp|\!) \subseteq \mathbb{T}_{M,term}$.*

This lemma states that if we find an interval trace $\wp$ on which $\mathcal{M}$ terminates then the set of refined standard traces $(\!|\wp|\!)$ are terminating for every term that refines $\mathcal{M}$. Suppose we are given a standard term M we can thus analyse $M^{2\mathfrak{J}}$ (the canonical embedding) and use any terminating interval traces found to obtain terminating traces for M.

Weight of Interval Traces We define the weight of an interval trace $\wp$, denoted by $\omega(\wp)$, in the obvious way:

$$\omega([a_1,b_1],\cdots,[a_n,b_n]) \triangleq \prod_{i=1}^{n}(b_i - a_i)$$

It is easy to see that for every $\wp \in \mathbb{S}_{\mathfrak{J}}$, $(\!|\wp|\!)$ is a measurable set of traces. Furthermore the measure on traces is defined in terms of Lebesgue measures, so the measure of any box is just the product of the side-lengths (the volume of the box). So for any $\wp \in \mathbb{S}_{\mathfrak{J}}$, $\mu_{\mathbb{S}}((\!|\wp|\!)) = \omega(\wp)$. When combining this with Lem. 4.4 we get that if $\wp \in \mathbb{T}^{\mathfrak{J}}_{\mathcal{M},\mathrm{term}}$ and $M \triangleleft \mathcal{M}$ then $\omega(\wp) \leq \mathbb{P}_{\mathrm{term}}(M)$. Every interval trace derives a lower bound for $\mathbb{P}_{\mathrm{term}}(M)$ for every refined program. This simple results alone already allows us to derive meaningful (in the sense of non-trivial) lower bounds on the probability of termination. As an example consider the SPCF term given in the introduction: the interval trace in Fig. 1.1c is terminating for the canonical embedding and all contained traces are terminating in the standard semantics.

Pairwise Compatible Traces Ideally, we would like to combine the weight of multiple terminating interval traces to derive even better bounds on the probability of termination. To ensure that the sum of the weights of intervals traces is indeed a lower bound on $\mathbb{P}_{\mathrm{term}}(M)$ we must ensure that the interval traces are disjoint, i.e., we do not account twice for the same standard trace. We call two measurable sets $A, B \subseteq \mathbb{R}^n$ *almost disjoint* if their intersection has Lebesgue measure 0. If A, B are both n-dimensional boxes this is equivalent to the fact

that the intersecting box has volume 0, i.e., is an at most $n-1$-dimensional object.

Definition 4.5: *Two interval traces $\wp$ and $\wp'$ are* compatible *if $|(\!|\wp|\!) \cap (\!|\wp'|\!)| \leq 1$, i.e., they share at most one refined standard trace.*

As interval traces can be seen as boxes we can easily check compatibility: $\wp$ and $\wp'$ are compatible if and only if either $|\wp| \neq |\wp'|$ or $(\!|\wp|\!)$ and $(\!|\wp'|\!)$ are almost disjoint[3]. As an example the four interval traces $[0,1][0,\frac{1}{3}]$, $[0,1][\frac{1}{3},\frac{1}{2}]$, $[0,1][\frac{3}{4},1]$ and $[0,1]$ are all pairwise compatible.

For a countable set of interval traces A we define $\omega(A) \triangleq \sum_{\wp\in A}\omega(\wp)$. We only take countable sums of interval traces so we actually do not need to worry about defining a proper σ-algebra on the (uncountable) set of interval traces. For a countable set $A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$ we also define the expected value of A, denoted $\mathbb{E}^{\mathcal{M}}(A)$ by

$$\mathbb{E}^{\mathcal{M}}(A) \triangleq \sum_{\wp\in A}\omega(\wp)\cdot \#^{\wp}_{\downarrow}(\mathcal{M})$$

We can now state soundness as follows:

Theorem 1 (Soundness of the Interval-based Semantics): *For every countable set of* pairwise compatible *traces $A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},term}$ and every $M \triangleleft \mathcal{M}$ we have the following:*

- $\omega(A) \leq \mathbb{P}_{term}(M)$
- $\mathbb{E}^{\mathcal{M}}(A) \leq \mathbb{E}_{term}(M)$

Proof As A is pairwise compatible the family $((\!|\wp|\!))_{\wp\in A}$ is pairwise almost disjoint. Now

$$\omega(A) = \sum_{\wp\in A}\omega(\wp) \overset{(1)}{=} \sum_{\wp\in A}\mu_{\mathbb{S}}\big((\!|\wp|\!)\big) \overset{(2)}{=} \mu_{\mathbb{S}}\Big(\bigcup_{\wp\in A}(\!|\wp|\!)\Big) \overset{(3)}{\leq} \mu_{\mathbb{S}}\big(\mathbb{T}_{M,\text{term}}\big) = \mathbb{P}_{\text{term}}(M)$$

where **(1)** follows from the definition of the Lebesgue measure on boxes, **(2)** from the fact that family is pairwise almost disjoint and thus differs by a countable union of null sets. **(3)** follows from Lem. 4.4.
For the second part we can observe that if $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$, $M \triangleleft \mathcal{M}$ and $\boldsymbol{s} \triangleleft \wp$, then $\#^{\boldsymbol{s}}_{\downarrow}(M) = \#^{\wp}_{\downarrow}(\mathcal{M})$. Now by definition $\mathbb{E}^{\mathcal{M}}(A) = \sum_{\wp\in A}\omega(\wp)\cdot\#^{\wp}_{\downarrow}(\mathcal{M})$ and

$$\sum_{\wp\in A}\omega(\wp)\cdot\#^{\wp}_{\downarrow}(\mathcal{M}) \overset{(1)}{=} \sum_{n=0}^{\infty}\omega(\{\wp\in A \mid \#^{\wp}_{\downarrow}(\mathcal{M}) = n\})\cdot n \overset{(2)}{\leq} \sum_{n=0}^{\infty}\mu_{\mathbb{S}}\Big(\mathbb{T}^{n}_{M,\text{term}}\Big)\cdot n \overset{(3)}{\leq} \mathbb{E}_{\text{term}}(M)$$

where **(1)** follows from simple reordering, **(2)** from the fact that every interval trace in $\{\wp\in A \mid \#^{\wp}_{\downarrow}(\mathcal{M}) = n\}$ we get $(\!|\wp|\!) \subseteq \mathbb{T}^{n}_{M,\text{term}}$ and the same reasoning as above. **(3)** is standard and can e.g. be inferred from the proof of Lem. 3.3. ■

[3]Note that almost disjointness of boxes means that there exist a dimension i, such that $\wp(i)$ and $\wp'(i)$ are almost disjoint intervals, i.e., at any position they overlap by at most one point.

$$\frac{}{\langle (\lambda x.M)V, \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle M[V/x], \boldsymbol{s}, \kappa\rangle} \qquad \frac{}{\langle \mathtt{sample}, r :: \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle \underline{r}, \boldsymbol{s}, \kappa\rangle}$$

$$\frac{}{\langle (\mu_x^{\varphi}.M)V, \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle M[V/x, (\mu_x^{\varphi}.M)/\varphi], \boldsymbol{s}, \kappa\rangle} \qquad \frac{r \geq 0}{\langle \mathtt{score}(\underline{r}), \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle \underline{r}, \boldsymbol{s}, \kappa\rangle}$$

$$\frac{r \leq 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}, \mathbf{L} :: \kappa\rangle \xrightarrow{co} \langle N, \boldsymbol{s}, \kappa\rangle} \qquad \frac{r > 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}, \mathbf{R} :: \kappa\rangle \xrightarrow{co} \langle P, \boldsymbol{s}, \kappa\rangle}$$

$$\frac{}{\langle f(\underline{r_1}, \cdots, \underline{r_{|f|}}), \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle \underline{f(r_1, \cdots, r_{|f|})}, \boldsymbol{s}, \kappa\rangle} \qquad \frac{\langle R, \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle M, \boldsymbol{s}', \kappa'\rangle}{\langle E[R], \boldsymbol{s}, \kappa\rangle \xrightarrow{co} \langle E[M], \boldsymbol{s}', \kappa'\rangle}$$

Figure 4.3: Small-step reduction relation with conditional oracles.

This soundness result is not really surprising but nevertheless offers a sound and powerful technique to infer lower bounds on the probably of termination.

Completeness of the interval semantics We have seen (Thm. 1) if there is an set of of interval traces A such that $M^{2\mathfrak{I}}$ terminates on every $\wp \in A$, and furthermore they are pairwise compatible, then the sum of the volume of the interval traces gives a lower bound on $\mathbb{P}_{\text{term}}(M)$. The remaining parts of this chapter are devoted to the proof that the interval method is also complete. In our context, this means that we can find a *countable* sequence of pairwise compatible interval traces whose sum *equals* $\mathbb{P}_{\text{term}}(M)$. In particular, this implies that we can get arbitrary tight lower bound on $\mathbb{P}_{\text{term}}(M)$ via finitely many interval traces.

> **Example 4.6:** As an example consider the following term $M \triangleq \mathtt{if}(\mathtt{sample} - \underline{0.5}, \underline{0}, \underline{1})$ which is clearly AST. In fact, we have $\mathbb{T}_{M,\text{term}} = \{s_1 \mid s_1 \in \mathbb{R}_{[0,1]}\}$, so the set of terminating traces is itself an interval. However, the interval trace $\wp = [0,1]$ is not terminating for $M^{2\mathfrak{I}}$ (formally $[0,1] \notin \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\text{term}}$). We can, however, define a countable family of interval traces that are pairwise disjoint and lead to termination: Let $(a_i)_{i\in\mathbb{N}}$ be any monotone decreasing sequence $a_i \in (0, \frac{1}{2}]$ with $a_0 = \frac{1}{2}$ and $\lim_{i\to\infty} a_i = 0$. For instance $a_i \triangleq \frac{1}{2^{i+1}}$. Then the family $\{\wp\} \cup \{\wp_i \mid i \in \mathbb{N}\}$ defined by $\wp \triangleq [0, \frac{1}{2}]$ and $\wp_i \triangleq [\frac{1}{2} + a_{i+1}, \frac{1}{2} + a_i]$ for $i \in \mathbb{N}$ is a countable family of terminating interval traces that are pairwise compatible and whose cumulative weights equals 1.

4.3 Conditional Oracle

The key step to constructing a countable set of interval traces is to focus on branching. We, therefore, annotate the reduction relation with explicit information which branch of a

conditional was taken. We define the set of *directions* by $D = \{\mathbf{L}, \mathbf{R}\}$. A *conditional oracle* is then a sequence $\kappa \in D^*$. To define the meaning of a conditional oracle we use a modified reduction relation $\xrightarrow{co} \subseteq (\Lambda \times \mathbb{S} \times D^*)^2$ via the rules in Fig. 4.3. The reduction relation agrees with the standard CbV reduction but explicitly annotates the branching decisions. For any terminating standard trace $\boldsymbol{s}$, that is $\langle M, \boldsymbol{s}\rangle \to^* \langle V, \epsilon\rangle$, there exists a *unique* conditional oracle $\kappa \in D^*$, s.t., $\langle M, \boldsymbol{s}, \kappa\rangle \xrightarrow{co}{}^* \langle V, \epsilon, \epsilon\rangle$. Call this observation **(1)**. We now partition the set of terminating traces according to their branching behavior. For $\kappa \in D^*$ we define

$$\mathbb{T}^{(\kappa)}_{M,\text{term}} \triangleq \{\boldsymbol{s} \in \mathbb{S} \mid \exists V : \langle M, \boldsymbol{s}, \kappa\rangle \xrightarrow{co}{}^* \langle V, \epsilon, \epsilon\rangle\}$$

I.e., all traces that branch according to κ. By our observation **(1)** it is easy to see that that the family $\left\{\mathbb{T}^{(\kappa)}_{M,\text{term}}\right\}_{\kappa\in D^*}$ forms a partition of the set of terminating traces: $\mathbb{T}_{M,\text{term}} = \biguplus_{\kappa\in D^*} \mathbb{T}^{(\kappa)}_{M,\text{term}}$. Note that by fixing the branching, we also fix the number of reduction steps and the number of samples: if $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathbb{T}^{(\kappa)}_{M,\text{term}}$, $\#^{\boldsymbol{s}_1}_{\downarrow}(M) = \#^{\boldsymbol{s}_2}_{\downarrow}(M)$ and $|\boldsymbol{s}_1| = |\boldsymbol{s}_2|$.

In Ex. 4.6, we have seen that there exit interval traces $\wp \in \mathbb{S}_{\mathfrak{I}}$ with $(\!|\wp|\!) \subseteq \mathbb{T}_{M,\text{term}}$ that are not terminating for the canonical embedding $M^{2^{\mathfrak{I}}}$ (i.e., $\wp \notin \mathbb{T}^{\mathfrak{I}}_{M^{2^{\mathfrak{I}}},\text{term}}$). We can, however, show that if all traces in $(\!|\wp|\!)$ follow the same branching $\wp \in \mathbb{T}^{\mathfrak{I}}_{M^{2^{\mathfrak{I}}},\text{term}}$ does hold:

Lemma 4.7: *If $\wp \in \mathbb{S}_{\mathfrak{I}}$, $\kappa \in D^*$ and $(\!|\wp|\!) \subseteq \mathbb{T}^{(\kappa)}_{M,term}$ then $\wp \in \mathbb{T}^{\mathfrak{I}}_{M^{2^{\mathfrak{I}}},term}$.*

Proof We show a more general statement: if $\mathcal{M}$ is any interval term, $\wp \in \mathbb{S}_{\mathfrak{I}}$ and $\kappa \in D^*$ and for all $M \lhd \mathcal{M}$, $(\!|\wp|\!) \subseteq \mathbb{T}^{(\kappa)}_{M,\text{term}}$ then $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$. The proof of this goes by easy induction on $\mathcal{M}$. As the only term refining $M^{2^{\mathfrak{I}}}$ is M itself we are done. A full proof can be found in the appendix: A. ■

Example 4.8: Take as an example term

$$M \triangleq \texttt{if sample} - .5 \,\texttt{then}\, \underline{0} \,\texttt{else}\, \left(\texttt{if sample} + \texttt{sample} - .6 \,\texttt{then}\, \underline{2} \,\texttt{else}\, \underline{3}\right)$$

Then for example $\langle M, [0, \frac{1}{2}, 0], \mathbf{RL}\rangle \xrightarrow{co}{}^* \langle \underline{2}, \epsilon, \epsilon\rangle$. When we look at the sets $\mathbb{T}^{(\kappa)}_{M,\text{term}}$ we get:

$$\mathbb{T}^{(\mathbf{L})}_{M,\text{term}} = \{s_1 \in \mathbb{R}_{[0,1]} \mid s_1 \leq .5\} \qquad \begin{aligned}\mathbb{T}^{(\mathbf{RL})}_{M,\text{term}} &= \{s_1 s_2 s_3 \in \mathbb{R}^3_{[0,1]} \mid s_1 > .5 \wedge s_2 + s_3 \leq .6\}\\ \mathbb{T}^{(\mathbf{RR})}_{M,\text{term}} &= \{s_1 s_2 s_3 \in \mathbb{R}^3_{[0,1]} \mid s_1 > .5 \wedge s_2 + s_3 > .6\}\end{aligned}$$

and $\mathbb{T}^{(\kappa)}_{M,\text{term}} = \emptyset$ for any other κ. For the interval trace $\wp \triangleq [\frac{3}{4}, 1][0, \frac{1}{2}][0, \frac{1}{10}]$ we get $(\!|\wp|\!) \subseteq \mathbb{T}^{(\mathbf{RL})}_{M,\text{term}}$ and indeed $\wp \in \mathbb{T}^{\mathfrak{I}}_{M^{2^{\mathfrak{I}}},\text{term}}$ as stated in general in Lem. 4.7.

4.4 Symbolic Terms and Symbolic Execution

What we show is that every set $\mathbb{T}^{(\kappa)}_{M,\text{term}}$ can be covered up to null-set by a countable set of interval traces. To formally prove this we need a more detailed understanding of the structure of $\mathbb{T}^{(\kappa)}_{M,\text{term}}$. We can achieve this insight by considering symbolic terms and symbolic execution. The idea of symbolic terms is to not evaluate a term on a fixed trace of real numbers but instead on a generic trace consisting of variables. Whenever we resolve a `sample`-statement we do not substitute in a real number but a variable. This does prohibit us from evaluating primitive functions or resolve conditionals. To circumvent the former we use symbolic values, which can be seen as partially evaluate primitive functions. To resolve the latter we make use of the conditional oracles. For an overview of a similar system of symbolic execution we refer the reader to [Mak et al. 2021].

Syntax of Symbolic Terms Let $\alpha_0, \alpha_1, \cdots$ be a denumerable set of *sample-variables* indexed by natural numbers. We use them to postpone every sample statement by instead substitution a fresh variable. Symbolic values and terms are defined by:

$$\mathfrak{V} \triangleq x \mid \underline{r} \mid \alpha_j \mid \lambda x.\mathfrak{M} \mid \mu^{\varphi}_x.\mathfrak{M} \mid \boxed{f}(\mathfrak{V}_1, \cdots, \mathfrak{V}_{|f|})$$

$$\mathfrak{M}, \mathfrak{N}, \mathfrak{P} \triangleq \mathfrak{V} \mid \mathfrak{M}\mathfrak{N} \mid \texttt{if}(\mathfrak{M}, \mathfrak{N}, \mathfrak{P}) \mid f(\mathfrak{M}_1, \cdots, \mathfrak{M}_{|f|}) \mid \texttt{sample} \mid \texttt{score}(\mathfrak{M})$$

Note that the only new syntactic additions, compared with standard SPCF, are the sample variables α_j and the symbolic primitive functions $\boxed{f}(\mathfrak{M}_1, \cdots, \mathfrak{M}_{|f|})$. We again focus on typeable terms. The simple type system for standard SPCF (given in Fig. 3.1) naturally extends to symbolic terms when we add the following two rules:

$$\frac{}{\Gamma \Vdash \alpha_j : \mathbf{R}} \qquad \frac{\Gamma \Vdash \mathfrak{M}_1 : \mathbf{R} \quad \cdots \quad \Gamma \Vdash \mathfrak{M}_{|f|} : \mathbf{R}}{\Gamma \Vdash \boxed{f}(\mathfrak{M}_1, \cdots, \mathfrak{M}_{|f|}) : \mathbf{R}}$$

Let Λ_{sym} be the set of all typeable symbolic terms. Note that any $M \in \Lambda$ directly corresponds to a symbolic term in the canonical way.

Symbolic Values It is worth to study symbolic values in some more detail. As sample variables are taken in for real-valued numerals, whenever we resolve a sample statement, we can no longer evaluate primitive functions as some of the arguments may be sample variables. A function symbol f applied to arguments, therefore, does not evaluate to the function value but instead we postpone the evaluation and use the symbolic construct $\boxed{f}$. In particular, a (closed) symbolic value of type $\mathbf{R}$ is no longer always a numeral. We can view $\boxed{f}$ as a function evaluation that is postponed. If we fix the value of the sample variables, a symbolic value, therefore, does again denotes a real number: Let $\mathfrak{V}$ be a symbolic value of type $\mathbf{R}$ (no λ or μ-abstraction) with sample-variables within $\{\alpha_0, \cdots, \alpha_{m-1}\}$. We can view a vector $\sigma \in \mathbb{R}^m_{[0,1]}$ as

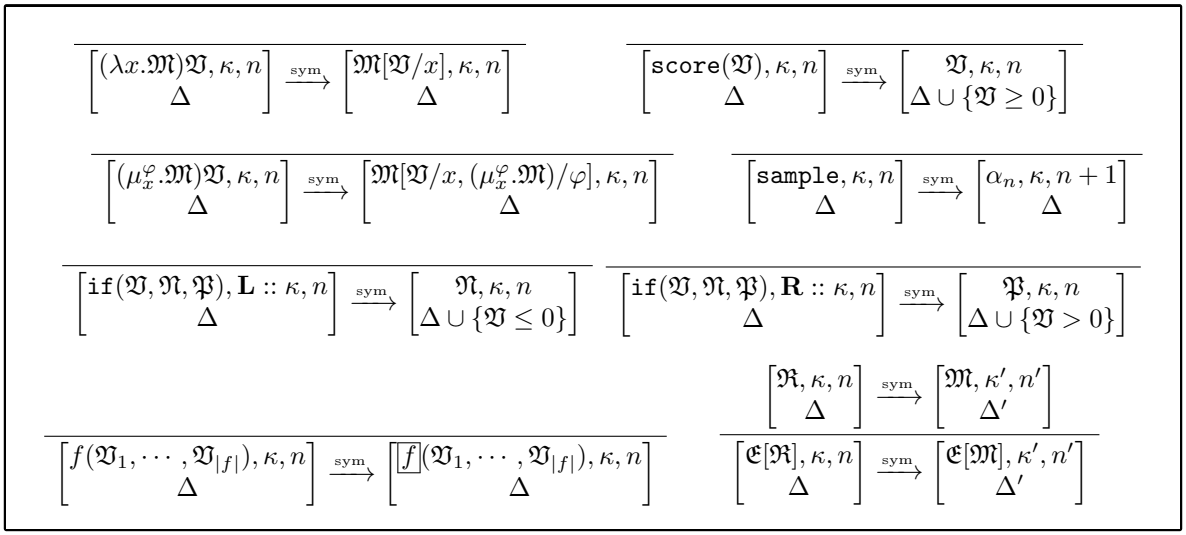


Figure 4.4: Small-step reduction for symbolic terms (symbolic execution).

a substitution and define $\mathfrak{V}[\sigma] \in \mathbb{R}$ in the obvious way by substituting in values and evaluating primitive functions. Given $A \subseteq \mathbb{R}$ we define $\mathfrak{V}^{-1}(A) \triangleq \{\sigma \in \mathbb{R}^m_{[0,1]} \mid \mathfrak{V}[\sigma] \in A\}$.

Symbolic Inequality We define a symbolic inequality as pairs of the from $(\mathfrak{V} \bowtie r)$ where $\mathfrak{V}$ is a symbolic value, $\bowtie \in \{\leq, <, \geq, >\}$ and $r \in \mathbb{R}$. A symbolic constraint Δ is a set of symbolic inequalities. Given a symbolic constraint $\Delta = \{(\mathfrak{V}_i \bowtie_i r_i)\}_{i\in[n]}$ with sample variables contained within $\alpha_0, \cdots, \alpha_{m-1}$ we can define

$$\mathsf{Sat}_m(\Delta) \triangleq \{\sigma \in \mathbb{R}^m_{[0,1]} \mid \forall i \in [n] : \mathfrak{V}_i[\sigma] \bowtie_i r_i\}$$

We can see every $\sigma \in \mathsf{Sat}_m(\Delta)$ also as an element in $\mathbb{S}^m$, i.e., a standard trace of length m.

Symbolic Execution We now give an operational small-step semantics to symbolic terms. This symbolic execution closely corresponds to reduction in the standard (CbV) semantics with the exception that every `sample`-statement is resolved by a sample variable. Symbolic redexes and evaluation contexts are defined as expected:

$$\mathfrak{R} \triangleq (\lambda x.\mathfrak{M})\mathfrak{V} \mid (\mu_x^{\varphi}.\mathfrak{M})\mathfrak{V} \mid \texttt{if}(\mathfrak{V},\mathfrak{N},\mathfrak{P}) \mid f(\mathfrak{V}_1,\cdots,\mathfrak{V}_{|f|}) \mid \texttt{sample} \mid \texttt{score}(\mathfrak{V})$$
$$\mathfrak{E} \triangleq [\cdot] \mid \mathfrak{E}\mathfrak{M} \mid (\lambda y.\mathfrak{M})\mathfrak{E} \mid (\mu_x^{\varphi}.\mathfrak{M})\mathfrak{E} \mid \texttt{if}(\mathfrak{E},\mathfrak{N},\mathfrak{P}) \mid f(\mathfrak{V}_1,\cdots,\mathfrak{V}_{k-1},\mathfrak{E},\mathfrak{M}_{k+1},\cdots,\mathfrak{M}_{|f|}) \mid \texttt{score}(\mathfrak{E})$$

Symbolic Configuration A symbolic configuration has the from $\left[\begin{smallmatrix}\mathfrak{M},\kappa,n\\ \Delta\end{smallmatrix}\right]$ where $\mathfrak{M}$ is a symbolic term, $\kappa \in D^*$ a sequence of directions $n \in \mathbb{N}$ a natural number and Δ a symbolic constraint. The conditional oracle κ is used to resolve branching. During execution the

constraints that a trace needs to satisfy to actually follow κ are recorded in the constraint Δ. The natural number in each configuration references the number of sample variables that have already been substituted. We define the symbolic small-step reduction relation $\xrightarrow{\text{sym}}$ via the rules in Fig. 4.4. Note we do not need to evaluate the symbolic value when we encounter a conditional as we can simply rely on the branching information given by the sequence of direction κ. Constraints are only added when resolving a conditional or a `score`-construct.

Remark: The symbolic execution we present here differs (especially on first glance) from the one used in [Mak et al. 2021]. In their semantics, a symbolic configuration at all times contains a set of traces that can take this path. In contrast, we annotate a symbolic configuration with an explicit set of symbolic inequalities. As we fixed the outcomes of conditionals beforehand our reductions is deterministic.

We can show the following correspondence theorem:

Proposition 4.9 (Symbolic Execution): *For any term M. If $\kappa \in D^*$ and there exist $\mathfrak{V}, n, \Delta$ (If they exists, they are unique) such that*

$$\begin{bmatrix} M, \kappa, 0 \\ \emptyset \end{bmatrix} \xrightarrow{sym}{}^{*} \begin{bmatrix} \mathfrak{V}, \epsilon, n \\ \Delta \end{bmatrix}$$

then $\mathsf{Sat}_n(\Delta) = \mathbb{T}^{(\kappa)}_{M,term}$ *otherwise* $\mathbb{T}^{(\kappa)}_{M,term} = \emptyset$.

Proof The proof is analogous to the proof in [Mak et al. 2021, Theorem 13]. Every symbolic configuration $\begin{bmatrix} \mathfrak{M}, \kappa, n \\ \Delta \end{bmatrix}$ can be seen as the pair $\langle\!\langle \mathfrak{M}, \_, \mathsf{Sat}_n(\Delta) \rangle\!\rangle$ in the setting of Mak et al. when we omit the weight parameter (denoted by $\_$). ■

Example 4.10: Consider $\mathfrak{M} \triangleq \texttt{if sample} + \texttt{sample} - \underline{1}\,\texttt{then}\, x\,\texttt{else}\,(\texttt{if}\ \underline{0}\,\texttt{then}\,\underline{3}\,\texttt{else}\,\underline{4})$. The reduction sequence starting at $\begin{bmatrix} \mathfrak{M}, \boldsymbol{RL}, 0 \\ \emptyset \end{bmatrix}$ is the following:

$$\begin{bmatrix} \mathfrak{M}, \boldsymbol{RL}, 0 \\ \emptyset \end{bmatrix} \xrightarrow{\text{sym}} \begin{bmatrix} \texttt{if}\ \alpha_0 + \texttt{sample} - \underline{1}\,\texttt{then}\, x\,\texttt{else}\,(\texttt{if}\ \underline{0}\,\texttt{then}\,\underline{3}\,\texttt{else}\,\underline{4}), \boldsymbol{RL}, 1 \\ \emptyset \end{bmatrix}$$
$$\xrightarrow{\text{sym}} \begin{bmatrix} \texttt{if}\ \alpha_0 + \alpha_1 - \underline{1}\,\texttt{then}\, x\,\texttt{else}\,(\texttt{if}\ \underline{0}\,\texttt{then}\,\underline{3}\,\texttt{else}\,\underline{4}), \boldsymbol{RL}, 2 \\ \emptyset \end{bmatrix}$$
$$\xrightarrow{\text{sym}}{}^{2} \begin{bmatrix} \texttt{if}\ \alpha_0 \boxed{+} \alpha_1 \boxed{-} \underline{1}\,\texttt{then}\, x\,\texttt{else}\,(\texttt{if}\ \underline{0}\,\texttt{then}\,\underline{3}\,\texttt{else}\,\underline{4}), \boldsymbol{RL}, 2 \\ \emptyset \end{bmatrix}$$
$$\xrightarrow{\text{sym}} \begin{bmatrix} \texttt{if}\ \underline{0}\,\texttt{then}\,\texttt{sample}\,\texttt{else}\,\underline{4}, \boldsymbol{L}, 2 \\ \{\alpha_0 \boxed{+} \alpha_1 \boxed{-} \underline{1} > 0\} \end{bmatrix} \xrightarrow{\text{sym}} \begin{bmatrix} \underline{3}, \epsilon, 2 \\ \{\alpha_0 \boxed{+} \alpha_1 \boxed{-} \underline{1} > 0, \underline{0} \leq 0\} \end{bmatrix}$$

And the solution Sat_2 of the symbolic constraint $\Delta \triangleq \{\alpha_0 \boxed{+} \alpha_1 \boxed{-} \underline{1} > 0, \underline{0} \leq 0\}$ is the set $\{s_0 s_1 \in \mathbb{S}^2 \mid s_0 + s_1 > 1\}$ which is exactly the set $\mathbb{T}^{(\boldsymbol{RL})}_{M,\text{term}}$ as stated in Prop. 4.9.

4.5 Completeness

Interval Separable Functions We have previously restricted the primitive functions to interval preserving functions to allow interval-based reasoning. For the interval-based semantics to be complete we require a further restriction that is, as before, satisfied by most interesting functions. For two measurable sets $A, B \subseteq \mathbb{R}^m$, we say that A *is almost-surely fully contained in* B, written $A \Subset B$, if $A \subseteq B$ and $\lambda_m(B \setminus A) = 0$[4].

> **Definition 4.11:** *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is called* interval separable *if for every interval* $[a, b] \in \mathfrak{I}$*, the set* $f^{-1}([a, b]) \subseteq \mathbb{R}^n$ *can be written almost as the countable union of boxes in* $\mathbb{R}^n$*. I.e., there exists a family of* boxes $\{B_i\}_{i\in\mathbb{N}}$ *with* $B_i \subseteq \mathbb{R}^n$ *such that* $\cup_i B_i \Subset f^{-1}(I)$*.*

The set of interval separable functions is very broad. Most interesting functions such as $+$, $*$, exp, etc. are interval separable. As we have done for interval preserving functions we can define the rational variant of interval separability ($\mathbb{Q}$-*interval separable*) where for every $[a, b] \in \mathfrak{I}^{\mathbb{Q}}$ there exists a family of boxes $\{B_i\}_{i\in\mathbb{N}}$ with rational endpoints, s.t., $\cup_i B_i \Subset f^{-1}([a, b])$. We can give the following sufficient (but not necessary) criterion for f being interval separable.

> **Lemma 4.12 :** *If* $f : \mathbb{R}^n \to \mathbb{R}$ *is continuous and for every* $y \in \mathbb{R}$*,* $f^{-1}(\{y\})$ *is a Lebesgue Null-set then* f *is* $\mathbb{Q}$*-interval separable.*

> **Proof** Let $I = [a, b]$ be an interval as in the definition of interval separable. We have $f^{-1}([a, b]) = f^{-1}((a, b)) \cup f^{-1}(\{a\}) \cup f^{-1}(\{b\})$. By assumption $f^{-1}(\{a\})$ and $f^{-1}(\{b\})$ are null-sets. As f is continuous and (a, b) is an open set we get that $f^{-1}(a, b)$ is an open set. A well know result in $\mathbb{R}^n$ is that every open-set can be covered exactly by a countable number of boxes that have rational endpoints. So there are boxes $B_1, B_2, \cdots$ (with rational endpoints) such that $\cup_i B_i = f^{-1}(a, b)$. ■

The criterion of Lemma 4.12 is only sufficient and not necessary. We can extended it further and cover all continuous functions f where $\mathbb{R}^n$ can be partitioned in countable boxes on all of which f is either constant or the preimage is a null-set as before.

Incompleteness of Interval-Based Reasoning Ideally, one would hope that every continuous function is interval separable. Unfortunately, this is not true, even in the case of $n = 1$. Let $C \subseteq \mathbb{R}$ be any Smith–Volterra–Cantor set, i.e., a set that has positive Lebesgue measure but is nowhere dense, i.e., there is no $[a, b]$ $(a < b)$ with $[a, b] \in C$. Note that C is a *closed set*. Now construct function $f_C : \mathbb{R} \to \mathbb{R}$ by $f_C(x) = d(x, C)$ where $d(x, C)$ denotes the distance of x to the nearest point in C. As C is closed the function is well defined and furthermore it is easy to see that the roots of f_C coincide with C. f_C is obviously continuous. But $f_C^{-1}([-1, 0]) = C$ and C is has positive measure so f_C is obviously not interval separable.

[4]Note that $B \setminus A$ is measurable as it can be written as the intersection of B and the complement of A.

Remark: This observation does not only provide the theoretical insight that not every continuous function is interval separable but also a concrete SPCF example where interval-based reasoning is incomplete. Consider the term $M \triangleq \texttt{if}\ f_c(\texttt{sample})\ \texttt{then}\ \underline{0}\ \texttt{else}\ \underline{1}$ where f_C is the function constructed earlier. This term is clearly AST. In the interval-based semantics, we can, however, never derive a termination probability of more than $1-\lambda(C) < 1$ as we can find no interval trace taking the left branch.

Completeness Proof We are now in a position to prove the statement we have been working towards. We assume the primitive functions to be interval separable. We begin with the following:

Lemma 4.13: *If $\mathfrak{V}$ is a symbolic value of type* **R** *with sample variables among $\{\alpha_0, \cdots, \alpha_{m-1}\}$ and $[a,b] \in \mathfrak{I}$ an interval then there exists a countable family of boxes $\{B_i\}_{i\in\mathcal{I}}$ ($B_i \subseteq \mathbb{R}^m_{[0,1]}$) such that $\bigcup_i B_i \Subset \mathfrak{V}^{-1}([a,b])$.*

Proof The proof goes by induction on the structure of $\mathfrak{V}$. In the case of α_j and $\underline{r}$ it is trivial. In the case of symbolic function application, we perform induction and use the fact that the countable product of a countable set is itself countable. The detailed proof can be found in the appendix: A ■

A further observation we make is that even if two boxes in $\mathbb{R}^n$ are not almost disjoint we can split into finite pairwise almost disjoint boxes covering the same set of points.

Lemma 4.14: *If A, B are two boxes in $\mathbb{R}^m$ then there exist finite boxes $\{C_i\}_{i\in[n]}$ that are pairwise almost disjoint and satisfy $A \cup B = \bigcup_i C_i$.*

We can now turn to our initial proof objective:

Theorem 2 (Completeness of the Interval Semantics): *For every M there exists a countable set of pairwise compatible interval traces $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},term}$ such that*

- $\omega(A) = \mathbb{P}_{term}(M)$
- $\sum_{n=0}^{\infty}\Big(1-\omega\big(\{\wp \in A \mid \#^{\wp}_{\downarrow}(M^{2\mathfrak{I}}) \leq n\}\big)\Big) = \mathbb{E}_{term}(M)$

If M is AST the second equation simplifies to $\mathbb{E}^{M^{2\mathfrak{I}}}(A) = \mathbb{E}_{term}(M)$.

