

Smart Choices and the Selection Monad

Martín Abadi and Gordon Plotkin
Google Research

Abstract

Describing systems in terms of choices and their resulting costs and rewards offers the promise of freeing algorithm designers and programmers from specifying how those choices should be made; in implementations, the choices can be realized by optimization techniques and, increasingly, by machine learning methods. We study this approach from a programming-language perspective. We define two small languages that support decision-making abstractions: one with choices and rewards, and the other additionally with probabilities. We give both operational and denotational semantics.

In the case of the second language we consider three denotational semantics, with varying degrees of correlation between possible program values and expected rewards. The operational semantics combine the usual semantics of standard constructs with optimization over spaces of possible execution strategies.

The denotational semantics, which are compositional and can also be viewed as an implementation by translation to a simpler language, rely on the selection monad, to handle choice, combined with an auxiliary monad, to handle other effects such as rewards or probability.

We establish adequacy theorems that the two semantics coincide in all cases. We also prove full abstraction at basic types, with varying notions of observation in the probabilistic case corresponding to the various degrees of correlation. We present axioms for choice combined with rewards and probability, establishing completeness at basic types for the case of rewards without probability.

Contents

1	Introduction	2
2	The selection monad and algebraic operations	6
2.1	The selection monad	6
2.2	Generic effects and algebraic operations	9
3	A general language with algebraic operations	18
3.1	Syntax	18
3.2	Operational semantics	19

3.3	Denotational semantics	24
3.4	Adequacy	26
3.5	Program equivalences and purity	26
4	A language of choices and rewards	27
4.1	Syntax	27
4.2	Rewards and additional effects	28
4.3	Operational semantics	29
4.4	Denotational semantics	32
4.5	Adequacy	33
4.6	Full abstraction, program equivalences, and purity	35
5	Adding probabilities	42
5.1	Syntax	42
5.2	Rewards and additional effects	43
5.3	Operational semantics	44
5.4	Denotational semantics	48
5.5	Adequacy	53
5.6	Full abstraction, program equivalences, and purity	54
6	Conclusion	63
	Acknowledgements	65

1 Introduction

Models and techniques for decision-making, such as Markov Decision Processes (MDPs) and Reinforcement Learning (RL), enable the description of systems in terms of choices and of the resulting costs and rewards. For example, an agent that plays a board game may be defined by its choices in moving pieces and by how many points these yield in the game. An implementation of such a system may aim to make the choices following a strategy that results in attractive costs and rewards, perhaps the best ones. For this purpose it may rely on classic optimization techniques or, increasingly, in forms of machine learning (ML). Deep RL has been particularly prominent in the last decade, but contextual bandits and ordinary supervised learning can also be useful.

In a programming context, several languages and libraries support choices, rewards, costs, and related notions in a general way (not specific to any application, such as a particular board game). McCarthy’s `amb` operator [34] may be seen as an early example of a construct for making choices. More recent work includes many libraries for RL (e.g., [9]), languages for planning such as DTGolog [7] and some descendants (e.g., [40]) of

the Planning Domain Definition Language [35], a “credit-assignment” compiler for learning to search built on the Vowpal-Rabbit learning library [12], and Dyna [42], a programming language for machine-learning applications based on MDPs. It also includes SmartChoices [11], an “approach to making machine learning (ML) a first class citizen in programming languages”, one of the main inspirations for our work. SmartChoices and several other recent industry projects in this space (such as Spiral [10]) extend mainstream programming languages and systems with the ability to make data-driven decisions by coding in terms of choices (or predictions) and feedback (in other words, perceived costs or rewards), and thus aim to have widespread impact on programming practice.

The use of decision-making abstractions has the potential to free algorithm designers and programmers from taking care of many details. For example, in an ordinary programming system, a programmer that implements quicksort should consider how to pick pivot elements and when to fall back to a simpler sorting algorithm for short inputs. Heuristic solutions to such questions abound, but they are not always optimal, and they require coding and sometimes maintenance when the characteristics of the input data or the implementation platform change. In contrast, SmartChoices enables the programmer to code in terms of choices and costs—or, equivalently, rewards, which we define as the opposite of costs—and to let the implementation of decision-making take care of the details [11, Section 4.3]. As another example, consider the program in Figure 1 that does binary search in a sorted array. This pseudocode is a simplified version of the one in [11, Section 4.2], which

```
let binsearch(x : Int, a : Array[Int], l : Int, r : Int) =
  if l > r then None // the special value None represents failure
  else let m : [l,r] = choice in \\ choose an integer in [l, r]
    if a[m] = x then m
    else cost(1); \\ pay to recurse
      if a[m] < x then binsearch(x, a, m+1, r)
      else binsearch(x, a, l, m-1)
```

Figure 1: Smart binary search

also includes a way of recording observations of the context of choices (in this example, x , $a[l]$, and $a[r]$) that facilitate machine learning. Here, a choice determines the index m where the array is split. Behind the scenes, a clever implementation can take into account the distribution of the data in order to decide exactly how to select m . For example, if x is half way between $a[l]$ and $a[r]$ but the distribution of the values in the array favors smaller values, then the selected m may be closer to r than to l . In order to inform the implementation, the programmer calls `cost`: each call to `cost` adds to the total cost of an execution, for the notion of cost that the programmer would wish to minimize. In this example, the total cost is the number of recursive calls. In other examples, the total cost could correspond, for instance, to memory requirements or to some application-specific

metric such as the number of points in a game.

In this paper, we study decision-making abstractions from a programming-language perspective. We define two small languages that support such abstractions, one with choices and rewards, and the other one additionally with probabilities. In the spirit of SmartChoices (and in contrast with DTGolog and Dyna, for instance), the languages are mostly mainstream: only the decision-making abstractions are special. We give them both operational and denotational semantics. In the case of the language with probabilities we provide three denotational semantics, covering with varying degrees of correlation between possible program values and expected rewards.

Their operational semantics combine the usual semantics of standard constructs with optimization over possible strategies (thinking of programs as providing one-person games). Despite the global character of optimization, our results include a tractable, more local formulation of their operational semantics (Theorems 4 and 8). Their denotational semantics are based on the selection monad [17, 19, 15, 18, 14, 25, 16, 5], which we explain below.

We establish that operational and denotational semantics coincide, proving adequacy results for both languages (Theorems 5 and 9). We also investigate questions of full abstraction (at basic types) and program equivalences. Our full abstraction results (particularly Theorems 6 and 10, and Corollary 3) provide further evidence of the match between denotational and operational semantics. We prove full abstraction results at basic types for each of our denotational semantics, in each case with respect to appropriate notions of observation. Program equivalences can justify program transformations, and develop proof systems for them. For example, one of our axioms concerns the commutation of choices and rewards. In particular, in the case of the language for rewards we establish (Theorem 6) the soundness and completeness of the proof system with respect to concepts of observational equivalence and semantic equivalence (at basic types). In the case of the language with probabilities, finding such completeness results is an open problem. However, in all cases we show that our proof systems are complete w.r.t. to proving effect-freeness (Corollary 1 and Theorem 11).

A brief, informal discussion of the semantics of `binsearch` may provide some intuition on the two semantics and on the role of the selection monad.

- If we are given the sequence of values picked by the `choice` construct in an execution of `binsearch`, a standard operational semantics straightforwardly allows us to construct the rest of the execution. We call this semantics the *ordinary operational semantics*. For each such sequence of values, the ordinary operational semantics implies a resulting total cost, and thus a resulting total reward. We define the *selection operational semantics* by requiring that the sequence of values be the one that maximizes this total reward.

Although they are rather elementary, these operational semantics are not always a convenient basis for reasoning, because (as usual for operational semantics) they are not compositional, and in addition the selection operational semantics is defined in

terms of sequences of choices and accumulated rewards in multiple executions. On the other hand, the chosen values are simply plain integers.

- In contrast, in the denotational semantics, we look at each choice of `binsearch` as being made locally, without implicit reference to the rest of the execution or other executions, by a higher-order function of type $(\text{Int} \rightarrow \text{R}) \rightarrow \text{Int}$ (where Int is a finite set of machine integers), whose expected argument is a reward function f that maps each possible value of the choice to the corresponding reward of type R of the rest of the program. We may view f as a reward continuation. One possible such higher-order function is the function `argmax` that picks a value for the argument x for f that yields the largest reward $f(x)$. (There are different versions of `argmax`, in particular with different ways of breaking ties, but informally one often identifies them all.)

The type $(\text{Int} \rightarrow \text{R}) \rightarrow \text{Int}$ of this example is a simple instance of the selection monad, $S(X) = (X \rightarrow \text{R}) \rightarrow X$, where X is any type, and `argmax` is an example of a selection function. More generally, we use $S(X) = (X \rightarrow \text{R}) \rightarrow T(X)$, where T is another, auxiliary, monad, which can be used to model other computational effects, for example, as we do here, rewards and probabilities. The monadic approach leads to a denotational semantics that is entirely compositional, and therefore facilitates proofs of program equivalences of the kind mentioned above. The denotational semantics may be viewed as an implementation by translation to a language in which there are no primitives for decision-making, and instead one may program with selection functions.

We develop our languages and their corresponding theory in stages. In Section 2, we review the selection monad, and investigate its algebraic operations. In Section 3, we present a general language with algebraic operations, give a basic adequacy theorem, and briefly discuss a calculus for program equivalences. This section is an adaptation of prior work, useful for our project but not specific to it. In Section 4, we define and study our first language with decision-making abstractions; it is a simply typed, higher-order λ -calculus, extended with a binary choice operation `— or —` and a construct for adding rewards. In Section 5, we proceed to our second language, which adds a probabilistic choice operator to the first. Probabilistic choices are not subject to optimization, but in combination with `— or —`, they enable us to imitate the choice capabilities of MDPs. Unlike MDPs, the language does not support infinite computations. We conjecture they can be treated via a metric approach to semantics; at any rate, there is no difficulty in adding a primitive recursion operator to the language, permitting MDP runs of arbitrary prescribed lengths.