Proof We can naturally identify traces in $\mathbb{S}^m$ with elements in $\mathbb{R}^m_{[0,1]}$. The definition of almost-surely fully contained, $\Subset$, naturally extends to traces. Fix any $\kappa \in D^*$. In the first step, we show that there exists a countable family of boxes $\{B_l\}_{l\in\mathcal{I}}$ ($B_l \subseteq \mathbb{R}^m_{[0,1]}$) such that $\bigcup_{l\in\mathcal{I}} B_l \Subset \mathbb{T}^{(\kappa)}_{M,\text{term}}$.
First: There either exists a value $\mathfrak{V}$ a natural number m and constraint Δ (all of them unique) such that $\begin{bmatrix}\mathfrak{M},\kappa,0\\ \emptyset\end{bmatrix} \xrightarrow{\text{sym}}{}^{*} \begin{bmatrix}\mathfrak{V},\epsilon,m\\ \Delta\end{bmatrix}$ or there exists none. In either case, we apply

Prop. 4.9. In the latter case, we are done as $\mathbb{T}^{(\kappa)}_{M,\mathrm{term}} = \emptyset$. In the former case, we get $\mathsf{Sat}_m(\Delta) = \mathbb{T}^{(\kappa)}_{M,\mathrm{term}}$. Note that $\mathbb{T}^{(\kappa)}_{M,\mathrm{term}} \subseteq \mathbb{S}^m$. Let $\Delta = \{(\mathfrak{V}_i \bowtie_i r_i)\}_{i\in[n]}$. Now by definition of Sat_m, $\mathsf{Sat}_m(\Delta) = \{\sigma \in \mathbb{R}^m_{[0,1]} \mid \forall i \in [n] : \mathfrak{V}_i[\sigma] \bowtie_i r_i\}$. We can write this as $\bigcap_{i\in[n]} \mathfrak{V}_i^{-1}(I_i)$ where I_i is one of (r_i, ∞), $[r_i, \infty)$, $(-\infty, r_i)$ or $(-\infty, r_i]$ depending on $\bowtie_i$. As all of these sets can be given as a countable union of closed bounded intervals, we can apply Lem. 4.13 and get a family $(B^i_k)_{k\in\mathcal{I}_i}$ such that $\bigcup_{k\in\mathcal{I}_i} B^i_k \Subset \mathfrak{V}_i^{-1}(I_i)$. Now the finite intersection of countable unions of boxes is itself a countable union of boxes (refer to the proof of Lem. 4.13). There thus exists a family $(B_l)_{l\in\mathcal{I}}$ with $\bigcup_{l\in\mathcal{I}} B_l \Subset \cap_{i\in[n]}\mathfrak{V}_i^{-1}(I_i) = \mathbb{T}^{(\kappa)}_{M,\mathrm{term}}$.

Second: So $\bigcup_{l\in\mathcal{I}} B_l \Subset \mathbb{T}^{(\kappa)}_{M,\mathrm{term}}$. By Lem. 4.14 we can assume that this family is pairwise almost disjoint. Now each box $B_l \subseteq \mathbb{R}^m_{[0,1]}$ can naturally be seen as an interval trace within $\mathbb{S}^m_{\mathfrak{I}}$. Let $A^{(\kappa)}$ be this set of interval traces. As the boxes are pairwise almost disjoint the traces are pairwise compatible. For each $\wp \in A^{(\kappa)}$ we have $(\!|\wp|\!) \in \mathbb{T}^{(\kappa)}_{M,\mathrm{term}}$ so by Lem. 4.7 we get that $\wp \in \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\mathrm{term}}$. So $A^{(\kappa)} \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\mathrm{term}}$. As the set of conditional oracles D^* is countable we can take the union of all interval traces $A^{(\kappa)}$ for all $\kappa \in D^*$. There thus exists a countable set of interval traces $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\mathrm{term}}$ such that $\bigcup_{\wp\in A}(\!|\wp|\!) \Subset \mathbb{T}_{M,\mathrm{term}}$. This already implies that $\omega(A) = \mu_{\mathbb{S}}(\mathbb{T}_{M,\mathrm{term}})$. For the expected time to termination recall that for all $\boldsymbol{s} \in (\!|\wp|\!)$, $\#^{\wp}_{\downarrow}(M^{2\mathfrak{I}}) = \#^{\boldsymbol{s}}_{\downarrow}(M)$. ■

Our completeness result (combined with soundness) allows us to reason entirely in the interval-based semantics. Interval-based reasoning is particularly well suited to derive lower bounds on the probability of termination. For an interval term $\mathcal{M}$ we define $\mathbb{P}_{\mathrm{term}}(\mathcal{M})$ as the least upper bound of $\omega(A)$ for all $A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\mathrm{term}}$ that are finite and pairwise compatible and similarly for $\mathbb{E}_{\mathrm{term}}(\mathcal{M})$. Then soundness and completeness give us $\mathbb{P}_{\mathrm{term}}(M) = \mathbb{P}_{\mathrm{term}}(M^{2\mathfrak{I}})$ and $\mathbb{E}_{\mathrm{term}}(M) = \mathbb{E}_{\mathrm{term}}(M^{2\mathfrak{I}})$ As we only need to consider countable sets of interval traces this has some interesting recursion-theoretical consequence which we discuss in the next section.

4.6 AST and PAST in the Arithmetic Hierarchy

If we only consider functions that are $\mathbb{Q}$-interval preserving and $\mathbb{Q}$-interval separable we can restrict the previous reasoning to intervals and boxes with rational endpoint. So for any term whose numerals are all rational, we get a countable set A of pairwise compatible interval traces traces with *rational endpoints* such that $A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\mathrm{term}}$ and $\omega(A) = \mathbb{P}_{\mathrm{term}}(M)$. We can, therefore, deduce the following[5]:

[5] In the following, computability of a primitive function f should not only be understood as pointwise computability of the value of f but also computability of $\hat{f}$.

Theorem 3 (Complexity of AST and PAST): *Let the set of primitive functions be $\mathbb{Q}$-interval preserving, $\mathbb{Q}$-interval separable and computable. For any term containing only rational numerals, deciding AST is in Π^0_2 and deciding PAST is in Σ^0_2. If the successors function $+1$ is constructable using the primitive functions, deciding AST is Π^0_2-complete and deciding PAST is Π^0_2-complete.*

Proof Let A ranges over *finite* sets of interval traces with rational endpoints. We can express M being AST by the following $\forall\exists$-formula (c.f [Kaminski and Katoen 2015]):

$$\forall \epsilon > 0 \in \mathbb{Q} : \exists A : A \text{ p.c.} \wedge A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\text{term}} \wedge \omega(A) \geq 1 - \epsilon$$

We can express that M is PAST by the $\exists\forall$-formula:

$$\exists c \in \mathbb{Q} : \forall m \in \mathbb{N} : \forall A : \left(A \text{ p.c.} \wedge A \subseteq \mathbb{T}^{\mathfrak{I}}_{M^{2\mathfrak{I}},\text{term}}\right) \Rightarrow \sum_{i=0}^{m} \left(1 - \omega\big(\{\wp \in A \mid \#^{\wp}_{\downarrow}(M^{2\mathfrak{I}}) \leq i\}\big)\right) \leq c$$

The hardness results follows from [Kaminski and Katoen 2015]. ■

In particular, note that this includes all programs using addition and multiplication. This gives a, to the best of our knowledge first, proof that both of these decision problems belong to the same level of the arithmetic hierarchy as for discrete languages.

Remark: In this chapter we explicitly worked with a CbV reduction. It is, however, not hard to see, that the exact reasoning can also be applied in a Call by Name setting.
We have argued (and Thm. 3 can be seen as evidence for that) that the interval-based semantics is an intriguing object of study and useful for many applications. It is somewhat surprising, that despite the continuous nature of SPCF, in the end we can reason about a countable number of traces. While we have seen that interval-based semantics is well suited to reason about programs, our counterexample constructed from the Cantor-set shows that interval-based methods are incomplete in the presence of arbitrary primitive functions. This is not surprising, as the restriction to a countable number must come at a price. We do, however, believe, that the set of primitive functions that guarantee completeness of the interval-based semantics include most useful functions (c.f. Lem. 4.2 and Lem. 4.12). An interesting direction for future work is to find different representation of uncountable sets in finite space (we have used intervals) such that function application is stable under the abstract representation (c.f. Lem. 4.2) and thereby support a larger (or just different) class of primitive functions.

Chapter 5

Intersection Type System

This chapter is devoted to study SPCF by means of intersection type derivations. Following Breuvart and Lago [2018], who established a complete intersection type system for the probabilistic (discrete) untyped λ-calculus, we aim to derive an extended system for SPCF. This has many advantages: first, a type system gives us a compositional way of reasoning about termination that is not phrased in an operational style but derived by structural induction on the program. A second advantage is that we present a type system where pairwise compatibility is "baked in": in the interval-based semantics, checking traces for pairwise compatibility is crucial to ensuring that the same standard trace is not account for twice. At the same time it is cumbersome to check any newly derived trace for compatibility. In our intersection type system, the probability of termination is obtained as the least upper bound of all derivations without having to check for side conditions such as compatibility. Our system builds upon the new interval-based semantics, in order to characterise AST by a *countable* number of derivations. Our system therefore operates on interval terms. We can always consider the canonical embedding $M^{2\mathfrak{I}}$ to reason about standard terms.

The Structure of this Chapter We begin this chapter by giving a very brief overview on intersection types in a deterministic (untyped) language. Following this, we outline some of the concepts needed to design an intersection system for a typed language with explicit recursion (mostly following [Ehrhard et al. 2014]). Afterwards we summarize some of the approaches and results of intersection types in a probabilistic setting [Breuvart and Lago 2018]. We then present the intersection type system and show it correct: the least upper bound over all derivations equals the probability of termination.

5.1 Background on Intersection Types

Intersection types are best understood in the untyped λ-calculus. It is well known (due to Tait) that if we endow the λ-calculus with a simple type system (see e.g. [Barendregt 1985]), the typeable terms are strongly normalising. Simple types are, however, far from complete for this class as not every normalising term is simply typeable. On the other hand, intersection types are a more expressive type system that cannot only ensure normalization, but *characterise* it: A term is typeable if and only if it is normalising. Intersection types thus offer a compositional system that can characterise properties that are defined via the operational behavior. The idea of intersection types is to summarize multiple types into an intersection, a set of types. Such a set of types can be seen as "having all types in that set". For an excellent overview on (non-idempotent) intersection type system that can characterize termination under different reduction relations see [Bucciarelli et al. 2017].

5.2 Intersection Types, Recursion and Conditionals

Intersection types are typically associated with languages similar to the untyped λ-calculus. For languages with an explicit recursion operator, such as SPCF, they do not seem as useful. For instance, SPCF is equipped with a simple type system but not every simply typed term is also terminating. It turns out, that intersection types are nevertheless applicable for such languages. But this comes with a cost: a desirable property of type systems is that they are purely compositional, meaning that in each rule the terms appearing in the premise are strict subterms of the term being typed in the conclusion. This does even hold for powerful intersection type systems in the λ-calculus[1].

In a language with explicit recursion, this is no longer possible. To type a fixpoint, we already require a derivation for the fixpoint itself. For an example of an intersection type system for a typed language (a variant of PCF) we refer to [Ehrhard et al. 2014, §5.1]. The reader should explicitly pay attention to the rule for fixpoints in the system of Ehrhard et al. that is not purely compositional. Apart from explicit recursion, conditionals pose a further obstacle. Any type system that characterises termination must handle branching efficiently. In particular, in the type derivation the decision which branch to type must be derivable from the type of the condition. This does, inevitable, require us to encode semantic information within types itself. If we for instance consider the types used in the PCF system by Ehrhard

[1] Even though the rules for intersection type systems are compositional, the size of a type derivation (defined e.g. in terms of symbols) is not bounded by the size of the term. There actually is an interesting correspondence between the size of a type derivation and the number of reduction steps ([de Carvalho 2018]). In [Bucciarelli et al. 2017], various properties about intersection types systems are based purely on an inductive argument on the size of a type derivation.

et al. [2014], they encoded natural numbers within the types, since natural numbers are the first-order datatype on which their language operates.

5.3 Intersection Types for a Probabilistic Language

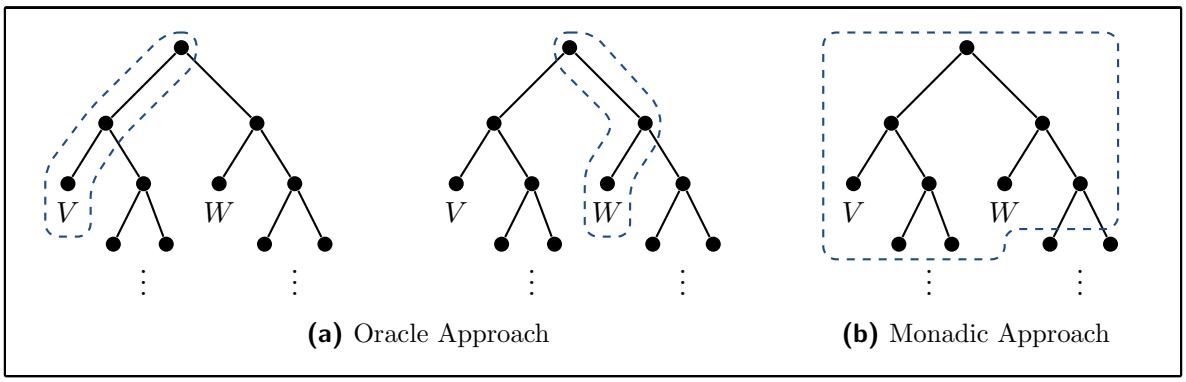
In the untyped λ-calculus, termination (normalization) is one of the most studied problems. As seen above, intersection type system are not only able to guarantee termination but also enjoy a completeness property, they *characterise* termination entirely. The natural question is whether this intriguing completeness does carry over to a probabilistic language where termination is usually understood as *Almost-sure Termination*.

Due to the intrinsic hardness of checking for AST (namely the Π_2^0-hardness shown by [Kaminski and Katoen 2015]), we cannot hope for a semi-decidable type system where a term is typeable if and only if it is AST. Every reasonable type system that is both sound and complete must inherit the hardness of checking AST. Recently, a first formal study of intersection types in probabilistic setting has been conducted by [Breuvart and Lago 2018]. They showed that intersection types can achieve similar completeness guarantees with respect to AST as they can for termination in a non-probabilistic setting. Due to the recursion-theoretical restrictions it is, however, not possible to obtain the probability of termination from a single type derivation but by countably many such. In their work, they considered two different approaches to the problem: In the first, they called *oracle types*, the probability of termination is achieved by taking the countable sum over all possible type derivations (weighted in an appropriate way). In the second approach, called *monadic*, the probability of termination is given as the least upper bound of all derivations. In their work, they considered the untyped λ-calculus with a binary probabilistic choice $\oplus$. Evaluation can be seen as a binary tree where each node corresponds to probabilistic decision. In the oracle approach, each type derivation corresponds to a probabilistic path leading to a value. In the monadic approach, each derivation can summarize multiple such paths. The situation is depicted in Fig. 5.1.

5.4 Intersection Type System For SPCF

The system we present here conceptually lies in between both approaches. On the one hand, our system relies on the idea of identifying terminating traces via type derivations and in that regard is similar to the oracle approach by Breuvart and Lago. On the other hand, as we summarize multiple traces, the probability of termination equals the least upper bound over all derivation, similar to their monadic system.

The profound difference between CbN and CbV evaluation has a direct impact on the



(a) Oracle Approach **(b)** Monadic Approach

Figure 5.1: Visual representation of the Oracle vs Monadic Intersection Type approach by Breuvart and Lago. In the oracle system, each type derivation identifies a terminating path. In the monadic system, a single type derivation can combine the weight of multiple derivations.. Picture is inspired by Breuvart and Lago [2018]

$$\frac{}{(\lambda x.\mathcal{M})\mathcal{N} \to_{\mathsf{det}} \mathcal{M}[\mathcal{N}/x]} \qquad \frac{}{(\mu^{\varphi}_{x}.\mathcal{M})\mathcal{N} \to_{\mathsf{det}} \mathcal{M}[\mathcal{N}/x, (\mu^{\varphi}_{x}.\mathcal{M})/\varphi]}$$

$$\frac{}{f(\underline{[a_1,b_1]},\cdots,\underline{[a_{|f|},b_{|f|}]}) \to_{\mathsf{det}} \underline{\hat{f}(a_1,b_1,\cdots,a_{|f|},b_{|f|})}} \qquad \frac{a \geq 0}{\mathtt{score}(\underline{[a,b]}) \to_{\mathsf{det}} \underline{[a,b]}}$$

$$\frac{b \leq 0}{\mathtt{if}(\underline{[a,b]},\mathcal{N},\mathcal{P}) \to_{\mathsf{det}} \mathcal{N}} \qquad \frac{a > 0}{\mathtt{if}(\underline{[a,b]},\mathcal{N},\mathcal{P}) \to_{\mathsf{det}} \mathcal{P}} \qquad \frac{\mathcal{R} \to_{\mathsf{det}} \mathcal{M}}{\mathcal{E}[\mathcal{R}] \to_{\mathsf{det}} \mathcal{E}[\mathcal{M}]}$$

$$\frac{}{\mathtt{sample} \to_{[a,b]} \underline{[a,b]}} \qquad \frac{\mathcal{R} \to_{[a,b]} \mathcal{M}}{\mathcal{E}[\mathcal{R}] \to_{[a,b]} \mathcal{E}[\mathcal{M}]}$$

Figure 5.2: Decomposed reduction into deterministic steps $\to_{\mathsf{det}}$ and probabilistic steps $\to_{[a,b]}$

typing systems. As already noted in [Breuvart and Lago 2018; Ehrhard et al. 2014], CbN systems are in general easier to handle in a compositional style. We, therefore, present a type system for CbN.

Decomposed Reduction The interval-based semantics is defined as a reduction relation of terms parametrized by an interval trace. To state properties of the type system it is actually easiest to decompose this reduction relation. A relation $\to_{\mathsf{det}}$, handling deterministic steps, and a relation $\to_{[a,b]}$ for $[a,b] \in \mathfrak{I}_{0,1}$ performing probabilistic steps. Those relations are defined in Fig. 5.2. Note that the evaluation contexts are Call by Name contexts.

While our type system is designed such that the least upper bound over all derivation equals

the probability of termination and thus looks very similar to the monadic system in Breuvart and Lago [2018], we have to approach on an entirely different way. The system by Breuvart and Lago relies on the countable nature of the execution tree and can state subject reduction by taking the weighted (finite) sum over the reduction relation. Due to the uncountable nature of SPCF, we can not follow this approach. Instead, in our system we allow for enumeration of terminating interval traces and make use of the soundness and completeness of the interval based semantics shown in Ch. 4.

Set Types We define *set types* by the following grammar:

$$\begin{aligned}\alpha &\triangleq [a,b] \mid a \to \mathcal{A}\\ \mathcal{A} &\triangleq \left\{\!\!\left\{(\alpha_1,\wp_1,\tau_1),\cdots,(\alpha_m,\wp_m,\tau_m)\right\}\!\!\right\}\\ a &\triangleq \{\mathcal{A}_1,\cdots,\mathcal{A}_n\}\end{aligned}$$

where each $\wp_i$ is an interval trace and τ_i a natural number. We refer to elements a as *intersections* and $\mathcal{A}$ as *set types*. We denote set types with a different symbol $\{\!\!\{\cdot\}\!\!\}$ to make differentiation easier, but treat $\{\!\!\{\cdot\}\!\!\}$ as a set of elements. The simplest type in our system is given by $\star \triangleq \emptyset \to \emptyset$. Note that we allow intervals as first-order types and, therefore, give types itself a semantic flavour. As discussed before this is needed to type conditionals. For a set type $\mathcal{A} = \{\!\!\{(\alpha_1,\wp_1,\tau_1),\cdots,(\alpha_n,\wp_n,\tau_n)\}\!\!\}$ we write $\mathcal{A}^{(\uparrow\wp,\tau)}$ for the set type $\{\!\!\{(\alpha_1,\wp\wp_1,\tau_1+\tau),\cdots,(\alpha_n,\wp\wp_n,\tau_n+\tau)\}\!\!\}$, i.e., the set obtained by appending $\wp$ to every trace and add τ to every count.

Type System Typing judgments are of the form $\Gamma \vdash \mathcal{M} : \mathcal{A}$ and given by the rules in Fig. 5.3. As an intuition the following is helpful: If $\vdash \mathcal{M} : \{\!\!\{(\alpha_1,\wp_1,\tau_1),\cdots,(\alpha_m,\wp_m,\tau_m)\}\!\!\}$ then $\wp_i$ are all terminating traces for M on which exactly τ_i steps are made until a value is reached. We advise the reader to compare this system with the monadic system given in [Breuvart and Lago 2018, §6.1]. We briefly discuss some of the interesting rules: the rule for abstractions is similar to the one in [Breuvart and Lago 2018] and the one for fixpoints builds on the observations made by [Ehrhard et al. 2014]. In particular, note that in order to type a fixpoint where the recursive abstraction has a non-empty intersection (called b in the typing rule) we need to type the fixpoint itself. When combined with $(\{\!\!\{\}\!\!\})$ we can type every λ- or μ-abstraction with set type $\{\!\!\{\star,\epsilon,0\}\!\!\}$. The **(app)** rule is closely related to [Breuvart and Lago 2018]. We note that the type of the term in the function (left) position fully determines the type of the application. This matches the intuition of CbN where the arguments are passed unevaluated. The **(sample)** -rule is particularly interesting. We can type a `sample`-statement with an arbitrary number of (one-element) interval traces as long as those traces are pairwise

$$\frac{\mathcal{A} \in a}{\Gamma; x : a \vdash x : \mathcal{A}}\ \textbf{(var)} \qquad \frac{}{\Gamma \vdash \mathcal{M} : \{\!\!\{\}\!\!\}}\ (\{\!\!\{\}\!\!\}) \qquad \frac{\Gamma; x : a \vdash \mathcal{M} : \mathcal{A}}{\Gamma \vdash \lambda x.\mathcal{M} : \left\{\!\!\left\{(a \to \mathcal{A}, \epsilon, 0)\right\}\!\!\right\}}\ \textbf{(abs)}$$

$$\frac{\Gamma; x : a; \varphi : b \vdash \mathcal{M} : \mathcal{A} \qquad \left\{\Gamma \vdash \mu_x^\varphi.\mathcal{M} : \mathcal{B} \mid \forall \mathcal{B} \in b\right\}}{\Gamma \vdash \mu_x^\varphi.\mathcal{M} : \left\{\!\!\left\{(a \to \mathcal{A}, \epsilon, 0)\right\}\!\!\right\}}\ \textbf{(fix)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \{\Gamma \vdash \mathcal{N} : \mathcal{C} \mid \forall (b \to \mathcal{B}, \wp, \tau) \in \mathcal{A}, \mathcal{C} \in b\}}{\Gamma \vdash \mathcal{M}\mathcal{N} : \bigcup_{(b \to \mathcal{B}, \wp, \tau) \in \mathcal{A}} \mathcal{B}^{(\uparrow \wp, \tau+1)}}\ \textbf{(app)}$$

$$\frac{}{\Gamma \vdash \underline{[a,b]} : \left\{\!\!\left\{([a,b], \epsilon, 0)\right\}\!\!\right\}}\ \textbf{(num)} \qquad \frac{\{[a_i, b_i]\}_{i \in [n]} \text{ are almost disjoint}}{\Gamma \vdash \texttt{sample} : \left\{\!\!\left\{([a_i, b_i], [a_i, b_i], 1) \mid i \in [n]\right\}\!\!\right\}}\ \textbf{(sample)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A}}{\Gamma \vdash \texttt{score}(\mathcal{M}) : \left\{\!\!\left\{([a,b], \wp, \tau + 1) \mid ([a,b], \wp, \tau) \in \mathcal{A}, a \geq 0\right\}\!\!\right\}}\ \textbf{(score)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \begin{array}{l}\{\Gamma \vdash \mathcal{N} : \mathcal{B}_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}, b \leq 0\} \\ \{\Gamma \vdash \mathcal{P} : \mathcal{C}_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}, a > 0\}\end{array}}{\Gamma \vdash \texttt{if}(\mathcal{M}, \mathcal{N}, \mathcal{P}) : \bigcup_{([a,b],\wp,\tau) \in \mathcal{A} \mid b \leq 0} \mathcal{B}^{(\uparrow \wp, \tau+1)}_{([a,b],\wp,\tau)} \cup \bigcup_{([a,b],\wp,\tau) \in \mathcal{A} \mid a > 0} \mathcal{C}^{(\uparrow \wp, \tau+1)}_{([a,b],\wp,\tau)}}\ \textbf{(if)}$$

$$\frac{\Gamma \vdash \mathcal{M} : \mathcal{A} \qquad \{\Gamma \vdash \mathcal{N} : \mathcal{B}_{([a,b],\wp,\tau)} \mid ([a,b], \wp, \tau) \in \mathcal{A}\}}{\Gamma \vdash f(\mathcal{M}, \mathcal{N}) : \bigcup_{([a,b],\wp,\tau)} \bigcup_{([c,d],\wp',\tau') \in \mathcal{B}_{([a,b],\wp,\tau)}} \left\{\!\!\left\{(\hat{f}(a,b,c,d), \wp\wp', \tau + \tau' + 1)\right\}\!\!\right\}}\ (f_2)$$

Figure 5.3: Intersection Type System for SPCF.

compatible. The **(if)** -rule for conditionals looks intimating at first. However, the subscript $([a,b], \wp, \tau)$ for each set type is simply used as an index, i.e., if $\Gamma \vdash \mathcal{M} : \mathcal{A}$ we can combine a different type derivation for every element in $\mathcal{A}$. Similar for the rules for primitive functions. We restricted to primitive functions with two arguments; however, the rules can easily be extended to handle higher arity. We omitted the general rule as it is gets chaotic. As a first easy result we can show:

Lemma 5.1 : *If* $\vdash \mathcal{M} : \mathcal{A}$ *and* $\mathcal{B} \subseteq \mathcal{A}$ *then* $\vdash \mathcal{M} : \mathcal{B}$

Proof Easy induction on $\vdash \mathcal{M} : \mathcal{A}$. ■

We now proof our type system correct and show that it actually matches the intuition given above. We show that if $\vdash \mathcal{M} : \{\!|(\alpha_1, \wp_1, \tau_1), \cdots, (\alpha_m, \wp_m, \tau_m)|\!\}$ then ever $\wp_i \in \mathbb{T}^{\mathfrak{J}}_{\mathcal{M},\text{term}}$ and furthermore $\tau_i = \#^{\wp_i}_{\downarrow}(\mathcal{M})$, i.e., τ_i gives the number of steps. This will later allow us to not only characterise AST but also PAST.

5.4.1 Subject Reduction and Soundness

We begin by stating a standard result for type systems: *subject reduction*. Informally it reads that type derivations are preserved under reduction steps. In our setting, the τ component gives the number of steps to termination. Matching this intuition, the τ decrease by 1 in each step. Furthermore as each $\wp$ is a terminating trace, each probabilistic reduction consumes the first element. Note that stating subject reduction is easier by using the decomposed semantics.

Lemma 5.2 (Subject Reduction): *If* $\vdash \mathcal{M} : \{\!|(\alpha, \wp, \tau)|\!\}$ *and* $\mathcal{M}$ *is not a value, then either*

- $\mathcal{M}$ *has a deterministic redex and* $\mathcal{M} \to_{det} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \{\!|(\alpha, \wp, \tau - 1)|\!\}$*, or*
- $\mathcal{M}$ *has a probabilistic redex then* $\wp = [a, b]\wp'$ *and we have* $\mathcal{M} \to_{[a,b]} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \{\!|(\alpha, \wp', \tau - 1)|\!\}$

Proof The proof then goes by induction on $\vdash \mathcal{M} : \{\!|(\alpha, \wp, \tau)|\!\}$, a substitution lemma and analysis of the typing rules. A proof can be found in the appendix: B. ■

Note that in general, not being a value does not mean that a reduction step can take place as our reduction does not enjoy progress. According to Lem. 5.2 typeable with a *non-empty* set type, does however guarantee progress, i.e., either being a value or making a reduction step. Using subject reduction we can now show that each type derivation does indeed identify terminating interval traces and gives the number of reduction steps for each. Note that due to the step count, which we use to characterise PAST, we can show this without taking the typical route and construct reducibility candidates.

Lemma 5.3: *If* $\vdash \mathcal{M} : \{\!|(\alpha_i, \wp_i, \tau_i) \mid i \in [n]|\!\}$ *then* $\wp_i \in \mathbb{T}^{\mathfrak{J}}_{\mathcal{M},term}$ *and* $\#^{\wp_i}_{\downarrow}(\mathcal{M}) = \tau_i$ *for all* $i \in [n]$

Proof We show the easier observation that if $\vdash \mathcal{M} : \{\!|\alpha, \wp, \tau|\!\}$, then $\wp \in \mathbb{T}^{\mathfrak{J}}_{\wp,\text{term}}$ and $\#^{\wp}_{\downarrow}(\mathcal{M}) = \tau$. The result then follows by Lem. 5.1 as we get $\vdash \mathcal{M} : \{\!|\alpha_i, \wp_i, \tau_i|\!\}$ for every $i \in [n]$.

As an easy corollary from Subject reduction (Lem. 5.2) combined with the obvious properties of the decomposed semantics, we get that if $\vdash \mathcal{M} : \{\!|(\alpha, \wp, \tau)|\!\}$ and $\langle \mathcal{M}, \wp \rangle \rightsquigarrow \langle \mathcal{M}', \wp' \rangle$ we have $\vdash \mathcal{M}' : \{\!|(\alpha, \wp', \tau - 1)|\!\}$. Call this observation **(1)**.

We first show $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$. Let $\langle M, \wp\rangle \triangleq \langle \mathcal{M}_0, \wp_0\rangle \rightsquigarrow \langle \mathcal{M}_1, \wp_1\rangle \rightsquigarrow \langle \mathcal{M}_2, \wp_2\rangle \rightsquigarrow \cdots$ be the possibly infinite reduction sequence. From **(1)** we get $\vdash \mathcal{M}_i : \{\!\{(\alpha, \wp_i, \tau - i)\}\!\}$. The sequence can thus make *at most* τ-steps and is hence finite. Let $\langle \mathcal{M}, \wp\rangle = \langle \mathcal{M}_0, \wp_0\rangle \rightsquigarrow \langle \mathcal{M}_1, \wp_1\rangle \rightsquigarrow \langle \mathcal{M}_2, \wp_2\rangle \rightsquigarrow \cdots \rightsquigarrow \langle \mathcal{M}_n, \wp_n\rangle$ be this finite, maximal sequence. We assume for contraction that $\mathcal{M}_n$ is not a value. As $\vdash \mathcal{M}_n : \{\!\{(\alpha, \wp_n, \tau - n)\}\!\}$ we can use subject reduction (Lem. 5.2) and get that $\langle \mathcal{M}_n, \wp_n\rangle$ can make a further step which contradicts the maximality. Now as M_n is a value, we can inspect the typing rules and get that $\wp_n = \epsilon$ as values can only be typed with an empty interval trace. This already shows that $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$.
Now by definition of the number of steps $\#^{\wp}_{\downarrow}(\mathcal{M}) = n$. As $\mathcal{M}_n$ is a value and $\vdash \mathcal{M}_n : \{\!\{(\alpha, \wp_n, \tau - n)\}\!\}$ we get by inspection that $\tau - n = 0$, so $\tau = n = \#^{\wp}_{\downarrow}(\mathcal{M})$. ■

As each type derivations identifies terminating traces, and, by construction, the interval traces used in the **(sample)** are pairwise compatible, we can infer the following:

Lemma 5.4 (Pairwise Compatibility): *If* $\vdash \mathcal{M} : \{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\}$ *then* $\{\wp_i\}_i$ *are pairwise compatible.*

Proof By induction on $\mathcal{M}$ using the fact that two *terminating*, compatible interval traces can never become non-compatible by appending arbitrary traces. A full proof can be found in the appendix: B. ■

We argued that if $\vdash \mathcal{M} : \{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\} \triangleq \mathcal{A}$ then each $\wp_i$ is terminating and the family is pairwise compatible. Every set type $\mathcal{A}$ thus gives an immediate lower bound on the probability of termination. At the same time, as τ_i equals the number of reduction steps, $\mathcal{A}$ can be given a natural notion of expectation. To formalize this we define:

- $\omega(\{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\}) \triangleq \sum_{i\in[n]} \omega(\wp_i)$
- $\mathbb{E}(\{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\}) \triangleq \sum_{i\in[n]} \omega(\wp_i) \cdot \tau_i$

Note that both quantities are defined purely in terms of the type itself and do not refer to a concrete term. $\omega(\mathcal{A})$ gives the joint probability of an set type while $\mathbb{E}(\mathcal{A})$ uses the τ-component to define an expectation. We can now state the soundness of the type system. We denote the least upper bound over a set of real numbers $\bigvee$. Recall that if a sequence $(a_n)_{n\in\mathbb{N}}$ satisfies $a_n \leq b$ then also $\bigvee_n a_n \leq b$ as b is a trivial upper bound[2].

Proposition 5.5 (Soundness): *For every interval term* $\mathcal{M}$ *and* $M \triangleleft \mathcal{M}$

- $\bigvee_{\vdash \mathcal{M}:\mathcal{A}} \omega(\mathcal{A}) \leq \mathbb{P}_{term}(M)$
- $\bigvee_{\vdash \mathcal{M}:\mathcal{A}} \mathbb{E}(\mathcal{A}) \leq \mathbb{E}_{term}(M)$

[2]Note, that this does not hold if we replace $\leq$ with the strict version $<$.

Proof Assume $\vdash \mathcal{M} : \mathcal{A}$ and $\mathcal{A} = \{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\}$. By Lem. 5.3 each $\wp_i \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$. Furthermore, by Lem. 5.4 the interval traces are pairwise compatible. By the Soundness of the interval-based semantics (Thm. 1) we, therefore, conclude that

$$\omega(\mathcal{A}) \leq \mathbb{P}_{\text{term}}(M)$$

For the second claim we can again use Thm. 1 and the fact that $\tau_i = \#^{\wp_i}_{\downarrow}(\mathcal{M})$ (shown in Lem. 5.3) and get

$$\mathbb{E}(\mathcal{A}) \leq \mathbb{E}_{\text{term}}(M)$$

As this holds for all $\vdash \mathcal{M} : \mathcal{A}$ it also holds for the least upper bound. ■

5.4.2 Subject Expansion and Completeness

Each derivation $\vdash \mathcal{M} : \mathcal{A}$ gives a lower bound on the probability of termination. We now show that least upper bound over all derivations *equals* the probability of termination, i.e., the completeness of our type system. We begin by showing a property that most intersection type systems enjoy: *subject expansion*. Informally it reads that typing judgments are preserved in the reverse reduction direction. While a term can reduce to multiple different terms due to the probabilistic nature, we show that the typing derivations for each obtained successor can be combined into a derivation for the original term as long as the reductions took place on pairwise almost disjoint intervals.

Lemma 5.6 (Subject Expansion):

- *If* $\vdash \mathcal{M} : \mathcal{A}$ *and* $\mathcal{N} \to_{\mathsf{det}} \mathcal{M}$ *then* $\vdash \mathcal{N} : \mathcal{A}^{(\uparrow\epsilon,1)}$
- *If* $\vdash \mathcal{M}_i : \mathcal{A}_i$ *and* $\mathcal{N} \to_{[a_i,b_i]} \mathcal{M}_i$ *where* $\{[a_i, b_i]\}_i$ *are almost disjoint then*

$$\vdash \mathcal{N} : \bigcup_i \mathcal{A}_i^{(\uparrow[a_i,b_i],1)}$$

Proof We first show reverse substitution. The proof then follows by induction. In the probabilistic case, we make use of the fact that in the rules **(if)** and (f_2) we can combine individual derivations. The proof can be found in the appendix: B. ■

Naïve Attempt on Completeness: We have seen in the soundness proof that if $\vdash \mathcal{M} : \{\!\{(\alpha_i, \wp_i, \tau_i) \mid i \in [n]\}\!\}$ each $\wp_i$ is a terminating trace. For completeness we would like to reverse that process and show that any pairwise compatible set of traces can be achieved via a type derivations: That is, if $\{\wp_i \mid i \in [n]\} \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$ are pairwise compatible then

$$\vdash \mathcal{M} : \{\!\{(\alpha_i, \wp_i, \#^{\wp_i}_{\downarrow}(\mathcal{M})) \mid i \in [n]\}\!\}$$

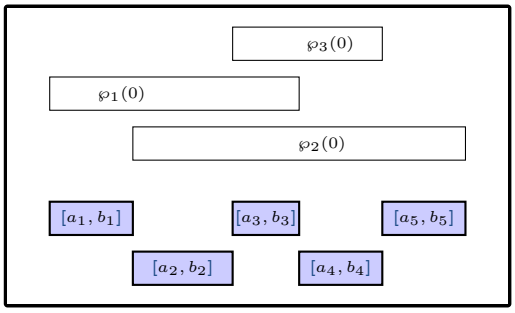


Figure 5.4: Visualisation of how to split intervals into unions of almost disjoint intervals as required in Lem. 5.8. The three top interval should be split into finite unions of intervals drawn from a family that is pairwise almost disjoint. We can split at every overlapping end position and obtain the 5 intervals given in blue. Clearly they are pairwise almost disjoint and each of the intervals above can be obtained as a finite union of blue intervals.

for some types $\{\alpha_i\}_{i\in[n]}$. This would immediately give us a completeness theorem as the interval-based semantics is itself complete. However, the above does **not** hold.

> **Example 5.7:** As an example consider the following simple term: $\mathcal{M} \triangleq \texttt{if}\big(\texttt{sample} - \frac{1}{2}, \texttt{sample}, \underline{0}\big)^{2\mathfrak{J}}$ Then the two interval traces $\wp_1 \triangleq [0,\frac{1}{2}][0,\frac{1}{2}]$, $\wp_2 \triangleq [0,\frac{1}{3}][\frac{1}{2},1]$ are clearly compatible but cannot be typed with the above system. The interested reader is advised to try find a typing derivation.

Strong Pairwise Compatibility To show completeness, we need to introduce the new concept of *strong compatibility*. We call $\wp_1, \wp_2$ *strongly compatible* if $\wp_1 \leftrightarrow \wp_2$ is derivable by the following rules

$$\frac{}{\epsilon \leftrightarrow [a,b]\wp} \qquad \frac{}{[a,b]\wp \leftrightarrow \epsilon} \qquad \frac{\wp_1 \leftrightarrow \wp_2}{[a,b]\wp_1 \leftrightarrow [a,b]\wp_2} \qquad \frac{[a,b],[c,d] \text{ are almost disjoint}}{[a,b]\wp_1 \leftrightarrow [c,d]\wp_2}$$

Strongly compatible traces are either pairwise almost disjoint in the first position or agree on the first position and the remainder is also strongly compatible. If two traces are strongly compatible they can thus share a common, identical prefix but must be pairwise almost disjoint at the first position where they differ. Clearly every strongly compatible pair of interval traces is also compatible but not the other way around. As an example the two traces in Ex. 5.7 are compatible but not strongly compatible. We can show the following:

> **Lemma 5.8 :** *If* $\{\wp_i \mid i \in [n]\} \subseteq \mathbb{S}_{\mathfrak{I}}$ *then there exists interval traces* $\{\wp'_j \mid j \in [m]\} \subseteq \mathbb{S}_{\mathfrak{I}}$ *that are pairwise* strongly *compatible with* $\bigcup_{i\in[n]} (\!|\, \wp_i \,|\!) = \bigcup_{j\in[m]} (\!|\, \wp'_j \,|\!)$ *and for each* $j \in [m]$, $(\!|\, \wp'_j \,|\!) \subseteq (\!|\, \wp_i \,|\!)$ *for some* $i \in [n]$.

> **Proof** We can give a constructive proof: We first analyse $\{\wp_i(0)\}_{i\in[n]}$, i.e., the interval at the first position. Clearly there exists intervals $\{[a_k, b_k]\}_{k\in\mathcal{K}}$ for a finite $\mathcal{K}$ that are all pairwise almost disjoint such that for each i there is a set $\mathcal{K}_i \subseteq \mathcal{K}$ with $\wp_i(0) = \bigcup_{k\in\mathcal{K}_i} [a_k, b_k]$. This holds as we can always partition at overlapping position as visualised in Fig. 5.4. We can thus replace every $\wp_i$ with $|\mathcal{K}_i|$ many interval traces by replacing the

first interval with the intervals in $[a_k, b_k]$ from $k \in \mathcal{K}_i$. The resulting set of standard traces agrees with the one we started from. The first position of those traces are either pairwise identical or almost disjoint as required by the definition of strong compatibility. For all traces that are identical on the first position we can proceed inductively. ■

The two traces in Ex. 5.7 are not strongly compatible but can be replaced by the 3 traces $\wp_1' \triangleq [0,\frac{1}{3}][0,\frac{1}{2}]$, $\wp_2' \triangleq [\frac{1}{3},\frac{1}{2}][0,\frac{1}{2}]$ and $\wp_3' \triangleq [0,\frac{1}{3}][\frac{1}{2},1]$ that denote the same set of standard traces but are pairwise strongly compatible.

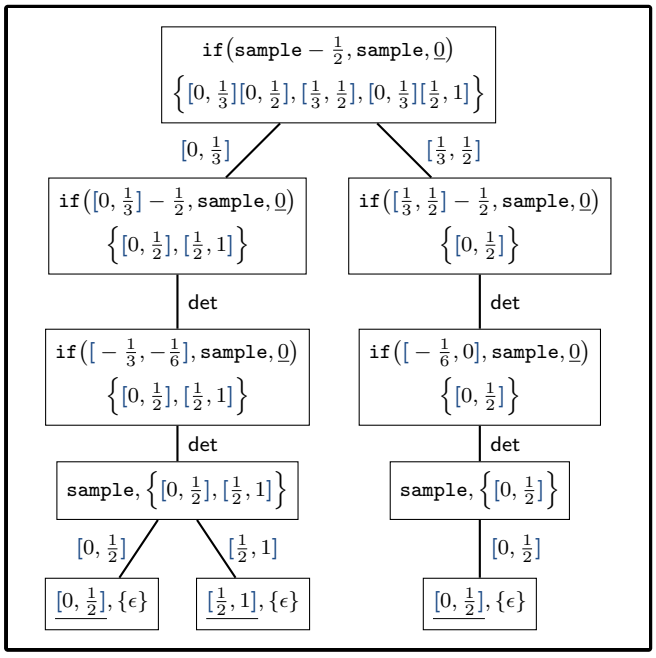


Figure 5.5: Example reduction for the term from Ex. 5.7 on a set of pairwise strongly compatible traces. Probabilistic and deterministic reduction steps are arranged as a tree.

Subject Expansion For Strongly Compatible Traces The crucial observation is that in the the statement of subject expansion (Lem. 5.6), the intervals in a probabilistic step should be pairwise almost disjoint. As we have seen in the example above pairwise compatible traces must not necessarily be almost disjoint in the first position. But pairwise strongly compatible traces are: the first position is either almost disjoint or identical. To make use of this idea we represent the reduction of a term given a set of interval traces as a tree. Nodes in the tree are of the from $(\mathcal{M}, A)$ we $\mathcal{M}$ is an interval term and $\emptyset \neq A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\mathrm{term}}$ a set of strongly compatible interval traces. The successors of a node are given by a relation $\rightsquigarrow$ where each transition is either labelled by det, to represents a deterministic reduction or by an interval $[a,b] \in \mathfrak{I}_{0,1}$:

$$\frac{\mathcal{M} \to_{\mathsf{det}} \mathcal{N}}{(\mathcal{M}, A) \rightsquigarrow_{\mathsf{det}} (\mathcal{N}, A)} \qquad \frac{\mathcal{M} \to_{[a,b]} \mathcal{N} \qquad B = \{\wp \mid [a,b]\wp \in A\} \neq \emptyset}{(\mathcal{M}, A) \rightsquigarrow_{[a,b]} (\mathcal{N}, B)}$$

If we again consider the example term fro Ex. 5.7 and the pairwise strongly compatible traces $\wp_1', \wp_2', \wp_3'$ from before we get the tree depicted in Fig. 5.5. Every det step corresponds to a deterministic reduction. For every probabilistic reduction the set of interval traces is stripped by its first position. As the set of traces is strongly compatible, the outgoing edges of every

node are labelled by almost disjoint intervals.

Proposition 5.9 (Completeness): *If* $\{\wp_i \mid i \in [n]\} \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},term}$ *are pairwise* strongly *compatible then* $\vdash \mathcal{M} : \{\!|(\alpha_i, \wp_i, \#^{\wp_i}_{\downarrow}(\mathcal{M})) \mid i \in [n]|\!\}$ *for some types* $\{\alpha_i\}_{i\in[n]}$

Proof We first make the following easy observation that follows immediately by the definition of strong compatibility: If $A \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$ is pairwise strongly compatible and $(\mathcal{M}, A) \rightsquigarrow^* (\mathcal{N}, B)$ and $(\mathcal{N}, B) \rightsquigarrow_{[a_i,b_i]} (\mathcal{N}_i, B_i)$ for $i \in [n]$ then $[a_i, b_i]$ are almost disjoint. Call this observation **(1)**. For our proof we consider the tree that is generated by $(\mathcal{M}, \{\wp_i \mid i \in [n]\})$. Note that this tree is finite. We claim that for every node $(\mathcal{N}, A)$ where $A = \{\dot\wp_i \mid i \in [k]\}$ in this tree we can type $\vdash \mathcal{N} : \{\!|(\alpha_i, \dot\wp_i, \#^{\dot\wp_i}_{\downarrow}(\mathcal{N})) \mid i \in [k]|\!\}$. We show this inductively by traversing the tree from the leafs up. Formally, we do induction on the shortest path to a leaf. In the base case, the node in question is a leaf: as by assumption each $\wp_i \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$ we get that each leaf of this tree has the from $(\mathcal{V}, \{\epsilon\})$ for some closed value $\mathcal{V}$. It is easy to check that for every value we can type $\vdash \mathcal{V} : \{\!|(\alpha, \epsilon, 0)|\!\}$ for some α, by either using **(num)** (in case of a numeral) or **(abs)** or **(fix)** followed by $(\{\!|\}\!)$ (in case of λ-or μ-abstraction). Now consider the case where $(\mathcal{N}, A)$ is a inner node. There are again two cases:

- $(\mathcal{N}, A) \rightsquigarrow_{\mathsf{det}} (\mathcal{P}, A)$, so $\mathcal{N} \to_{\mathsf{det}} \mathcal{P}$. Write $A = \{\dot\wp_i \mid i \in [k]\}$. By induction we can type $\vdash \mathcal{P} : \{\!|(\alpha_i, \dot\wp_i, \#^{\dot\wp_i}_{\downarrow}(\mathcal{P})) \mid i \in [k]|\!\}$. Now by Subject Expansion (Lem. 5.6) we can type $\vdash \mathcal{N} : \{\!|(\alpha_i, \dot\wp_i, \#^{\dot\wp_i}_{\downarrow}(\mathcal{P}) + 1) \mid i \in [k]|\!\}$ as required,
- In the other case, $(\mathcal{N}, A) \rightsquigarrow_{[a_i,b_i]} (\mathcal{P}_i, B_i)$ for $i \in [m]$. Lets write $B_i = \{\dot\wp^j_i \mid j \in [k_i]\}$. We have $A = \bigcup_{i\in[m]}\{[a_i, b_i]\dot\wp^j_i \mid j \in [k_i]\}$. By induction we $\vdash \mathcal{P}_i : \{\!|(\alpha^i_j, \dot\wp^i_j, \#^{\dot\wp^i_j}_{\downarrow}(\mathcal{P})) \mid j \in [k_i]|\!\} \triangleq \mathcal{A}_i$. By **(1)** we get that the $[a_i, b_i]$ are pairwise almost disjoint. By Lem. 5.6 we can thus type $\vdash \mathcal{N} : \bigcup_i \mathcal{A}_i^{(\uparrow[a_i,b_i],1)}$ as required.

■

Soundness and Completeness:

We can combine soundness and completeness of our type system in the following:

Theorem 4 (Soundness and Completeness): *For every term* M*,*

$$\bigvee_{\vdash M^{2\mathfrak{I}}:\mathcal{A}} \omega(\mathcal{A}) = \mathbb{P}_{term}(M)$$

if we assume M *to be AST then*

$$\bigvee_{\vdash M^{2\mathfrak{I}}:\mathcal{A}} \mathbb{E}(\mathcal{A}) = \mathbb{E}_{term}(M)$$

Proof We first show the first part: We already showed $\bigvee_{\vdash M^{2\mathfrak{J}}:\mathcal{A}} \omega(\mathcal{A}) \leq \mathbb{P}_{\text{term}}(M)$ in Prop. 5.5 as $M \triangleleft M^{2\mathfrak{J}}$. It remains to show that they are actually equal. Let $\epsilon > 0$. We show that there exist a $\vdash M^{2\mathfrak{J}} : \mathcal{A}$ such that $\omega(\mathcal{A}) \geq \mathbb{P}_{\text{term}}(M) - \epsilon$. Using the completeness of the interval-based semantics (Thm. 2), we get a *finite* set of pairwise compatible interval traces $\{\wp'_j \mid j \in [n]\} \subseteq \mathbb{T}^{\mathfrak{J}}_{M^{2\mathfrak{J}},\text{term}}$ such that $\sum_{j\in[n]} \omega(\wp'_j) \geq \mathbb{P}_{\text{term}}(M) - \epsilon$. Now from Lem. 5.8 there exists a finite set of interval traces with the same weight that is furthermore pairwise strongly compatible. Let $\{\wp_i \mid i \in [m]\}$ be this set. Note that $\{\wp_i \mid i \in [m]\} \subseteq \mathbb{T}^{\mathfrak{J}}_{M^{2\mathfrak{J}},\text{term}}$ as byLem. 5.8 for every $i \in [m]$, $(\!|\, \wp_i \,|\!) \subseteq (\!|\, \wp'_j \,|\!)$ for some $j \in [n]$. By Prop. 5.9 we get that $\vdash M^{2\mathfrak{J}} : \{\!|(\alpha_i, \wp_i, \#^{\wp_i}_{\downarrow}(M)) \mid i \in [m]|\!\} \triangleq \mathcal{A}$ for some types α_i. Now obviously $\omega(\mathcal{A}) = \sum_i \omega(\wp'_i) \geq \mathbb{P}_{\text{term}}(M) - \epsilon$, so we are done as we can let ϵ tend to 0.
For the second part we can proceed as before by using the second part of Thm. 2. ■

In our type system, the probability of termination equals the least upper bound over all derivations. This gives a recursion-theoretically optimal characterisation of AST that is purely based on the type system. One of the many novel features of our system compared to the work by Breuvart and Lago lies in the fact that we explicitly reason about execution time. This gives an alternative recursion theoretically optimal characterisation of PAST[3]. The idea of annotating types by a step count is also applicable in the setting of Breuvart and Lago, giving a strict generalisation of their system.

Remark: Lastly a few remarks: Firstly, while the system does look intimidating at first, this is mainly due to the the many language constructs. Each rule taken individually is actually simple, in the sense that no complex operations are needed. Our system can easily be generalized to the untyped λ-calculus considered in [Borgström et al. 2016]. This would also make our system truly compositional as the fixpoint rule is no longer needed. Secondly, we can observe that in each rule the interval traces in a set type are not important and only needed to derive the weight of the type (ω). We could thus see each set type $\mathcal{A}$ as a distribution on pairs of types and natural numbers by identifying every $\mathcal{A} = \{\!|(\alpha_i, \wp_i, \tau_i) \mid i \in [n]|\!\}$ with the distribution $\sum_{i\in[n]} \omega(\wp_i) \cdot \delta_{(\alpha_i,\tau_i)}$ on pairs. We would thus arrive at a system that is similar to the distribution type system by [Breuvart and Lago 2018], especially if we drop the second step count and consider distributions over types. The annotation by interval traces is, however, paramount to show subject reduction/expansion and employ soundness and completeness of the interval-based semantics. As we remarked at the beginning of this chapter, we cannot show correctness with the ideas of Breuvart and Lago as a weighted sum over the reduction relation due to the continuous nature of

[3]Similar to Thm. 2 the type system can provide a characterisation of PAST even if we do not assume that the term in question is already AST. In Thm. 4 we assume AST to make the statement more intuitive.

SPCF.

In this chapter we explicitly worked in a CbN setting and, unlike in the previous chapter, the system cannot straightforwardly be adapted to different evaluation strategies. Nonetheless, we do conjecture that the ideas from our system can also be applied in a CbV setting. As the work by Breuvart and Lago for a comparatively simple language already suggests, handling CbV requires even more sophisticated systems. We leave a CbV version of our system as future work.

Part II

Non-affine Recursion

Chapter 6

Counting-based Analysis of Non-affine Recursion

The interval based semantics and the local representation of it in form of an intersection type system allow reasoning about the termination behavior of SPCF and, for instance, derive complexity bounds on various decision problems. While the first part did provide an alternative ("countable") perspective on AST and is very convenient to derive lower bounds on the probability of termination, it does not lead to any practical AST verification method.

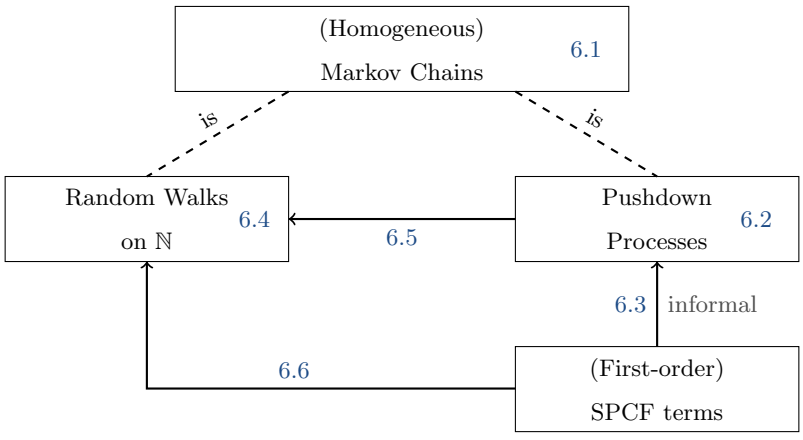
In this second part, we develop a framework that gives a sound method for AST analysis and can be used for actual analysis (in the style of the long list of related work [Fioriti and Hermanns 2015; McIver et al. 2018; Lago and Grellois 2019; Agrawal et al. 2018]). After presenting the theoretical underpinnings we give a practical proof system that can give AST guarantees. A particular strength of our system is that it is easy to automate. Existing approaches to AST analysis (many based on ranking functions) try to capture a broad range of programs and provide completeness properties for certain classes (e.g., [Fu and Chatterjee 2019]). However, AST proofs in these system must often be constructed by hand, by e.g. coming up with a ranking function (c.f. for instance [McIver et al. 2018]). There are, of course, some approaches that try to automate this as far as possible (see e.g. [Chatterjee et al. 2018; Chen and He 2020]).

Non-affine Recursion As already evident from the example in the introduction, making more than 1 recursive call severely complicates AST analysis and causes methods to become unsound ([Lago and Grellois 2019]). In this work, we provide a first formal study of AST analysis and lay particular emphasis on the non-affine nature. We do not aim for a complete proof system but instead present a system that is easy to implement and highlights the essence

of non-affine recursion, which is the possibility of making more than one recursive call. Our approach is based on a coarse over-approximation of recursion where we we do not consider the exact arguments of the recursive call but focus on the *number* of recursive calls. We explicitly focus on SPCF fixpoints that operate on first-order arguments (real numbers). Note that while we restrict the fixpoint itself to a first-order type, the surrounding computations can still be of higher-order.

Our work is closely related to [Olmedo et al. 2016], in that they also study recursive programs and allow for non-affine behavior. Our work differs nonetheless in several key aspects: While they considered a discrete imperative language with state, we work in a purely functional setting with continuous distributions. Their proposed rules can be used to show lower bounds on the probability of termination, doing so is, however, cumbersome as we have to given an explicit sequence $(l_n)_{n\in\mathbb{N}}$ that approaches the probability of termination where l_n gives the termination probability after n fixpoint unfoldings. Their rule informally reads: if, for all n, we assume that each recursive call terminates with probability l_n after n fixpoint unfoldings, and we show that it terminates with probability at least l_{n+1} after $n+1$ unfoldings, then the programs terminates with probability at least $\sup_n l_n$ (c.f. [Olmedo et al. 2016, Theorem 4.2]). This rule both requires the user to manually come up with a sequence $(l_n)_n$ and to directly reason on the program, whereas our system provides a sound reduction to a random walk that can be analysed efficiently in linear time.

Structure of this Chapter The structure of this chapter can graphically be represented by the following:



We begin by introducing markov chains on general state spaces (Section 6.1). Instead of considering SPCF directly, we first illustrate our counting-based approach on a probabilistic pushdown process. A pushdown process is a markov chain on stacks, where in each step the

first element of the stack popped and replaced by a (possibly empty) stack of new arguments (Section 6.2). We *informally* argue that it is intuitive to view recursion of first-order SPCF fixpoints as pushdown processes (Section 6.3). We next lay the foundations for random walks on the natural numbers and show linear AST decidability in Section 6.4. We then present a reduction from pushdown processes to random walks that we show sound (Section 6.5). As we used pushdown processes only as an intermediate representation to illustrate the counting-based approach, we then give a sound, direct extraction of a random walk from a first-order SPCF term in Section 6.6. This gives a sound method for AST analysis, that we turn into an effective proof system in the next chapter.

6.1 Homogeneous Markov Chains on General State Space

In this section, we provide the necessary preliminaries for discrete time Markov Chains. We focus on homogeneous chains, i.e., chains that are time invariant in the sense that the successor states are generated via a fixed transition matrix or kernel. Instead of introducing discrete and continuous state space chains separately, we directly give the construction for chains on an arbitrary (continuous) space. If the reader is not familiar with Markov chains on a discrete state space we refer her to [Baier and Katoen 2008, §10.1] for a gentle introduction.

For chains on a discrete state space the transition behavior is often given as a transition matrix, i.e., an explicit representation of the transition probability from each state given as a mass function on successor states. For a Markov Chain on a continuous state space (say the real numbers) we use a transition kernel instead of a transition matrix. Composition of steps which is seen as matrix multiplication in the discrete case is replaced by kernel composition. Our construction mostly follows the excellent book [Meyn and Tweedie 2009]. We choose a slightly different notation to keep the presentation as close as possible to the discrete case and the presentation used by Baier and Katoen. Assume that X is any set and Σ_X a σ-algebra on X. A Markov chain is a pair $\mathfrak{M} = (X, \kappa)$ where $\kappa : X \times \Sigma_X \to \mathbb{R}_{[0,1]}$ is a Markov kernel. Recall that this means that $\kappa(x, \cdot)$ is a probability measure for every $x \in X$ and $\kappa(\cdot, A)$ is a measurable function for every $A \in \Sigma_X$.

Canonical Measure Space With each chain $\mathfrak{M}$ we associate a canonical measure space $(\Omega, \Sigma_\Omega, \mu)$: The state space consist of all infinite sequences, i.e., $\Omega \triangleq X^\omega$. For a finite sequence measurable sets $A_0, \cdots, A_{n-1} \subseteq X$ we define the cylinder set $Cyl(A_0, \cdots, A_{n-1}) \triangleq \{w \in \Omega \mid \forall i \in [n] : w(i) \in A_i\}$. We then define $\Sigma_\Omega \triangleq \sigma(Cyl(A_0, \cdots, A_{n-1}) \mid n \in \mathbb{N}, \forall i \in [n] : A_i \in \Sigma_X)$, i.e., the smallest σ-algebra containing all cylinders[1]. For every $x \in X$ there exists a probability

[1] On first glance this construction differs from the construction in [Meyn and Tweedie 2009] and is similar to the construction in e.g. [Baier and Katoen 2008]. We can however show, that the resulting σ-algebra is

measure μ_x on (Ω, Σ_Ω) such that for every measurable sets $A_0, \cdots, A_{n-1}$ it holds that

$$\mu_x\big(Cyl(A_0, \cdots, A_{n-1})\big) = \int_{A_0} \kappa(x, dy_0) \int_{A_1} \kappa(y_0, dy_1) \cdots \kappa(y_{n-2}, A_{n-1})$$

For a proof of the existence of this measure see [Meyn and Tweedie 2009, Theorem 3.4.1].

Self Composition We can compose the transition kernel with itself and let κ^n denote the n-fold self composition of κ. $\kappa^n(x, A)$ gives the probability of, starting at x, being in A after exactly n steps. We can characterise $\kappa^n(x, A)$ in our canonical measure space as

$$\kappa^n(x, A) = \mu_x\big(Cyl(\underbrace{X, \cdots, X}_{n-1}, A)\big)$$

Remark: For discrete state spaces (i.e., when X is countable) the transition probability if often given as *transition matrix* $\mathfrak{P} : X \times X \to \mathbb{R}_{[0,1]}$ that satisfies $\sum_{y \in X} \mathfrak{P}(x, y) = 1$ for every $x \in X$. $\mathfrak{P}$ is also called a *stochastic matrix*. With $\mathfrak{P}^n$ we denote the n-fold matrix product of $\mathfrak{P}$. We can interpret $\mathfrak{P}^n(x, y)$ as the probability of moving from x to y in n steps. To reconcile this with the definition on a general state space, we can view $\mathfrak{P}$ as a transition kernel κ on the powerset of X, defined by $\kappa(x, A) = \sum_{y \in A} \mathfrak{P}(x, y)$ for $x \in X, A \in 2^X$. In the following, when working with a discrete state space, particular on the natural numbers, we often give chains via transition matrices. As we use the powerset as the σ-algebra every singleton set is also measurable, so the set $Cyl(\{x_0\}, \cdots, \{x_{n-1}\})$ is a measurable set of paths and $\mu_x\big(Cyl(\{x_0\}, \cdots, \{x_{n-1}\})\big) = \mathfrak{P}(x, x_0) \prod_{i=0}^{n-2} \mathfrak{P}(x_i, x_{i+1})$, which matches the usual construction in the discrete case ([Baier and Katoen 2008, Definition 10.9]).

6.2 Pushdown Process

First-order recursion is naturally modelled by a pushdown system, where the first element in a stack of future tasks is processed and replaced by list of new elements. For an example of how recursion (in a discrete language) can be modelled as a pushdown system, see e.g. [Olmedo et al. 2016, Section 6]. In this section, we provide the formal underpinnings of a probabilistic pushdown system on a continuous state space by describing a pushdown process as a Markov chain. Our system is related to the study of probabilistic pushdown automata (see e.g. [Brázdil et al. 2013]) but simpler in the sense that our system has no control state. Let (X, Σ_X) be any measurable space (for us, this will usually be $(\mathbb{R}, \Sigma_\mathbb{R})$). We define the set of stacks on X,

identical to the one by Meyn and Tweedie. This way, we hope to achieve a clearer presentation if the reader is familiar with the (usual) construction for discrete state spaces.

by $\mathsf{Stack}(X) \triangleq X^*$, i.e., finite sequences over X. With $\mathsf{Stack}^n(X)$ we denote the stacks with exactly n elements, i.e., the set X^n. We denote the empty stack by [] and the concatenation of stacks ϱ and φ by $\varrho\varphi$. We define a σ-algebra on $\mathsf{Stack}(X)$ by $\Sigma_{\mathsf{Stack}(X)} \triangleq \{\biguplus_{n\in\mathbb{N}} A_n \mid A_n \in \Sigma_{X^n}\}$ where Σ_{X^n} is the σ-algebra on X^n obtained as the n-fold product algebra of Σ_X. This is similar to the σ-algebra on traces (Sec. 3.2 or [Borgström et al. 2016]).

Generating kernel of Pushdown Process A pushdown process is a markov chain on stacks that is fully determined by an operation on the top element of the stack. A step kernel S is a subprobability kernel $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$. For any x, $S(x,\cdot)$ is a distribution on stacks that give the successor tasks. In the context of SPCF, we can think of $S(r,\cdot)$ as the distribution over recursive calls made on argument r. The missing probability mass $(1 - S(x, \mathsf{Stack}(X)))$, is interpreted as (the mass of) failure. We therefore define $\mathsf{Stack}(X)_\perp \triangleq \mathsf{Stack}(X) \cup \{\perp\}$, as stacks with a dedicated error state. The σ-algebra $\Sigma_{\mathsf{Stack}(X)}$ naturally extends to a σ-algebra $\Sigma_{\mathsf{Stack}(X)_\perp}$ on $\mathsf{Stack}(X)_\perp$[2].

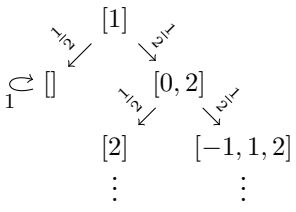
Definition 6.1: *Let $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$ be a subprobability kernel. We refer to this kernel as the* step kernel. *We define the associated kernel on stacks, $\kappa_S : \mathsf{Stack}(X)_\perp \times \Sigma_{\mathsf{Stack}(X)_\perp} \to \mathbb{R}_{[0,1]}$, called a* pushdown process, *by $\kappa_S([], A) \triangleq \delta_{[]}(A)$, $\kappa_S(\perp, A) \triangleq \delta_\perp(A)$ and*

$$\kappa_S(x :: \varphi, A) \triangleq S(x, \{\varrho \mid \varrho\varphi \in A\}) + \big(1 - S(x, \mathsf{Stack}(X))\big) \cdot \delta_\perp(A)$$

On the empty stack, [], and the error state, $\perp$, the chain is absorbing. For a non-empty stack $x :: \varphi$, the first element x is replaced according to the generating kernel S. With the missing probability of S $(1 - S(x, \mathsf{Stack}(X)))$ we move to the error state $\perp$. We note that the above definition is well defined as $\{\varrho \mid \varrho\varphi \in A\}$ is measurable provided A is. We can show:

Lemma 6.2 : *For every step kernel S the pushdown system κ_S is a Markov kernel.*

Example 6.3: As an example consider the step kernel $S : \mathbb{R} \times \Sigma_{\mathbb{R}} \to \mathbb{R}_{[0,1]}$ defined by $S(x,\cdot) = \frac{1}{2}\delta_{[x-1,x+1]} + \frac{1}{2}\delta_{[]}$. In every step, the first element x is replaced either with $[x-1, x+1]$ or by the empty stack, each with probability $\frac{1}{2}$. For the generated pushdown process from initial state [1] the set of reachable states is countable and we can depict an initial fragment on the right.



Termination of a Pushdown Process A pushdown process continually replaces the top element of the stack with segments of elements. Again think of an SPCF programs adding further recursive calls. We say that the generated stack process terminates when it reaches

[2]The elements in $\Sigma_{\mathsf{Stack}(X)_\perp}$ are exactly those that have the from A or $A \cup \{\perp\}$ for some $A \in \Sigma_{\mathsf{Stack}(X)}$.

the empty stack. We thus define:

Definition 6.4: *A step kernel* $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$ *is AST if for the induced pushdown process* κ_S *we have* $\lim_{n\to\infty} \kappa_S^n(\varrho, \{[]\}) = 1$ *for every* $\varrho \in \mathsf{Stack}(X)$

Note that the limit always exists (and lies between 0 and 1). The step kernel in Ex. 6.3 is AST.

6.3 First-Order SPCF as a Pushdown Process

Let $\mu_x^\varphi.M$ be a fixpoint term with no nested fixpoints. We call a fixpoint term $\mu_x^\varphi.M$ first-order if $\Vdash \mu_x^\varphi.M : \mathbf{R} \to \mathbf{R}$, i.e., recursion occurs on real numbers. A pushdown system is a natural model for recursive computations, as recursive calls of a program are processed in accord with the stack discipline. We briefly and informally argue that first-order SPCF terms naturally denote pushdown processes on the real numbers. For every argument $r \in \mathbb{R}$ we can analyse the recursive calls made when evaluating $(\mu_x^\varphi.M)\underline{r}$. We are only interested in calls made on the same recursion level from distinct call-sites, i.e., not in calls that are made when evaluating a recursive call itself. This naturally gives us a step kernel where $S(r, A)$ is the probability that, when evaluating $(\mu_x^\varphi.M)\,\underline{r}$, recursive calls to $(\mu_x^\varphi.M)$ are made to a stack of arguments within A[3]. This step kernel must not be a proper kernel, as the computation may fail (due to a failed `score`-construct). This motivates, why in a pushdown process the missing probability mass is interpreted as the mass of failure.

Example 6.5: As an example consider the SPCF term $\mu_x^\varphi.\underline{0} \oplus \big(\varphi(x-1) + \varphi(x+1)\big)$. This term intuitively denotes the step kernel S given by $S(x,\cdot) = \frac{1}{2}\delta_{[]} + \frac{1}{2}\delta_{[x-1,x+1]}$ which, incidentally, is the one we studied in Ex. 6.3. As a second example take the term $\mu_x^\varphi.\underline{0} \oplus \big(\varphi\texttt{sample} + \varphi\texttt{sample}\big)$. Here clearly on an argument r the distribution on stacks is [] with probability $\frac{1}{2}$ and otherwise $[r_1, r_2]$ where r_1, r_2 are i.i.d. on $[0,1]$.

It is well known, that higher-order recursion can be characterised as pushdown systems [Hague et al. 2017]. The takeaway of this section is that first-order SPCF terms naturally denote (probabilistic) pushdown processes. This extraction can be made formal[4]. As a detailed extraction is not required later and as a formal description takes up a significant amount of space, we choose to omit the construction in this work. The idea of viewing SPCF terms as

[3] Note that this differs substantially from the approach of e.g. [Olmedo et al. 2016] where the pushdown system operates on terms. In our system, S gives the actual arguments of recursive calls. Our pushdown system, therefore, does not require any control state as we solely operate on the first element of the stack. This is in contrast to e.g. the discrete system studied in [Brázdil et al. 2013].

[4] In the sense that we can explicitly construct a step kernel for every closed SPCF term of type $\mathbf{R} \to \mathbf{R}$ (that terminates with at least some positive probability on every input) and show that AST of the pushdown process coincides with AST of the program.

pushdown process is a useful intuition for the counting-based method we employ later.

6.4 Random Walk on $\mathbb{N}$

The previous informal reasoning suggests that analysing AST of a pushdown system is interesting and corresponds to the analysis of first-order SPCF. In our counting-based approach, we neglect the concrete values in a stack and simply focus on the number of elements. In this section we lay the necessary foundations to model counting as a random walk on $\mathbb{N}$.

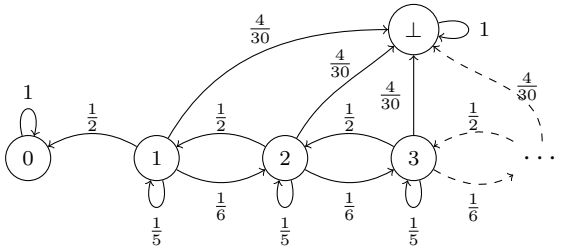
A particularly interesting and well-studied class of Markov Chains are those where the underlying state space consists of the natural numbers. For the purposes of this work it actually suffices to study a particular class of such chains, namely the ones that can be defined in terms of a distribution on the *relative change* that is invariant under the current state, often referred to as a random walk. The relative change in each step is given by *step distribution* which is a (subprobability) mass function $p : \mathbb{Z} \to \mathbb{R}_{[0,1]}$. We call p *finite* if it has finite support. As before, we interpret the missing probability, $(1 - \sum_{i\in\mathbb{Z}} p(i))$, as failure. We define the extended natural numbers $\mathbb{N}_\perp$ by $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$ for a fresh element $\perp$.

Definition 6.6: *Let $p : \mathbb{Z} \to \mathbb{R}_{[0,1]}$ be a (sub)probability mass function, subsequently called a* step distribution. *We define the generated transition matrix $\mathfrak{P}_p$ on $\mathbb{N}_\perp$ by:*

$$\mathfrak{P}_p(0,0) = 1 \qquad \mathfrak{P}_p(n,m) = p(m-n) \qquad n,m > 0$$
$$\mathfrak{P}_p(0,n) = 0 \qquad n > 0 \qquad \mathfrak{P}_p(\perp,\perp) = 1$$
$$\mathfrak{P}_p(n,0) = \textstyle\sum_{i\le -n} p(i) \qquad n > 0 \qquad \mathfrak{P}_p(n,\perp) = 1 - \sum_{i\in\mathbb{Z}} p(i) \qquad n > 0$$

The step distribution p gives the relative change in each step. For example the probability of moving from state 1 to 2 is $p(1)$ and moving from 6 to 1 is $p(-5)$. From every state (apart from 0) the missing mass of p moves to the error state $\perp$. As we walk on the natural numbers, the walk is truncated at 0, which is also an absorbing state. The probability of moving from n to 0 for an $n > 0$ is, therefore, the probability of moving from n to anything that is less than (or equal) 0. E.g. the probability of moving from 2 to 0 is $p(-2) + p(-3) + p(-4) + \cdots$.