In sum, we regard the main contributions of this paper as being (1): the connection between programming languages with decision-making abstractions and the selection monad, and (2): the definition and study of operational and denotational semantics for those languages, and the establishment of adequacy and full abstraction theorems for them. The

adequacy theorems show that global operationally-defined optimizations can be characterized compositionally using a semantics based on the selection monad.

As described above, the selection operational semantics and the denotational semantics with the argmax selection function both rely on maximizing rewards. In many cases, optimal solutions are expensive. Even in the case of `binsearch`, an optimal solution that immediately picks `m` such that `a[m]` equals `x` (without ever recursing) seems unrealistic. For efficiency, the optimization may be approximate and data-driven. In particular, as an alternative to the use of maximization in the selection operational semantics, we may sometimes be able to make the choices with contextual-bandit techniques, as in [11, Section 4.2]. In the denotational semantics, assuming that `R` is the type of real numbers, we may use other selection functions instead of argmax. (The use of argmax is convenient, but our approach does not require it.) For example, instead of computing $\text{argmax}(f)$, we may approximate f by a differentiable function over the real numbers, represented by a neural network with learned parameters, and then find a local maximum of this approximation by gradient ascent. We have explored such approximations only informally so far; Section 6 briefly mentions aspects of this and other subjects for further work.

2 The selection monad and algebraic operations

In this section we present material on the selection monad and on generic effects and algebraic operations, including a discussion of algebraic operations for the selection monad.

2.1 The selection monad

The selection monad

$$S(X) = (X \rightarrow R) \rightarrow X$$

introduced in [17], is a strong monad available in any cartesian closed category, for simplicity discussed here only in the category of sets. One can think of $F \in S(X)$ as *selection functions* which choose an element $x \in X$, given a *reward function* $\gamma : X \rightarrow R$, viewing `R` as a type of *rewards*. In a typical example, the choice x optimizes, maximizing the reward $\gamma(x)$ in some sense.

Computationally, we may understand $F \in S(X)$ as producing x given a *reward continuation* γ , i.e., a function giving the reward of the remainder of the computation. Indeed there is a close connection to the continuation monad $K(X) = (X \rightarrow R) \rightarrow R$: there is a monad morphism $\theta_{S,K} : S \rightarrow K$ sending $F \in S(X)$ to $\lambda\gamma \in (X \rightarrow R). \gamma(F(\gamma))$ (see Lemma 1 below). The selection monad has strong connections to logic. For example, as explained in [18], whereas logic translations using K , taking `R` to be \perp , verify the double-negation law $\neg\neg P \supset P$, translations using S verify the instance $((P \supset R) \supset P) \supset P$ of Peirce’s law. Again, with `R` the truth values, elements of $K(X)$ correspond to quantifiers, and elements of $S(X)$ correspond to selection operators, such as Hilbert’s ε -operator.

The selection monad has unit $(\eta_S)_X : X \rightarrow S(X)$, where $\eta_S(x) = \lambda\gamma \in X \rightarrow R. x$ (here, and below, we may drop suffices when they are evident from the context). The Kleisli extension $f^{\dagger_S} : S(X) \rightarrow S(Y)$ of a function $f : X \rightarrow S(Y)$ is a little involved, so we explain it in stages. We need a function

$$S(X) \xrightarrow{f^{\dagger_S}} S(Y)$$

Equivalently, we need to pick an element of Y , given a reward continuation $\gamma : Y \rightarrow R$ and a computation $F \in S(X)$, and do so as follows:

- Given γ , f yields a final result in Y as a function of X , viz. $res_x = f(x)(\gamma)$. We think of this as the optimal choice of $y \in Y$ starting from x (optimal given the reward function γ , that is).
- As we know the reward for such a final result, we know the reward for this particular application of f , viz. $rew_x = \gamma(res_x)$.
- This reward function is in turn the reward continuation of F , and we can choose the optimal element of X for this reward function, viz. $opt = F(rew)$.
- Now that we know the best choice of x , we use it to get the desired element of Y , viz. res_{opt} .

Intuitively, F chooses the $x \in X$ which gives the optimal $y \in Y$, and then f uses that x .

Writing all this out, we find:

$$\begin{aligned} f^{\dagger_S} F \gamma &= res_{opt} \\ &= res_{F(rew)} \\ &= res_{F(\lambda x \in X. \gamma(res_x))} \\ &= res_{F(\lambda x \in X. \gamma(f(x)(\gamma)))} \\ &= f F(\lambda x \in X. \gamma(f(x)(\gamma))) \gamma \end{aligned}$$

The selection monad has strength $(st_S)_{X,Y} : X \times S(Y) \rightarrow S(X \times Y)$ where:

$$(st_S)_{X,Y}(x, F) = \lambda\gamma \in X \times Y \rightarrow R. \langle x, F(\lambda y \in Y. \gamma(x, y)) \rangle$$

There is a generalization of the above “basic” selection monad, incorporating a given strong *auxiliary* monad T . This generalization will prove useful when we wish to combine various additional effects with selection. Suppose that R is a T -algebra with algebra map $\alpha_T : T(R) \rightarrow R$. Then, as essentially proved in [16] for any cartesian closed category, we can define a strong monad S_T (which may just be written S , when T is understood) by setting:

$$S_T(X) = (X \rightarrow R) \rightarrow T(X)$$

It has unit $(\eta_{S_T})_X : X \rightarrow S_T(X)$ where $(\eta_{S_T})_X(x) = \lambda\gamma \in X \rightarrow R. (\eta_T)(x)$. The Kleisli extension $f^{\dagger_{S_T}} : S_T(X) \rightarrow S_T(Y)$ of a function $f : X \rightarrow S_T(Y)$ is given, analogously to the above, by:

$$f^{\dagger_{S_T}} F\gamma = (\lambda x \in X. f(x)(\gamma))^{\dagger_T} (F(\lambda x \in X. (\alpha_T \circ T(\gamma))(f(x)(\gamma)))) \quad (1)$$

So for $(\mu_T)_X = \text{id}_{S_T(X)}^{\dagger_{S_T}}$ we get:

$$(\mu_T)_X F\gamma = (\lambda d \in S_T(X). d\gamma)^{\dagger_T} (F(\lambda d \in S_T(X). (\alpha_T \circ T(\gamma))(d\gamma))) \quad (2)$$

The selection monad has strength $(\text{st}_{S_T})_{X,Y} : X \times S_T(Y) \rightarrow S_T(X \times Y)$ where:

$$(\text{st}_{S_T})_{X,Y}(x, F) = \lambda\gamma \in X \times Y \rightarrow R. (\text{st}_T)_{X,Y}(x, F(\lambda y \in Y. \gamma(x, y)))$$

As with the basic selection monad, there is a monad morphism to the continuation monad. Define $\theta_{S_T, K} : S \rightarrow K$ by:

$$(\theta_{S_T, K})_X(F) = \lambda\gamma \in X \rightarrow R. \alpha_T(T(\gamma)(F\gamma)) \quad (F \in S(X))$$

Lemma 1. $\theta_{S_T, K}$ is a monad morphism.

Proof. Preservation of unit is evident. For preservation of the multiplication, we first note that, for $(\mu_S)_X = \text{id}_{S(X)}^{\dagger_S} : S(S(X)) \rightarrow S(X)$, we have:

$$(\mu_S)_X F = \lambda\gamma \in X \rightarrow R. (\lambda d \in S(X). d\gamma)^{\dagger_T} (F(\lambda d \in S(X). (\alpha_T \circ T(\gamma))(d\gamma)))$$

We have to show that:

$$(\mu_S)_X \circ \theta_{S(X)} \circ T(\theta_X) = \theta_X \circ (\mu_T)_X$$

For $a \in T(T(X))$, setting $b = T(\theta_X)a$ we have:

$$\begin{aligned} ((\mu_S)_X \circ \theta_{S(X)} \circ T(\theta_X))a &= \lambda\gamma \in X \rightarrow R. (\lambda d \in S(X). d\gamma)^{\dagger_T} ((\theta_{S(X)}b)(\lambda d \in S(X). (\alpha_T \circ T(\gamma))(d\gamma))) \\ &= \lambda\gamma \in X \rightarrow R. (\lambda d \in S(X). d\gamma)^{\dagger_T} (b) \\ &= \lambda\gamma \in X \rightarrow R. (\lambda d \in S(X). d\gamma)^{\dagger_T} (T(\theta_X)a) \\ &= \lambda\gamma \in X \rightarrow R. ((\lambda d \in S(X). d\gamma) \circ \theta_X)^{\dagger_T} (a) \\ &= \lambda\gamma \in X \rightarrow R. (\text{id}_{T(X)})^{\dagger_T} (a) \\ &= \lambda\gamma \in X \rightarrow R. (\mu_T)_X(a) \\ &= (\theta_X \circ (\mu_T)_X)a \end{aligned}$$

□

2.2 Generic effects and algebraic operations

In order to be able to give semantics to effectual operations such as probabilistic choice, we use the apparatus of generic effects and algebraic operations in the category of sets discussed (in a much more general setting) in [38]. Suppose that M is a (necessarily strong) monad on the category of sets. An *generic effect* g with arity (I, O) (written $g : (I, O)$) for M is just a Kleisli map:

$$g : O \rightarrow M(I)$$

and an *algebraic operation* op with arity (I, O) (written $\text{op} : (I, O)$) for M is a family

$$\text{op}_X : O \times M(X)^I \rightarrow M(X)$$

natural with respect to Kleisli maps in the sense that the following diagram commutes for all $e : X \rightarrow M(Y)$:

$$\begin{array}{ccc} O \times M(X)^I & \xrightarrow{\text{op}_X} & M(X) \\ \downarrow O \times (e^\dagger_M)^I & & \downarrow e^\dagger_M \\ O \times M(Y)^I & \xrightarrow{\text{op}_Y} & M(Y) \end{array}$$

There is a 1-1 correspondence between (I, O) -ary generic effects and (I, O) -ary algebraic operations. Given g one may define:

$$\text{op}_X(o, a) = a^{\dagger_M}(g(o))$$

and conversely one sets:

$$g(o) = \text{op}_I(o, (\eta_M)_I)$$

Naturality implies a weaker but useful property, that the above diagram commutes for maps $M(f)$, for any $f : X \rightarrow Y$. In other words, such maps are homomorphisms $M(f) : M(X) \rightarrow M(Y)$, if we regard $M(X)$ and $M(Y)$ as algebras equipped with (any) corresponding algebraic operation components.