Example 6.7: As an example consider the step distribution p defined by $p = \frac{1}{2}\delta_{-1} + \frac{1}{5}\delta_0 + \frac{1}{6}\delta_1$. Then the Markov chain $(\mathbb{N}_\perp, \mathfrak{P}_p)$ is sketched on the right.



Termination of Random Walks We say the random walk terminates if it reaches the absorbing state 0. It is AST if its does so almost-surely:

> **Definition 6.8:** *A step distribution p is called AST if for the associated transition matrix $\mathfrak{P}_p$, we have $\lim_{n\to\infty} \mathfrak{P}_p^n(m,0) = 1$ for every $m \in \mathbb{N}$.*

Note that the limit in the definition above does always exist (and lies between 0 and 1) as the sequence is monotone *increasing* and obviously bounded. From our definition above, the natural question arises whether a step distribution p is AST. For obvious reasons we assume that the support of p is finite and the image of p a subset of the rationals. A similar problem was already considered in [Lago and Grellois 2019][5]. In their work, Lago and Grellois showed that the problem can be reduced to a one-counter Markov decision process for which termination can be decided in polynomial time [Brázdil et al. 2010]. Their reasoning, however, yields a coarse upper bound, namely, polynomial time. We present an alternative proof that is both simpler (in the sense that we avoid the detour to one-counter MDPs) and gives a tighter (in fact optimal) complexity upper bound. The crux lies in a simple and checkable (decidable, if p finite) characterisation of AST:

> **Theorem 5 (Conditions for AST):** *A finite step distribution p is AST if and only if all of the following hold*
>
> *a)* $\sum_{i\in\mathbb{Z}} p(i) = 1$ *b)* $p \neq \delta_0$ *c)* $\sum_{i\in\mathbb{Z}} i \cdot p(i) \leq 0$

> **Proof** The interesting direction is the one that all three conditions imply AST. We do a case analysis on c) for equality or strict inequality. In case of strict inequality, the expectation of the Markov chain decreases by a lower bounded value in each step. We can then use Chebyshev's inequality to derive that the chain eventually reaches 0 a.s. In the case of equality, we make use of the famous result by George Pólya that states that any random walk on the integers with zero mean is recurrent [Novak 2014]. A full proof can be found in the Appendix: C. ■

For a finite step distribution p taking on only rational values, the above three conditions, and thus AST of p, can be checked in linear time.

Uniform AST In the example above, the relative change is given by the same step distribution. We also model the case where in each time step, a different distribution can be (non-deterministically) chosen from an available set of step distributions[6].

[5] The only real difference is a different interpretation of the missing probability mass of p. While we interpret this mass as failure, Lago and Grellois interpret it as instant termination.

[6] We could also frame this problem as a MDP where in each step a scheduler can choose the distribution.

Definition 6.9: *A family of discrete step distribution* $\{p_i\}_{i\in\mathcal{I}}$ *is called* uniform AST *if*

$$\lim_{n\to\infty}\left(\inf_{i_1,\cdots,i_n}\mathfrak{P}_{p_{i_1}}\cdots\mathfrak{P}_{p_{i_n}}(m,0)\right)=1$$

for every $m\in\mathbb{N}$.

Informally it reads that if step count n tends to ∞, no matter which step distribution from $\{p_i\}_{i\in\mathcal{I}}$ is chosen at each step, the walk eventually reaches 0 almost surely. Obviously, uniform AST implies AST for each of the p_i but, in general, not the other way around[7]. We can, however, show:

Lemma 6.10: *If* $\{p_i\}_{i\in\mathcal{I}}$ *is a* finite *family of discrete step distribution and each* p_i *is AST then* $\{p_i\}_{i\in\mathcal{I}}$ *is uniform AST.*

Proof The proof goes via the the definition of a limit and the pigeon hole principle. A proof can be found in the appendix: C ■

6.5 AST of Pushdown Process by Counting

We now have the necessary background on random walks on $\mathbb{N}$ to give a counting-based method for proving AST of a pushdown process. As argued before, we ignore the specific values of the elements replacing the top element of the stack; rather we simply count the number of elements added by the step kernel. While general pushdown processes are hard to analyse, we have seen in the previous section (especially in Thm. 5) that random walks enjoy easy computational characteristics. The counting-based method reduces pushdown system to random walks, making this nice analysis applicable to pushdown system.

Definition 6.11: *Given a step kernel* $S : X\times\Sigma_{\mathsf{Stack}(X)}\to\mathbb{R}_{[0,1]}$*, we define the family of subprobability mass functions* $\{p_{S,x}:\mathbb{N}\to\mathbb{R}_{[0,1]}\}_{x\in X}$*, called the* counting pattern of S*, by*

$$p_{S,x}(n)\triangleq S(x,\mathsf{Stack}^n(X))$$

That is, for each $x\in X$, $p_{S,x}(n)$ is the probability that $S(x,\cdot)$ replaces the top-of-stack x by a segment of n elements. For example, the counting pattern of the pushdown process from Ex. 6.4 is given by $p_{S,r}=\frac{1}{2}\delta_0+\frac{1}{2}\delta_2$ for every $r\in\mathbb{R}$.

Shifted Counting Pattern The counting pattern $p_{S,x}(n)$ for a $x\in X$ gives the probability mass of S adding n new elements to the stack. In the end, we try to analyse the number of elements in the stack and are thus interested in the relative change of the number of elements.

[7] As an counter example take the family $\{p_i\}_{i\geq 2}$ where $p_i\triangleq\frac{1}{i^2}\delta_{-1}+(1-\frac{1}{i^2})\delta_0$ which is obviously AST when considered individually. The family is, however, not uniform AST: take $m=1$. Now $\inf_{i_1,\cdots,i_n}\mathfrak{P}_{p_{i_1}}\cdots\mathfrak{P}_{p_{i_n}}(m,0)\leq\mathfrak{P}_{p_2}\mathfrak{P}_{p_3}\cdots\mathfrak{P}_{p_{n+1}}(m,0)=\left(1-\prod_{i=2}^{n+1}(1-\frac{1}{i^2})\right)$. But $\prod_{i=2}^{\infty}(1-\frac{1}{i^2})=\frac{1}{2}>0$.

Replacing the top-of-stack by a segment of n elements corresponds to an change in the size of the stack by $n-1$. In particular, replacement by a segment of 0 element (corresponding to pop) effectively removes one element. We therefore need to shift our mass function: for any mass function $p : \mathbb{N} \to \mathbb{R}_{[0,1]}$ on the natural numbers, we define the shifted mass function $\overline{p} : \mathbb{Z} \to \mathbb{R}_{[0,1]}$, by $\overline{p}(z) \triangleq p(z+1)$ for $z \geq -1$ and $\overline{p}(z) \triangleq 0$ otherwise.

AST Analysis Using Counting Patten The shifted pattern $\{\overline{p_{S,x}} : \mathbb{Z} \to \mathbb{R}_{[0,1]}\}_{x \in X}$ directly corresponds to the change of elements in the pushdown system. We can use this family as a step distribution family that denotes a random walk on $\mathbb{N}$. In particular, note the identical treatment of missing probability mass. In both the pushdown system and the random walk it is interpreted as immediate failure. We can show the following result that captures the essence of the counting-based method:

Theorem 6 (AST Analysis via Counting Pattern): *If step kernel $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$ has counting pattern $\{p_{S,x} : \mathbb{N} \to \mathbb{R}_{[0,1]}\}_{x \in \mathbb{R}}$, and $\{\overline{p_{S,x}} : \mathbb{Z} \to \mathbb{R}_{[0,1]}\}_{x \in \mathbb{R}}$ is uniform AST then the pushdown process generated by S is AST.*

Proof We claim the following for all $n, m \in \mathbb{N}$:

$$\inf_{x_1,\cdots,x_n \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, m) \leq \kappa_S^n(\varphi, \mathsf{Stack}^m(X))$$

We can prove this statement via induction on n with m universally quantified by careful study of kernel composition. A full proof (including this induction) that is worth reading can be found in the appendix: C. Once we have established the above, we can set $m = 0$ and get

$$\inf_{x_1,\cdots,x_n \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, 0) \leq \kappa_S^n(\varphi, \{[]\})$$

Now by assumption $\{\overline{p_{S,x}} : \mathbb{Z} \to \mathbb{R}_{[0,1]}\}_{x \in \mathbb{R}}$ is uniform AST so if $n \to \infty$ the left hand side tends to 1. So $\lim_{n\to\infty} \kappa_S^n(\varphi, \{[]\}) = 1$: the stack process is AST. ■

Via Thm. 6 we have the full power of AST analysis of random walk from Sec. 6.4 at our disposal. For instance, we can combine Thm. 6, Thm. 5 and Lem. 6.10 and get the following corollary[8]:

Corollary *If a step kernel $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$ has counting pattern $\{p_{S,x}\}_{x \in X}$ with finitely many distinct elements and for every x, $p_{S,x}(0) > 0$, $\sum_i p_{S,x}(i) = 1$ and $\sum_i i \cdot p_{S,x}(i) \leq 1$ then S is AST.*

As an example of the kind of programs the above theorem gives us consider the following:

[8] Note the changed condition of ≤ 1 compared to Thm. 6 as we phrase the corollary directly on the counting pattern and not on the shifted version ($\overline{p_{S,x}}$).

Example 6.12: Consider the SPCF term:

$$\mu_x^{\varphi}.\texttt{if}\ x\ \texttt{then}\ x\ \texttt{else}\left(\left(\underline{0}\oplus\varphi(x-1)\right)\oplus\left(\texttt{if}\ x-\underline{3}\ \texttt{then}\ \underline{0}\ \texttt{else}\ x\oplus\varphi(\varphi(\varphi(x)))\right)\right)$$

Let S be the extracted step kernel in the style of Sec. 6.3. Then $p_{S,r}=\delta_0$ for $r\leq 0$, $p_{S,r}=\frac{3}{4}\delta_0+\frac{1}{4}\delta_1$ for $0<r\leq 3$ and $p_{S,r}=\frac{1}{2}\delta_0+\frac{1}{4}\delta_1+\frac{1}{4}\delta_3$ for $r>3$. By using the above corollary, we can thus deduce that the term above is AST for every input. Note that while we only informally argued the connection of SPCF and pushdown processes, we make this formal in the next Sec. 6.6 and can then formally show the above term AST.

6.6 Counting-based Extraction of Random Walks

In the previous section, we have seen that counting gives a sound method for AST verification of pushdown processes. Due to the close correspondence between pushdown systems and first-order SPCF terms this suggests counting-based method are applicable to checking AST. While we did give a sound proof of the counting-based method for pushdown systems we have only informally sketched the relation between SPCF and pushdown systems. This section is devoted to closing this gap. Instead of taking the detour and first extract a stack process only to analyse it via counting, we take the direct path: given a first-order SPCF term we count the number of recursive calls directly. We show the soundness of this approach: AST of the extracted random walk implies AST of the SPCF term.

The problem with formally counting the number of recursive calls is that the returned value of a prior call can influence not only what the next calls are but also how many calls are made. As an example consider $\mu_x^{\varphi}.\texttt{if}\, fx\,\texttt{then}\, fx\,\texttt{else}\, fx+fx$ where the number of recursive calls is either 2 or 3 depending on the outcome of the first.

Avoiding Recursion in Conditionals Lets fix a first-order fixpoint of the form $\mu_x^{\varphi}.M$ with *no* nested fixpoints. For simplicity we replace the fixpoint with a more concise version $\boxed{\mu}$. The term we analyse is thus $\mathbf{body}_{\mu_x^{\varphi}.M}(r)\triangleq M[\underline{r}/x,\boxed{\mu}/\varphi]$, i.e., the body of the fixpoint with a fixed argument and $\boxed{\mu}$ for all recursive calls.

The crux of the counting-based approach is now to disallow the outcome of recursive calls to influence branching in the programs, i.e., recursive outcomes may not be used inside guards of conditionals or `score`-constructs. Obviously, this cannot be characterised purely syntactically as the property we seek is semantic in nature. We enforce this by a more involved simple type system where we add a dedicated type $\mathbf{R}^{\top}$ for recursive outcomes that cannot be used within guards. We define simple types in Fig. 6.1a. The idea of $\mathbf{R}^{\top}$ being more restrictive than $\mathbf{R}$ can

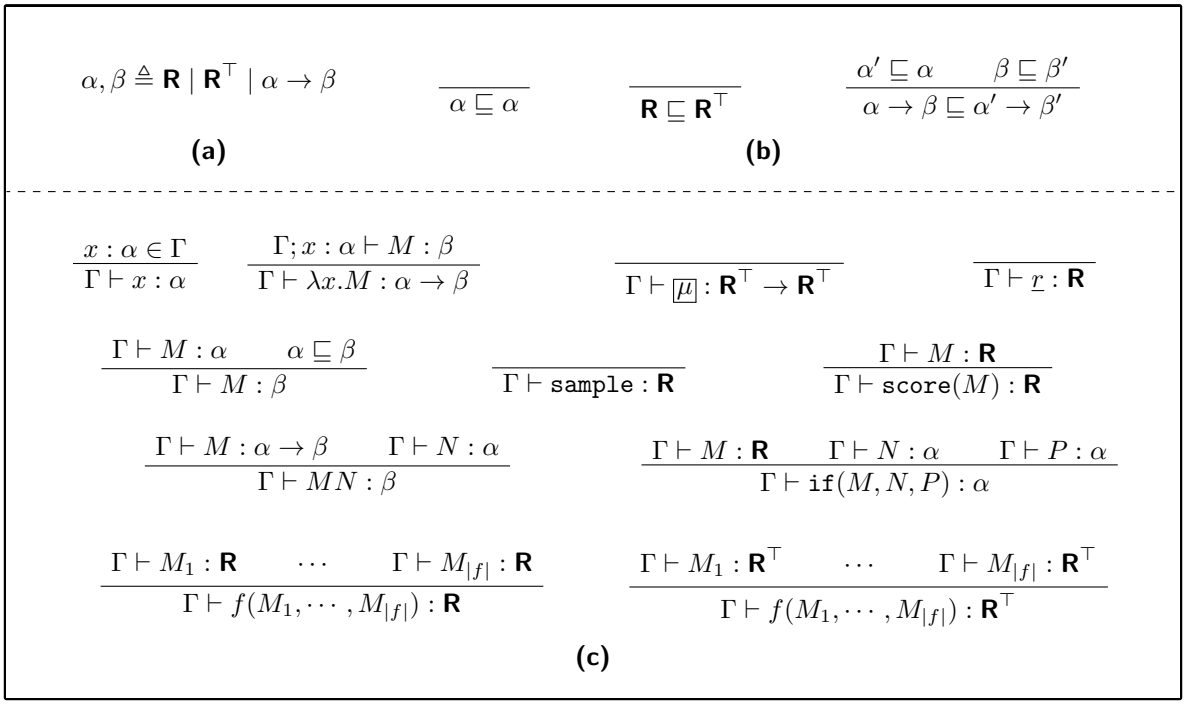


Figure 6.1: Simple typing judgments that guarantee that recursive outcomes are never used inside inside conditionals.

be formalized via a subtyping relation given in Fig. 6.1b. Typing judgments are of the form $\Gamma \vdash M : \alpha$ and given in Fig. 6.1c. The crucial step is the rule for conditionals combined with the fixpoint rule. For conditionals we require that the term in the guard position has $\mathbf{R}$ and at the same time the recursive abstraction $\boxed{\mu}$ has the more restrictive return type $\mathbf{R}^\top$. Combined with subtyping this gives a semantic guarantee that the recursive abstraction cannot be used inside conditionals. Note that the type system works with the simplified fixpoint constructs, $\boxed{\mu}$. For now on we assume that the fixed $\mu_x^\varphi.M$ satisfies $\vdash \mathbf{body}_{\mu_x^\varphi.M}(r) : \mathbf{R}^\top$ for some r[9].

A Recursion Ignoring Reduction By not having to consider probabilistic outcomes in conditionals we can ignore the concrete recursive outcome entirely. We extend the SPCF syntax by a new value construct $\star$ with $\Gamma \Vdash \star : \mathbf{R}$, that can be seen as an unknown numeral. Whenever we encounter a recursive abstraction $\boxed{\mu}$ applied to an argument we just substitute in the dummy value $\star$. If we evaluate a primitive function and any argument is $\star$, the functions value itself is $\star$. Apart from a term and a trace we annotate each configuration with a number

[9] We obviously have that $\mathbf{body}_{\mu_x^\varphi.M}(r)$ is typeable for some r iff it is typeable for all r. We could have also used the direct fixpoint rule: $\frac{\Gamma; \varphi : \mathbf{R}^\top \to \mathbf{R}^\top; x : \mathbf{R} \vdash M : \mathbf{R}^\top}{\Gamma \vdash \mu_x^\varphi.M : \mathbf{R}^\top \to \mathbf{R}^\top}$ and type $\mu_x^\varphi.M$ directly

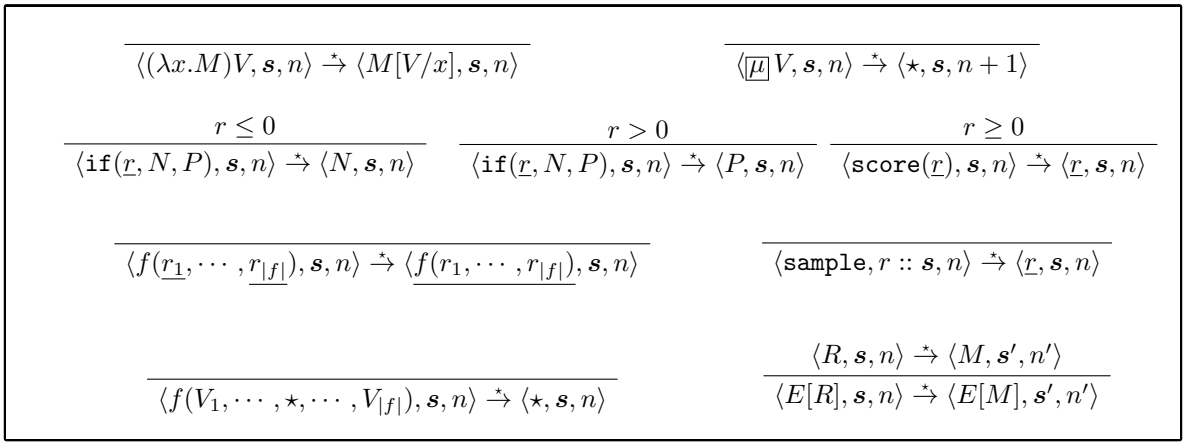


Figure 6.2: Small-step reduction rules for $\xrightarrow{\star}$.

that counts the number of recursive abstractions resolved that way. The reduction relation $\xrightarrow{\star}$ is given in Fig. 6.2. It operates on configurations $\langle M, \boldsymbol{s}, n\rangle$ where M is a term, $\boldsymbol{s}$ a trace and n a natural number that counts the number of recursive calls.

The simple type system (Fig. 6.1c) is crucial for the reduction, as terms of the from $\texttt{if}(\star, N, P)$ or $\texttt{score}(\star)$ cannot reduce any further. We can observe that if $\mathbf{body}_{\mu_x^{\varphi}.M}(r)$ is typeable in Fig. 6.1c such terms are never reachable. That is, if $\langle \mathbf{body}_{\mu_x^{\varphi}.M}(r), \boldsymbol{s}, n\rangle \xrightarrow{\star}{}^{*} \langle N, \boldsymbol{s}', n'\rangle$ then N is either a value or of the from $E[R]$[10]. To show this, we extend the system in Fig. 6.1c by the axiom $\Gamma \vdash \star : \mathbf{R}^{\top}$, show subject reduction and can conclude as terms containing $\texttt{if}(\star, N, P)$ or $\texttt{score}(\star)$ are not typeable.

Counting Recursive Calls For any term N typeable in the system in Fig. 6.1c we define:

$$\mathbb{T}^{\star}_{N;n} \triangleq \{\boldsymbol{s} \in \mathbb{S} \mid \exists V : \langle N, \boldsymbol{s}, 0\rangle \xrightarrow{\star} \langle V, \epsilon, n\rangle\}$$

These are all traces on which exactly n calls are made. As the reduction relation is deterministic we get that $\{\mathbb{T}^{\star}_{N;n}\}_n$ are pairwise disjoint. It is easy to see that $\mathbb{T}^{\star}_{N;n}$ is a measurable set of traces (similar arguments as in [Borgström et al. 2016]).

> **Definition 6.13:** *Given a term $\mu_x^{\varphi}.M$ we define the family $\{[\![\mu_x^{\varphi}.M \mid r]\!] : \mathbb{N} \to \mathbb{R}_{[0,1]}\}_{r \in \mathbb{R}}$, called the* counting pattern of $\mu_x^{\varphi}.M$, *by*
>
> $$[\![\mu_x^{\varphi}.M \mid r]\!](n) \triangleq \mu_{\mathbb{S}}\big(\mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^{\varphi}.M}(r);n}\big)$$

[10] Note, that in our extended system we treat $\star$ as a value and terms of the form $f(V_1, \cdots, \star, \cdots, V_{|f|})$ as a redex.

$[\![\mu^{\varphi}_x.M \mid r]\!] (n)$ gives the probability of making n calls on argument r.

> **Example 6.14:** As an example consider the term $M \triangleq x \oplus \big(\varphi(x) \oplus_x \varphi(\varphi(x))\big)$. We get $[\![\mu^{\varphi}_x.M \mid r]\!] (0) = \frac{1}{2}$, $[\![\mu^{\varphi}_x.M \mid r]\!] (1) = \frac{1}{2} \cdot \min(1, \max(r, 0))$, $[\![\mu^{\varphi}_x.M \mid r]\!] (2) = \frac{1}{2} \cdot (1 - \min(1, \max(r, 0)))$ and $[\![\mu^{\varphi}_x.M \mid r]\!] (n) = 0$ for $n > 2$.

We can easily see that for any r we have $\sum_n [\![\mu^{\varphi}_x.M \mid r]\!] (n) \leq 1$ by the same argument as in [Mak et al. 2021, Lemma 7]. It is important to note, that even for terms that are not AST the above sum may be 1, i.e., the sum does not represent the termination probability. E.g., the term $\mu^{\varphi}_x.\varphi(x) \oplus \varphi(\varphi x)$ is clearly not AST but the above sum does equal 1. (The interested reader is advised to check this herself). The above sum can be less than 1 because of failed `score`-constructs. E.g. for the term $\mu^{\varphi}_x.\texttt{score}(\underline{-1}) \oplus x$ the sum equals $\frac{1}{2}$.

> **Remark:** We can make the following informal observation that clarifies the meaning of $[\![\mu^{\varphi}_x.M \mid r]\!] (n)$. If we assume that $S : \mathbb{R} \times \Sigma_{\mathsf{Stack}(\mathbb{R})} \to \mathbb{R}_{[0,1]}$ is the step kernel that models the recursive calls of $\mu^{\varphi}_x.M$ in the style of Sec. 6.3 then the counting pattern extracted for S agrees with $[\![\mu^{\varphi}_x.M \mid r]\!]$, i.e., $p_{S,r} = [\![\mu^{\varphi}_x.M \mid r]\!]$.

Termination via Counting Patterns We can show that the extracted counting pattern provides a sound way to reason about AST:

> **Theorem 7 (AST via Counting Pattern):** *If* $\{\overline{[\![\mu^{\varphi}_x.M \mid r]\!]}\}_{r \in \mathbb{R}}$ *is uniform AST then* $\mu^{\varphi}_x.M$ *is AST on every argument.*

This theorem allows us to e.g. formally justify the observations in Ex. 6.12 and show this program to be AST on every input.

6.6.1 Correctness Proof

Proving Thm. 7 requires some care and takes up the remainder of this section. We begin by giving a rough outline of the proof for orientation. The fundamental idea is to decompose the set of terminating traces according to the number of recursive calls made (not only on the first level). We formalize this decomposition by a special kind of tree structure called number tree. We then show a direct correspondence between number trees and and terminating runs of the random walk generated by $\{\overline{[\![\mu^{\varphi}_x.M \mid r]\!]}\}_{r \in \mathbb{R}}$. Henceforth fix a term $\mu^{\varphi}_x.M$.

Number Trees We define a *number tree* by the following:

$$\mathcal{S} \triangleq n \triangleright [\mathcal{S}_1, \cdots, \mathcal{S}_n]$$

where $n \in \mathbb{N}$. We can depict each number tree by viewing n as the label of the node and $\mathcal{S}_1, \cdots, \mathcal{S}_n$ as the children as depicted in Fig. 6.3a. Note that the simplest tree is given by $0 \triangleright []$. Two (distinct) example trees are given in Fig. 6.3b and Fig. 6.3c.

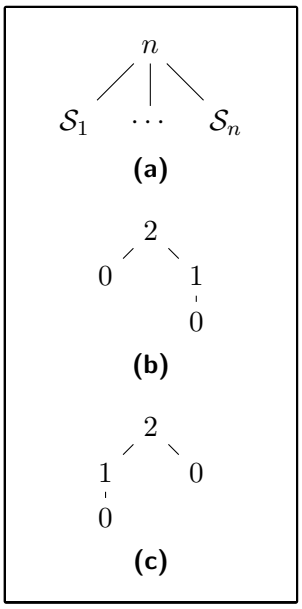


Figure 6.3: Example number trees.

Summary Semantics We define a *summary* as an element $\square^r_{r'}$ for $r, r' \in \mathbb{R}$. With $\mathfrak{O}$ we denote the set of all summaries and with $\mathbb{S}^{\square} \triangleq (\mathfrak{O} \cup \mathbb{R}_{[0,1]})^*$ the set of summary traces. We can define a summary semantics working on summary traces in Fig. 6.4. For every recursive call, we substitute in a summary, i.e., an abbreviation for a trace, on the traces. Note that this semantics closely corresponds to the semantics in Fig. 6.2 used to count the recursive calls. The only difference is that we do not blindly substitute in a dummy value $\star$ but predefine the outcome via a summary[11]. We set

$$\mathbb{T}^{\square}_{r \mapsto r'} = \{\boldsymbol{s} \in \mathbb{S}^{\square} \mid \langle \mathbf{body}_{\mu^{\varphi}_x.M}(r), \boldsymbol{s} \rangle \xrightarrow{\square}{}^{*} \langle \underline{r'}, \epsilon \rangle\}$$

as all summary traces on which the term on argument r evaluates to argument r'.

Number Trees as Traces The summary semantics explicitly lists recursive calls, as we can view the summary $\square^r_{r'}$ as a placeholder for a trace such that $(\mu^{\varphi}_x.M)\underline{r}$ evaluates to $\underline{r'}$. The summaries allow us to partition the set of terminating traces according to the number of calls made on each level. We can specify the number of calls by a number tree. For a tree $n \triangleright [\mathcal{S}_1, \cdots, \mathcal{S}_n]$ there should be n direct recursive calls and inside those calls the number of calls is inductively specified by $\mathcal{S}_i$. We consider the following example for some intuition:

> **Example 6.15:** Consider the term $\mu^{\varphi}_x.\varphi(\varphi x) \oplus \left(\underline{0} \oplus \varphi x\right)$ and the number tree in Fig. 6.3b. All traces that correspond to this tree should make 2 recursive calls in the first level. In the first of those calls, no further call is made and on the second a single one is made and afterwards none. This corresponds to the following set of terminating traces:
>
> $$\{s \in \mathbb{S}^7 \mid s_1 \in [0, \tfrac{1}{2}], s_2 \in (\tfrac{1}{2}, 1], s_3 \in [0, \tfrac{1}{2}], s_4 \in (\tfrac{1}{2}, 1], s_5 \in (\tfrac{1}{2}, 1], s_6 \in (\tfrac{1}{2}, 1], s_7 \in [0, \tfrac{1}{2}]\}$$

[11] Note that, unlike in $\xrightarrow{}$ we do not need to count the number of calls, as we can simply count the number of summaries in a trace which equals the number of calls made.

$$\frac{}{\langle (\lambda x.M)V, \boldsymbol{s}\rangle \xrightarrow{\square} \langle M[V/x], \boldsymbol{s}\rangle} \qquad \frac{}{\langle (\boxed{\mu})\underline{r}, \square^{r}_{r'} :: \boldsymbol{s}\rangle \xrightarrow{\square} \langle \underline{r'}, \boldsymbol{s}\rangle} \qquad \frac{}{\langle \mathtt{sample}, r :: \boldsymbol{s}\rangle \xrightarrow{\square} \langle \underline{r}, \boldsymbol{s}\rangle}$$

$$\frac{r \leq 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}\rangle \xrightarrow{\square} \langle N, \boldsymbol{s}\rangle} \qquad \frac{r > 0}{\langle \mathtt{if}(\underline{r}, N, P), \boldsymbol{s}\rangle \xrightarrow{\square} \langle P, \boldsymbol{s}\rangle} \qquad \frac{r \geq 0}{\langle \mathtt{score}(\underline{r}), \boldsymbol{s}\rangle \xrightarrow{\square} \langle \underline{r}, \boldsymbol{s}\rangle}$$

$$\frac{}{\langle f(\underline{r_1}, \cdots, \underline{r_{|f|}}), \boldsymbol{s}\rangle \xrightarrow{\square} \langle \underline{f(r_1, \cdots, r_{|f|})}, \boldsymbol{s}\rangle} \qquad \frac{\langle R, \boldsymbol{s}\rangle \xrightarrow{\square} \langle M, \boldsymbol{s}'\rangle}{\langle E[R], \boldsymbol{s}\rangle \xrightarrow{\square} \langle E[M], \boldsymbol{s}'\rangle}$$

Figure 6.4: Small-step reduction rules for $\xrightarrow{\square}$.

Definition 6.16: *For each number tree $\mathcal{S}$ we can define a family of sets of traces $\{\mathbb{A}^{\mathcal{S}}_{r\mapsto r'}\}_{r,r'\in\mathbb{R}}$ by induction on $\mathcal{S}$ as follows:*

$$\frac{\boldsymbol{s}_1 \square^{r_1}_{r'_1} \boldsymbol{s}_2 \cdots \square^{r_n}_{r'_n} \boldsymbol{s}_{n+1} \in \mathbb{T}^{\square}_{r\mapsto r'} \qquad \left\{\dot{\boldsymbol{s}}^{r_i}_{r'_i} \in \mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto r'_i}\right\}_{i=1}^{n}}{\boldsymbol{s}_1 \dot{\boldsymbol{s}}^{r_1}_{r'_1} \boldsymbol{s}_2 \cdots \dot{\boldsymbol{s}}^{r_n}_{r'_n} \boldsymbol{s}_{n+1} \in \mathbb{A}^{n\triangleright[\mathcal{S}_1,\cdots,\mathcal{S}_n]}_{r\mapsto r'}}$$

Elements in $\mathbb{A}^{n\triangleright[\mathcal{S}_1,\cdots,\mathcal{S}_n]}_{r\mapsto r'}$ are thus obtained by taking every summary trace with exactly n summaries that takes r to r'. For the ith summary (the ith recursive call) we then substitute in a trace from $\mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto r'_i}$ that is recursively obtained from the ith child ($\mathcal{S}_i$). Every number tree thus defines a specific set of traces. By induction it is easy to see that for distinct trees the obtained sets of traces are disjoint. The interested reader is advised to match this definition with Ex. 6.15. We define $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}} \triangleq \bigcup_{r'\in\mathbb{R}} \mathbb{A}^{\mathcal{S}}_{r\mapsto r'}$ as all terminating traces with recursion according to $\mathcal{S}$. It is easy to see that $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$ is measurable.

Probability Distributions on Number Trees We can view a number tree as being sampled from a step distribution $p : \mathbb{N} \to \mathbb{R}_{[0,1]}$. For every node we sample a number n according to p, add n nodes and continue by sampling the child nodes. Every step distribution $p : \mathbb{N} \to \mathbb{R}_{[0,1]}$ thus gives a natural probability to a number tree $\mathcal{S}$ as just the product over all nodes in $\mathcal{S}$. More generally we define:

Definition 6.17: *For a family of step distributions $\{p_k\}_k$ and a number tree $\mathcal{S}$ we define $\mathbb{P}^{\{p_k\}_k}_{inf}(\mathcal{S})$ by induction as*

$$\mathbb{P}^{\{p_k\}_k}_{inf}(n \triangleright [\mathcal{S}_1, \cdots, \mathcal{S}_n]) \triangleq \left(\inf_k p_k(n)\right) \cdot \prod_{i=1}^{n} \mathbb{P}^{\{p_k\}_k}_{inf}(\mathcal{S}_i)$$

where we follow the usual convention that $\prod_{i=1}^{0} = 1$. Note that if $\{p_k\}_k$ consist of a single

element $\mathbb{P}_{\text{inf}}^{\{p_k\}_k}(\mathcal{S})$ for a tree $\mathcal{S}$ is the product of the probability of every node in that tree. Taking the infimum follows the general scheme as e.g. in the definition uniform AST (Definition 6.9). We show that if we take the family $(\llbracket \mu_x^\varphi.M \mid r \rrbracket)_{r\in\mathbb{R}}$ the probability of tree is a lower bound on the measure of $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$.

Example 6.18: For Example 6.15 we get that the family $(\llbracket \mu_x^\varphi.M \mid r \rrbracket)_{r\in\mathbb{R}}$ comprises a single element, namely the function p defined by $p(2) = \frac{1}{2}$, $p(1) = \frac{1}{4}$ and $p(0) = \frac{1}{4}$. The probability of the tree $\mathcal{S}$ in Fig. 6.3b, as defined above then equals $\frac{1}{2}\frac{1}{4}\frac{1}{4}\frac{1}{4} = \frac{1}{128}$ which is less than or equal (in this case equal) to the measure of $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$.