We will generally obtain the algebraic operations we need via their generic effects. When O is a product $O_1 \times \dots \times O_m$, we obtain semantically useful functions

$$\text{op}_X^\dagger : (M(O_1) \times \dots \times M(O_m)) \times M(X)^I \rightarrow M(X)$$

from an algebraic operation

$$\text{op}_X : (O_1 \times \dots \times O_m) \times M(X)^I \rightarrow M(X)$$

This can be done by applying iterated Kleisli extension to the curried version

$$\mathbf{op}'_X : O_1 \rightarrow \dots O_m \rightarrow M(X)^I \rightarrow M(X)$$

of \mathbf{op} , or, equivalently, using Kleisli extension and the monoidal structure

$$(m_T)_{X,Y} : M(X) \times M(Y) \rightarrow M(X \times Y)$$

induced by the monadic strength (see [31]).

When $O = \mathbb{1}$, we may ignore it and equivalently take $g \in M(I)$ and $\mathbf{op}_X : M(X)^I \rightarrow M(I)$, and write $g : I$ and $\mathbf{op} : I$; Note that (I, O) -ary algebraic operations \mathbf{op} are in an evident correspondence with indexed families \mathbf{op}_o ($o \in O$) of I -ary algebraic operations; in particular, when $O = [n]$ (as usual, $[n] = \{i \mid i < n\}$), the \mathbf{op}_o can be considered to be families $(\mathbf{op}_o)_X : M(X)^n \rightarrow M(X)$ of n -ary functions (here, and below, it is convenient to confuse $[n]$ with n).

We say that algebraic operations $\mathbf{op}_1 : [n_1], \dots, \mathbf{op}_k : [n_k]$ *satisfy* equations over function symbols $f_1 : n_1, \dots, f_k : n_k$ iff for any X , $(\mathbf{op}_1)_X, \dots, (\mathbf{op}_k)_X$ do, in the usual sense, i.e., if the equations hold with f_i interpreted as $(\mathbf{op}_i)_X$ for $i = 1, k$. In the case where M is the free-algebra monad for an equational theory Th with function symbols $\mathbf{op} : n$ of given arity, the $\mathbf{op}_X : M(X)^n \rightarrow M(X)$ form $[n]$ -ary algebraic operations (indeed, in this case all algebraic operations occur as combinations of these ones). These algebraic operations satisfy all the equations of Th .

Given a monad morphism $\theta : M \rightarrow M'$, any generic effect $g : O \rightarrow M(I)$ yields a generic effect $g' = \theta_I \circ g$ for M' . Then, see [28], θ is a homomorphism of the corresponding algebraic operations, $\mathbf{op}_X : O \times M(X)^I \rightarrow M(X)$ and $\mathbf{op}'_X : O \times M'(X)^I \rightarrow M'(X)$ in the sense that, for all sets X , the following diagram commutes:

$$\begin{array}{ccc} O \times M(X)^I & \xrightarrow{\mathbf{op}_X} & M(X) \\ \downarrow O \times (\theta_X)^I & & \downarrow \theta_X \\ O \times M'(X)^I & \xrightarrow{\mathbf{op}'_X} & M'(X) \end{array}$$

Given an M -algebra, $\alpha : M(X) \rightarrow X$, and an M -algebraic operation $\mathbf{op} : (I, O)$ we can induce a corresponding map $\mathbf{op}_\alpha : X^I \rightarrow X^O$, by setting

$$\mathbf{op}_\alpha = \alpha^O \circ \mathbf{op}_{M(X)} \circ (\eta_M)^I$$

and α is then a homomorphism between $\mathbf{op}_{M(X)}$ and the induced map. Given a collection of operations $\mathbf{op}_1 : [n_1], \dots, \mathbf{op}_k : [n_k]$, the corresponding induced maps satisfy the same equations the operations do. So, in particular, if M is the free-algebra monad for an equational theory Th with function symbols $\mathbf{op}_i : n_i$, X becomes a model of the theory via

the $(\mathbf{op}_i)_{M(X)}$. Conversely, if X is a model of the theory then we can define a corresponding M-algebra by setting $\alpha = \text{id}_X^{\dagger M}$. These two correspondences yield an isomorphism between the categories of M-algebras and models of the theory (the isomorphism is the identity on morphisms).

We next consider algebraic operations for the selection monad S_T . Modulo currying, $(I, O \times R^I)$ -ary generic effects $g : (O \times R^I) \rightarrow T(I)$ for T are in bijective correspondence with (I, O) -ary generic effects $\tilde{g} : O \rightarrow T(I)^{R^I}$ for S_T . There is therefore a corresponding bijective correspondence between $(I, O \times R^I)$ -ary algebraic operations \mathbf{op} for T and (I, O) -ary algebraic operations $\widetilde{\mathbf{op}}$ for S_T .

This correspondence has a pleasing componentwise expression going from T to S_T :

Proposition 1. *Let \mathbf{op} be an $(I, O \times R^I)$ -ary algebraic operation for T . The corresponding (I, O) -ary algebraic operation $\widetilde{\mathbf{op}}$ for S_T is given by:*

$$\widetilde{\mathbf{op}}_X(o, a) = \lambda \gamma \in R^X. \mathbf{op}_X(\langle o, \lambda i \in I. \theta_{S_T, K}(ai)\gamma \rangle, \lambda i \in I. ai\gamma)$$

Proof. The generic effect $g : O \times R^I \rightarrow T(I)$ corresponding to \mathbf{op} is given by:

$$g(o, \overline{\gamma}) = \mathbf{op}_I(\langle o, \overline{\gamma} \rangle, (\eta_T)_I)$$

Currying, we obtain $\tilde{g} : O \rightarrow S(I)$ where:

$$\tilde{g}(o) = \lambda \overline{\gamma} \in R^I. \mathbf{op}_I(\langle o, \overline{\gamma} \rangle, (\eta_T)_I)$$

and we have:

$$\widetilde{\mathbf{op}}_X(o, a) = a^{\dagger_{S_T}}(\tilde{g}(o))$$

We next choose a reward continuation $\gamma \in R^X$ and examine $a^{\dagger_{S_T}}(\tilde{g}(o))\gamma$. To this end we first obtain a reward continuation in R^I from a and γ , namely:

$$\overline{\gamma} =_{\text{def}} \lambda i \in I. \theta_{S_T, K}(ai)\gamma$$

and, setting

$$u =_{\text{def}} (\lambda i \in I. ai\gamma) \in I \rightarrow T(X)$$

we have:

$$\begin{aligned} a^{\dagger_{S_T}}(\tilde{g}(o))\gamma &= u^{\dagger_T}(\tilde{g}(o)(\overline{\gamma})) \\ &= u^{\dagger_T}(\mathbf{op}_I(\langle o, \overline{\gamma} \rangle, (\eta_T)_I)) \\ &= \mathbf{op}_X(\langle o, \overline{\gamma} \rangle, u^{\dagger_T}(\eta_T)_I) \quad (\text{by the Kleisli naturality of algebraic operations}) \\ &= \mathbf{op}_X(\langle o, \overline{\gamma} \rangle, u) \end{aligned}$$

Putting these facts together, we have:

$$\begin{aligned} (\widetilde{\mathbf{op}})_X(o, a)\gamma &= a^{\dagger_{S_T}}(g_{S_T}(o)) \\ &= \mathbf{op}(\langle o, \overline{\gamma} \rangle, u) \\ &= \mathbf{op}(\langle o, \lambda i \in I. \theta_{S_T, K}(ai)\gamma \rangle, \lambda i \in I. ai\gamma) \end{aligned}$$

as required. □

Note that, as is natural, $(\mathbf{op}_{S_T})_X$ uses the reward function in R^I which assigns to $i \in I$ the reward obtained by following the i th branch.

Using Proposition 1, we can reduce questions of equational satisfaction over S_T to corresponding questions over T . Fix an $(I, O \times R^I)$ -ary algebraic operation \mathbf{op} for T . For $o \in O$, X , and $\gamma : X \rightarrow R$ define an *auxiliary function* $\mathbf{aux}_{o,X,\gamma} : T(X)^I \rightarrow T(X)$ by:

$$\mathbf{aux}_{o,X,\gamma}(u) = \mathbf{op}(\langle o, \lambda i \in I. \alpha_T(T(\gamma)u_i) \rangle, u)$$

By the proposition we then have:

$$\widetilde{\mathbf{op}}_X(o, a)\gamma = \mathbf{aux}_{o,X,\gamma}(\lambda i \in I. ai\gamma)$$

Proposition 2. *Let \mathbf{op}_i be $(I_i, O_i \times R^{[n_i]})$ -ary T -algebraic operations, and choose $o_i \in O_i$ ($i = 1, n$). Then an equation is satisfied by $(\widetilde{\mathbf{op}}_1)_{o_1}, \dots, (\widetilde{\mathbf{op}}_n)_{o_n}$, iff, for all $X, \gamma : X \rightarrow R$, it is satisfied by $(\mathbf{aux}_1)_{o_1,X,\gamma}, \dots, (\mathbf{aux}_n)_{o_n,X,\gamma}$.*

We can use Proposition 1 to obtain (I, O) -ary algebraic operations $\widetilde{\mathbf{op}}$ for S_T from (I, O) -ary algebraic operations \mathbf{op} for T . Specifically, we obtain:

$$\widetilde{\mathbf{op}}_X(o, a) = \lambda \gamma \in R^X. \mathbf{op}_X(o, \lambda i \in I. ai\gamma)$$

which is the natural pointwise definition. In the case $I = [n]$ this can be written as:

$$\widetilde{\mathbf{op}}_X(o, F_1, \dots, F_n) = \lambda \gamma \in R^X. \mathbf{op}_X(o, F_1\gamma, \dots, F_n\gamma) \quad (3)$$

So we can use algebraic operations for T -effects to obtain corresponding ones for S_T -effects. In particular, as can be seen from a straightforward pointwise argument, if T is the free algebra monad for an equational theory Th , the algebraic operations for S_T we obtain satisfy all the equations of Th .

Another way to obtain algebraic operations is to start from the basic selection monad S . Consider an $(I, O \times R^I)$ -ary generic effect $g : O \times R^I \rightarrow I$ for the identity monad (equivalent via currying to an (I, O) -ary generic effect \tilde{g} for S). Viewing as a T -generic effect, via the unit for T , we obtain an $(I, O \times R^I)$ -ary algebraic operation \mathbf{op}_g for T where:

$$(\mathbf{op}_g)_X(\langle o, \bar{\gamma} \rangle, u) = u(g(o, \bar{\gamma})) \quad (o \in O, \bar{\gamma} \in R^I, u \in T(X)^I)$$

and, in turn, via Proposition 1, an (I, O) -ary algebraic operation $\widetilde{\mathbf{op}}_g$ for T where:

$$(\widetilde{\mathbf{op}}_g)_X(o, a)\gamma = a(g(o, \lambda i \in I. \theta_{S_T, K}(ai)\gamma))\gamma \quad (o \in O, a \in S_T(X)^I, \gamma \in R^X) \quad (4)$$

Say that a family of functions

$$f_X : O \times S_T(X)^I \rightarrow S_T(X)$$

is an (I, O) -ary-selection operation if for every $o \in O$, and every $\bar{\gamma} \in \mathbf{R}^I$ there is an $i_0 \in I$ such that for all sets X , $a \in \mathbf{S}_T(X)^I$, and $\gamma \in \mathbf{R}^X$:

$$\bar{\gamma} = \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\gamma \implies f_X(o, a)\gamma = ai_0\gamma$$

that is, each component \mathbf{op}_X of the family selects a fixed argument, depending only on the parameter $o \in P$ and the reward associated to each branch ai of f 's argument a . For example, $\widetilde{\mathbf{op}}_g$, as given by Equation 4, is such a selection operation, where the argument selected is the $g(o, \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\gamma)$ -th.

Theorem 1. *Every selection operation has the form $\widetilde{\mathbf{op}}_g$ for some basic selection monad generic effect $g : O \times \mathbf{R}^I \rightarrow I$.*

Proof. Let f be an (I, O) -ary-selection operation. This means that for every $o \in O$ and $\bar{\gamma} \in \mathbf{R}^I$, there is an $i_0 \in I$ such that for all sets X , $a \in \mathbf{S}_T(X)^I$, and $\gamma \in \mathbf{R}^X$:

$$\bar{\gamma} = \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\gamma \implies f_X(o, a)\gamma = ai\gamma$$

We consider the I component f_I . Fix $o \in O$ and $\bar{\gamma} \in \mathbf{R}^I$, and take $a(i) = (\eta_{\mathbf{S}_T})_I(i)$. Then we have:

$$\theta_{\mathbf{S}_T, \mathbf{K}}(ai)\bar{\gamma} = \alpha_T \circ T(\bar{\gamma})(ai\bar{\gamma}) = \alpha_T \circ T(\bar{\gamma})((\eta_T)_I(i)) = \alpha_T(\bar{\gamma}(i)) = \bar{\gamma}(i)$$

So $\bar{\gamma} = \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\bar{\gamma}$, and we therefore have:

$$f_I(o, a)\bar{\gamma} = ai_0\bar{\gamma} = (\eta_T)_I(i_0)$$

As monad units are always monos, we see there is only one such i_0 , and so there is a function $g : O \times \mathbf{R}^I \rightarrow I$ such that for all $o \in O$, $\bar{\gamma} \in \mathbf{R}^I$, sets X , $a \in \mathbf{S}_T(X)^I$, and $\gamma \in \mathbf{R}^X$:

$$\bar{\gamma} = \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\gamma \implies f_X(o, a)\gamma = ag(o, \bar{\gamma})\gamma$$

Substituting, we see that for all $o \in O$, sets X , $a \in \mathbf{S}_T(X)^I$, and $\gamma \in \mathbf{R}^X$:

$$f_X(o, a)\gamma = ag(o, \lambda i \in I. \theta_{\mathbf{S}_T, \mathbf{K}}(ai)\gamma)\gamma$$

But this identifies f as the algebraic operation $\widetilde{\mathbf{op}}_g$ arising by Proposition 1 from \mathbf{op}_g . \square

We next consider a particular case, binary selection algebraic operations. Here $O = \mathbb{1}$ and $I = [2]$, and $[2]$ -ary generics $g : \mathbf{R}^{[2]} \rightarrow [2]$ for the basic selection monad correspond to relations B on \mathbf{R} where:

$$rBs \equiv_{\text{def}} g(\bar{\gamma}) = 0$$

where $\bar{\gamma}(0) = r$ and $\bar{\gamma}(1) = s$ (read rBs as “ r beats s ”). We replace \mathbf{op}_g with the equivalent (O, \mathbf{R}^2) -ary T-algebraic operation \mathbf{op}_B , where:

$$\mathbf{op}_B(\langle o, \langle r_0, r_1 \rangle \rangle, u_0, u_1) = \mathbf{op}_g(\langle o, \lambda i \in [2]. r_i \rangle, u_0, u_1)$$

and we have:

$$\text{op}_B(\langle o, r_0, r_1 \rangle, u_0, u_1) = \begin{cases} u_0 & (r_0 B r_1) \\ u_1 & (\neg r_0 B r_1) \end{cases}$$

We write $\widetilde{\text{op}}_B$ for $\widetilde{\text{op}}_g$. In terms of B we have:

$$(\widetilde{\text{op}}_B)_X(G_0, G_1)(\gamma) = \begin{cases} G_0\gamma & \text{if } (\theta_{S_T, K} G_0 \gamma) B (\theta_{S_T, K} G_1 \gamma) \\ G_1\gamma & \text{otherwise} \end{cases}$$

In particular, for $x_0, x_1 \in X$ we have:

$$(\widetilde{\text{op}}_B)_X(\eta_{S_T}(x_0), \eta_{S_T}(x_1))(\gamma) = \begin{cases} \eta_T(x_0) & \text{if } \gamma(x_0) B \gamma(x_1) \\ \eta_T(x_1) & \text{otherwise} \end{cases}$$

Looking ahead to verifying equations, expressing the auxiliary $(\text{aux}_B)_{X, \gamma} : T(X)^2 \rightarrow T(X)$ in terms of B we have:

$$(\text{aux}_B)_{X, \gamma}(u_0, u_1) = \begin{cases} u_0 & \text{if } \alpha_T(T(\gamma)(u_0)) B \alpha_T(T(\gamma)(u_1)) \\ u_1 & \text{otherwise} \end{cases}$$

For optimization purposes it is natural to assume B is a linear order \geq . We set or to be the resulting binary algebraic operation on S_T ; it is this operation that we will use for the semantics of decision-making in our two languages.

Taking B to be a linear order is equivalent to using a version of argmax as a generic effect. For finite linearly ordered sets I , assuming a linear order \geq on \mathbb{R} , define $\text{argmax}_I \in S(I) = (I \rightarrow \mathbb{R})$, by taking $\text{argmax}_I \gamma$ to be the least $i \in I$ among those maximizing $\gamma(i)$. Then B corresponds to $\text{argmax}_{\mathbb{B}}$, with \mathbb{B} ordered by setting $0 < 1$. We could as well have used generics picking from any finite set, as in the example in Figure 1, with resulting choice functions of corresponding arity; however, binary choice is sufficiently illustrative. A binary maximization function will prove useful: for any $\gamma : X \rightarrow \mathbb{R}$ define $\text{max}_{X, \gamma} : X^2 \rightarrow X$ (written infix) by:

$$x \text{ max}_{X, \gamma} y = \begin{cases} x & \text{if } \gamma(x) \geq \gamma(y) \\ y & \text{otherwise} \end{cases}$$

Note that $\text{max}_{X, \gamma}$ is $(\text{aux}_B)_{X, \gamma}$, when T is the identity.

Say that a binary function f is *left-biased* if the following equation holds:

$$f(x, f(y, x)) = f(x, y)$$

and is *right-biased* if the following equation holds:

$$f(f(x, y), x) = f(y, x)$$

and say a relation R is *connex* iff, for all x, y , either $x R y$ or $y R x$. We say that an algebraic operation obeys an equation if every component does.