Proposition 6.19 : *For every number tree $\mathcal{S}$ and any r we have*

$$\mathbb{P}_{inf}^{\{\llbracket \mu_x^\varphi.M|r'\rrbracket\}_{r'}}(\mathcal{S}) \leq \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\big)$$

Proof By induction on $\mathcal{S}$ with r universally quantified. Let $\mathcal{S} = n \triangleright [\mathcal{S}_1, \cdots, \mathcal{S}_n]$. We first analyse $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$: by definition every $\boldsymbol{s} \in \mathbb{A}^{\mathcal{S}}_{r\mapsto r'}$ has the from $\boldsymbol{s} = \boldsymbol{s}_1 \dot{\boldsymbol{s}}^{r_1}_{r'_1} \boldsymbol{s}_2 \cdots \dot{\boldsymbol{s}}^{r_n}_{r'_n} \boldsymbol{s}_{n+1}$ for some $\boldsymbol{s}_1 \Box^{r_1}_{r'_1} \boldsymbol{s}_2 \cdots \Box^{r_n}_{r'_n} \boldsymbol{s}_{n+1} \in \mathbb{T}^{\Box}_{r\mapsto r'}$ and $\dot{\boldsymbol{s}}^{r_i}_{r'_i} \in \mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto r'_i}$. We can observe that $\{\boldsymbol{s}_1 \cdots \boldsymbol{s}_{n+1} \mid \boldsymbol{s}_1 \Box^{r_1}_{r'_1} \boldsymbol{s}_2 \cdots \Box^{r_n}_{r'_n} \boldsymbol{s}_{n+1} \in \mathbb{T}^{\Box}_{r\mapsto r'}, r' \in \mathbb{R}\} = \mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^\varphi.M}(r);n}$ by comparing the relation $\xrightarrow{\star}$ and $\xrightarrow{\Box}$. If we concatenate two sets of traces the measure of the new set is the product of the individual measures. As we know that n traces are substituted we can take the infimum over all possible arguments which gives us a trivial lower bound on $\mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\big)$:

$$\mu_{\mathbb{S}}\big(\mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^\varphi.M}(r);n}\big) \cdot \inf_{r_1,\cdots,r_n} \prod_{i=1}^{n} \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto\mathbb{R}}\big) \leq \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\big) \tag{1}$$

By induction we get that $\mathbb{P}_{\text{inf}}^{\{\llbracket \mu_x^\varphi.M|r'\rrbracket\}_{r'}}(\mathcal{S}_i) \leq \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}_i}_{\dot{r}\mapsto\mathbb{R}}\big)$ for every i and every $\dot{r}$. Note that the left hand side does not depend on $\dot{r}$ so in particular

$$\prod_{i=1}^{n} \mathbb{P}_{\text{inf}}^{\{\llbracket \mu_x^\varphi.M|r'\rrbracket\}_{r'}}(\mathcal{S}_i) \leq \inf_{r_1,\cdots,r_n} \prod_{i=1}^{n} \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto\mathbb{R}}\big) \tag{2}$$

We can now put this all together and get:

$$\begin{aligned}
\mathbb{P}_{\text{inf}}^{\{\llbracket \mu_x^\varphi.M|r'\rrbracket\}_{r'}}(\mathcal{S}) &= \inf_{r'} \llbracket \mu_x^\varphi.M \mid r' \rrbracket(n) \cdot \prod_{i=1}^{n} \mathbb{P}_{\text{inf}}^{\{\llbracket \mu_x^\varphi.M|r'\rrbracket\}_{r'}}(\mathcal{S}_i) && \text{def} \\
&\leq \llbracket \mu_x^\varphi.M \mid r \rrbracket(n) \cdot \inf_{r_1,\cdots,r_n} \prod_{i=1}^{n} \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto\mathbb{R}}\big) && \text{inf and } (2) \\
&= \mu_{\mathbb{S}}\big(\mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^\varphi.M}(r);n}\big) \cdot \inf_{r_1,\cdots,r_n} \prod_{i=1}^{n} \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}_i}_{r_i\mapsto\mathbb{R}}\big) && \text{def of } \llbracket \mu_x^\varphi.M \mid r' \rrbracket \\
&\leq \mu_{\mathbb{S}}\big(\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\big) && (1)
\end{aligned}$$

as required. ■

We can thus lower bound the measure of $\mathbb{A}^{\mathcal{S}}_{r'\mapsto\mathbb{R}}$ by means $\{[\![\mu^{\varphi}_{x}.M \mid r']\!]\}_{r'}$. Note that unlike the measure of $\mathbb{A}^{\mathcal{S}}_{r'\mapsto\mathbb{R}}$ we can compute $\mathbb{P}^{\{[\![\mu^{\varphi}_{x}.M|r']\!]\}_{r'}}_{\text{inf}}(\mathcal{S})$ efficiently.

Number Trees as Terminating Runs It is easy to see that for every family of subprobability mass functions on the natural numbers $\{p_k\}_k$, $\sum_{\mathcal{S}} \mathbb{P}^{\{p_k\}_k}_{\text{inf}}(\mathcal{S}) \leq 1$. What we can show is the following:

Lemma 6.20 : *If $\{p_k\}_k$ is a family of step distributions and $\{\overline{p_k}\}_k$ is uniform AST then*

$$\sum_{\mathcal{S}} \mathbb{P}^{\{p_k\}_k}_{inf}(\mathcal{S}) = 1$$

Proof We can establish a bijective correspondence between number trees and terminating runs of the markov chain generated by $(\overline{p_k})_k$ started from state 1. As an example the tree Fig. 6.3b would denote the run $1, 2, 1, 1, 0$, i.e., start at 1, then increase by $2-1$, then by $0-1$ then by $1-1$ and then by $0-1$ (where we traverse the tree in pre-order). The sum in question then equals the probability of all terminating runs which by assumption is 1. A detailed proof can be found in the appendix: C ■

We can now finally proof the theorem:

Restatement of Theorem 7 : *If $\{\overline{[\![\mu^{\varphi}_{x}.M \mid r]\!]}\}_{r\in\mathbb{R}}$ is uniform AST then $\mu^{\varphi}_{x}.M$ is AST on every argument.*

Proof We obviously have $\mathbb{T}_{(\mu^{\varphi}_{x}.M)\underline{r},\text{term}} \supseteq \biguplus_{\mathcal{S}} \mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$ (in fact they are equal but we do not require this for the proof). We can thus deduce:

$$\mu_{\mathbb{S}}\left(\mathbb{T}_{(\mu^{\varphi}_{x}.M)\underline{r},\text{term}}\right) \geq \mu_{\mathbb{S}}\left(\biguplus_{\mathcal{S}} \mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\right) \overset{(1)}{=} \sum_{\mathcal{S}} \mu_{\mathbb{S}}\left(\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}\right) \overset{(2)}{\geq} \sum_{\mathcal{S}} \mathbb{P}^{\{[\![\mu^{\varphi}_{x}.M|r']\!]\}_{r'}}_{\text{inf}}(\mathcal{S}) \overset{(3)}{=} 1$$

where **(1)** follows from the fact that $\mathbb{A}^{\mathcal{S}}_{r\mapsto\mathbb{R}}$ is disjoint for distinct number trees, **(2)** from Prop. 6.19 and **(3)** from Lem. 6.20. ■

6.7 ϵ-Recursion Avoiding Fixpoint Terms

Thm. 7 gives a sound method for checking AST of a first-order program. In this section, we focus on a particular class of programs where we do not need to extract the entire counting pattern but only focus on the probability of avoiding recursion, i.e., $[\![\mu^{\varphi}_{x}.M \mid r]\!](0)$. We assume that no `score`-constructs in $\mu^{\varphi}_{x}.M$ can fail[12]. We are interested in programs where this probability is lower bounded by some ϵ, i.e., for every possible input the probability of

[12]This implies that $\sum_n [\![\mu^{\varphi}_{x}.M \mid r]\!](n) = 1$.

$$\frac{}{\{x:[\alpha]\}\vdash x:\alpha} \qquad \frac{\Gamma;x:a\vdash M:\alpha}{\Gamma\vdash\lambda x.M:a\to\alpha} \qquad \frac{\Gamma\vdash M:[\alpha_i]\to\beta \qquad \{\Gamma_i\vdash N:\alpha_i\}}{\uplus_i\Gamma_i\uplus\Gamma\vdash MN:\beta}$$

$$\frac{\Gamma\vdash M:\mathbf{R} \qquad \Delta\vdash N:\alpha}{\Gamma\uplus\Delta\vdash \mathtt{if}(M,N,P):\alpha} \qquad \frac{\Gamma\vdash M:\mathbf{R} \qquad \Delta\vdash P:\alpha}{\Gamma\uplus\Delta\vdash \mathtt{if}(M,N,P):\alpha} \qquad \frac{}{\emptyset\vdash \mathtt{sample}:\mathbf{R}}$$

$$\frac{\Gamma_1\vdash M_1:\mathbf{R} \qquad \cdots \qquad \Gamma_{|f|}\vdash M_{|f|}:\mathbf{R}}{\Gamma_1\uplus\cdots\uplus\Gamma_{|f|}\vdash f(M_1,\cdots,M_{|f|}):\mathbf{R}} \qquad \frac{}{\emptyset\vdash \underline{r}:\mathbf{R}} \qquad \frac{\Gamma\vdash M:\mathbf{R}}{\Gamma\vdash \mathtt{score}(M):\mathbf{R}}$$

Figure 6.5: Non-idempotent Intersection Type System that counts the number of semantic uses of a variable. Note that this system is not syntax-guided due to the two rules for conditionals.

avoiding making any recursive call is at least ϵ:

> **Definition 6.21:** *A fixpoint term $\mu_x^{\varphi}.M$ is called* ϵ-Recursion Avoiding (ϵ-RA) *if for every $r\in\mathbb{R}$, $[\![\mu_x^{\varphi}.M \mid r]\!](0)\geq\epsilon$.*

Now in general ϵ-RA for a positive ϵ does not imply AST, as e.g., $\mu_x^{\varphi}.\underline{0}\oplus_{.1}\big(\varphi(x)+\varphi(x)\big)$ is clearly 0.1-RA but not AST. The concept of RA is nevertheless intriguing as checking RA is easier than extracting the entire counting pattern. We can show that ϵ-RA does imply AST if the ϵ is "big enough", where the size of ϵ must naturally be related to the number of recursive calls. A natural estimate is the *maximal* number of calls possible.

Counting System via Non-idempotent intersection Types To count the number of occurrences we employ a non-idempotent intersection (NII) type system. Intersection types are defined by the mutually recursive grammar $\alpha,\beta\triangleq\mathbf{R}\mid a\to\alpha$ and $a\triangleq[\alpha_1,\cdots,\alpha_n]$ where $[\alpha_1,\cdots,\alpha_n]$ denotes a *multiset*. A typing context Γ is a partial map from variables to intersections. The disjoint union of two contexts Γ,Δ denoted by $\Gamma\uplus\Delta$ is the elementwise disjoint union of multiset. For an overview on non-idempotent intersection types see [Bucciarelli et al. 2017]. Typing judgments are of the from $\Gamma\vdash M:\alpha$ and given by the rules in Fig. 6.5. Due to the non-idempotent nature, for each type derivation we can read of the number of semantic occurrences as the cardinality of the intersection type. E.g. $\mu_x^{\varphi}.M$ is a first-order fixpoint and $\{\varphi:a,x:b\}\vdash M:\mathbf{R}$ we get a path in which φ is used exactly $|a|$-many times. Here $|a|$ denote the cardinality of an intersection type. We can use the type system to define the recursive rank of a fixpoint construct, i.e., the maximal number of possible uses of the recursive abstraction.

> **Definition 6.22:** *Let $\mu_x^{\varphi}.M$ be a first-order fixpoint term. The* recursive rank *of $\mu_x^{\varphi}.M$ is defined as* $\max_{\{\varphi:a,x:b\}\vdash M:\mathbf{R}}|a|$.

Via a subject reduction argument it is easy to see that if m is the recursive rank then $[\![\mu_x^{\varphi}.M \mid r]\!] (n) = \emptyset$ for all $n > m$ and all r, i.e., there are never more than m recursive calls made. We can thus use Thm. 5 combined with Thm. 7 to get an easy corollary:

> **Corollary** *If $\mu_x^{\varphi}.M$ is such that no* `score`*-constructs can fail, has recursive rank m (and thus has no nested fixpoints) and is ϵ-RA for an $\epsilon > 0$ that satisfies $m(1-\epsilon) \leq 1$ then $\mu_x^{\varphi}.M$ is AST on every argument.*

Special Case: Affine Recursion While the emphasis of our system is on non-affine recursion, the reasoning is obviously still valid in the affine case. As in particular every affine program $\mu_x^{\varphi}.M$ has recursive rank at most 1 we get that if it is ϵ-RA for *any* positive ϵ it is already AST. This is a trivial consequence from the previous corollary. This affine result can be seen as analogous to a known fact for imperative languages discussed by McIver and Morgan. They called it the *zero-one-law for termination* (c.f. [McIver and Morgan 2005, Section 2.6]). Informally, it reads that if there is a lower bounded probability of leaving the loop (in our case by ϵ) then the loop is left eventually with certainty. Our framework subsumes this result as a trivial corollary. The idea of lower bounding the probability of termination at each iteration has also been used to construct languages whose programs are guaranteed to be AST. As an example [Lew et al. 2020] used stochastic-while loops, which are loops where at each evaluation of the guard a biased (with a user defined bias in $(0, 1)$) coin can exit the loop immediately. Lew et al. demonstrated that despite this restrictions many programs, including complex neural networks, can be approximate with stochastic while loops. In this spirit, our result can be used to design extensions of SPCF whose programs are guaranteed to be AST by construction. Due to possible non-affineness the bias of leaving a fixpoint must be related to the maximal number of recursive calls as outlined in the corollary. This is a static property that can be checked very efficiently.

Interval-Based Reasoning and Recursion Avoidance The interval-based semantics (Ch. 4) is well suited to derive lower bounds on the measure of various sets of traces. For instance, we get that for a fixed δ the set of programs that is ϵ-RA for an $\epsilon > \delta$ is recursive enumerable (under the same mild assumptions as in Thm. 3). For some classes of primitive functions checking ϵ-RA is even decidable (see the next chapter). This once again shows that interval-based reasoning is useful for applications beyond direct computation of the termination probability as lower bounds on certain sets of traces are often needed to apply various AST methods.

Chapter 7

A Proof System For Non-affine Recursion

The aim of this section is to turn the theoretical method presented in the last chapter into a practical proof system. While the framework gives an effective method to verify AST of many programs, it exhibits one behavior that prohibits its automation. Thm. 7 relies on the counting pattern $\{[]\mu_x^\varphi.M \mid r[]\}_{r\in\mathbb{R}}$ which we extract from the program directly. This family can, however, contain uncountably many different distributions making it impractical for analysis. As a running example in this chapter we consider the following:

$$\mu_x^\varphi.\big(\texttt{score}(\underline{\tfrac{1}{3}}) \oplus \varphi\, x\big) \oplus_{sig(x)} \Big(\texttt{let}\, p = \texttt{sample}\, \texttt{in}\, \underline{0} \oplus_p \big(\varphi\, x \oplus_{sig(x)+p} \varphi(\varphi\, x)\big)\Big)$$

where $sig(x) \triangleq \frac{1}{1+e^{-x}}$ is the sigmoid function which we use to squeeze the real line into $[0,1]$. Checking if this program is AST is not easy as the analysis depends on a complex interplay between the value of x and the probabilistic outcomes especially in the right branch where the sampled value p influences the number of recursive call-sites depending on the value of $sig(x)$. In particular, note that the counting pattern $[]\mu_x^\varphi.M \mid r[]$ for the above term is different for every $r \in \mathbb{R}$, i.e., depending on the input the distribution on the number of calls differs.

We show that we can ignore the actual parameter of the recursive function calls, and instead employ environment strategies that resolve branching, giving us a sound and practical AST method. As a rule of thumb, our system can verify all programs that exhibit an AST counting pattern which is independent of the (exact values of the) actual parameters (of the recursive function in question). Our system can, for example, handle the example above.

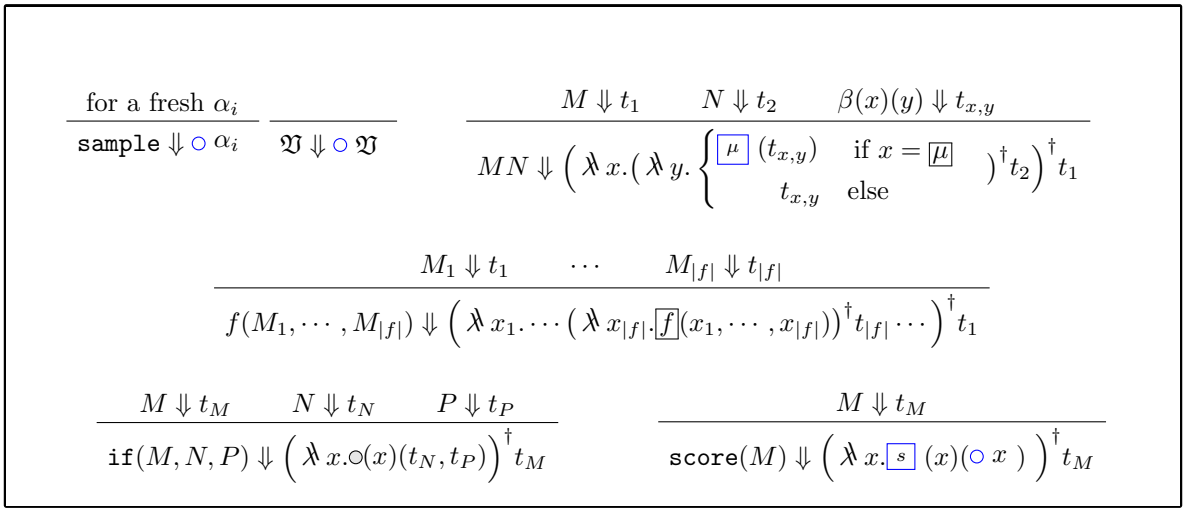


Figure 7.1: Big-step symbolic execution where symbolic terms denote execution trees.

7.1 Big-step Symbolic Execution

In the small-step symbolic execution (see Sec. 4.4), we fixed a conditional oracle $\kappa \in \{\mathbf{L}, \mathbf{R}\}^*$ beforehand and collected constraints, i.e., symbolic inequalities of the from $\mathfrak{V} \bowtie r$ at each conditional or `score`-construct. In our big-step system, we now represent every possible conditional oracle κ in the form of a single binary tree where each branch is annotated with the symbolic value that decides which branch to follow. As we are interested in the number of recursive calls, we mark every such call as a dedicated node.

Symbolic Execution Trees We define (symbolic) executions trees by:

$$\mathit{ETree} \ni \mathfrak{T} \quad \triangleq \quad \circ\, V \ \mid \boxed{\mu}\ (\mathfrak{T}) \ \mid \boxed{s}\ (V)(\mathfrak{T}) \ \mid \circ(V)(\mathfrak{T}_1, \mathfrak{T}_2)$$

where $\mathfrak{V} \in \mathit{Val}$ is a symbolic value[1]. We choose a more space-economical way to present trees. We occasionally depict execution trees as tree of degree 2 where $\circ(V)(\mathfrak{T}_1, \mathfrak{T}_2)$ represents a binary branch. Note that an execution tree condenses all of the information we are interested in. For branching, it records the symbolic value as the condition; for score, it records the symbolic value that is scored, and finally every recursive call is recorded. To construct an execution tree from a program it remains to fold[2] execution trees:

[1] As we have done in Sec. 6.6, we extend symbolic values by a special symbol $\star$.

[2] Tree folding is standard in functional programming. In our case, fold traverses the tree and replaces every leaf with a tree given the folded function.

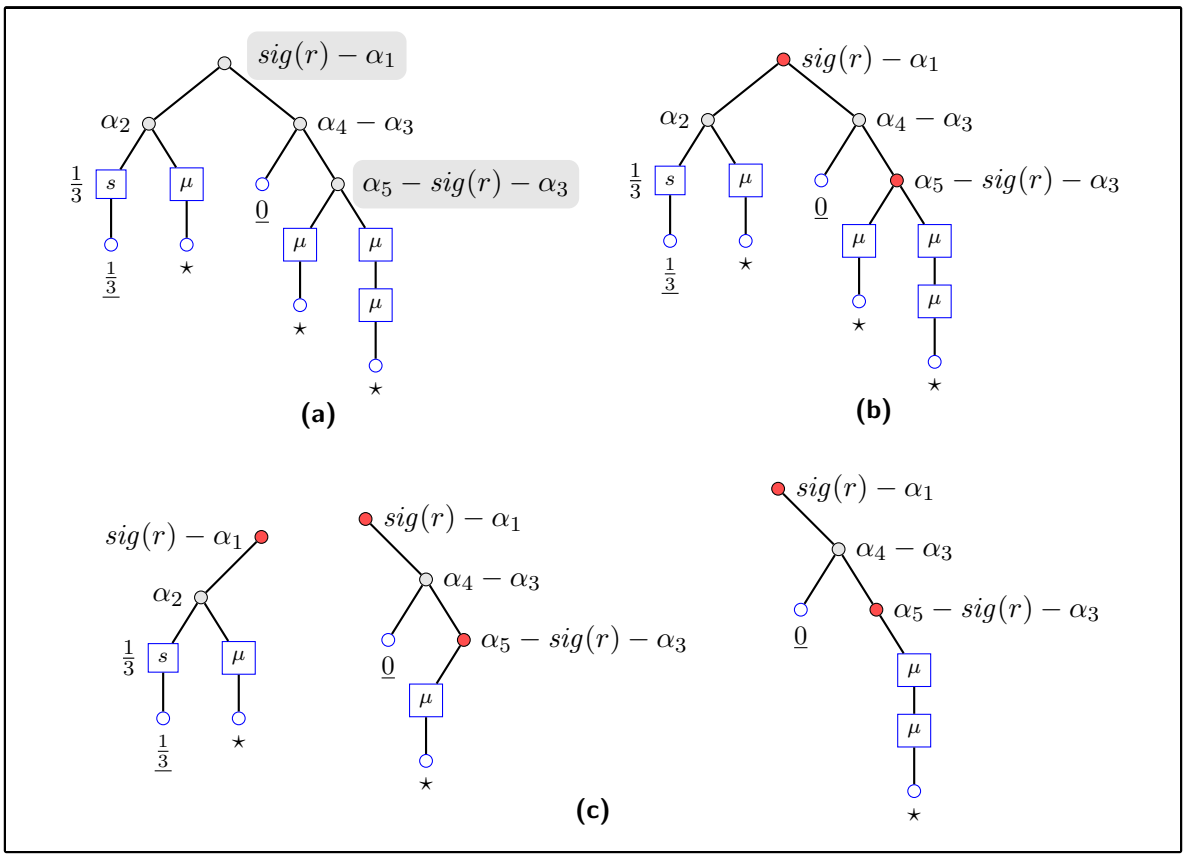


Figure 7.2: Symbolic execution trees for the running example and all possible strategies **(c)**.

Definition 7.1: *Given a function $H : Val \to ETree$ we can define the lifted tree fold $H^\dagger : ETree \to ETree$ by induction as follows:*

$$H^\dagger(\circ\, \mathfrak{V}\ ) = H(\mathfrak{V}) \qquad H^\dagger\left(\boxed{s}\ (\mathfrak{V})(\mathfrak{T})\ \right) = \boxed{s}\ (\mathfrak{V})(H^\dagger\mathfrak{T})$$

$$H^\dagger(\boxed{\mu}\ (\mathfrak{T})\ ) = \boxed{\mu}\ (H^\dagger\mathfrak{T}) \qquad H^\dagger\left(\circ(\mathfrak{V})(\mathfrak{T}_1, \mathfrak{T}_2)\right) = \circ(\mathfrak{V})(H^\dagger\mathfrak{T}_1, H^\dagger\mathfrak{T}_2)$$

We can now define a big-step semantics by giving a symbolic execution tree for each program, denoted $M \Downarrow \mathfrak{T}$. The big-step rules are given in Fig. 7.1 where $\beta(x)(y)$ performs a β-step, i.e., $\beta(\lambda x.M)(V) \triangleq M[V/x]$ and $\beta(\boxed{\mu})(V) \triangleq \star$. Note that this system inherits the structure of a standard big-step semantic (see e.g. [Borgström et al. 2016]). As we execute symbolically and can not resolve branching, we operate on trees and fold each reduction step. For each resolved conditional we introduce a binary branch $\circ(V)(\mathfrak{T}_1, \mathfrak{T}_2)$ at every conditional, a $\boxed{s}\ (V)(\mathfrak{T})$ for every score construct, and a $\boxed{\mu}\ (\mathfrak{T})$ for every recursive call. For every term M there exist a $M \Downarrow \mathfrak{T}$ and $\mathfrak{T}$ is unique up to reordering of the sample variables.

$$\mathsf{Const}(\circ\,\mathfrak{V}\,,\epsilon) \triangleq \mathbb{R}^m_{[0,1]} \qquad \mathsf{Const}(\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2),\boldsymbol{L}\kappa) \triangleq \mathsf{Const}(\mathfrak{T}_1,\kappa)\cap\mathfrak{V}^{-1}(-\infty,0]$$
$$\mathsf{Const}(\boxed{\mu}\,(\mathfrak{T})\,,\kappa) \triangleq \mathsf{Const}(\mathfrak{T},\kappa) \qquad \mathsf{Const}(\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2),\boldsymbol{R}\kappa) \triangleq \mathsf{Const}(\mathfrak{T}_2,\kappa)\cap\mathfrak{V}^{-1}(0,\infty)$$
$$\mathsf{Const}(\boxed{s}\,(\mathfrak{V})(\mathfrak{T})\,,\kappa) \triangleq \mathsf{Const}(\mathfrak{T},\kappa)\cap\mathfrak{V}^{-1}[0,\infty)$$

Figure 7.3: Inductive definition of $\mathsf{Const}(\mathfrak{T},\kappa)$ for $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T})$.

Example 7.2: As an example consider our running example for this chapter. The term we analyse in our big-step system is $\mathbf{body}_{\mu^{\varphi}_{x}.M}(r)$ which in our case is:

$$\left(\mathtt{score}(\underline{\tfrac{1}{3}}) \oplus \boxed{\mu}\underline{r}\right) \oplus_{sig(\underline{r})} \left(\mathtt{let}\, p = \mathtt{sample}\,\mathtt{in}\,\underline{0} \oplus_p \left(\boxed{\mu}\underline{r} \oplus_{sig(\underline{r})+p} \boxed{\mu}(\boxed{\mu}\underline{r})\right)\right)$$

The tree $\mathfrak{T}$ with $\mathbf{body}_{\mu^{\varphi}_{x}.M}(r) \Downarrow \mathfrak{T}$ is depicted in Fig. 7.2a. The interested reader is advised to check the construction herself.

Correspondence We now show a correspondence between the big-step symbolic execution and the reduction relation $\xrightarrow{\star}$, similar to the correspondence formalized in Prop. 4.9. For Prop. 4.9 we fixed a conditional oracle κ and extracted a set of constraints of the from $\mathfrak{V} \bowtie r$. Our big-step execution tree summarizes all such paths κ and every path that leads to a node naturally denotes a set of constraints via the node it traverses. Assume that $\mathbf{body}_{\mu^{\varphi}_{x}.M}(r) \Downarrow \mathfrak{T}$ and all symbolic values within $\mathfrak{T}$ are within $\{\alpha_0,\cdots,\alpha_{m-1}\}$.

Recall conditional oracles (in the following called paths) are sequences $\kappa \in \{\boldsymbol{L},\boldsymbol{R}\}^*$. We call a paths terminating if, by following its branching, a leaf is reached. In the example tree in Fig. 7.2a, possible terminating paths include $\boldsymbol{LR}, \boldsymbol{RRR}, \cdots$. With $\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T})$ we denote the set of terminating paths. For any tree $\mathfrak{T}$ and terminating path $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T})$ we count the numbers of recursive calls on that path, i.e., the number of times that a fixpoint node ($\boxed{\mu}\,(\cdot)\,$) is traversed. We denote this number with $|\boxed{\mu}|(\mathfrak{T},\kappa) \in \mathbb{N}$. For a set C of natural numbers we abbreviate $\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},C) \triangleq \{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}) \mid |\boxed{\mu}|(\mathfrak{T},\kappa) \in C\}$. Now each path $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T})$ denotes a measurable subset of $\mathbb{R}^m_{[0,1]}$ in the natural way. We denote this set with $\mathsf{Const}(\mathfrak{T},\kappa) \subseteq \mathbb{R}^m_{[0,1]}$ and it is defined by induction in Fig. 7.3. For every $\mathfrak{T}$ and κ, $\mathsf{Const}(\mathfrak{T},\kappa) \subseteq \mathbb{R}^m_{[0,1]}$ denotes the set of assignments for $\alpha_0,\cdots,\alpha_{m-1}$ such that the branching and `score`-constructs are evaluated according to κ. It is easy to see that $\mathsf{Const}(\mathfrak{T},\kappa)$ is measurable. We define $\mathbb{P}(\mathfrak{T},\kappa) \triangleq \lambda_m\big(\mathsf{Const}(\mathfrak{T},\kappa)\big)$. We can conclude the following proposition that can be proved similar to Prop. 4.9:

Proposition 7.3 : *If $r \in \mathbb{R}$ and $\boldsymbol{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}$ and $n \in \mathbb{N}$ then,*

$$\sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},\{n\})} \mathbb{P}(\mathfrak{T},\kappa) = \mu_{\mathbb{S}}\left(\mathbb{T}^{\star}_{\boldsymbol{body}_{\mu_x^{\varphi}.M}(r);n}\right)$$

This proposition states, that if we are interested in the number of recursive calls, say n. Then the set of paths $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},\{n\})$ are all paths on which n calls are made and the constraints along those paths characterize exactly the traces on which n calls are made in the $\xrightarrow{\star}$ semantics (Fig. 6.2). Note that not all traces in $\mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^{\varphi}.M}(r);n}$ are of length at most m.

Remark: The symbolic execution tree thus offers an alternative way to compute the counting pattern of $\mu_x^{\varphi}.M$. Namely, for every $r \in \mathbb{R}$, if $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}$ then we have $[\![\mu_x^{\varphi}.M \mid r]\!] (n) = \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},\{n\})} \mathbb{P}(\mathfrak{T},\kappa)$. Depending on the set of primitive functions this sum can be computed effectively.

7.2 Replacing Probabilistic Branching by Nondeterminisms

So far we have seen, that symbolic execution can be used to compute the counting pattern. This has, however, not solved the fundamental issue that the counting pattern relies on the concrete argument provided to the fixpoint. This section is devoted to change that. We show that we can replace each probabilistic branching that depends on the concrete argument by a non-deterministic choice (by using environment strategies) and thus lose the dependency on the concrete argument. The approximation relies on the following observation:

Lemma 7.4 : *Let $p, q : \mathbb{N} \to \mathbb{R}_{[0,1]}$ be finite subprobability mass functions. If $\overline{p}$ is an AST step distribution and for all n: $\sum_{m \leq n} p(m) \leq \sum_{m \leq n} q(m)$, then $\overline{q}$ is an AST step distribution.*

Proof The proof hinges on the equivalence proved in Thm. 5. In particular, the combinatorial equation $\sum_{i \in \mathbb{N}} i \cdot p(i) = \sum_{i \in \mathbb{N}} \sum_{j > i} p(j)$ allows us to show that the expectation of q is less or equal to the expectation of p (as required by Thm. 5). A full proof can be found in the appendix: D. ■

We try to find a function $p : \mathbb{N} \to \mathbb{R}_{[0,1]}$ such that for every r, $\sum_{m \leq n} p(m) \leq \sum_{m \leq n} [\![\mu_x^{\varphi}.M \mid r]\!] (m)$ for every n. If $\overline{p}$ is then AST then each individual step distribution is AST. It is easy to extend the above proof and show that in this case the family $\{\overline{[\![\mu_x^{\varphi}.M \mid r]\!]}\}_{r \in \mathbb{R}}$ is uniform AST.

Colouring Problematic Branching We can observe that if $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}_r$ and $\mathbf{body}_{\mu_x^{\varphi}.M}(r') \Downarrow \mathfrak{T}_{r'}$ for every $r, r' \in \mathbb{R}$ then $\mathfrak{T}_r$ and $\mathfrak{T}_{r'}$ agree structurally and in particular share the same paths. They can, however, differ in some of the symbolic values at nodes.

$$\mathsf{Const}^\star(\circ\,\mathfrak{V}\,,\epsilon) \triangleq \mathbb{R}^m_{[0,1]} \qquad \mathsf{Const}^\star(\bullet(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2), \boldsymbol{L}\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T}_1,\kappa)$$
$$\mathsf{Const}^\star(\boxed{\mu}\,(\mathfrak{T})\,,\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T},\kappa) \qquad \mathsf{Const}^\star(\bullet(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2), \boldsymbol{R}\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T}_2,\kappa)$$
$$\mathsf{Const}^\star(\boxed{s}\,(\mathfrak{V})(\mathfrak{T})\,,\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T},\kappa) \cap \mathfrak{V}^{-1}[0,\infty)$$
$$\mathsf{Const}^\star(\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2), \boldsymbol{L}\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T}_1,\kappa) \cap \mathfrak{V}^{-1}(-\infty,0]$$
$$\mathsf{Const}^\star(\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2), \boldsymbol{R}\kappa) \triangleq \mathsf{Const}^\star(\mathfrak{T}_2,\kappa) \cap \mathfrak{V}^{-1}(0,\infty)$$

Figure 7.4: Inductive definition of $\mathsf{Const}^\star(\mathfrak{T},\kappa)$ for $\kappa \in \mathsf{Paths}^\blacktriangledown(\mathfrak{T})$.

In the example in Fig. 7.2a, we have surrounded those values by grey boxes. In the computation of $\mathbb{P}(\mathfrak{T}_r,\kappa)$, used e.g. in Prop. 7.3 the values that are different are obviously important, i.e., if in general $\mathbb{P}(\mathfrak{T}_r,\kappa) \neq \mathbb{P}(\mathfrak{T}_{r'},\kappa)$.

We begin by identifying the branching nodes in a tree that do depend on the concrete value r. We extend executions trees with a new binary node $\bullet(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$ that otherwise behaves identical to $\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$. In particular, Const extends naturally to such nodes and behaves like $\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$. We now replace every node of the from $\circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$ whose symbolic value depends on r with $\bullet(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$. For our example in Fig. 7.2 we have given the coloured tree in Fig. 7.2b[3].

The construction of Const does not treat red nodes any different than normal branching. For every $\kappa \in \mathsf{Paths}^\blacktriangledown(\mathfrak{T})$ we now define a set $\mathsf{Const}^\star(\mathfrak{T},\kappa) \subseteq \mathbb{R}^m_{[0,1]}$ inductively in Fig. 7.4 which is similar to $\mathsf{Const}(\mathfrak{T},\kappa)$ but ignores all red nodes. It is easy to see that $\mathsf{Const}(\mathfrak{T},\kappa) \subseteq \mathsf{Const}^\star(\mathfrak{T},\kappa)$. Analogously to before we define $\mathbb{P}^\star(\mathfrak{T},\kappa) \triangleq \lambda_m\big(\mathsf{Const}^\star(\mathfrak{T},\kappa)\big)$.

> **Remark:** By definition, $\mathsf{Const}^\star$ does not depend on the symbolic values at nodes coloured in red. In particular, if $\mathbf{body}_{\mu^\varphi_x.M}(r) \Downarrow \mathfrak{T}_r$ and $\mathbf{body}_{\mu^\varphi_x.M}(r') \Downarrow \mathfrak{T}_{r'}$ and $\mathfrak{T}_r$ is coloured such that it agrees with $\mathfrak{T}_{r'}$ on all non-red nodes (see e.g. Fig. 7.2) then $\mathsf{Const}^\star(\mathfrak{T}_r,\kappa) = \mathsf{Const}^\star(\mathfrak{T}_{r'},\kappa)$ and in particular for an $C \subseteq \mathbb{N}$: $\sum_{\kappa\in\mathsf{Paths}^\blacktriangledown(\mathfrak{T}_r,C)} \mathbb{P}^\star(\mathfrak{T}_r,\kappa) = \sum_{\kappa\in\mathsf{Paths}^\blacktriangledown(\mathfrak{T}_{r'},C)} \mathbb{P}^\star(\mathfrak{T}_{r'},\kappa)$, i.e., $\mathbb{P}^\star$ is fully independent of the concrete choice of r.

As the value of $\mathbb{P}^\star$ does not depend on the concrete r we can see this as a first step towards a system where we do not need to consider every possible input. As $\mathsf{Const}(\mathfrak{T},\kappa) \subseteq \mathsf{Const}^\star(\mathfrak{T},\kappa)$ we get $\sum_{m\leq n} [\![\mu^\varphi_x.M \mid r]\!](m) = \sum_{\kappa\in\mathsf{Paths}^\blacktriangledown(\mathfrak{T}_r,\{0,\cdots,n\})} \mathbb{P}(\mathfrak{T}_r,\kappa) \leq \sum_{\kappa\in\mathsf{Paths}^\blacktriangledown(\mathfrak{T}_r,\{0,\cdots,n\})} \mathbb{P}^\star(\mathfrak{T}_r,\kappa)$.

[3] To formally identify the nodes that should be coloured, we extend symbolic values by yet another symbol $\boxed{\star}$. We can then compute $\mathbf{body}_{\mu^\varphi_x.M}(\boxed{\star}) \Downarrow \mathfrak{T}_\star$. Now all nodes labelled with a value containing $\boxed{\star}$ are coloured red. We can observe that if we replace each $\boxed{\star}$ symbol with $\underline{r}$ (denoted by $\mathfrak{T}_\star[\underline{r}/\boxed{\star}]$) we obtain the execution tree for $\mathbf{body}_{\mu^\varphi_x.M}(r)$, i.e., $\mathbf{body}_{\mu^\varphi_x.M}(r) \Downarrow \mathfrak{T}_\star[\underline{r}/\boxed{\star}]$.