Theorem 2. *For every binary relation B on R we have:*

1. $\widetilde{\text{op}}_B$ is idempotent.
2. $\widetilde{\text{op}}_B$ is associative iff B and its complement is transitive.
3. $\widetilde{\text{op}}_B$ is left-biased iff B is connex.
4. $\widetilde{\text{op}}_B$ is right-biased iff the complement of B is connex.
5. $\widetilde{\text{op}}_B$ is not commutative (assuming R non-empty).

Proof. Throughout the proof, we use the fact that, like any monad unit, all components of η_{S_T} are 1-1.

1. This is evident.
2. (a) Suppose op_B is associative, and choose $r_0, r_1, r_2 \in R$. Set $f = (\text{op}_B)_{[2]}$, and $x_i = (\eta_{S_T})_{\mathbb{B}}(i)$ for $i = 0, 2$. Note that the x_i are all different. Define $\gamma : [2] \rightarrow B$ by: $\gamma(i) = r_i$, for $i = 0, 2$.
 Suppose first that r_0Br_1 and r_1Br_2 . As r_0Br_1 and r_1Br_2 , $f(x_0, f(x_1, x_2)) = f(x_0, x_1) = x_0$, and $f(f(x_0, x_1), x_2) = f(x_0, x_2)$. So, as f is associative $f(x_0, x_2) = x_0$. As $x_0 \neq x_2$, we have x_0Bx_2 , as required. Suppose next that $\neg r_0Br_1$ and $\neg r_1Br_2$. Then $f(x_0, f(x_1, x_2)) = f(x_0, x_2)$ and $f(f(x_0, x_1), x_2) = f(x_1, x_2) = x_2$, and, similarly to before, we conclude that $\neg r_0Br_2$.
 (b) For the converse, suppose that B and its complement is transitive. By Proposition 2 it suffices to prove that every $f = (\text{aux}_B)_{X, \gamma} : T(X)^2 \rightarrow T(X)$ is associative. Choose $u_i \in T(X)$ ($i = 0, 2$) and set $r_i = \alpha_T(T(\gamma)(u_i))$. The proof divides into cases according as each of r_0Br_1 and r_1Br_2 does or does not hold:

- i. Suppose that r_0Br_1 and r_1Br_2 (and so r_0Br_2). We then have:

$$f(u_0, f(u_1, u_2)) = f(u_0, u_1) = u_0 = f(u_0, u_2) = f(f(u_0, u_1), u_2)$$

- ii. Suppose that r_0Br_1 and $\neg r_1Br_2$. Then:

$$f(u_0, f(u_1, u_2)) = f(u_0, u_2) = f(f(u_0, u_1), u_2)$$

- iii. Suppose that $\neg r_0Br_1$ and r_1Br_2 . Then:

$$f(u_0, f(u_1, u_2)) = f(u_0, u_1) = u_1 = f(u_1, u_2) = f(f(u_0, u_1), u_2)$$

- iv. Suppose that $\neg r_0Br_1$ and $\neg r_1Br_2$. Then $\neg r_0Br_2$, and we have:

$$f(u_0, f(u_1, u_2)) = f(u_0, u_2) = u_2 = f(u_1, u_2) = f(f(u_0, u_1), u_2)$$

So in all cases we have

$$f(u_0, f(u_1, u_2)) = f(f(u_0, u_1), u_2)$$

and so f is associative, as required.

3. (a) Suppose op_B is left-biased and choose $r_0, r_1 \in R$. Set $f = (\text{op}_B)_{\mathbb{B}}$, and $x_i = (\eta_{S_T})_{\mathbb{B}}(i)$ for $i = 0, 1$. Note that $x_0 \neq x_1$. Define $\gamma : \mathbb{B} \rightarrow B$ by: $\gamma(i) = r_i$, for $i = 0, 1$. Suppose that $\neg r_1 Br_0$. Then we have:

$$f(x_0, x_1) = f(x_0, f(x_1, x_0)) = f(x_0, x_0) = x_0$$

and so, as $x_0 \neq x_1$, $r_0 Br_1$.

- (b) For the converse, suppose that B is connex. By Proposition 2 it suffices to prove that every $f = (\text{aux}_B)_{X, \gamma} : T(X)^2 \rightarrow T(X)$ is left-biased. Choose $u_i \in T(X)$ ($i = 0, 1$) and set $r_i = \alpha_T(T(\gamma)(u_i))$. Suppose first that $r_1 Br_0$ holds. Then $f(u_0, f(u_1, u_0)) = f(u_0, u_1)$. Otherwise, as B is connex, we have $\neg r_1 Br_0$ and $r_0 Br_1$, and so, $f(u_0, f(u_1, u_0)) = f(u_0, u_0) = u_0 = f(u_0, u_1)$. So in either case we have $f(u_0, f(u_1, u_0)) = f(u_0, u_1)$ as required.
4. (a) Suppose op_B is right-biased and choose $r_0, r_1 \in R$. Set $f = (\text{op}_B)_{\mathbb{B}}$, and $x_i = (\eta_{S_T})_{\mathbb{B}}(i)$ for $i = 0, 1$. Note that $x_0 \neq x_1$. Define $\gamma : \mathbb{B} \rightarrow B$ by: $\gamma(i) = r_i$, for $i = 0, 1$. Suppose that $r_0 Br_1$. Then we have:

$$f(x_1, x_0) = f(f(x_0, x_1), x_0) = f(x_0, x_0) = x_0$$

and so, as $x_0 \neq x_1$, $\neg r_1 Br_0$.

- (b) For the converse, suppose that $\neg B$ is connex. By Proposition 2 it suffices to prove that every $f = (\text{aux}_B)_{X, \gamma} : T(X)^2 \rightarrow T(X)$ is right-biased. Choose $u_i \in T(X)$ ($i = 0, 1$) and set $r_i = \alpha_T(T(\gamma)(u_i))$. Suppose first that $r_0 \neg Br_1$ holds. Then $f(f(x_0, x_1), x_0) = f(x_1, x_0)$.

Otherwise, as B is connex, we have $r_0 Br_1$ and $\neg r_1 Br_0$, and so, $f(u_0, f(u_1, u_0)) = f(u_0, u_0) = u_0 = f(u_0, u_1)$. So in either case we have $f(u_0, f(u_1, u_0)) = f(u_0, u_1)$ as required.

Suppose first that $r_1 Br_0$ holds. Then

$$f(u_0, f(u_1, u_0)) = f(u_0, u_1)$$

Otherwise, as B is connex, we have $\neg r_1 Br_0$ and $r_0 Br_1$, and so:

$$f(u_0, f(u_1, u_0)) = f(u_0, u_0) = u_0 = f(u_0, u_1)$$

So in either case we have

$$f(u_0, f(u_1, u_0)) = f(u_0, u_1)$$

$$f(f(x, y), x) = f(y, x)$$

as required.

5. Choose $r \in R$, set $f = (\text{op}_B)_\mathbb{B}$, $x_i = (\eta_{S_T})(i)$, for $i = 0, 1$, and define $\gamma : \mathbb{B} \rightarrow R$ by setting $\gamma(0) = \gamma(1) = r$. Note that $x_0 \neq x_1$. In case rBr holds, we have:

$$f(x_0, x_1)(\gamma) = x_0 \neq x_1 = f(x_1, x_0)(\gamma)$$

In case rBr does not hold, we have:

$$f(x_0, x_1)(\gamma) = x_1 \neq x_0 = f(x_1, x_0)(\gamma)$$

In either case $(\text{op}_B)_\mathbb{B}$ is not commutative.

□

Given a binary relation \mathbb{B} on R and an algebraic operation $\text{op} : [n]$ for S_T , we say that op *distributes* over $\widetilde{\text{op}}_B$ iff for all X , $i \in [n]$, $F_j \in S_T(X)$ ($j \in [n], j \neq i$), and $G_0, G_1 \in S_T(X)$, we have:

$$\begin{aligned} \text{op}_X(F_0, \dots, F_{i-1}, \widetilde{\text{op}}_B(G_0, G_1), F_{i+1}, \dots, F_{n-1}) = \\ \widetilde{\text{op}}_B(\text{op}_X(F_0, \dots, F_{i-1}, G_0, F_{i+1}, \dots, F_{n-1}), \text{op}_X(F_0, \dots, F_{i-1}, G_1, F_{i+1}, \dots, F_{n-1})) \end{aligned}$$

Also, given a binary relation B on R and a function $f : R^n$ we say that f *distributes* over B iff for all $0 \leq i < n$, $r_j \in R$ ($0 \leq j < n, j \neq i$), and $s_0, s_1 \in R$ we have:

$$s B t \iff f(r_0, \dots, r_{i-1}, s_0, r_{i+1}, \dots, r_{n-1}) B f(r_0, \dots, r_{i-1}, s_1, r_{i+1}, \dots, r_{n-1})$$

We remark that when B is a preorder, a function distributes over B iff it preserves and reflects the preorder.

Proposition 3. *Let $\text{op} : [n]$ be an algebraic operation $\text{op} : [n]$ for T and let B be a binary relation on R . If op_{α_T} distributes over B then $\widetilde{\text{op}}$ distributes over $\widetilde{\text{op}}_B$.*

Proof. To keep notation simple we suppose that $\widetilde{\text{op}}$ is binary and establish distributivity in its second argument. That is, we prove, for any X , that:

$$\widetilde{\text{op}}(F, \widetilde{\text{op}}_B(G, H)) = \widetilde{\text{op}}_B(\widetilde{\text{op}}(F, G), \widetilde{\text{op}}(F, H)) \quad (F, G, H \in S_T(X))$$

To do so we use Proposition 2 and establish the corresponding equation for the auxiliary functions of these operations. The auxiliary function $\text{aux}_{X, \gamma} : T(X)^2 \rightarrow T(X)$ of $\widetilde{\text{op}}$ is op_X . So we need to show for any $\gamma : X \rightarrow R$ that

$$\text{op}(u, (\text{aux}_B)_{X, \gamma}(v_0, v_1)) = (\text{aux}_B)_{X, \gamma}(\text{op}(u, v_0), \text{op}(u, v_1)) \quad (u, v_0, v_1 \in T(X))$$

From the definition of the auxiliary function of $\widetilde{\text{op}}_B$ we see that each side of this equation is either $\text{op}(u, v_0)$ or $\text{op}(u, v_1)$, and that the LHS is $\text{op}(u, v_0)$ iff

$$\alpha_T(T(\gamma)(v_0)) B \alpha_T(T(\gamma)(v_1)) \quad (*)$$

and that the RHS is $\text{op}(u, v_0)$ iff

$$\alpha_T(T(\gamma)(\text{op}(u, v_0))) B \alpha_T(T(\gamma)(\text{op}(u, v_1))) \quad (**)$$

As both $T(\gamma)$ and α_T are homomorphisms, this last condition is equivalent to:

$$\text{op}_\alpha(\alpha_T(T(\gamma)(u), \alpha_T(T(\gamma)(v_0)))) B \text{op}_\alpha(\alpha_T(T(\gamma)(u), \alpha_T(T(\gamma)(v_1))))$$

and we see, using the fact B distributes over op_{α_T} , that the conditions $(*)$ and $(**)$ are equivalent. \square

3 A general language with algebraic operations

The goal of this section is to give some definitions and results, in particular an adequacy theorem, for a general language with algebraic operations. We treat our two languages of later sections as instances of this language via such algebraic operations.

3.1 Syntax

We make use of a standard call-by-value λ -calculus equipped with algebraic operations. Our language is a convenient variant of the one in [37] (itself a variant of Moggi's computational λ -calculus [36]). The somewhat minor differences are that we allow a variety of basic types, our algebraic operations may have parameters, and we make use of general big-step transition relations as well as small-step ones.

Specifically, the types σ, \dots of the language are given by:

$$\sigma ::= b \mid \text{Unit} \mid \sigma \times \sigma \mid \sigma \rightarrow \sigma$$

and the terms L, M, N, \dots are given by:

$$\begin{aligned} M ::= & x \mid c \mid f(M_1, \dots, M_m) \mid \text{if } L \text{ then } M \text{ else } N \mid \\ & \text{op}(N_1, \dots, N_n; M_1, \dots, M_m) \mid \\ & * \mid \langle M, N \rangle \mid \text{fst}(M) \mid \text{snd}(M) \mid \lambda x \in \sigma. M \mid MN \end{aligned}$$

The types and terms are built from:

- a *basic vocabulary*, consisting of:

1. *basic types*, b (including **Bool**);

2. *constants*, $c : b$ of given basic type b (including $\texttt{tt}, \texttt{ff} : \text{Bool}$); and
3. first-order *function symbols*, $f : b_1 \dots b_m \rightarrow b$, of given arity $b_1 \dots b_m$ and co-arity b (including equality symbols $=_b : b \times b \rightarrow \text{Bool}$),

together with

- *algebraic operation symbols* $\text{op} : b_1 \dots b_n; m$, with given parameter basic types $b_1 \dots b_n$ and arity $m \in \mathbb{N}$.

We write BTypes for the set of basic types and Con_b for the set of constants of type b . The languages considered in the next two sections provide examples of this general setup.

We work up to α -equivalence, as usual, and free variables and substitution are also defined as usual. The typing rules are standard, and omitted, except for that for the algebraic operation symbols, which, aside from their parameters, are polymorphic:

$$\frac{\Gamma \vdash N_1 : b_1, \dots, \Gamma \vdash N_n : b_n \quad \Gamma \vdash M_1 : \sigma, \dots, \Gamma \vdash M_m : \sigma}{\Gamma \vdash \text{op}(N_1, \dots, N_n; M_1, \dots, M_m) : \sigma} \quad (\text{op} : b_1 \dots b_n; m)$$

where $\Gamma = x_1 : \sigma_1, \dots, x_n : \sigma_n$ is an environment. We write $M : \sigma$ for $\vdash M : \sigma$ and say then that the (closed) term M is *well-typed*; such terms are the *programs* of our language. We employ standard notation, for example writing $\text{let } x : \sigma \text{ be } M \text{ in } N$ for $(\lambda x : \sigma. N)M$. We also use a cases form **cases** $M_1 \Rightarrow N_1 \mid \dots \mid M_n \Rightarrow N_n$ **else** N_{n+1} (for $n \geq 0$), defined by iterated conditionals (the M_i are boolean).

3.2 Operational semantics

The operational semantics of programs is given in three parts: a small-step semantics, a big-step semantics, and an evaluation function. We make use of evaluation contexts, following [20]. The set of *values* V, W, \dots is given by:

$$V ::= c \mid * \mid \langle V, W \rangle \mid \lambda x : \sigma. M$$

where we restrict $\lambda x : \sigma. M$ to be closed. We write Val_σ for the set of values of type σ .

The *evaluation contexts* are given by:

$$\begin{aligned} \mathcal{E} ::= & [] \mid f(V_1, \dots, V_{k-1}, \mathcal{E}, M_{k+1}, \dots, M_m) \mid \text{if } \mathcal{E} \text{ then } M \text{ else } N \mid \\ & \text{op}(V_1, \dots, V_{k-1}, \mathcal{E}, N_{k+1}, \dots, N_n; M_1, \dots, M_m) \\ & \langle \mathcal{E}, N \rangle \mid \langle V, \mathcal{E} \rangle \mid \text{fst}(\mathcal{E}) \mid \text{snd}(\mathcal{E}) \mid \mathcal{E}N \mid (\lambda x : \sigma. M)\mathcal{E} \end{aligned}$$

and are restricted to be closed. The *redexes* are defined by:

$$\begin{aligned} R ::= & f(c_1, \dots, c_m) \mid \text{if } \texttt{tt} \text{ then } M \text{ else } N \mid \text{if } \texttt{ff} \text{ then } M \text{ else } N \mid \\ & \text{op}(c_1, \dots, c_n; M_1, \dots, M_m) \\ & \text{fst}(\langle M, N \rangle) \mid \text{snd}(\langle M, N \rangle) \mid (\lambda x : \sigma. M)V \end{aligned}$$

and are restricted to be closed. Any program is of one of two mutually exclusive forms: it is either a value V or else has the form $\mathcal{E}[R]$ for a unique evaluation context \mathcal{E} and a unique redex R .

We define two small-step transition relations on redexes, *ordinary* transitions and algebraic operation symbol transitions:

$$R \rightarrow N \quad \text{and} \quad R \xrightarrow[\text{op}_i]{c_1, \dots, c_n} N \quad (\text{op} : b_1 \dots b_n; m \text{ and } i = 1, m)$$

The idea of the algebraic operation symbol transitions is to indicate with which parameters an operation is being executed, and which of its arguments is then being followed. The definition of the first kind of transition is standard; we just mention that for each function symbol $f : b_1 \dots b_m \rightarrow b$ and constants $c_1 : b_1, \dots, c_m : b_m$, we assume given a constant $\text{val}_f(c_1, \dots, c_m) : b$, where, in the case of equality, we have:

$$\text{val}_{=b}(c_1, c_2) = \begin{cases} \text{tt} & (\text{if } c_1 = c_2) \\ \text{ff} & (\text{otherwise}) \end{cases}$$

We then have the ordinary transitions:

$$f(c_1, \dots, c_m) \rightarrow c \quad (\text{val}_f(c_1, \dots, c_m) = c)$$

The algebraic operation symbol transition relations are given by the following rule:

$$\text{op}(c_1, \dots, c_n; M_1, \dots, M_m) \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i$$

We can then extend these transition relations to corresponding ordinary and algebraic operation symbol transition relations on programs

$$M \rightarrow M' \quad \text{and} \quad M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M'$$

To do so, we use evaluation contexts in a standard way by means of the following rules:

$$\frac{R \rightarrow M'}{\mathcal{E}[R] \rightarrow \mathcal{E}[M']} \quad \frac{R \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M'}{\mathcal{E}[R] \xrightarrow[\text{op}_i]{c_1, \dots, c_n} \mathcal{E}[M']}$$

These transition relations are all deterministic.

For any program M which is not a value, exactly one of two mutually exclusive possibilities holds:

- For some program M'

$$M \rightarrow M'$$

In this case M' is determined and of the same type as M .

- For some $\text{op} : b_1 \dots b_n; m$ and $c_1 : b_1, \dots, c_n : b_n$

$$M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i$$

for all $i = 1, n$ and some M_i . In this case op , the c_j and the M_i are uniquely determined and the M_i have the same type as M .

We say a program M is *terminating* if there is no infinite chain of (small-step) transitions from M .

Lemma 2. *Every program is terminating.*

Proof. This is a standard computability argument; see the proof of Theorem 1 in [37] for some detail. One defines a computability predicate on values, by induction on types, and then extends it to well-typed terms by taking them to be computable if there is no infinite chain of (small-step) transitions from M , and every terminating sequence of small-step transitions from M ends in a computable value. (See the proof of Theorem 1 in [37] for more detail.) \square

Using the small-step relations one defines big-step ordinary and algebraic operation symbol transition relations by:

$$\frac{M \rightarrow^* V}{M \Rightarrow V} \quad \frac{M \rightarrow^* M' \quad M' \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M''}{M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M''}$$

For any program M which is not a value, exactly one of two mutually exclusive possibilities holds:

- For some value V

$$M \Rightarrow V$$

In this case V is determined and of the same type as M .