While the right hand side is independent of the concrete r this inequality does *not* allow us Lem. 7.4 as we require a *lower* bound on $\sum_{m \leq n} [\![\mu_x^{\varphi}.M \mid r]\!] (m)$.

Strategies We now show that we can replace each red node with a non-deterministic choice. We introduce strategies, which can restrict execution trees by choosing one branch for each red node. We define symbolic strategy trees by

$$\mathfrak{S} \triangleq \circ\, V \mid \boxed{\mu}\, (\mathfrak{S}) \mid \boxed{s}\, (V)(\mathfrak{S}) \mid \circ(V)(\mathfrak{S}_1, \mathfrak{S}_2) \mid \bullet(V)(\mathfrak{S}, \times) \mid \bullet(V)(\times, \mathfrak{S})$$

A strategy $\mathfrak{S}$ is compatible with an execution tree $\mathfrak{T}$ (written $\mathfrak{S} \prec \mathfrak{T}$) if it matches the structure. For each red node, a strategy can decide which branch to follow and neglect the other. The concept of path and the definitions of $\mathsf{Const}^\star$ and $\mathbb{P}^\star$ extends naturally to strategies. For example, *all* strategies for the tree in Fig. 7.2b are listed in Fig. 7.2c. If we assume $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}$ and all nodes that depend on r are coloured red, we can see that $\mathbb{P}^\star(\mathfrak{S}, \kappa)$ for a strategy $\mathfrak{S} \prec \mathfrak{T}$ does not depend on the concrete choice of r.

We call an execution tree $\mathfrak{T}$ *sufficiently independent* just if every sample variable that occurs in a red node does not occur in the subtree rooted at that node. The tree in Fig. 7.2b is sufficiently independent. The next theorem now shows, that despite the fact that $\mathsf{Const}(\mathfrak{T}, \kappa) \subseteq \mathsf{Const}^\star(\mathfrak{T}, \kappa)$ and therefore $\mathbb{P}(\mathfrak{T}, \kappa) \leq \mathbb{P}^\star(\mathfrak{T}, \kappa)$ we can always find a strategy that gives lower bounds on $\sum_{\kappa \in \mathsf{Paths}^\blacktriangledown(\mathfrak{T}, C)} \mathbb{P}(\mathfrak{T}, \kappa)$. Note that the next theorem does work on arbitrary execution trees.

Proposition 7.5 (Strategies For Lower Bounds): *If $\mathfrak{T}$ is sufficiently independent and $C \subseteq \mathbb{N}$ then there exists a strategy $\mathfrak{S} \prec \mathfrak{T}$, s.t.,*

$$\sum_{\kappa \in \mathsf{Paths}^\blacktriangledown(\mathfrak{S}, C)} \mathbb{P}^\star(\mathfrak{S}, \kappa) \leq \sum_{\kappa \in \mathsf{Paths}^\blacktriangledown(\mathfrak{T}, C)} \mathbb{P}(\mathfrak{T}, \kappa)$$

Proof For a measurable set $A \subseteq \mathbb{R}^m_{[0,1]}$ we define $\mathbb{P}_A(\mathfrak{T}, \kappa) \triangleq \lambda_m(\mathsf{Const}(\mathfrak{T}, \kappa) \cap A)$ and similarly for $\mathbb{P}^\star_A$. Note that $\mathbb{P}_{\mathbb{R}^m_{[0,1]}}(\mathfrak{T}, p) = \mathbb{P}(\mathfrak{T}, p)$. We generalize the statement by using $\mathbb{P}^\star_A$ and $\mathbb{P}_A$. The general statement can be proved by induction on $\mathfrak{T}$ with A universally quantified. The crucial observation is that sufficient independence of $\mathfrak{T}$ gurantess that the symbolic value in each red node is conditionally independent from its subtree. A full proof can be found in the appendix: D ■

7.3 The Proof System

We can turn the observation above into a proof system. We first compute $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}_r$ for any r of our choosing and colour all nodes that depend on r in red. We assume that $\mathfrak{T}_r$

is sufficiently independent. We define a distribution $\mathbb{P}_{\text{cum}}$ by the following:

$$\mathbb{P}_{\text{cum}}(n) \triangleq \min_{\mathfrak{S} \prec \mathfrak{T}_r} \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}, \{0,\cdots,n\})} \mathbb{P}^{\star}(\mathfrak{S}, \kappa)$$

The value $\mathbb{P}_{\text{cum}}(n)$ is the probability of making *at most* n calls taken as the minimum over all of the (finitely many) strategies for $\mathfrak{T}_r$. Note that $\mathbb{P}_{\text{cum}}$ does not depend on our concrete choice of r. We define $\mathbb{P}_{\text{approx}}(0) \triangleq \mathbb{P}_{\text{approx}}(0)$ and $\mathbb{P}_{\text{approx}}(n) \triangleq \mathbb{P}_{\text{cum}}(n) - \mathbb{P}_{\text{cum}}(n-1)$ for $n \geq 1$[4].

Example 7.6: Consider all strategies for the running example listed in Fig. 7.2c. We can compute $\mathbb{P}_{\text{cum}}(0) = \mathbb{P}_{\text{cum}}(1) = \frac{1}{2}$, $\mathbb{P}_{\text{cum}}(n) = 1$ for $n \geq 2$. The reader is strongly advised to reproduce this. Then $\mathbb{P}_{\text{approx}}(0) = \mathbb{P}_{\text{approx}}(2) = \frac{1}{2}$ and $\mathbb{P}_{\text{approx}}(n) = 0$ for all other n.

Theorem 8 (Sound Proof System): *If $\overline{\mathbb{P}_{\text{approx}}}$ is an AST step distribution, then $\mu_x^{\varphi}.M$ is AST on every argument.*

Proof Choose any $r \in \mathbb{R}$ and any $n \in \mathbb{N}$. Let $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}_r$. By Prop. 7.5 there exists a strategy $\mathfrak{S} \prec \mathfrak{T}_r$ such that $\sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}, \{0,\cdots,n\})} \mathbb{P}^{\star}(\mathfrak{S}, \kappa) \leq \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_r, \{0,\cdots,n\})} \mathbb{P}(\mathfrak{T}_r, \kappa)$. Note that the argument used in the definition of $\mathbb{P}_{\text{approx}}$ is completely arbitrary and does not matter. So, for convenience, we just assume it was r.

$$\begin{aligned}
\sum_{m \leq n} \mathbb{P}_{\text{approx}}(m) = \mathbb{P}_{\text{cum}}(n) &= \min_{\mathfrak{S}' \prec \mathfrak{T}_r} \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}', \{0,\cdots,n\})} \mathbb{P}^{\star}(\mathfrak{S}', \kappa) \\
&\leq \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}, \{0,\cdots,n\})} \mathbb{P}^{\star}(\mathfrak{S}, \kappa) \\
&\overset{(1)}{\leq} \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_r, \{0,\cdots,n\})} \mathbb{P}(\mathfrak{T}_r, \kappa) \\
&\overset{(2)}{=} \sum_{m \leq n} \mu_{\mathbb{S}}\left(\mathbb{T}^{\star}_{\mathbf{body}_{\mu_x^{\varphi}.M}(r)}; m\right) \overset{(3)}{=} \sum_{m \leq n} [\![\mu_x^{\varphi}.M \mid r]\!] (m)
\end{aligned}$$

where **(1)** follows from the choice of $\mathfrak{S}$, **(2)** follows from Prop. 7.3 and **(3)** by definition of $[\![\mu_x^{\varphi}.M \mid r]\!]$. Now by assumption $\overline{\mathbb{P}_{\text{approx}}}$ is an AST step distribution, so by Lem. 7.4 we get that $\{\overline{[\![\mu_x^{\varphi}.M \mid r]\!]}\}_{r \in \mathbb{R}}$ is uniform AST. By the theoretical insight proved in the last chapter (Thm. 7) this implies that $\mu_x^{\varphi}.M$ is AST on every input. ■

Using Thm. 8 and the values computed in Ex. 7.6 allow us to deduce that our running example is AST on every argument. To the best of our knowledge, no existing method can show the running example AST due to the complex interplay of non-affine recursion while our approach

[4]The observant reader might ask why we have not defined $\mathbb{P}_{\text{approx}}(n)$ directly by $\mathbb{P}_{\text{approx}}(n) = \min_{\mathfrak{S} \prec \mathfrak{T}_r} \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}, \{n\})} \mathbb{P}^{\star}(\mathfrak{S}, \kappa)$. If we were to use this definition, $\mathbb{P}_{\text{approx}}$ may not be proper mass functions. As an example consider Example 7.6 with this alternative definition.

can do so and is furthermore easy to automate.

In this chapter, we showed how to compute the counting pattern via symbolic execution and presented a technique that requires us to only check a single tree (by introducing strategies). Our algorithm computes $\mathbf{body}_{\mu_x^{\varphi}.M}(r) \Downarrow \mathfrak{T}_r$ for any r of our choosing, checks sufficient independence and afterwards extracts $\mathbb{P}_{\text{approx}}$. We can then check AST for this distribution in linear time by using the fundamental result from Thm. 5. Our system is sound and can verify many interesting examples as we saw with the running example. As a good rule of thumb, our system works if the counting pattern of a program is AST *no matter* how branching, where the symbolic value depend on the concrete argument, is resolved. The side condition of sufficiently independence means that every probabilistic outcome used together with the argument may not be used for subsequent branching. In our experience, most interesting programs yield sufficiently independent execution trees. Note that our method can only prove that a program is AST but never refute it.

Prototype Implementation The proof system as presented is easy to implement as all operations are based on functional operations on a tree. We have designed a prototype implementation in F#[5]. The challenging part for an implementation is to compute probability associated with a path, which depends on the set of primitive functions. A simple class of primitive functions is $\{\_ + \_, c \cdot \_\}$, i.e., addition and multiplication by a *fixed* constant $c \in \mathbb{R}$ (subtraction can easily be encoded). Note that checking AST with these primitive functions is already Π_2^0-hard [Kaminski and Katoen 2015]. If the set of primitive functions is restricted as such, every symbolic value $\mathfrak{V}$ is a *linear* function in the sample variables. The probability of a path is thus the Lebesgue measure/volume of a convex polytopes (a subset of $\mathbb{R}^d$ of the from $\{\vec{x} \mid A\vec{x} \leq b\}$). Computing volumes of convex polytopes is a long studied problem. There does exist analytical expression by recursion on the dimension [Lasserre 1983] which is well suited for automation. The computation has been implemented in the tool Vinci [Büeler et al. 2000]. Our prototype implementation uses Vinci for volume computations and checks AST of the obtained step distribution via Thm. 5. Due to the low dimension of the polytopes, our tool is very efficient and provides an answer in less than a second.

[5] We only report on our implementation to emphasise that, unlike most existing approaches, our system is easy to implement. The implementation itself should not be considered part of this project, which is also why we did not provide any additional material such as source code.

Chapter 8

Conclusion

In this work, we have provided both theoretical insights into the complexity of AST and presented a sound method to reason about AST for functional programs with non-affine (first-order) recursion.

Our novel interval-based semantics enables sound and complete reasoning while only needing to consider countably many traces. As an immediate consequence we obtained a precise characterisation of the AST and PAST decision problem in the arithmetic hierarchy. Our results show, that despite the intrinsic continuous nature of SPCF reasoning about termination is (under mild restrictions) not harder than in the discrete case. Internal-based reasoning is also applicable for imperative continuous languages (as e.g. used in [Fioriti and Hermanns 2015]) establishing identical complexity results. Remarkably, our intersection type system shows, that despite its continuous nature, a compositional system can characterises AST as the least upper bound of finite derivations. Our type system also provides a direct generalization of the system by Breuvart and Lago to reason about PAST.

In the second part, we conducted a first study of termination in a setting with non-affine recursion. Many of the widely used languages for probabilistic programming are functional, which is in stark contrast to the fact that most proof systems focus on imperative languages. For many applications, the ability to write programs with non-affine recursion, is a distinct and useful feature of functional languages compared to typical loops in an imperative language. While termination as a *qualitative* property does not depend on the number of recursive calls (and thus non-affinity), we showed that for the quantitative notion of almost-sure termination the number of calls matters and is intricate. Our counting-based framework is a first step towards a better understanding of the quantitative nature of non-affine recursion. As our proof system shows, our approach is easy to implement and can be used for actual AST

analysis. We can verify a broad class of programs which can not be handled by existing approaches. In the special case of possible recursion avoidance (Sec. 6.7), we have outlined how the interval-based reasoning from the first part can be used to obtain lower bounds on the recursion-avoidance probability. This again shows that the theoretical insights from the first part are useful for practical AST proof systems.

8.1 Future Work

Our results give a comprehensive study of various aspects of AST. In this last section, we give some directions for future work.

Relaxing Interval Preservation and Interval Separability Our interval-based semantics hinges on the assumption that the primitive functions are interval preserving and interval separable. While we have seen that this covers most functions a programmer would use, we want to extend this class to as large a family of functions as possible. Interval preservation, i.e., the requirement that the image of a box is an interval, was crucial to allow interval-based reasoning for primitive functions. We can generalize interval preservation and require that the image of every box is a *finite* union of intervals. This would add support for e.g. step-functions such as $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$. To encompass such functions we need to relax the notion of interval terms and allow finite (non-empty) sets of intervals as numerals. Our soundness, completeness and complexity results extend easily. As discussed at the end of Ch. 4 it is also interesting to investigate representation of infinite sets of real numbers different from intervals and therefore extend to a broader class of primitive functions.

Verification in a Continuous Language Throughout this work we focussed on a particular kind of correctness property: almost-sure termination. Especially in the area of probabilistic algorithm, one is usually interested in more expressive correctness properties, say as an example "the return value is positive a.s.". The interval-based semantics may provide a sound and complete system to answer such questions[1] and is particularly well suited for refutation by e.g. providing a trace with non-zero probability on which a safety-property is violated. An interesting direction for future work is to study the (continuous) probabilistic verification problem extensively and identify a suitable class of specifications and the position of their decision problems in the arithmetic hierarchy (using e.g. the interval-based semantics).

[1] Checking most probabilistic safety-specifications (such as a.s. termination with a positive value) inherits the Π^0_2-hardness of AST.

Quantitative Notion of Termination In this work, we mostly focussed on almost-sure termination, i.e., termination with *maximal* probability. An equally interesting question is to check arbitrary lower bounds on the probability of termination. As an example consider the term $\mu_x^\varphi.\underline{0} \oplus \varphi(\varphi(\varphi(x)))$ which is known not be AST. It turns out that this program terminates with probability $\frac{\sqrt{5}-1}{2}$. In our counting-based framework, we only focus on AST of random walks, but could also analyse lower bounds. E.g. the walk generated by the step distribution $\frac{1}{2}\delta_{-1} + \frac{1}{2}\delta_2$ started in 1 terminates with probability $\frac{\sqrt{5}-1}{2}$. We can extend our framework from the second part to prove such non-maximal lower bounds.

Beyond First-Order Fixpoints We have focused on first-order recursion to demonstrate the counting-based approach. It is interesting to investigate to what extend counting-based methods are applicable to higher-order fixpoints. In our system, we have focussed on showing that a first-order fixpoint terminates on every input. Due to the first-order nature such a (closed fixpoint) can be freely embedded into any non-recursive program and the program remains AST[2]. For higher-order fixpoints, AST on every input is often times not strong enough of a property. As an example consider the term $\mu_x^\varphi.M \triangleq \mu_x^\varphi.(\lambda y.\varphi\, x\, y)$. Then $(\mu_x^\varphi.M)\underline{0}$ is clearly AST as it evaluates to the value $\lambda y.(\mu_x^\varphi.M)\underline{0}\, y$ without any recursive call. But $(\mu_x^\varphi.M)\underline{0}\,\underline{1}$ diverges deterministically.

Pushdown System We have informally argued that first-order SPCF programs denote (probabilistic) pushdown processes. Following this observation it is interesting to, on the one hand, investigate properties of probabilistic pushdown systems in the continuous setting (in the style of [Brázdil et al. 2013]) and, on the other hand, study the correspondence between higher-order SPCF and pushdown systems. One idealised form to study higher-order recursion, are higher-order recursion schemes. Recently, [Kobayashi et al. 2019] gave a first attempt to formalize (discrete) higher-order recursion in a probabilistic setting as probabilistic higher-order recursion schemes. In a non-probabilistic setting, it is known that order-n higher order recursion schemes (or equivalently order-n simply-typed λ-terms with general recursion) correspond to order-n collapsible pushdown automata[3] [Hague et al. 2017] (for expressing trees). In this work, we argued that first-order SPCF denotes a (first-order) pushdown process. We conjecture that this correspondence extends to higher orders, i.e., that SPCF terms with higher-order recursion correspond to higher-order (collapsible) probabilistic pushdown systems, with identical termination behavior. This is particularly interesting for

[2] As an example, if we know that $\mu_x^\varphi.M$ is AST on every input we can also deduce that $(\mu_x^\varphi.M)\underline{0} + (\mu_x^\varphi.M)\underline{1}$ is AST.

[3] A collapsible pushdown automaton operates on order-n stacks (i.e., stacks of stacks of stacks etc. n times, where each element in the stacks has a link to a stack of order less than n) with a special collapse operation that reduces a stack to the prefix linked to by its first element.

continuous languages and the resulting continuous pushdown systems.

Functional vs. Imperative Languages Most of the existing approaches on AST methods (mostly based on ranking functions) work with imperative languages. A general direction of research should be to check if and to what extend AST methods for imperative languages can be used in a functional setting, a question that is still wide open. A particularly promising idea is to use ranking functions directly on a pushdown process. This would allow us to reuse some of the advanced results in the area of ranking functions (hitherto almost exclusively stated for an imperative language) to the domain of functional programs. Especially the powerful rule from [McIver et al. 2018] would allow for the verification of many interesting functional programs. We believe, the method presented in the first part of this work to be useful for most language with continuous distributions, as lower bounds on certain events are required for most methods (both in functional and imperative languages). The reasoning about non-affine recursion is also applicable for recursive imperative languages (such as the one studied in [Olmedo et al. 2016]).

Bibliography

Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 34:1–34:32. https://doi.org/10.1145/3158122

Robert B. Ash and Catherine A. Doleans-Dade. 1999. *Probability and Measure Theory* (second ed.). Harcourt/Academic Press.

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking.* MIT Press.

Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics.* Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016.* ACM, 33–46. https://doi.org/10.1145/2951913.2951942

Tomás Brázdil, Václav Brozek, Kousha Etessami, Antonín Kucera, and Dominik Wojtczak. 2010. One-Counter Markov Decision Processes. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010.* SIAM, 863–874. https://doi.org/10.1137/1.9781611973075.70

Tomás Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kucera. 2013. Analyzing probabilistic pushdown automata. *Formal Methods Syst. Des.* 43, 2 (2013), 124–163. https://doi.org/10.1007/s10703-012-0166-0

Flavien Breuvart and Ugo Dal Lago. 2018. On Intersection Types and Probabilistic Lambda Calculi. In *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018.* ACM, 8:1–8:13. https://doi.org/10.1145/3236950.3236968

Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. 2017. Non-idempotent intersection types for the Lambda-Calculus. *Log. J. IGPL* 25, 4 (2017), 431–464. https://doi.org/10.1093/jigpal/jzx018

Benno Büeler, Andreas Enge, and Komei Fukuda. 2000. *Exact Volume Computation for Polytopes: A Practical Study.* Birkhäuser Basel, Basel, 131–154. https://doi.org/10.1007/978-3-0348-8438-9_6

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, Vol. 8044)*. Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. https://doi.org/10.1145/3174800

Jianhui Chen and Fei He. 2020. Proving almost-sure termination by omega-regular decomposition. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*. ACM, 869–882. https://doi.org/10.1145/3385412.3386002

Daniel de Carvalho. 2018. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.* 28, 7 (2018), 1169–1203. https://doi.org/10.1017/S0960129516000396

Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. ACM, 309–320. https://doi.org/10.1145/2535838.2535865

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*. ACM, 489–501. https://doi.org/10.1145/2676726.2677001

Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11388)*. Springer, 468–490. https://doi.org/10.1007/978-3-030-11245-5_22

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. AUAI Press, 220–229. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1346&proceeding_id=24

Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*. ACM, 167–181. https://doi.org/10.1145/2593882.2593900

Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. 2017. Collapsible Pushdown Automata and Recursion Schemes. *ACM Trans. Comput. Log.* 18, 3 (2017), 25:1–25:42. https://doi.org/10.1145/3091122

Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost-Sure Termi-

nation. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9234)*. Springer, 307–318. https://doi.org/10.1007/978-3-662-48057-1_24

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (2018), 30:1–30:68. https://doi.org/10.1145/3208102

S. C. Kleene. 1955. Hierarchies of number-theoretic predicates. *Bull. Amer. Math. Soc.* 61 (05 1955), 193–213. https://doi.org/10.1090/S0002-9904-1955-09896-3

Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. 2019. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 1–14. https://doi.org/10.1109/LICS.2019.8785679

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

Ugo Dal Lago and Charles Grellois. 2019. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Trans. Program. Lang. Syst.* 41, 2 (2019), 10:1–10:65. https://doi.org/10.1145/3293605

Jean B Lasserre. 1983. An analytical expression and an algorithm for the volume of a convex polyhedron in $\mathbb{R}^n$. *Journal of optimization theory and applications* 39, 3 (1983), 363–377.

Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. 2018. Reparameterization Gradient for Non-differentiable Models. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 5558–5568. https://proceedings.neurips.cc/paper/2018/hash/b096577e264d1ebd6b41041f392eec23-Abstract.html

Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2020. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.* 4, POPL (2020), 19:1–19:32. https://doi.org/10.1145/3371087

Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. 2021. Densities of Almost Surely Terminating Probabilistic Programs are Differentiable Almost Everywhere. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*. Springer, 432–461. https://doi.org/10.1007/978-3-030-72019-3_16

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

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

Sean Meyn and Richard L. Tweedie. 2009. *Markov Chains and Stochastic Stability* (2nd ed.). Cambridge University Press.

Akihiko Nishimura, David B Dunson, and Jianfeng Lu. 2020. Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods. *Biometrika* 107, 2 (2020), 365–380.

Jonathan Novak. 2014. Pólya's Random Walk Theorem. *Am. Math. Mon.* 121, 8 (2014), 711–716. http://www.jstor.org/stable/10.4169/amer.math.monthly.121.08.711

Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. ACM, 672–681. https://doi.org/10.1145/2933575.2935317

Gordon D. Plotkin. 1977. LCF Considered as a Programming Language. *Theor. Comput. Sci.* 5, 3 (1977), 223–255. https://doi.org/10.1016/0304-3975(77)90044-5

Tom Rainforth. 2017. *Automating Inference, Learning, and Design Using Probabilistic Programming.* Ph.D. Dissertation. University of Oxford.

Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2018. Denotational validation of higher-order Bayesian inference. *Proc. ACM Program. Lang.* 2, POPL (2018), 60:1–60:29. https://doi.org/10.1145/3158148

David Tolpin, Jan-Willem van de Meent, and Frank D. Wood. 2015. Probabilistic Programming in Anglican. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9286)*. Springer, 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.* 3, POPL (2019), 36:1–36:29. https://doi.org/10.1145/3290349

Yuan Zhou, Bradley J. Gram-Hansen, Tobias Kohn, Tom Rainforth, Hongseok Yang, and Frank Wood. 2019. LF-PPL: A Low-Level First Order Probabilistic Programming Language for Non-Differentiable Models. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan (Proceedings of Machine Learning Research, Vol. 89)*. PMLR, 148–157. http://proceedings.mlr.press/v89/zhou19b.html

Appendix A

Additional Proofs - Chapter 4

Restatement of Lemma 4.3 : *If* $\langle \mathcal{M}, \wp \rangle \rightsquigarrow^* \langle \mathcal{N}, \wp' \rangle$ *and* $M \triangleleft \mathcal{M}$ *and* $\boldsymbol{s} \triangleleft \wp$ *then there exists a* $N \triangleleft \mathcal{N}$ *and* $\boldsymbol{s}' \triangleleft \wp'$ *such that* $\langle M, \boldsymbol{s} \rangle \rightarrow^* \langle N, \boldsymbol{s}' \rangle$.

Proof We first observe the following obvious result: If $M \triangleleft \mathcal{M}$ and $N_i \triangleleft \mathcal{N}_i$ for $i \in [n]$ then $M[N_i/x_i]_{i\in[n]} \triangleleft \mathcal{M}[\mathcal{N}_i/x_i]_{i\in[n]}$ which can be proved by induction on M (or $\mathcal{M}$). Call this observation **(1)**.

We now show the statement without the transitive reflexive closure. The transitive statement the follows by an trivial induction. We do structural induction on $\mathcal{M}$.

- If $\mathcal{M} = (\lambda x.\mathcal{P})\mathcal{V}$: then $\mathcal{N} = \mathcal{P}[\mathcal{V}/x]$ and as $M \triangleleft \mathcal{M}$, $M = (\lambda x.P)V$ for some $P \triangleleft \mathcal{P}, V \triangleleft \mathcal{V}$ and $\wp' = \wp$. Define $\boldsymbol{s}' \triangleq \boldsymbol{s}$ and $N \triangleq P[V/x]$. Clearly $\langle M, \boldsymbol{s} \rangle \rightarrow \langle N, \boldsymbol{s}' \rangle$. Now $\boldsymbol{s}' \triangleleft \wp'$ is obvious and from **(1)** we also get $N \triangleleft \mathcal{N}$.
- If $\mathcal{M} = (\mu_x^{\wp}.\mathcal{P})\mathcal{V}$: similar to the previous case.
- $\mathcal{M} = f\big(\underline{[a_1, b_1]}, \cdots, \underline{[a_{|f|}, b_{|f|}]}\big)$. The $\mathcal{N} = \underline{\hat{f}(a_1, b_1, \cdots, a_{|f|}, b_{|f|})}$. As $M \triangleleft \mathcal{M}$ we get $M = f(r_1, \cdots, r_{|f|})$ and $r_i \in [a_i, b_i]$. Define $\boldsymbol{s}' \triangleq \boldsymbol{s}$ and $N \triangleq \underline{f(r_1, \cdots, r_n)}$. Clearly $\langle M, \boldsymbol{s} \rangle \rightarrow \langle N, \boldsymbol{s}' \rangle$. As f is interval preserving we also get that $f(r_1, \cdots, r_{|f|}) \in \underline{\hat{f}(a_1, b_1, \cdots, a_{|f|}, b_{|f|})}$ so $N \triangleleft \mathcal{N}$.
- $\mathcal{M} = \texttt{if}\big([a, b], \mathcal{P}, \mathcal{Q}\big)$. Assume $b \leq 0$, the case where $a > 0$ is analogous. So $\mathcal{N} = \mathcal{P}$. As $M \triangleleft \mathcal{M}$, $M = \texttt{if}\big(r, P, Q\big)$ for some $r \in [a, b]$ and $P \triangleleft \mathcal{P}$. So we can choose $N = P$.
- $\mathcal{M} = \texttt{sample}$ so $\wp = [a, b] :: \wp'$ and $\mathcal{N} = [a, b]$. Now $M = \texttt{sample}$ and as $\boldsymbol{s} \triangleleft \wp$, $\boldsymbol{s} = r :: \boldsymbol{s}'$ for $r \in [a, b]$. Now define $N = \underline{r}$.
- $\mathcal{M} = \texttt{score}(\underline{[a, b]})$: So $M = \texttt{score}(\underline{r})$ and $r \in [a, b]$. So $N = \underline{r}$.

- $\mathcal{M} = \mathcal{E}[\mathcal{R}]$ for $\mathcal{E} \neq [\cdot]$: Then $\langle \mathcal{R}, \wp \rangle \to \langle \mathcal{M}, \wp' \rangle$. As $M \triangleleft \mathcal{M}$ we have $M = E[R]$ for $R \triangleleft \mathcal{R}$. Now by induction on $\mathcal{R}$ we get a M with $M \triangleleft \mathcal{M}$ and $\boldsymbol{s}' \triangleleft \wp'$ such that $\langle R, \boldsymbol{s} \rangle \to \langle M, \boldsymbol{s}' \rangle$. Now $E[M] \triangleleft \mathcal{E}[\mathcal{M}]$ as $\triangleleft$ is obviously closed under context closure. And $\langle E[R], \boldsymbol{s} \rangle \to \langle E[M], \boldsymbol{s}' \rangle$ as required.

■

Lemma A.1 : *Let $\mathcal{M}$ be any term not already a value, $\kappa, \kappa' \in D^*$ and $\wp \in \mathbb{S}_\mathfrak{I}$ and $n \in \mathbb{N}$. If for every pair $(M, \boldsymbol{s})$ with $M \triangleleft \mathcal{M}$ and $\boldsymbol{s} \triangleleft \wp$*

$$\langle M, \boldsymbol{s}, \kappa \rangle \xrightarrow{co}{}^{n} \langle M_{(M,\boldsymbol{s})}, \boldsymbol{s}_{(M,\boldsymbol{s})}, \kappa' \rangle$$

(Note that $M_{(M,\boldsymbol{s})}$ and $\boldsymbol{s}_{(M,\boldsymbol{s})}$ are uniquely determined). Then $\langle \mathcal{M}, \wp \rangle \rightsquigarrow^n \langle \mathcal{M}', \wp' \rangle$ for some $\mathcal{M}'$ and $\wp'$ such that for every pair $(M, \boldsymbol{s})$, $M_{(M,\boldsymbol{s})} \triangleleft \mathcal{M}'$ and $\boldsymbol{s}_{(M,\boldsymbol{s})} \triangleleft \wp'$.

Proof We show the result for $n = 1$. The case for $n = 0$ is trivial and for $n > 1$ follows by easy induction using the case for $n = 1$. The proof goes by induction on $\mathcal{M}$. We only consider the case were $\mathcal{M}$ is itself a redex. The case where $\mathcal{M} = \mathcal{E}[\mathcal{R}]$ follow by induction on $\mathcal{R}$. The only interesting case is where $\mathcal{M}$ is a conditional redex: So lets focus on the case where $\mathcal{M} = \texttt{if}([a, b], \mathcal{N}, \mathcal{P})$: Now any $M \triangleleft \mathcal{M}$ must have the from $M = \texttt{if}(r_M, N_M, P_M)$ and there exist at least on such (as we work with closed, non, empty intervals). As any such M can reduce via $\xrightarrow{co}$ by assumption we get that $\kappa = \boldsymbol{L} :: \kappa'$ or $\kappa = \boldsymbol{R} :: \kappa'$ as otherwise no reduction can take place. W.l.o.g. assume $\kappa = \boldsymbol{L}\kappa'$. We now claim that $b \leq 0$. Assume for contradiction that $b > 0$. Then choose the term $\dot{M} \triangleleft \texttt{if}(b, N, P)$ where $N \triangleleft \mathcal{N}$ and $P \triangleleft \mathcal{P}$ are arbitrary (they always exist). Now $\langle \dot{M}, \boldsymbol{s}, \mathbf{L} :: \kappa' \rangle$ cannot make a reduction step via $\xrightarrow{co}$ which contradicts the assumption. So $b \leq 0$.
This means that $\mathcal{M} = \texttt{if}([a, b], \mathcal{N}, \mathcal{P}) \rightsquigarrow \mathcal{N}$. Now any $M \triangleleft \mathcal{M}$ of the form $M = \texttt{if}(r_M, N_M, P_M)$ and reduces to N_M. So $M_{(M,\boldsymbol{s})} = N_M \triangleleft \mathcal{N} = \mathcal{M}'$ as required and obviously $\boldsymbol{s}_{(M,\boldsymbol{s})} = \boldsymbol{s} \triangleleft \wp = \wp'$. ■

Restatement of Lemma 4.7 : *If $\wp \in \mathbb{S}_\mathfrak{I}$, $\kappa \in D^*$ and $(\!|\wp|\!) \subseteq \mathbb{T}^{(\kappa)}_{M,term}$ then $\wp \in \mathbb{T}^\mathfrak{I}_{M^{2\mathfrak{I}},term}$.*

Proof We show the following stronger lemma which immediately implies the result as the only term refining $M^{2\mathfrak{I}}$ is M itself: If $\mathcal{M}$ is any interval term, $\wp \in \mathbb{S}_\mathfrak{I}$ and $\kappa \in D^*$ and for all $M \triangleleft \mathcal{M}$, $(\!|\wp|\!) \subseteq \mathbb{T}^{(\kappa)}_{M,\text{term}}$ then $\wp \in \mathbb{T}^\mathfrak{I}_{\mathcal{M},\text{term}}$.
We can proof this as follows: As for any $M \triangleleft \mathcal{M}$, $(\!|\wp|\!) \subseteq \mathbb{T}^{(\kappa)}_{M,\text{term}}$ we get that every $M \triangleleft \mathcal{M}$ and $\boldsymbol{s} \triangleleft \wp$, $\langle M, \boldsymbol{s}, \kappa \rangle \xrightarrow{co}{}^{n} \langle V_{(M,\boldsymbol{s})}, \epsilon, \epsilon \rangle$ for a fixed n (As soon as κ is fixed each reduction takes the same number of steps). We can thus apply Lem. A.1 and get that $\langle \mathcal{M}, \wp \rangle \rightsquigarrow^n \langle \mathcal{V}, \epsilon \rangle$,

so $\wp \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},\text{term}}$ as required. ■

Restatement of Lemma 4.13 : *If $\mathfrak{V}$ is a symbolic value of type* **R** *with sample variables among $\{\alpha_0, \cdots, \alpha_{m-1}\}$ and $[a,b] \in \mathfrak{I}$ an interval then there exists a countable family of boxes $\{B_i\}_{i\in\mathcal{I}}$ ($B_i \subseteq \mathbb{R}^m_{[0,1]}$) such that $\bigcup_i B_i \Subset \mathfrak{V}^{-1}([a,b])$.*

Proof By induction on the structure of $\mathfrak{V}$:

- $\mathfrak{V} = \underline{r}$: Then choose $B_1 \triangleq \mathbb{R}^m_{[0,1]}$ if $r \in [a,b]$ and $B_1 \triangleq \emptyset$ otherwise.
- $\mathfrak{V} = \alpha_j$. Then choose $B_1 \triangleq \mathbb{R}^j_{[0,1]} \times (\mathbb{R}_{[0,1]} \cap [a,b]) \times \mathbb{R}^{m-j-1}_{[0,1]}$.
- $\mathfrak{V} = \boxed{f}(\mathfrak{V}_1, \cdots, \mathfrak{V}_{|f|})$. As f is by assumption interval-separable there exist countable boxes $(B_i)_{i\in\mathcal{I}}$ s.t., $\cup_i B_i \Subset f^{-1}([a,b])$ for some countable set $\mathcal{I}$. Each B_i is a box and can thus be written as $B_i = [a_i^1, b_i^1] \times \cdots \times [a_i^{|f|}, b_i^{|f|}]$. Now define $C_i \triangleq \bigcap_{1\le j\le|f|} \mathfrak{V}_j^{-1}([a_i^j, b_i^j]) \subseteq \mathbb{R}^m_{[0,1]}$. These are all assignments such that each $\mathfrak{V}_j$ takes on a value in $[a_i^j, b_i^j]$. As the countable union of Lebesgue null sets is a null we get $\bigcup_{i\in\mathcal{I}} C_i \Subset \mathfrak{V}^{-1}([a,b])$. Call this fact **(1)**.