- For some $\text{op} : b_1 \dots b_n; m$ and $c_1 : b_1, \dots, c_n : b_n$

$$M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i$$

for all $i = 1, n$ and some N_i . In this case op , the c_j and the M_i are uniquely determined and the M_i have the same type as M .

The big-step transition relations from a given program M form a finite tree with values at the leafs, all transitions, except for those leading to values being algebraic operation symbol transitions, and with algebraic operation symbol transitions of type $(w; n)$ branching n -fold. We write $\|M\|$ for the height of this tree.

Rather than use trees, we follow [37] and use *effect values* E . These give the same information and, conveniently, form a subset of our programs. They are defined as follows:

$$E ::= V \mid \text{op}(c_1, \dots, c_n; E_1, \dots, E_m)$$

(Our effect values are a finitary version of the interaction trees of [43]). Every program $M : \sigma$ can be given an effect value $\text{Op}(M) : \sigma$ defined as follows using the big-step transition relations:

$$\text{Op}(M) = \begin{cases} V & (\text{if } M \Rightarrow V) \\ \text{op}(c_1, \dots, c_n; \text{Op}(M_1), \dots, \text{Op}(M_m)) & (\text{if } M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i \text{ for } i = 1, m) \end{cases}$$

This definition can be justified by induction on $\|M\|$. Note that $\text{Op}(E) = E$, for any effect value $E : \sigma$. Note too that the transitions of programs and their evaluations closely parallel each other, indeed we have:

$$M \Rightarrow V \iff \text{Op}(M) = V \quad (5)$$

and

$$M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i \iff \text{Op}(M) \xrightarrow[\text{op}_i]{c_1, \dots, c_n} \text{Op}(M_i) \quad (6)$$

We next give a proof-theoretic account of this evaluation function to help us prove the basic adequacy of our operational semantics. There is a natural equational theory for the operational semantics, with evident rules, which establishes judgments of the form $\vdash_o M = N : \sigma$, well formed in case $M : \sigma$ and $N : \sigma$. The axioms are the small-step reductions for the redexes together with a commutation schema that algebraic operations commute with evaluation contexts; they are given (omitting type information) in Figure 2.

$$\begin{aligned} f(c_1, \dots, c_m) = c & \quad (\text{val}_f(c_1, \dots, c_m) = c) \\ \text{if tt then } M \text{ else } N = M & \quad \text{if ff then } M \text{ else } N = N \\ \text{fst}(\langle V, W \rangle) = V & \quad \text{snd}(\langle V, W \rangle) = W \\ (\lambda x : \sigma. M)V = M[V/x] & \\ \mathcal{E}[\text{op}(c_1, \dots, c_n; M_1, \dots, M_m)] = \text{op}(c_1, \dots, c_n; \mathcal{E}[M_1], \dots, \mathcal{E}[M_m]) & \end{aligned}$$

Figure 2: Axioms

We then have:

Lemma 3. *For any well-typed term $M : \sigma$ we have:*

1.

$$M \rightarrow M' \implies \vdash_o M = M' : \sigma$$

2.

$$M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i, \text{ for } i = 1, m \implies \vdash_o M = \text{op}(c_1, \dots, c_n; M_1, \dots, M_m) : \sigma$$

3.

$$M \Rightarrow V \implies \vdash_o M = V : \sigma$$

4.

$$M \xrightarrow[\text{op}_i]{c_1, \dots, c_n} M_i, \text{ for } i = 1, m \implies \vdash_o M = \text{op}(c_1, \dots, c_n; M_1, \dots, M_m) : \sigma$$

The following proposition is an immediate consequence of this lemma:

Proposition 4. *For any program $M : \sigma$ we have:*

$$\vdash_o M = \text{Op}(M) : \sigma$$

A substitution lemma will prove useful later. Given any effect value $E : b$, a nonempty finite set $u \subseteq \text{Con}_b$ that includes all the constants of type b in E , and a function g from u to programs of type b' , define $E[g] : b'$, the substitution g of programs for constants, homomorphically by:

$$\begin{aligned} c[g] &= g(c) \\ \text{op}(c_1, \dots, c_n; E_1, \dots, E_m)[g] &= \text{op}(c_1, \dots, c_n; E_1[g], \dots, E_m[g]) \end{aligned}$$

Let c_1, \dots, c_n enumerate u (the order does not matter) and define $F_g : b \rightarrow b'$ to be

$$\lambda x : b. \text{cases } x = c_1 \Rightarrow g(c_1) \mid \dots \mid x = c_{n-1} \Rightarrow g(c_{n-1}) \text{ else } g(c_n)$$

With this notation we have:

Lemma 4.

$$\text{Op}(F_g E) = \text{Op}(E[g])$$

Proof. The proof is a structural induction on E . For E a constant c we have:

$$\begin{aligned} \text{Op}(F_g E) &= \text{Op}(\text{cases } c = c_1 \Rightarrow g(c_1) \mid \dots \mid c = c_{n-1} \Rightarrow g(c_{n-1}) \text{ else } g(c_n)) \\ &= \text{Op}(g(c)) \\ &= \text{Op}(c[g]) \end{aligned}$$

and for E of the form $\text{op}(c_1, \dots, c_n; E_1, \dots, E_m)$ we have:

$$\begin{aligned}
\text{Op}(F_g E) &= \text{op}(c_1, \dots, c_n; \text{Op}(F_g E_1), \dots, \text{Op}(F_g E_m)) \\
&= \text{op}(c_1, \dots, c_n; \text{Op}(E_1[g]), \dots, \text{Op}(E_m[g])) \\
&= \text{Op}(\text{op}(c_1, \dots, c_n; E_1[g], \dots, E_m[g])) \\
&= \text{Op}(\text{op}(c_1, \dots, c_n; E_1, \dots, E_m)[g]) \\
&= \text{Op}(E[g])
\end{aligned}$$

□

3.3 Denotational semantics

The semantics of our language makes use of a given strong monad, following that of Moggi's computational λ -calculus [36]. In order to be able to give semantics to effectual operations we use the apparatus of generic effects and algebraic operations as discussed above. For the sake of simplicity we work in the category of sets, although the results go through much more generally, for example in a cartesian closed category with binary sums.

To give the semantics of our language a number of ingredients are needed. We assume given:

- a (necessarily) strong monad M on the category of sets,
- nonempty sets $\llbracket b \rrbracket$ for the basic types b (with $\llbracket \text{Bool} \rrbracket = \mathbb{B} =_{\text{def}} \{0, 1\}$),
- elements $\llbracket c \rrbracket$ of $\llbracket b \rrbracket$ for constants $c : b$ (with $\llbracket \text{tt} \rrbracket = 1$ and $\llbracket \text{ff} \rrbracket = 0$),
- functions $\llbracket f \rrbracket : \llbracket b_1 \rrbracket \times \dots \times \llbracket b_m \rrbracket \rightarrow \llbracket b \rrbracket$ for function symbols $f : b_1 \dots b_m \rightarrow b$, and
- generic effects

$$g_{\text{op}} : \llbracket b_1 \rrbracket \times \dots \times \llbracket b_m \rrbracket \rightarrow M([n])$$

for algebraic operation symbols $\text{op} : b_1 \dots b_n; m$.

We further assume that different constants receive different denotations (so we can think of constants as just names for their denotations, similarly to how one thinks of numerals), and that the given denotations of function symbols are consistent with their operational semantics in that:

$$\text{val}_f(c_1, \dots, c_m) = c \implies \llbracket f \rrbracket(\llbracket c_1 \rrbracket, \dots, \llbracket c_m \rrbracket) = \llbracket c \rrbracket$$

With these ingredients, we can give our language its semantics. Types are interpreted by putting:

$$\begin{aligned}
\mathcal{M}[\llbracket b \rrbracket] &= \llbracket b \rrbracket \\
\mathcal{M}[\llbracket \sigma \times \tau \rrbracket] &= \mathcal{M}[\llbracket \sigma \rrbracket] \times \mathcal{M}[\llbracket \tau \rrbracket] \\
\mathcal{M}[\llbracket \sigma \rightarrow \tau \rrbracket] &= \mathcal{M}[\llbracket \sigma \rrbracket] \rightarrow M(\mathcal{M}[\llbracket \tau \rrbracket])
\end{aligned}$$

To every term

$$\Gamma \vdash N : \sigma$$

we associate a function

$$\mathcal{M}[\Gamma \vdash N : \sigma] : \mathcal{M}[\Gamma] \rightarrow \mathbf{M}(\mathcal{M}[\sigma])$$

where $\mathcal{M}[x_1 : \sigma_1, \dots, x_n : \sigma_n] =_{\text{def}} \mathcal{M}[\sigma_1] \times \dots \times \mathcal{M}[\sigma_n]$. When the typing $\Gamma \vdash N : \sigma$ is understood, we generally write $\mathcal{M}[N]$ rather than $\mathcal{M}[\Gamma \vdash N : \sigma]$.