  Now by induction for each $1 \le j \le |f|$ there exists a family of boxes $(B_k^{i,j})_{k\in\mathcal{I}_i^j}$ for some countable index set $\mathcal{I}_i^j$, such that $\bigcup_{k\in\mathcal{I}_i^j} B_k^{i,j} \Subset \mathfrak{V}_j^{-1}([a_i^j, b_i^j])$. Now $\bigcap_{1\le j\le|f|} \bigcup_{k\in\mathcal{I}_i^j} B_k^{i,j} = \bigcup_{(k_1,\cdots,k_{|f|})\in\mathcal{I}_i^1\times\cdots\times\mathcal{I}_i^{|f|}} B_{k_1}^{i,1} \cap \cdots B_{k_{|f|}}^{i,|f|}$ by distributing the intersection over the union. We can put this together and get the following by again using the fact that the countable union of null sets is a null set:

$$\bigcup_{(k_1,\cdots,k_{|f|})\in\mathcal{I}_i^1\times\cdots\times\mathcal{I}_i^{|f|}} B_{k_1}^{i,1} \cap \cdots B_{k_{|f|}}^{i,|f|} = \bigcap_{1\le j\le|f|} \bigcup_{k\in\mathcal{I}_i^j} B_k^{i,j} \Subset \bigcap_{1\le j\le|f|} \mathfrak{V}_j^{-1}([a_i^j, b_i^j]) = C_i$$

  Note that the index set $\mathcal{I}_i^1 \times \cdots \times \mathcal{I}_i^{|f|}$ is countable. Combined with **(1)** we get

$$\bigcup_{i\in\mathcal{I}} \bigcup_{(k_1,\cdots,k_{|f|})\in\mathcal{I}_i^1\times\cdots\times\mathcal{I}_i^{|f|}} B_{k_1}^{i,1} \cap \cdots B_{k_{|f|}}^{i,|f|} \Subset \mathfrak{V}^{-1}([a,b])$$

  Note that the set $\{(i, k_1, \cdots, k_{|f|}) \mid i \in \mathcal{I}, (k_1, \cdots, k_{|f|}) \in \mathcal{I}_i^1 \times \cdots \times \mathcal{I}_i^{|f|}\}$ is also countable as the countable product of countable sets. Also note that the finite intersection of boxes in the equation above is again a box. We are thus done.

■

Appendix B

Additional Proofs - Chapter 5

Subject Reduction

In this section, we show that the Intersection Type System (Fig. 5.3) enjoys subject reduction.

> **Restatement of Lemma 5.2 (Subject Reduction):** *If* $\vdash \mathcal{M} : \{\!\{(\alpha, \wp, \tau)\}\!\}$ *and* $\mathcal{M}$ *is not a value, then either*
>
> - $\mathcal{M}$ *has a deterministic redex so* $\mathcal{M} \to_{det} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \{\!\{(\alpha, \wp, \tau - 1)\}\!\}$*, or*
> - $\mathcal{M}$ *has a probabilistic redex then* $\wp = [a, b]\wp'$ *and we have* $\mathcal{M} \to_{[a,b]} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \{\!\{(\alpha, \wp', \tau - 1)\}\!\}$

We again begin with a substitution lemma:

> **Lemma B.1 (Substitution):** *If* $\Gamma; \{x_i : a_i\}_{i\in[n]} \vdash \mathcal{M} : \mathcal{A}$ *for distinct* x_i *and for all* $i \in [n]$ *and* $\mathcal{B} \in a_i$*,* $\Gamma \vdash \mathcal{N}_i : \mathcal{B}$ *then* $\Gamma \vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i\in[n]} : \mathcal{A}$

> **Proof** An easy induction on $\mathcal{M}$. ■

We then split our initial lemma into two: as $\mathcal{M} = \mathcal{E}[\mathcal{R}]$ for a deterministic or probabilistic redex $\mathcal{R}$. Note that being a value does not necessary mean that a reduction step can take place. It is however if a term is typeable with anything other than $\{\!\{\}\!\}$. E.g. a term $\mathtt{if}([-1, 1], \mathcal{N}, \mathcal{P})$ is not typeable with anything other than $\{\!\{\}\!\}$. For deterministic subject reduction it is actually more convenient to show the following stronger statement:

> **Lemma B.2 (Deterministic Subject Reduction):** *If* $\vdash \mathcal{M} : \mathcal{A}$*,* $\mathcal{A} \neq \{\!\{\}\!\}$ *and* $\mathcal{M}$ *has a deterministic redex and then* $\mathcal{M} \to_{det} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \mathcal{A}^{(\uparrow\epsilon,-1)}$*.*

Proof Induction on $\mathcal{M} \to_{\mathsf{det}} \mathcal{M}'$. Case analysis on $\mathcal{M}$.

- $\mathcal{M} = (\lambda x.\mathcal{N})\mathcal{P} \to_{\mathsf{det}} \mathcal{N}[\mathcal{P}/x]$: Then the last step must have been:

$$\dfrac{\dfrac{\{x : b\} \vdash \mathcal{N} : \mathcal{B}}{\vdash \lambda x.\mathcal{N} : \{\!|(b \to \mathcal{B}, \epsilon, 0)|\!\}}\,\textbf{(abs)} \qquad \{\vdash \mathcal{P} : \mathcal{C} \mid \forall \mathcal{C} \in b\}}{\vdash (\lambda x.\mathcal{N})\mathcal{P} : \mathcal{B}^{(\uparrow\epsilon,1)} = \mathcal{A}}\,\textbf{(app)}$$

  By substitution (Lem. B.1) we can type $\vdash \mathcal{N}[\mathcal{P}/x] : \mathcal{B} = \mathcal{A}^{(\uparrow\epsilon,-1)}$ as required.

- $\mathcal{M} = (\mu_x^{\varphi}.\mathcal{N})\mathcal{P} \to_{\mathsf{det}} \mathcal{N}[\mathcal{P}/x, (\mu_x^{\varphi}.\mathcal{N})/\varphi]$: Then the last step must have been via **(app)** and **(fix)** , similar to above. We conclude via (Lem. B.1).

- $\mathcal{M} = \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}) \to_{\mathsf{det}} \mathcal{N}$ and $b \leq 0$: Then the last step must have been:

$$\dfrac{\overline{\vdash \underline{[a,b]} : \{\!|([a,b], \epsilon, 0)|\!\}} \qquad \vdash \mathcal{N} : \mathcal{B}_{([a,b],\epsilon,0)}}{\vdash \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}) : \mathcal{B}^{(\uparrow\epsilon,1)}_{([a,b],\epsilon,0)}}\,\textbf{(if)}$$

  So $\vdash \mathcal{N} : \mathcal{B}_{([a,b],\epsilon,0)} = \mathcal{A}^{(\uparrow\epsilon,-1)}$.

- $\mathcal{M} = \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}) \to_{\mathsf{det}} \mathcal{P}$ and $a > 0$. Similar to the previous case.

- $\mathcal{M} = \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P})$ and $a < 0$ and $b \geq 0$. Note possible as by assumption $\mathcal{A} \neq \{\!|\,|\!\}$.

- $\mathcal{M} = f(\underline{[a,b]}, \underline{[c,d]}) \to_{\mathsf{det}} \underline{\hat{f}(a,b,c,d)}$. Then the last step must have been via (f) and **(num)** and $\mathcal{A} = \{\!|(\hat{f}(a,b,c,d), \epsilon, 1)|\!\}$ we can type $\vdash \underline{\hat{f}(a,b,c,d)} : \{\!|(\hat{f}(a,b,c,d), \epsilon, 0)|\!\}$ via **(num)** .

- $\mathcal{M} = \mathtt{score}([a,b]) \to_{\mathsf{det}} [a,b]$ and $a \geq 0$. Then the last step must have been via **(score)** and **(num)** to $\mathcal{A} = \{\!|([a,b], \epsilon, 1)|\!\}$ and we can type $\vdash \underline{[a,b]} : \{\!|([a,b], \epsilon, 0)|\!\}$ via **(num)** as required.

- $\mathcal{M} = \mathtt{score}([a,b])$ and $a < 0$. Not possible as by assumption $\mathcal{A} \neq \{\!|\,|\!\}$.

- $\mathcal{M} = \mathcal{N}\mathcal{P} \to_{\mathsf{det}} \mathcal{N}'\mathcal{P}$ and $\mathcal{N} \to_{\mathsf{det}} \mathcal{N}'$. Then the last step must have been:

$$\dfrac{\vdash \mathcal{N} : \mathcal{B} \qquad \{\vdash \mathcal{P} : \mathcal{D} \mid \forall (b \to \mathcal{C}, \wp, \tau) \in \mathcal{B}, \mathcal{D} \in b\}}{\vdash \mathcal{N}\mathcal{P} : \bigcup_{(b \to \mathcal{C}, \wp, \tau) \in \mathcal{B}} \mathcal{C}^{(\uparrow\wp, \tau+1)}}\,\textbf{(app)}$$

  By induction we get $\vdash \mathcal{N}' : \mathcal{B}^{(\uparrow\epsilon,-1)}$. We can conclude using **(app)** , by choosing the same type derivations for each element in $\mathcal{B}^{(\uparrow\epsilon,-1)}$ as in the original derivation..

- $\mathcal{M} = \mathtt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q}) \to_{\mathsf{det}} \mathtt{if}(\mathcal{N}', \mathcal{P}, \mathcal{Q})$ and $\mathcal{N} \to_{\mathsf{det}} \mathcal{N}'$: Then the last step must have been via **(if)** :

$$\frac{\vdash \mathcal{N} : \mathcal{B} \qquad \begin{array}{l}\{\vdash \mathcal{P} : \mathcal{C}_{([a,b],\wp,\tau)} \mid ([a,b],\wp,\tau) \in \mathcal{B}, b \leq 0\} \\ \{\vdash \mathcal{Q} : \mathcal{D}_{([a,b],\wp,\tau)} \mid ([a,b],\wp,\tau) \in \mathcal{B}, a > 0\}\end{array}}{\vdash \mathtt{if}(\mathcal{N},\mathcal{P},\mathcal{Q}) : \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}\mid b\leq 0} \mathcal{C}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)} \bigcup \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}\mid a>0} \mathcal{D}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}} \textbf{(if)}$$

By induction we get $\vdash \mathcal{N}' : \mathcal{B}^{(\uparrow\epsilon,-1)}$ and can again conclude via **(if)** by choosing the same derivations.

- $\mathcal{M} = \mathtt{score}(\mathcal{N}) \to_{\mathsf{det}} \mathtt{score}(\mathcal{N}')$: Then the last step must have been via **(score)** we can apply induction on N and again conclude via **(score)** .
- $\mathcal{M} = f(\mathcal{N},\mathcal{P}) \to_{\mathsf{det}} f(\mathcal{N}',\mathcal{P})$ and $\mathcal{N} \to_{\mathsf{det}} \mathcal{N}'$: Then the last step must have been via (f_2) . Similar to the case before can apply induction, use the same derivations as in the original one and conclude via (f_2)
- $\mathcal{M} = f(\underline{[a,b]},\mathcal{N}) \to_{\mathsf{det}} f(\underline{[a,b]},\mathcal{N}')$ and $\mathcal{N} \to_{\mathsf{det}} \mathcal{N}'$: The last step must have been via and **(num)** . We can apply induction and conclude back via (f_2) .

■

Lemma B.3 (Probabilistic Subject Reduction): *If* $\vdash \mathcal{M} : \{\!|(\alpha,\wp,\tau)|\!\}$ *and* $\mathcal{M}$ *has a probabilistic redex then* $\wp = [a,b]\wp'$ *and we have* $\mathcal{M} \to_{[a,b]} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \{\!|(\alpha,\wp',\tau-1)|\!\}$

Proof Case analysis on $\mathcal{M}$.

- $\mathcal{M} = \mathtt{sample}$. Then the last step is:

$$\frac{}{\vdash \mathtt{sample} : \{\!|([a,b],[a,b],1)|\!\}} \textbf{(sample)}$$

for some a, b. We get $\mathtt{sample} \to_{[a,b]} \mathcal{M}' \triangleq \underline{[a,b]}$ and can obviously type: $\vdash \mathcal{M}' : \{\!|([a,b],\epsilon,0)|\!\}$ using **(num)** .

- $\mathcal{M} = \mathcal{N}\mathcal{P}$ and $\mathcal{N}$ has a probabilistic redex. The last step must have been:

$$\frac{\vdash \mathcal{N} : \{\!|(b \to \{\!|(\alpha,\wp_3,\tau_3)|\!\},\wp_1,\tau_1)|\!\} \qquad \{\vdash \mathcal{P} : \mathcal{C} \mid \forall \mathcal{C} \in b\}}{\vdash \mathcal{N}\mathcal{P} : \{\!|(\alpha,\wp_1\wp_3,\tau_1+\tau_3+1)|\!\}} \textbf{(app)}$$

By induction we get that $\wp_1 = [a,b]\wp_2$, $\mathcal{N} \to_{[a,b]} \mathcal{N}'$ and $\vdash \mathcal{N} : \{\!|(b \to \{\!|(\alpha,\wp_3,\tau_3)|\!\},\wp_2,\tau_1-1)|\!\}$. So $\wp_1\wp_3 = [a,b]\wp_2\wp_3$. Now $\mathcal{N}\mathcal{P} \to_{[a,b]} \mathcal{N}'\mathcal{P}$ and we can conclude $\vdash \mathcal{N}'\mathcal{P} : \{\!|(\alpha,\wp_2\wp_3,\tau_1+\tau_3)|\!\}$ via **(app)** .

- All the other closre cases, i.e., $\mathcal{M} = \mathtt{if}(\mathcal{N},\mathcal{P},\mathcal{Q})$, $\mathcal{M} = f(\mathcal{N},\mathcal{P})$, $\mathcal{M} = f(\underline{[a,b]},\mathcal{N})$ and $\mathcal{M} = \mathtt{score}(\mathcal{N})$ where $\mathcal{N}$ has a probabilistic redex follow in the same fashion as above.

■

Pairwise Compatible

In this part, we show that each type derivation in our system gives pairwise compatible traces. We first observe the following fact which states that if we have interval traces that are terminating for some term and pairwise compatible, we can append arbitrary traces to each and still be pairwise compatible.

Lemma B.4: *Let $\mathcal{M}$ be a closed interval term. Assume $\{\wp_i, \mid i \in [n]\} \subseteq \mathbb{T}^{\mathfrak{I}}_{\mathcal{M},term}$ are pairwise compatible. Let $\{\wp'_i \mid i \in [n]\} \subseteq \mathbb{S}_{\mathfrak{I}}$ be arbitrary interval traces (neither necessary pairwise compatible nor terminating for some term). Then $\{\wp_i\wp'_i \mid i \in [n]\}$ is pairwise compatible.*

Proof Fix any two $i \neq j \in [n]$. By assumption $\wp_i$ and $\wp_j$ are compatible. In the first case, we assume $|\wp_i| = |\wp_j|$. Then there exists a $k \in |\wp_i|$ such that $\wp_i(k)$ and $\wp_j(k)$ are almost disjoint. Now obviously no matter what trace we append to either of them they remain compatible as the kth position does not change.
In the second case, we get w.l.o.g. $|\wp_i| < |\wp_j|$. Now we make use of the assumption that both are terminating for the same term M. Define $\hat{\wp} = \wp_j[0, \cdots, |\wp_i| - 1]$ i.e., the first $|\wp_i|$ positions of $\wp_j$. We claim that $\hat{\wp}$ and $\wp_i$ are compatible. If this were the case the statement follows from the observation in the first case.
To show that $\hat{\wp}$ and $\wp_i$ are compatible, we show that $(\!|\hat{\wp}|\!) \cap (\!|\wp_i|\!) = \emptyset$ which is even stronger than requiring compatibility (for compatibility this intersection may contain up to one element). Assume for contradiction there exist a standard trace $\boldsymbol{s}$ in the intersection. In particular, $\boldsymbol{s} \triangleleft \wp_i$, so $\boldsymbol{s}$ is terminating for M (where $M \triangleleft \mathcal{M}$ is any term). Now as $\boldsymbol{s} \in (\!|\hat{\wp}|\!)$ and by definition of $\hat{\wp}$ there exists a $\boldsymbol{s}' \neq \epsilon$ such that $\boldsymbol{s}\boldsymbol{s}' \triangleleft \wp_j$, so $\boldsymbol{s}\boldsymbol{s}'$ is also terminating for M. This is a clear contradiction as the extnsion of any terminating trace cannot be terminating. ■

We can then show the desired result:

Restatement of Lemma 5.4 (Pairwise Compatibility): *If $\vdash \mathcal{M} : \mathcal{A}$ and $\mathcal{A} = \{\!|(\alpha_i, \wp_i, \tau_i) \mid i \in [n]|\!\}$ then $\{\wp_i\}_i$ are pairwise compatible.*

Proof In the case of $\mathcal{A} = \{\!|\,|\!\}$ the statement is trivial, so we can assume that the first step is not ($\{\!|\,|\!\}$). The order on which we do induction is rather strange. For all but one case simple induction on $\mathcal{M}$ suffices. In the case of application, we do, however, need a stronger inductive hypothesis. We define an order $\prec$ as follows

$$\frac{\mathcal{M} \text{ is a subterm of } \mathcal{N}}{\mathcal{M} \prec \mathcal{N}} \qquad \frac{\langle \mathcal{N}, \wp\rangle \rightsquigarrow^* \langle \mathcal{V}, \epsilon\rangle \text{ for some } \wp}{\mathcal{V}\mathcal{P} \prec \mathcal{N}\mathcal{P}}$$

It is easy to see that $\prec$ is well founded. We now do induction on $\mathcal{M}$ w.r.t. $\prec$. We do a

case analysis on $\mathcal{M}$.

- If $\mathcal{M} = \lambda x.\mathcal{N}$ or $\mathcal{M} = \mu_x^{\wp}.\mathcal{N}$ or $\mathcal{M} = \underline{[a,b]}$: Trivial as $\mathcal{A}$ is a singleton
- If $\mathcal{M} = \texttt{sample}$: By construction of the (`sample`) rule each of the $\wp_i$ are pairwise compatible.
- If $\mathcal{M} = \texttt{score}(\mathcal{N})$: So $\vdash \mathcal{N} : \mathcal{B}$ for some $\mathcal{B}$ and each $\wp_i$ in $\mathcal{A}$ is also an interval trace from $\mathcal{B}$. The statement then follows by induction.
- If $\mathcal{M} = \texttt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q})$: So the last step must have been:

$$\frac{\vdash \mathcal{N} : \mathcal{A} \qquad \begin{array}{ll}\{\vdash \mathcal{P} : \mathcal{B}_{([a,b],\wp,\tau)} & \mid ([a,b],\wp,\tau) \in \mathcal{A}, b \leq 0\} \\ \{\vdash \mathcal{Q} : \mathcal{C}_{([a,b],\wp,\tau)} & \mid ([a,b],\wp,\tau) \in \mathcal{A}, a > 0\}\end{array}}{\vdash \texttt{if}(\mathcal{N},\mathcal{P},\mathcal{Q}) : \bigcup_{([a,b],\wp,\tau)\in\mathcal{A}\mid b\leq 0} \mathcal{B}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)} \bigcup \bigcup_{([a,b],\wp,\tau)\in\mathcal{A}\mid a>0} \mathcal{C}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}} \textbf{(if)}$$

  For each $([a,b],\wp,\tau) \in \mathcal{A}$ with $b \leq 0$, we have $\vdash P : \mathcal{B}_{([a,b],\wp,\tau)}$. Now by induction on $\mathcal{P}$ all interval traces in $\mathcal{B}_{([a,b],\wp,\tau)}$ are pairwise compatible. We can thus conclude that $\mathcal{B}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}$ is also pairwise compatible as we append the same trace to each element. With identical reasoning we also get that the traces in $\mathcal{C}_{([a,b],\wp,\tau)}$ for $a > 0$ are pairwise compatible. What remains now is to show that the interval traces in

$$\bigcup_{([a,b],\wp,\tau)\in\mathcal{A}\mid b\leq 0} \mathcal{B}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)} \bigcup \bigcup_{([a,b],\wp,\tau)\in\mathcal{A}\mid a>0} \mathcal{C}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}$$

  are pairwise compatible. Assume for contradiction they are not, so there are two traces that are not compatible. By the preceding observation they cannot both come from $\mathcal{B}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}$ or $\mathcal{C}^{(\uparrow\wp,\tau+1)}_{([a,b],\wp,\tau)}$. They must thus come from two distinct such sets.

  Now by induction the trace in $\mathcal{A}$ are also pairwise compatible and furthermore by Lem. 5.3 each of them is terminating for $\mathcal{N}$. We can thus apply Lem. B.4 and get that two traces from distinct sets as above must be compatible.

- If $\mathcal{M} = f(\mathcal{N}, \mathcal{P})$. The last step must have been:

$$\frac{\vdash \mathcal{N} : \mathcal{A} \qquad \{\vdash \mathcal{P} : \mathcal{B}_{([a,b],\wp,\tau)\in\mathcal{A}} \mid ([a,b],\wp,\tau) \in \mathcal{A}\}}{\vdash f(\mathcal{N},\mathcal{P}) : \bigcup_{([a,b],\wp,\tau)} \bigcup_{([c,d],\wp',\tau')\in\mathcal{B}_{([a,b],\wp,\tau)}} \left\{\!\!\left\{ (\hat{f}(a,b,c,d), \wp\wp', \tau+\tau'+1) \right\}\!\!\right\}} (f_2)$$

  By induction we get that each trace in $\mathcal{B}_{([a,b],\wp,\tau)\in\mathcal{A}}$ for $([a,b],\wp,\tau) \in \mathcal{A}$ is pairwise compatible and thus also in $\bigcup_{([c,d],\wp',\tau')\in\mathcal{B}_{([a,b],\wp,\tau)}} \left\{\!\!\left\{ (\hat{f}(a,b,c,d), \wp\wp', \tau+\tau'+1) \right\}\!\!\right\}$.

  Now by induction the trace in $\mathcal{A}$ are pairwise compatible and by Lem. 5.3 also terminating for $\mathcal{N}$ so we can, as in the previous case, use Lem. B.4 to conclude.

- If $\mathcal{M} = \mathcal{N}\mathcal{P}$: We get that the last step must have been:

$$\frac{\vdash \mathcal{N} : \mathcal{A} \qquad \{\vdash \mathcal{P} : \mathcal{C} \mid \forall (b \to \mathcal{B}, \wp, \tau) \in \mathcal{A}, \mathcal{C} \in b\}}{\vdash \mathcal{N}\mathcal{P} : \bigcup_{(b \to \mathcal{B}, \wp, \tau) \in \mathcal{A}} \mathcal{B}^{(\uparrow \wp, \tau+1)}} \textbf{(app)}$$

  As ensured by the simple type system each type $\mathcal{A}$ is a function type. Lets write $\mathcal{A} = \{\!\!\{(b_i \to \mathcal{B}_i, \wp_i', \tau_i') \mid i \in [m]\}\!\!\}$. By induction we get that the interval traces $(\wp_i', i \in [m])$ in $\mathcal{A}$ are pairwise compatible. By Lem. 5.3 each of them is terminating for $\mathcal{N}$, i.e., $\wp_i' \in \mathbb{T}^{\mathfrak{I}}_{N,\text{term}}$.

  Fix any $i \in [m]$. As $\wp_i' \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{N},\text{term}}$ we get $\langle \mathcal{N}, \wp_i' \rangle \rightsquigarrow^* \langle \mathcal{V}_i, \epsilon \rangle$ for some value $\mathcal{V}_i$ (this must be a λ or μ-abstraction). By Subject Reduction (Lem. 5.2) we get $\vdash \mathcal{V}_i : \{\!\!\{(b_i \to \mathcal{B}_i, \epsilon, 0)\}\!\!\}$. Using **(app)** we can thus type $\vdash \mathcal{V}_i\mathcal{P} : \mathcal{B}_i^{(\uparrow \epsilon, 1)}$. By induction (note that $\mathcal{V}_i\mathcal{P} \prec \mathcal{N}\mathcal{P}$) we get that the traces in $\mathcal{B}_i^{(\uparrow \epsilon, 1)}$ are pairwise compatible. So $\mathcal{B}_i^{(\uparrow \wp_i', \tau_i'+1)}$ is also pairwise compatible.

  As each $\wp_i' \in \mathbb{T}^{\mathfrak{I}}_{\mathcal{N},\text{term}}$ we can also apply Lem. B.4 and get that every trace combination from $\mathcal{B}_i^{(\uparrow \wp_i', \tau_i'+1)}$ and $\mathcal{B}_i^{(\uparrow \wp_i', \tau_i'+1)}$ for $i \neq j$ must be compatible. So the traces in $\bigcup_{i \in [m]} \mathcal{B}_i^{(\uparrow \wp_i', \tau_i'+1)} = \bigcup_{(b \to \mathcal{B}, \wp, \tau) \in \mathcal{A}} \mathcal{B}^{(\uparrow \wp, \tau+1)}$ are pairwise compatible as required.

■

Subject Expansion

In this section, we show that the set type system enjoys subject Expansion:

Restatement of Lemma 5.6 (Subject Expansion):

- *If* $\vdash \mathcal{M} : \mathcal{A}$ *and* $\mathcal{N} \to_{\mathit{det}} \mathcal{M}$ *then* $\vdash \mathcal{N} : \mathcal{A}^{(\uparrow \epsilon, 1)}$
- *If* $\vdash \mathcal{M}_i : \mathcal{A}_i$ *and* $\mathcal{N} \to_{[a_i, b_i]} \mathcal{M}_i$ *where* $\{[a_i, b_i]\}_i$ *are almost disjoint then*

$$\vdash \mathcal{N} : \bigcup_i (\mathcal{A}_i)^{(\uparrow [a_i, b_i], 1)}$$

We again by giving a reverse substitution lemma:

Lemma B.5 (Reverse Substitution): *If* $\vdash \mathcal{M}[\mathcal{N}_i/x_i]_{i \in [n]} : \mathcal{A}$ *for distinct* x_i *then there exist a* $\{a_i\}_{i \in [n]}$*, s.t.,* $\{x_i : a_i\}_{i \in [n]} \vdash \mathcal{M} : \mathcal{A}$ *and for all* $i \in [n]$ *and* $\mathcal{B} \in a_i$*,* $\vdash \mathcal{N}_i : \mathcal{B}$

Proof Standard. By induction on $\mathcal{M}$. ■

We then split the lemma into two part we can prove separately:

Lemma B.6 (Deterministic Subject Expansion): *If* $\vdash \mathcal{M} : \mathcal{A}$ *and* $\mathcal{M}' \to_{det} \mathcal{M}$ *then* $\vdash \mathcal{M}' : \mathcal{A}^{(\uparrow\epsilon,1)}$

Proof We assume w.l.o.g. that $\mathcal{A} \neq \emptyset$. Induction on $\vdash \mathcal{M} : \mathcal{A}$. Case analysis on $\mathcal{M}'$.

- $\mathcal{M}' = (\lambda x.\mathcal{N})\mathcal{P} \to_{\mathsf{det}} \mathcal{N}[\mathcal{P}/x]$: So $\vdash \mathcal{N}[\mathcal{P}/x] : \mathcal{A}$. By Lem. B.5 we get an a, s.t., $\{x : a\} \vdash \mathcal{N} : \mathcal{A}$ and for all $\mathcal{B} \in a$, $\vdash \mathcal{P} : \mathcal{B}$. We can conclude $\vdash \lambda x.\mathcal{N} : \{\!|(a \to \mathcal{A}, \epsilon, 1)|\!\}$ using **(abs)** and can the derive $\vdash (\lambda x.\mathcal{N})\mathcal{P} : \mathcal{A}^{(\uparrow\epsilon,1)}$ via **(app)** .
- $\mathcal{M}' = (\mu^{\varphi}_{x}.\mathcal{N})\mathcal{P} \to_{\mathsf{det}} \mathcal{N}[\mathcal{P}/x, (\mu^{\varphi}_{x}.\mathcal{N})/\varphi]$. So $\vdash \mathcal{N}[\mathcal{P}/x, (\mu^{\varphi}_{x}.\mathcal{N})/\varphi] : \mathcal{A}$. By Lem. B.5 we get a_x, a_φ such that $\{x : a_x, \varphi : a_\varphi\} \vdash \mathcal{N} : \mathcal{A}$ and for all $\mathcal{B} \in a_x$, $\vdash \mathcal{P} : \mathcal{B}$ and all $\mathcal{B} \in a_\varphi$, $\vdash \mu^{\varphi}_{x}.\mathcal{N} : \mathcal{B}$. We can thus type $\vdash \mu^{\varphi}_{x}.\mathcal{N} : \{\!|(a_x \to \mathcal{A}, \epsilon, 0)|\!\}$ using **(fix)** and conclude $\vdash (\mu^{\varphi}_{x}.\mathcal{N})\mathcal{P} : \mathcal{A}^{(\uparrow\epsilon,1)}$ via **(app)** .
- $\mathcal{M}' = \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}) \to_{\mathsf{det}} \mathcal{N}$ and $b \leq 0$ and $\vdash \mathcal{N} : \mathcal{A}$. We can type $\vdash \underline{[a,b]} : \{\!|([a,b], \epsilon, 0)|\!\}$ via **(num)** and as $b \leq 0$ we can derive $\vdash \mathtt{if}(\underline{[a,b]}, \mathcal{N}, \mathcal{P}) : \mathcal{A}^{(\uparrow\epsilon,1)}$ using **(if)** .
- $\mathcal{M}' = \mathtt{if}(\underline{a,b}, \mathcal{N}, \mathcal{P}) \to_{\mathsf{det}} \mathcal{P}$ and $a > 0$. Similar as the case above.
- $\mathcal{M}' = f(\underline{[a,b]}, \underline{[c,d]}) \to_{\mathsf{det}} \underline{\hat{f}(a,b,c,d)}$: So $\vdash \underline{\hat{f}(a,b,c,d)} : \mathcal{A}$, so we get that $\mathcal{A} = \{\!|(\hat{f}(a,b,c,d), \epsilon, 0)|\!\}$ as only **(num)** is applicable. We can type $\vdash \underline{[a,b]} : \{\!|([a,b], \epsilon, 0)|\!\}$ via **(num)** and similar for $\underline{[a,b]}$ and can conclude using (f_2)
- $\mathcal{M}' = \mathcal{N}'\mathcal{P} \to_{\mathsf{det}} \mathcal{N}\mathcal{P}$ and $\mathcal{N}' \to_{\mathsf{det}} \mathcal{N}$. As $\vdash \mathcal{N}\mathcal{P} : \mathcal{A}$ we get that the last step must have been:

$$\frac{\vdash \mathcal{N} : \mathcal{B} \qquad \{\vdash \mathcal{P} : \mathcal{D} \mid \forall (c \to \mathcal{C}, \wp, \tau) \in \mathcal{B}, \mathcal{D} \in c\}}{\vdash \mathcal{N}\mathcal{P} : \bigcup_{(c \to \mathcal{C}, \wp, \tau) \in \mathcal{B}} \mathcal{C}^{(\uparrow\wp, \tau+1)} = \mathcal{A}} \textbf{(app)}$$

  By IH we get $\vdash \mathcal{N}' : \mathcal{B}^{(\uparrow\epsilon,1)}$ and conclude using **(app)** by choosing the same derivations as in the original derivation.
- $\mathcal{M}' = \mathtt{if}(\mathcal{N}', \mathcal{P}, \mathcal{Q}) \to_{\mathsf{det}} \mathtt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q})$ and $\mathcal{N}' \to_{\mathsf{det}} \mathcal{N}$. As $\vdash \mathtt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q})$ we get that the last step must have been via **(if)** . As in the previous case we can apply induction choose the same derivations for $\mathcal{P}$ and $\mathcal{Q}$ and conclude back via **(if)** .
- $\mathcal{M}' = f(\mathcal{N}', \mathcal{P}) \to_{\mathsf{det}} f(\mathcal{N}, \mathcal{P})$ and $\mathcal{N}' \to_{\mathsf{det}} \mathcal{P}$. The last step must have been via (f_2) and we can apply our induction hypothesis as we have before.
- $\mathcal{M}' = f(\underline{[a,b]}, \mathcal{N}') \to_{\mathsf{det}} f(\underline{[a,b]}, \mathcal{N})$ and $\mathcal{N}' \to_{\mathsf{det}} \mathcal{N}$. Similar to the previous cases.
- $\mathcal{M}' = \mathtt{score}(\mathcal{N}') \to_{\mathsf{det}} \mathtt{score}(\mathcal{N})$ and $\mathcal{N}' \to_{\mathsf{det}} \mathcal{N}$. Similar to the previous cases.

■

Lemma B.7 (Probabilistic Subject Expansion): *If* $\vdash \mathcal{M}_i : \mathcal{A}_i$ *and* $\mathcal{M} \to_{[a_i,b_i]} \mathcal{M}_i$ *where* $\{[a_i, b_i]\}_i$ *are almost disjoint then*

$$\vdash \mathcal{M} : \bigcup_i \mathcal{A}_i^{(\uparrow[a_i,b_i],1)}$$

Proof We can assume that $\mathcal{A}_i \neq \{\!\!\{\,\}\!\!\}$ as this case is trivial. By induction on $\mathcal{M}$.

- $\mathcal{M} = \texttt{sample}$: Then $\mathcal{M}_i = [a_i, b_i]$ and as $\vdash \mathcal{M}_i : \mathcal{A}_i$ we get that $\mathcal{A}_i = \{\!\!\{([a_i, b_i], \epsilon, 0)\}\!\!\}$ as the last step must be via **(num)** . As $[a_i, b_i]$ are almost disjoint we can use the **(sample)** -rule to type $\texttt{sample}$as required.
- $\mathcal{M} = \mathcal{N}\mathcal{P}$ and $\mathcal{N}$ does a reduction step, i.e., $\mathcal{N} \to_{[a_i,b_i]} \mathcal{N}_i$ and $\mathcal{M}_i = \mathcal{N}_i\mathcal{P}$. As $\vdash \mathcal{N}_i\mathcal{P} : \mathcal{A}_i$ we get that the last step must have been:

$$\frac{\vdash \mathcal{N}_i : \mathcal{B}_i \qquad \{\vdash \mathcal{P} : \mathcal{D} \mid \forall (c \to \mathcal{C}, \wp, \tau) \in \mathcal{B}_i, \mathcal{D} \in c\}}{\vdash \mathcal{N}_i\mathcal{P} : \bigcup_{(c\to\mathcal{C},\wp,\tau)\in\mathcal{B}_i} \mathcal{C}^{(\uparrow\wp,\tau+1)} = \mathcal{A}_i}\ \textbf{(app)}$$

  By induction we get $\vdash \mathcal{N} : \bigcup_i \mathcal{B}_i^{(\uparrow[a_i,b_i],1)} \triangleq \mathcal{B}$- Define $\mathcal{B} \triangleq \bigcup_i (\mathcal{B}_i)^{(\uparrow[a_i,b_i],1)}$. We get $\{\vdash \mathcal{P} : \mathcal{D} \mid \forall (c \to \mathcal{C}, \wp, \tau) \in \mathcal{B}, \mathcal{D} \in c\}$ as $\mathcal{B}$ is just the concatenation of all $\mathcal{B}_i$, i.e., every type in $\mathcal{B}$ is in at least on $\mathcal{B}_i$. By using **(app)** we can thus type $\vdash \mathcal{N}\mathcal{P} : \bigcup_{(c\to\mathcal{C},\wp,\tau)\in\mathcal{B}} \mathcal{C}^{(\uparrow\wp,\tau+1)}$. The statement follows by observing that

$$\bigcup_{(c\to\mathcal{C},\wp,\tau)\in\mathcal{B}} \mathcal{C}^{(\uparrow\wp,\tau+1)} = \bigcup_i \bigcup_{(c\to\mathcal{C},\wp,\tau)\in\mathcal{B}_i} \mathcal{C}^{\uparrow([a_i,b_i]\wp,\tau+1+1)} = \bigcup_i \mathcal{A}_i^{(\uparrow[a_i,b_i],1)}$$

  as required

- $\mathcal{M} = \texttt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q})$ and $\mathcal{N}$ does a reduction step, i.e., $\mathcal{N} \to_{[a_i,b_i]} \mathcal{N}_i$ and $\mathcal{M}_i = \texttt{if}(\mathcal{N}_i, \mathcal{P}, \mathcal{Q})$. The last step in each derivation must have been:

$$\frac{\vdash \mathcal{N}_i : \mathcal{B}_i \qquad \begin{array}{ll}\{\vdash \mathcal{P} : \mathcal{C}_{([a,b],\wp,\tau),i} & \mid ([a, b], \wp, \tau) \in \mathcal{B}_i, b \leq 0\} \\ \{\vdash \mathcal{Q} : \mathcal{D}_{([a,b],\wp,\tau),i} & \mid ([a, b], \wp, \tau) \in \mathcal{B}_i, a > 0\}\end{array}}{\vdash \texttt{if}(\mathcal{N}_i, \mathcal{P}, \mathcal{Q}) : \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}_i \mid b\leq 0} \mathcal{C}^{(\uparrow\wp,\tau)}_{([a,b],\wp,\tau),i} \cup \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}_i \mid a>0} \mathcal{D}^{(\uparrow\wp,\tau),i}_{([a,b],\wp,\tau)} = \mathcal{A}_i}\ \textbf{(if)}$$

  By induction we get $\vdash \mathcal{N} : \bigcup_i \mathcal{B}_i^{(\uparrow[a_i,b_i],1)} \triangleq \mathcal{B}$. Now each element $([a, b], \wp, \tau) \in \mathcal{B}$ stems from exactly one category i (from one $\mathcal{B}_i$). So elements in $\mathcal{B}$ can be seen as having the from $([a, b], \wp, \tau), i \in \mathcal{B}$. (This is just needed to take care of the indices). Using **(if)** we can type

$$\begin{aligned}\vdash \texttt{if}(\mathcal{N}, \mathcal{P}, \mathcal{Q}) : &\bigcup_{([a,b],\wp,\tau),i\in\mathcal{B}\mid b\leq 0} \mathcal{C}^{(\uparrow\wp,\tau)}_{([a,b],\wp,\tau),i} \cup \bigcup_{([a,b],\wp,\tau),i\in\mathcal{B}\mid a>0} \mathcal{D}^{(\uparrow\wp,\tau)}_{([a,b],\wp,\tau),i} \\ = \bigcup_i \Big(&\bigcup_{([a,b],\wp,\tau)\in\mathcal{B}_i\mid b\geq 0} \mathcal{C}^{(\uparrow\wp,\tau)}_{([a,b],\wp,\tau),i} \cup \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}_i\mid a<0} \mathcal{D}^{(\uparrow\wp,\tau)}_{([a,b],\wp,\tau),i}\Big) = \bigcup_i \mathcal{A}_i^{(\uparrow[a_i,b_i],1)}\end{aligned}$$

as required.

- $\mathcal{M} = f(\mathcal{N}, \mathcal{P})$ and $\mathcal{N}$ does a reduction step, i.e., $\mathcal{N} \to_{[a_i,b_i]} \mathcal{N}_i$ and $\mathcal{M}_i = f(\mathcal{N}_i, \mathcal{P})$. As $\vdash f(\mathcal{N}_i, \mathcal{P}) : \mathcal{A}_i$ we get that the last step must have been:

$$\frac{\vdash \mathcal{N}_i : \mathcal{B}_i \qquad \{\vdash \mathcal{P} : \mathcal{C}_{([a,b],\wp,\tau),i} \mid ([a,b],\wp,\tau) \in \mathcal{B}_i\}}{\vdash f(\mathcal{N}_i, \mathcal{P}) : \bigcup_{([a,b],\wp,\tau)\in\mathcal{B}_i} \bigcup_{([c,d],\wp',\tau')\in\mathcal{C}_{([a,b],\wp,\tau),i}} \{\!|(\hat{f}(a,b,c,d), \wp\wp', \tau+\tau'+1)|\!\}} (f_2)$$

  By induction we get $\vdash \mathcal{N} : \bigcup_i \mathcal{B}_i^{(\uparrow[a_i,b_i],1)} \triangleq \mathcal{B}$. We can now conclude using (f_2) as in the previous cases.

- $\mathcal{M} = f(\underline{[a,b]}, \mathcal{N})$ and $\mathcal{N}$ does a reduction step, i.e., $\mathcal{N} \to_{[a_i,b_i]} \mathcal{N}_i$ and $\mathcal{M}_i = f(\underline{[a,b]}, \mathcal{N}_i)$. The last step must thus have been:

$$\frac{\dfrac{}{\vdash \underline{[a,b]} : \{\!|([a,b],\epsilon,0)|\!\}}\ \textbf{(num)} \qquad \vdash \mathcal{N}_i : \mathcal{B}_{([a,b],\epsilon,0),i}}{\vdash f(\underline{[a,b]}, \mathcal{N}_i) : \bigcup_{([c,d],\wp,\tau)\in\mathcal{B}_{([a,b],\epsilon,0),i}} \{\!|(\hat{f}(a,b,c,d), \wp, \tau+1)|\!\}} (f_2)$$

  By induction we get $\vdash \mathcal{N} : \bigcup_i \mathcal{B}_i^{(\uparrow[a_i,b_i],1)} \triangleq \mathcal{B}$. We can trivially conclude via (f_2) and **(num)** .

- $\mathcal{M} = \texttt{score}(\mathcal{N})$ and $\mathcal{N}$ does a reduction step, i.e., $\mathcal{N} \to_{[a_i,b_i]} \mathcal{N}_i$ and $\mathcal{M}_i = \texttt{score}(\mathcal{N}_i)$. The last step must have been via **(score)** . We can apply induction and trivially conclude via **(score)** .

■

Appendix C

Additional Proofs - Chapter 6

Restatement of Theorem 5 (Conditions for AST): *A finite step distribution p is AST if and only if all of the following hold*

a) $\sum_{i\in\mathbb{Z}} p(i) = 1$ *b)* $p \neq \delta_0$ *c)* $\sum_{i\in\mathbb{Z}} i \cdot p(i) \leq 0$

Proof First Direction: We begin with the (arguably more interesting direction) that the three conditions together imply AST.
We first note that due to condition a) the error state, $\bot$, is never reachable. We can thus concentrate on paths consisting of natural numbers and can neglect the possibly of moving to the error state. Instead of considering the random walk on the half line we, we consider the more general wok on the integers, i.e., we remove the truncation at 0. That is the markov chain $\mathfrak{M} = (\mathbb{Z}, \mathfrak{P})$ where $\mathfrak{P}$ is defined by $\mathfrak{P}(x, y) = p(y - x)$. It is easy to see that p is AST if and only if $\mathfrak{M}$ eventually visits the non-positive numbers a.s.
We then begin by checking the third condition (c)) for equality of strict inequality:

- In the case of strict inequality, we have $\sum_{i\in\mathbb{Z}} i \cdot p(i) < 0$:

  Fix any starting state m as in the definition of AST. We define integer valued random variables $X_0, X_1, \cdots$ by $X_i = m + \sum_{k=1}^{i} Y_i$ where Y_i are independent random variables that are distributed according to p. It is easy to see that by the construction of the markov chain $\mathfrak{M}$ we have $\mathfrak{P}^n(m, y) = \mathbb{P}(X_n = y)$, i.e., for the random variable X_i the probability of $X_i = y$ is the probability of being in state y after n steps. Here $\mathbb{P}$ is the probability distribution on the underlying (not specified) measurable space on which the X_is are defined.

  With $\mathbb{E}(X_i)$ we denote the expectation of X_i and with $Var(X_i)$ the variance defined

in the standard way. With $\mathbb{E}(p)$ we denote the expectation of p. We obviously have $\mathbb{E}(X_i) = m + i \cdot \mathbb{E}(p)$ and as each of the Y_i are independent also $Var(X_i) = i \cdot Var(p)$. By assumption $\mathbb{E}(p) = \sum_{i \in \mathbb{Z}} i \cdot p(i) < 0$. Let $\epsilon = -\sum_{i \in \mathbb{Z}} i \cdot p(i) > 0$. So $\mathbb{E}(X_i) = m - i \cdot \epsilon$.

All that remains now is to apply an appropriate concentration bound. Let N be such that for every $i > N$ we have $\mathbb{E}(X_i) < 0$, which exists as $\mathbb{E}(X_i) = m + i \cdot \epsilon$. For each $i > N$ we have:

$$\mathbb{P}(X_i > 0) \leq \mathbb{P}\Big(|X_i - \mathbb{E}(X_i)| > -\mathbb{E}(X_i)\Big) \overset{(1)}{\leq} \frac{Var(X_i)}{(-\mathbb{E}(X_i))^2} = \frac{i \cdot Var(p)}{(m - \epsilon \cdot i)^2} \propto \frac{i}{i+i^2}$$

where **(1)** follows from Chebyshev's inequality. If we let $i \to \infty$ we thus get that $\mathbb{P}(X_i > 0)$ converges to 0.

- In the case of equality, we have $\sum_{i \in \mathbb{Z}} i \cdot p(i) = 0$:

  First note that in this case the above reposing does not work. It does not even hold that $\mathbb{P}(X_i > 0)$ tends to 0 as it has in the previous case. We again use the same construction of the RV X_i as before. We will show that X_i does eventually become negative at least once. From condition c) together with b) we get that there exists an $i^* < 0$ with $p(i^*) > 0$. From any state k we can now reach a non-positive number in $\lceil k/i^* \rceil$ steps with probability at least $p(i^*)^{\lceil k/i^* \rceil} > 0$ (just take the relative change i^* so many times). Our proof now hinges on a famous theorem proved by George Pólya that states that any random walk on $\mathbb{Z}^1$ or $\mathbb{Z}^2$ with zero mean is recurrent (For a modern proof see e.g. [Novak 2014]). As our random walk starts in m, i.e., $X_0 = m$ we thus get that the process $(X_i)_i$ does return to m with probability 1. We can then use the strong markov property that states that if we have any stopping time τ (in our case the first time we revisit m) the process after τ is identical to the original one. We thus get that as $(X_i)_i$ is recurrent, i.e., visits m again almost-surely, it also visits m infinity many times a.s. As we have just shown, every time we visit m there is a (lower bounded) positive probability (of $p(i^*)^{\lceil k/i^* \rceil} > 0$) of visiting a negative numbers. So we eventually visit a negative number with certainty (see the zero-one law in [McIver and Morgan 2005]).

Second Direction: We prove this by contraposition. It is easy to see that if $\sum_i p(i) < 1$ the walk is not AST as we have a positive probability of moving to the error state from any state. Similar if $p = \delta_0$ we are obviously not terminating. Lastly if $\sum_{i \in \mathbb{Z}} i \cdot p(i) > 0$ we can follow similar reasoning as in the case of strictly negative expectation and show that the expectation *increases* in each step. ■

Restatement of Lemma 6.10 : *If $\{p_i\}_{i\in\mathcal{I}}$ is a* finite *family of discrete step distribution and each p_i is AST then $\{p_i\}_{i\in\mathcal{I}}$ is uniform AST.*

Proof Fix any m. Fix any $\epsilon > 0$. By assumption $\lim_{n\to\infty} \mathfrak{P}^n_{p_i}(m,0) = 1$ for every i. So for any i there exist a $N_i \in \mathbb{N}$ such that for every $n \geq N_i$, $\mathfrak{P}^n_{p_i}(m,0) \geq 1-\epsilon$ (By definition of the limit). Now define $N = \sum_i N_i$ which is finite as $\mathcal{I}$ is finite.
Now choose any $n \geq N$. We claim $\inf_{i_1,\cdots,i_n} \mathfrak{P}_{p_{i_1}} \cdots \mathfrak{P}_{p_{i_n}}(m,0) \geq 1-\epsilon$ which would immediately give us the result. Choose arbitrary indices $i_1,\cdots,i_n$. As $n \geq N$ there must exists a $i_* \in \mathcal{I}$ that occurs at least N_{i_*}-many times among $i_1,\cdots,i_n$ by the pigeon hole principle.
As 0 is an absorbing state we can see that $\mathfrak{P}_{p_i}\mathfrak{P}(m,0) \geq \mathfrak{P}(m,0)$ for any stochastic matrix $\mathfrak{P}$ **(1)**. Note that multiplication is commutative, i.e., $\mathfrak{P}_{p_i}\mathfrak{P}_{p_j}(m,0) = \mathfrak{P}_{p_j}\mathfrak{P}_{p_i}(m,0)$ (This does *not* hold for general matrix multiplication but holds for $\mathfrak{P}_{p_i}$ as a random walk is invariant of the current state). We can thus reorder the indices $i_1,\cdots,i_n$ such that i_* fills the last N_{i_*} positions. Together with **(1)** we thus get

$$\inf_{i_1,\cdots,i_n} \mathfrak{P}_{p_{i_1}} \cdots \mathfrak{P}_{p_{i_n}}(m,0) \geq \mathfrak{P}^{N_{i_*}}_{p_{i_*}}(m,0) \geq 1-\epsilon$$

as required. ∎

Restatement of Theorem 6 : *If step kernel $S : X \times \Sigma_{\mathsf{Stack}(X)} \to \mathbb{R}_{[0,1]}$ has counting pattern $\{p_{S,x} : \mathbb{N} \to \mathbb{R}_{[0,1]}\}_{x\in\mathbb{R}}$ and $\{\overline{p_{S,x}} : \mathbb{Z} \to \mathbb{R}_{[0,1]}\}_{x\in\mathbb{R}}$ is uniform AST then the pushdown process generated by S is AST.*

Proof We claim the following for all $n, m \in \mathbb{N}$:

$$\inf_{x_1,\cdots,x_n\in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, m) \leq \kappa^n_S(\varphi, \mathsf{Stack}^m(X))$$

We can prove this statement via induction on n with m universally quantified. The base case for $n = 0$ is easy as both sides are 1 iff $m = |\varphi|$ and otherwise 0. In the inductive step, we observe the following:

$$\begin{aligned}
\kappa_S^{n+1}(\varphi, \mathsf{Stack}^m(X)) &= \int_{\mathsf{Stack}(X)} \kappa_S^n(\varphi, d\,x)\kappa_S(x, \mathsf{Stack}^m(X)) && \textbf{(1)}\\
&\geq \int_{\mathsf{Stack}^{\geq 1}(X)} \kappa_S^n(\varphi, d\,x)\kappa_S(x, \mathsf{Stack}^m(X))\\
&= \sum_{k=1}^{\infty} \int_{\mathsf{Stack}^k(X)} \kappa_S^n(\varphi, d\,x :: \varrho) S(x, \mathsf{Stack}^{m-|\varrho|}(X)) && \textbf{(2)}\\
&= \sum_{k=1}^{\infty} \int_{\mathsf{Stack}^k(X)} \kappa_S^n(\varphi, d\,x :: \varrho) p_{S,x}(m-k+1) && \textbf{(3)}\\
&\geq \sum_{k=1}^{\infty} \int_{\mathsf{Stack}^k(X)} \kappa_S^n(\varphi, d\,x :: \varrho) \inf_{x'} p_{S,x'}(m-k+1)\\
&= \sum_{k=1}^{\infty} \kappa_S^n(\varphi, \mathsf{Stack}^k(X)) \cdot \inf_{x'} p_{S,x'}(m-k+1)\\
&= \sum_{k=1}^{\infty} \inf_{x_1,\cdots,x_n \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, k) \cdot \inf_{x'} p_{S,x'}(m-k+1) && \textbf{(4)}\\
&= \sum_{k=1}^{\infty} \inf_{x_1,\cdots,x_n \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, k) \cdot \inf_{x'} \mathfrak{P}_{\overline{p_{S,x'}}}(k, m) && \textbf{(5)}\\
&= \sum_{k=1}^{\infty} \inf_{x_1,\cdots,x_{n+1} \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_{n+1}}}}\right)(|\varphi|, m)
\end{aligned}$$

where **(1)** is the definition of kernel composition, **(2)** follows from the definition of κ_S, **(3)** from the definition of $p_{S,x}$, **(4)** from the IH as we assume m to be universally quantified. Finally **(5)** follows from the fact that $\mathfrak{P}_{\overline{p_{S,x'}}}(k,m) = \overline{p_{S,x'}}(m-k) = p_{S,x'}(m-k+1)$ by definition of $\mathfrak{P}_{\cdot}$ and shifting.
For the special case of $m = 0$ we get

$$\inf_{x_1,\cdots,x_n \in X} \left(\mathfrak{P}_{\overline{p_{S,x_1}}} \cdots \mathfrak{P}_{\overline{p_{S,x_n}}}\right)(|\varphi|, 0) \leq \kappa_S^n(\varphi, \{[]\})$$

Now by assumption $\{\overline{p_{S,x}} : \mathbb{Z} \to \mathbb{R}_{[0,1]}\}_{x \in \mathbb{R}}$ is uniform AST so if $n \to \infty$ the left hand side tends to 1. So $\lim_{n\to\infty} \kappa_S^n(\varphi, \{\epsilon\}) = 1$, i.e., the stack process is AST. ■

Restatement of Lemma 6.20 : *If $\{p_k\}_k$ is a family of step distributions and $(\overline{p_k})_k$ is uniformly AST then*

$$\sum_{\mathcal{S}} \mathbb{P}_{inf}^{\{p_k\}_k}(\mathcal{S}) = 1$$

Proof We define the following set of *absolute* runs, i.e., sequences of states:

$$Runs_A \triangleq \{U \in \mathbb{N}^* \mid U(i+1) - U(i) \geq -1, U(0) = 1, U(|u|-1) = 0, \forall 1 \leq i < |U| - 1 : U(i) \neq 0\}$$

Think of elements in Runs_A as terminating runs of the markov chain that start in state 1 and eventually reach state 0. The condition $U(i+1) - U(i) \geq -1$ is there to ensure that in each step the value never decrease by more than 1 (Note that $\overline{p_k}$ never assign positive probability to values less than -1). We associate a probability, $\mathbf{P}(U)$ to elements $U \in \mathit{Runs}_A$ by:

$$\mathbf{P}(U) = \inf_{k_0,\cdots,k_{|U|-2}} \prod_{i=0}^{|U|-2} \mathfrak{P}_{\overline{p_{k_i}}}(U(i), U(i+1))$$

which matches the construct for the canonical state space for Markov chains. What we observe now is that

$$\lim_{n\to\infty} \left(\inf_{k_1,\cdots,k_n} \mathfrak{P}_{\overline{p_{k_1}}} \cdots \mathfrak{P}_{\overline{p_{k_n}}}(1,0) \right) = \sum_{U \in \mathit{Runs}_A} \mathbf{P}(U)$$

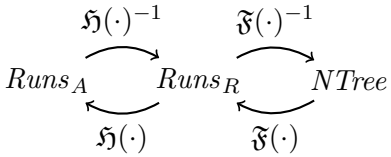
, i.e., the probability of eventually reaching 0 from 1 is the same as the sum over the probability of each path that terminates starting in 1. By assumption $(\overline{p_k})_k$ is uniformly AST so the left hand side equals 1. Call this **(1)**. Instead of analysis the absolute path we can also consider the relative change in each step. We define

$$\mathit{Runs}_R \triangleq \{u \in (\mathbb{N} \cup \{-1\})^* \mid \sum_{i=0}^{|u|-1} u(i) = -1 \wedge \forall m < |u| - 1 \sum_{i=0}^{m} u(i) > -1\}$$

Each element $u \in \mathit{Runs}_R$ gives the relative change in each step such that starting from state 1 we eventual terminate. The sum of the relative change should thus be -1 but the sum of every strict prefix is at least 0 (so that termination only occurs in the last step). There exists a bijective correspondence between elements in Runs_A and Runs_R: For each $u \in \mathit{Runs}_R$, define $\mathfrak{H}(u) \in \mathit{Runs}_A$ as the sequence of length $|u|+1$ defined by $\mathfrak{H}(u)(i) \triangleq 1 + \sum_{j=0}^{i-1} u(j)$. It is easy to verify that $\mathfrak{H}(\cdot)$ is a bijection.
Now lastly we observe that there is a bijection between the set of number trees and Runs_R. For each number tree S we inductively define a sequence of integers $\mathfrak{F}(\mathcal{S}) \in \mathit{Runs}_R$ by

$$\mathfrak{F}(n \triangleright [\mathcal{S}_1, \cdots, \mathcal{S}_n]) = (n-1) :: \mathfrak{F}(\mathcal{S}_1) \cdots \mathfrak{F}(\mathcal{S}_n)$$

It is an easy proof to show that $\mathfrak{F}(\cdot)$ forms a bijection. We thus have the bijective situation depicted on the right. It is now easy to see that for every number tree $\mathcal{S}$, $\mathbb{P}_{\inf}^{\{p_k\}_k}(\mathcal{S}) = \mathbf{P}(\mathfrak{H}(\mathfrak{F}(\mathcal{S})))$ as $\mathbb{P}_{\inf}^{\{p_k\}_k}$ gives probability of the relative change $\{p_k\}_k$ which is exactly the same as weighting the transition directly as in the definition of $\mathbf{P}$.



As $\mathfrak{H} \circ \mathfrak{F}$ is a bijection and by **(1)** we thus get $\sum_{\mathcal{S}} \mathbb{P}_{\inf}^{\{p_k\}_k}(\mathcal{S}) = 1$ as required. ■

Appendix D

Additional Proofs - Chapter 7

Restatement of Lemma 7.4 : *Let $p, q : \mathbb{N} \to \mathbb{R}_{[0,1]}$ be finite subprobability mass functions. If $\overline{p}$ is an AST step distribution and for all n,*

$$\sum_{m \leq n} p(m) \leq \sum_{m \leq n} q(m)$$

Then $\overline{q}$ is an AST step distribution.

Proof For convenience lets write $\hat{p}(i) \triangleq \sum_{j \leq i} p(j)$ similarly for $\hat{q}$. By assumption we have $\hat{p}(i) \leq \hat{q}(i)$ for every i. We now use Thm. 5: As $\overline{p}$ is an AST step distribution we have $\sum_i i \cdot p(i) = 1$, $\sum_i i \cdot p(i) \leq 1$ and $p \neq \delta_1$. We now show that q satisfies all those conditions as well and can then use the other direction from Thm. 5.
As p is finite we can view $p : \{0, \cdots, k\} \to \mathbb{R}_{[0,1]}$ for some k. As $\sum_i i \cdot p(i) = 1$ we get $\hat{p}(k) = 1$ and thus by assumption $\sum_i i \cdot q(i) = \hat{q}(k) = 1$ **(1)**. We can hence also view q as a function $q : \{0, \cdots, k\} \to \mathbb{R}_{[0,1]}$. As $\sum_i i \cdot p(i) \leq 1$ and $p \neq \delta_1$ we get that $\hat{p}(0) = p(0) > 0$, so $q(0)$ is also positive and thus $q \neq \delta_1$ **(2)**.
In the following, it remains to show that $\sum_i i \cdot q(i) \leq 1$. We use the combinatorial fact that $\sum_{i \in \mathbb{N}} i \cdot p(i) = \sum_{i \in \mathbb{N}} \sum_{j > i} p(j)$ and show:

$$\sum_{i \in \mathbb{N}} i \cdot p(i) = \sum_{i \in \mathbb{N}} \sum_{j > i} p(j) = \sum_i \big(1 - \sum_{j \leq i} p(j)\big) = \sum_{i=0}^{k} \big(1 - \hat{p}(i)\big) = (k+1) - \sum_{i=0}^{k} \hat{p}(i)$$

Analogously $\sum_{i\in\mathbb{N}} i\cdot q(i) = (k+1) - \sum_{i=0}^{k}\hat{q}(i)$. As $\hat{p}(i)\leq\hat{q}(i)$ for all i we get:

$$\sum_i i\cdot q(i) = (k+1) - \sum_{i=0}^{k}\hat{q}(i) \leq (k+1) - \sum_{i=0}^{k}\hat{p}(i) = \sum_i i\cdot p(i) \leq 1 \qquad \textbf{(3)}$$

We are done as **(1)**, **(2)** and **(3)** imply that $\overline{q}$ is AST by Thm. 5. ■

Proof of Prop. 7.5

We need the following simple fact:

Lemma D.1 : *If $\dot{a}\leq a$ and $\dot{b}\leq b$ and $p\in\mathbb{R}_{[0,1]}$ then $\dot{a}\leq pa+(1-p)b$ or $\dot{b}\leq pa+(1-p)b$.*

Proof Assume for contradiction $pa+(1-p)b<\dot{a}$ and $pa+(1-p)b<\dot{b}$ then $pa+(1-p)b<p\dot{a}+(1-p)\dot{b}$. But obviously also $p\dot{a}+(1-p)\dot{b}\leq pa+(1-p)b$, a contradiction. ■

And can then prove:

Restatement of Proposition 7.5 : *If $\mathfrak{T}$ is sufficiently independent and $C\subseteq\mathbb{N}$ then there exists a strategy $\mathfrak{S}\prec\mathfrak{T}$, s.t.,*

$$\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S},C)}\mathbb{P}^{\star}(\mathfrak{S},\kappa) \leq \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},C)}\mathbb{P}(\mathfrak{T},\kappa)$$

Proof We generalize the statement. For a measurable set $A\subseteq\mathbb{R}^m_{[0,1]}$ we define $\mathbb{P}_A(\mathfrak{T},\kappa)\triangleq\lambda_m(\mathsf{Const}(\mathfrak{T},\kappa)\cap A)$ and $\mathbb{P}^{\star}_A(\mathfrak{S},p)\triangleq\lambda_m(\mathsf{Const}^{\star}(\mathfrak{S},p)\cap A)$. Note that $\mathbb{P}_{\mathbb{R}^m_{[0,1]}}(\mathfrak{T},p)=\mathbb{P}(\mathfrak{T},p)$ and $\mathbb{P}^{\star}_{\mathbb{R}^m_{[0,1]}}(\mathfrak{S},p)=\mathbb{P}^{\star}(\mathfrak{S},p)$.
We now show that the statement holds with $\mathbb{P}_A$ instead of $\mathbb{P}$ and $\mathbb{P}^{\star}_A$ instead of $\mathbb{P}^{\star}$ for any measurable $A\subseteq\mathbb{R}^m_{[0,1]}$ which obviously subsumes our initial obligation.
The proof goes by induction on $\mathfrak{T}$ with $A\subseteq\mathbb{R}^m_{[0,1]}$ universally quantified.

- If $\mathfrak{T}=\circ\,\mathfrak{V}$ then define $\mathfrak{S}\triangleq\circ\,\mathfrak{V}$. It is easy to check that this strategy does satisfy the condition.
- If $\mathfrak{T}=\boxed{\mu}\,(\mathfrak{T}')$. By induction there is a $\mathfrak{S}'\prec\mathfrak{T}'$ that satisfies the conditions. Define $\mathfrak{S}\triangleq\boxed{\mu}\,(\mathfrak{S}')$ which trivial satisfies the condition.
- If $\mathfrak{T}=\boxed{s}\,(\mathfrak{V})(\mathfrak{T}')$. Define $A'\triangleq\mathfrak{V}^{-1}[0,\infty)\cap A$ which is obviously measurable. Now by induction there is a $\mathfrak{S}'\prec\mathfrak{T}'$ such that

$$\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}',C)}\mathbb{P}^{\star}_{A'}(\mathfrak{S}',\kappa) \leq \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}',C)}\mathbb{P}_{A'}(\mathfrak{T}',\kappa)$$

Define $\mathfrak{S} \triangleq \boxed{s}\,(\mathfrak{V})(\mathfrak{S}')$. Now for every $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S})$ we have

$$\begin{aligned}\mathbb{P}^{\star}_{A}(\mathfrak{S},\kappa) &= \lambda_m\big(\mathsf{Const}^{\star}(\mathfrak{S},\kappa)\cap A\big) = \lambda_m\big(\mathsf{Const}^{\star}(\mathfrak{S}',\kappa)\cap \mathfrak{V}^{-1}[0,\infty)\cap A\big)\\ &= \mathbb{P}^{\star}_{\mathfrak{V}^{-1}[0,\infty)\cap A}(\mathfrak{S}',\kappa) = \mathbb{P}^{\star}_{A'}(\mathfrak{S}',\kappa)\end{aligned}$$

Analogously $\mathbb{P}_{A}(\mathfrak{T},\kappa) = \mathbb{P}_{A'}(\mathfrak{T}',\kappa)$. So using the IH we get

$$\begin{aligned}\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S},C)} \mathbb{P}^{\star}_{A}(\mathfrak{S},\kappa) &= \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}',C)} \mathbb{P}^{\star}_{A'}(\mathfrak{S}',\kappa)\\ &\leq \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}',C)} \mathbb{P}_{A'}(\mathfrak{T}',\kappa) = \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},C)} \mathbb{P}_{A}(\mathfrak{T},\kappa)\end{aligned}$$

- If $\mathfrak{T} = \circ(\mathfrak{V})(\mathfrak{T}_1,\mathfrak{T}_2)$: We define the set $A_1 \triangleq \mathfrak{V}^{-1}(-\infty,0]\cap A$ and $A_2 \triangleq \mathfrak{V}^{-1}(0,\infty)\cap A$. Both are measurable. By induction there are strategies $\mathfrak{S}_1,\mathfrak{S}_2$ such that

$$\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_i,C)} \mathbb{P}^{\star}_{A_i}(\mathfrak{S}_i,\kappa) \leq \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_i,C)} \mathbb{P}_{A_i}(\mathfrak{T}_i,\kappa) \tag{1}$$

  for $i\in\{1,2\}$. We define $\mathfrak{S} \triangleq \circ(\mathfrak{V})(\mathfrak{S}_1,\mathfrak{S}_2)$ and claim that this fulfils the criterion. We observe the following, for any $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_1)$ we have:

$$\begin{aligned}\mathbb{P}^{\star}_{A}(\mathfrak{S},\boldsymbol{L}\kappa) &= \lambda_m\big(\mathsf{Const}^{\star}(\mathfrak{S},\boldsymbol{L}\kappa)\cap A\big)\\ &= \lambda_m\big(\mathsf{Const}^{\star}(\mathfrak{S}_1,\kappa)\cap \mathfrak{V}^{-1}(-\infty,0]\cap A\big)\\ &= \mathbb{P}^{\star}_{\mathfrak{V}^{-1}(-\infty,0]\cap A}(\mathfrak{S}_1,\kappa) = \mathbb{P}^{\star}_{A_1}(\mathfrak{S}_1,\kappa)\end{aligned}$$

  and analogously for every $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_2)$, $\mathbb{P}^{\star}_{A}(\mathfrak{S},\boldsymbol{R}\kappa) = \mathbb{P}^{\star}_{A_2}(\mathfrak{S}_2,\kappa)$. The same also holds for $\mathbb{P}$ instead of $\mathbb{P}^{\star}$. We can now check:

$$\begin{aligned}\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S},C)} \mathbb{P}^{\star}_{A}(\mathfrak{S},\kappa) &= \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_1,C)} \mathbb{P}^{\star}_{A}(\mathfrak{S},L\kappa) + \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_2,C)} \mathbb{P}^{\star}_{A}(\mathfrak{S},R\kappa)\\ &= \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_1,C)} \mathbb{P}^{\star}_{A_1}(\mathfrak{S}_1,\kappa) + \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_2,C)} \mathbb{P}^{\star}_{A_2}(\mathfrak{S}_2,\kappa)\end{aligned}$$

  And using the same reasoning we have

$$\sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T},C)} \mathbb{P}_{A}(\mathfrak{T},\kappa) = \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_1,C)} \mathbb{P}_{A_1}(\mathfrak{T}_1,\kappa) + \sum_{\kappa\in\mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_2,C)} \mathbb{P}_{A_2}(\mathfrak{T}_2,\kappa)$$

  We can now conclude using the inequalities we obtained via induction (1).

- If $\mathfrak{T} = \bullet(f)(\mathfrak{T}_1,\mathfrak{T}_2)$: We can assume that $\lambda_m(A) > 0$ as otherwise the statement is obvious as any strategy would work since both sides are equal to zero.

Let $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_1)$: We make use of the assumption of sufficient independence. As by assumption $\mathfrak{V}$ does not contain sample variables occurring in $\mathfrak{T}_1$ (and thus A), we get that $\mathfrak{V}^{-1}(-\infty, 0]$ and $\mathsf{Const}(\mathfrak{T}_1, \kappa)$ are conditionally independent w.r.t. to A λ_m. In particular,

$$\lambda_m\big(\mathfrak{V}^{-1}(-\infty,0] \cap \mathsf{Const}(\mathfrak{T}_1,\kappa) \mid A\big) = \lambda_m\big(\mathfrak{V}^{-1}(-\infty,0] \mid A\big) \cdot \lambda_m\big(\mathsf{Const}(\mathfrak{T}_1,\kappa) \mid A\big)$$

We can multiply both sides by $\lambda_m(A)$ and derive

$$\lambda_m\big(\mathfrak{V}^{-1}(-\infty,0] \cap \mathsf{Const}(\mathfrak{T}_1,\kappa) \cap A\big) = \lambda_m\big(\mathsf{Const}(\mathfrak{T}_1,\kappa) \cap A\big) \cdot \lambda_m\big(\mathfrak{V}^{-1}(-\infty,0] \mid A\big)$$

We can now derive:

$$\begin{aligned}\mathbb{P}_A(\mathfrak{T}, \boldsymbol{L}\kappa) &= \lambda_m\big(\mathsf{Const}(\mathfrak{T}, \boldsymbol{L}\kappa) \cap A\big) = \lambda_m\big(\mathsf{Const}(\mathfrak{T}_1, \kappa) \cap \mathfrak{V}^{-1}(-\infty, 0] \cap A\big) \\ &= \lambda_m(\mathsf{Const}(\mathfrak{T}_1, \kappa) \cap A) \cdot \lambda_m(\mathfrak{V}^{-1}(-\infty, 0] \mid A) \\ &= \mathbb{P}_A(\mathfrak{T}_1, \kappa) \cdot \lambda_m(\mathfrak{V}^{-1}(-\infty, 0] \mid A)\end{aligned}$$

Analogously $\mathbb{P}_A(\mathfrak{T}, \boldsymbol{R}\kappa) = \mathbb{P}_A(\mathfrak{T}_2, \kappa) \cdot \mu(\mathfrak{V}^{-1}(0, \infty) \mid A)$ for $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_2)$. Now:

$$\begin{aligned}\sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}, C)} \mathbb{P}_A(\mathfrak{T}, \kappa) &= \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_1, C)} \mathbb{P}^{\star}_A(\mathfrak{T}, \boldsymbol{L}\kappa) + \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_2, C)} \mathbb{P}^{\star}_A(\mathfrak{T}, \boldsymbol{R}\kappa) \\ = \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_1, C)} \mathbb{P}_A(\mathfrak{T}_1, \kappa) \lambda_m(\mathfrak{V}^{-1}(-\infty, 0] \mid A) &+ \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_2, C)} \mathbb{P}_A(\mathfrak{T}_2, \kappa) \lambda_m(\mathfrak{V}^{-1}(0, \infty) \mid A)\end{aligned}$$

By the IH there are strategies $\mathfrak{S}_1, \mathfrak{S}_2$ such that

$$\sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_i, C)} \mathbb{P}^{\star}_A(\mathfrak{S}_i, \kappa) \leq \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}_i, C)} \mathbb{P}_A(\mathfrak{T}_i, \kappa)$$

for $i \in \{1, 2\}$. Now as $\lambda_m(\mathfrak{V}^{-1}(-\infty, 0] \mid A) + \lambda_m(\mathfrak{V}^{-1}(0, \infty) \mid A) = 1$ we can apply Lem. D.1. So there exists $i^* \in \{1, 2\}$ such that

$$\sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_{i^*}, C)} \mathbb{P}^{\star}_A(\mathfrak{S}_{i^*}, \kappa) \leq \sum_{\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{T}, C)} \mathbb{P}_A(\mathfrak{T}, \kappa)$$

In case where $i^* = 1$, we define $\mathfrak{S} = \bullet(\mathfrak{V})(\mathfrak{S}_1, \times)$. We can observe that for all $\kappa \in \mathsf{Paths}^{\blacktriangledown}(\mathfrak{S}_1)$ we have $\mathbb{P}^{\star}_A(\mathfrak{S}_1, \kappa) = \mathbb{P}^{\star}_A(\mathfrak{S}, \boldsymbol{L}\kappa)$ as $\mathsf{Const}^{\star}$ does not add any constraint. So $\mathfrak{S}$ does satisfy the desired property. In the case of $i^* = 2$, define $\mathfrak{S} = \bullet(\mathfrak{V})(\times, \mathfrak{S}_2)$.

■