The semantic clauses for conditionals and the product and function space terms are standard, and we omit them. For constants $c : b$ we put:

$$\mathcal{M}[c](\rho) = (\eta_{\mathbf{M}})_{[b]}([c])$$

For function symbol applications $f(M_1, \dots, M_m)$, where $f : b_1 \dots b_m \rightarrow b$, we put:

$$\mathcal{M}[f(M_1, \dots, M_m)](\rho) = [f]^\sim([M](M_1)(\rho), \dots, [M](M_m)(\rho))$$

where

$$[f]^\sim : \mathbf{M}([b_1]) \times \dots \times \mathbf{M}([b_m]) \rightarrow \mathbf{M}([b])$$

is obtained from $[f]$ in a standard way via Kleisli extension and the monadic monoidal structure. For terms $\Gamma \vdash \text{op}(N_1, \dots, N_n; M_1, \dots, M_m) : \sigma$, where $\text{op} : b_1 \dots b_n; m$, we make use of the algebraic operation

$$\text{op}_X : ([b_1] \times \dots \times [b_n]) \times \mathbf{M}(X)^{[m]} \rightarrow \mathbf{M}(X)$$

corresponding to the generic effects g_{op} and put:

$$\begin{aligned} \mathcal{M}[\text{op}(N_1, \dots, N_n; M_1, \dots, M_m)](\rho) = \\ \text{op}_{[\sigma]}^\dagger(\langle \mathcal{M}[N_1](\rho), \dots, \mathcal{M}[N_n](\rho) \rangle \langle \mathcal{M}[M_1](\rho), \dots, \mathcal{M}[M_m](\rho) \rangle) \end{aligned}$$

We further give values $V : \sigma$ an *effect-free* semantics $[V]_\sigma \in \mathcal{M}[\sigma]$, extending that for constants:

$$\begin{aligned} \langle [V, V'] \rangle_{\tau \times \tau'} &= \langle [V]_\tau, [V']_{\tau \times \tau'} \rangle \\ [\lambda x : \tau. N]_{\tau \rightarrow \tau'} &= \mathcal{M}[x : \tau \vdash N : \tau'] \end{aligned}$$

This effect-free semantics of values $V : \sigma$ determines their denotational semantics:

$$\mathcal{M}[V](\rho) = (\eta_{\mathbf{M}})_{\mathcal{M}[\sigma]}([V]_\sigma)$$

Below, we may regard the effect-free semantics as providing functions:

$$[-]_\sigma : \text{Val}_\sigma \rightarrow \mathcal{M}[\sigma]$$

We may also drop the type subscript.

3.4 Adequacy

Our proof system is consistent relative to our denotational semantics:

Lemma 5. *If $\vdash_o M = N : \sigma$ then $\mathcal{M}[[M]] = \mathcal{M}[[N]]$.*

The naturality condition with respect to Kleisli morphisms for algebraic operations is used here to establish the soundness of the commutation schema.

Our basic adequacy theorem now follows immediately from Proposition 4 and Lemma 5:

Theorem 3. *For any program N we have: $\mathcal{M}[[N]] = \mathcal{M}[[\text{Op}(N)]]$.*

This adequacy theorem differs from the usual ones where the denotational semantics determines termination and the denotation of any final result; further, for basic types it generally determines the value produced by the operational semantics. In our case the first part is not relevant as terms always terminate. We do have that the denotational semantics determines the denotation of any final result. For basic types (as at any type) it determines the effect values produced up to their denotation, though the extent of that determination will depend on the choice of the generic effects.

3.5 Program equivalences and purity

The equational system in Section 3.2, helps prove adequacy, but is too weak to provide a means of reasoning about our calculus. Moggi gave a suitable consistent and complete system for his computational λ -calculus in [36]. His system has equational and purity (meaning effect-free) assertions $\Gamma \vdash M = N : \sigma$ and $\Gamma \vdash M \downarrow_\sigma$; we always assume that the terms are appropriately typed, and may omit types or environments when the context makes them clear. One can substitute a term M for a variable only if one can prove $M \downarrow_\sigma$.

Our λ -calculus is an extension of Moggi's and we extend his logic correspondingly; an alternate approach, well worth pursuing, would be to use instead the purely equational fine-grained variant of the computational λ -calculus: see [32]. We keep Moggi's axioms and rules, but extended to our language.

For conditionals we add:

$$\text{if } \text{tt} \text{ then } M \text{ else } N = M \quad \text{if } \text{ff} \text{ then } M \text{ else } N = N$$

$$f(x) = \text{if } x \text{ then } f(\text{tt}) \text{ else } f(\text{ff})$$

For the algebraic operations we add two equations, one:

$$g(\text{op}(y_1, \dots, y_n; M_1, \dots, M_n)) = \text{op}(y_1, \dots, y_n; g(M_1), \dots, g(M_n)) \quad (7)$$

expressing their naturality (and generalizing the commutation schema of Figure 2), and the other:

$$\text{op}(N_1, \dots, N_n; M_1, \dots, M_m) = \begin{array}{l} \text{let } y_1 : b_1, \dots, y_n : b_n \text{ be } N_1, \dots, N_n \\ \text{in op}(y_1, \dots, y_n; M_1, \dots, M_m) \end{array} \quad (\text{no } y_j \text{ in any FV}(M_i)) \quad (8)$$

expressing the order of evaluation of the parameter arguments of `op`. For function symbols and constants we add the purity axiom $c \downarrow$ and the equation in Figure 2. This equation enables us to evaluate function symbol applications to constants within our proof system. One could certainly add further axioms and even rules (e.g., that a function is commutative or a form of induction if the natural numbers were a basic type). However, such extensions are not needed for our purposes which are to establish completeness results for programs of basic type.

We write

$$\Gamma \vdash_{\text{Ax}} M = N : \sigma \text{ and } \Gamma \vdash_{\text{Ax}} M \downarrow_\sigma$$

to mean $M = N$ (resp. $M \downarrow_\sigma$) is provable from a given set of equational or purity axioms Ax (where $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$). In particular all the axioms of Figure 2 are provable. An equational or purity assertion is true (or holds) in \mathcal{M} , written $\Gamma \models_{\mathcal{M}} M = N : \sigma$ or $\Gamma \models_{\mathcal{M}} M \downarrow_\sigma$, if $\mathcal{M} \llbracket M \rrbracket = \mathcal{M} \llbracket N \rrbracket$ or $\exists a \in \mathcal{M} \llbracket \sigma \rrbracket. \mathcal{M} \llbracket M \rrbracket(\rho) = \eta_M(a)$, respectively; a *theory*, i.e., set of axioms, Ax is *valid* in \mathcal{M} if all the assertions in Ax are true in \mathcal{M} .

Equational consistency holds, meaning that, for Ax valid in \mathcal{M} :

$$\Gamma \vdash_{\text{Ax}} M = N : \sigma \implies \Gamma \models_{\mathcal{M}} M = N : \sigma$$

as does the analogous *purity consistency*.

We can use Ax to give axioms for the particular operations. For example, we will consider languages with a binary decision algebraic operation symbol `or` : $\varepsilon; 2$ and the associative axioms

$$(L \text{ or } M) \text{ or } N = L \text{ or } (M \text{ or } N)$$

hold as every component of the algebraic operation family employed is associative, as shown by Theorem 2.

4 A language of choices and rewards

Building on the framework of Section 3, in this section we define and study a language with constructs for choices and rewards.

4.1 Syntax

For the basic vocabulary of our language, in addition to the Boolean primitives of Section 3, we assume available: a basic type `Rew` with $\llbracket \text{Rew} \rrbracket = \mathbf{R}$; a constant $0 : \text{Rew}$; function symbols

$+$: $\text{Rew} \text{Rew} \rightarrow \text{Rew}$ and \leq : $\text{Rew} \text{Rew} \rightarrow \text{Bool}$. We further assume that \mathbb{R} is *expressively nontrivial* in that there is a $c : \text{Rew}$ with $\llbracket c \rrbracket \neq 0$. There are exactly two algebraic operation symbols: a *choice operation* $\text{or} : \varepsilon; 2$ to make choices, and a *reward operation* $\text{reward} : \text{Rew}; 1$, to prescribe rewards. We leave any other basic type symbols, constants, or function symbols unspecified.

We may use infix for $+$ and \leq . Similarly, we may use infix notations $M_0 \text{ or } M_1$ or $N \cdot M$ for the algebraic operation terms $\text{or}(\cdot; M_0, M_1)$ and $\text{reward}(N; M)$. The signature $\text{or} : \varepsilon; 2$ means that M_0 and M_1 must have the same type and that is then the type of $M_0 \text{ or } M_1$; the signature $\text{reward} : \text{Rew}; 1$ means $N \cdot M$ has the same type as M and that N must be of type Rew . For example, assuming that 5 and 6 are two constants of type Rew , we may write the tiny program:

$$(5 \cdot \text{tt}) \text{ or } (6 \cdot \text{ff}) : \text{Bool}$$

Intuitively, this program could potentially return either tt or ff , with respective rewards 5 and 6. In the intended semantics that maximizes rewards, then, the program returns ff with reward 6.

4.2 Rewards and additional effects

We assume that the linearly ordered set of rewards \mathbb{R} additionally has a commutative monoid structure, written additively, and that this addition preserves and reflects the order, in that:

$$r \leq s \iff r + t \leq s + t \quad (r, s, t \in \mathbb{R})$$

For example, \mathbb{R} could be the reals (or the nonnegative reals) with addition, or the positive reals with multiplication, in all cases with the usual order. We use the rewards, so equipped, for the monad employed to handle additional effects, as indicated in Section 2.

The so-called *writer* monad $\mathbf{W}(X) = \mathbb{R} \times X$, is defined using the commutative monoid structure on \mathbb{R} . It will prove useful for both our operational and denotational semantics. The operational semantics we will define evaluates programs M of type σ to pairs $\langle r, V \rangle$, with $r \in \mathbb{R}$ and $V : \sigma$, that is to elements of $\mathbf{W}(\text{Val}_\sigma)$. The writer monad is the free-algebra monad for \mathbb{R} -actions, i.e., algebras with an \mathbb{R} -indexed family of unary operations, which we write as $\text{reward}(r, -)$ or $r \cdot -$, satisfying the equations

$$0 \cdot x = x \quad r \cdot s \cdot x = (r + s) \cdot x \tag{9}$$

The resulting algebraic operation $(\text{reward}_\mathbf{W})_X : \mathbb{R} \times \mathbf{W}(X) \rightarrow \mathbf{W}(X)$ is given by:

$$(\text{reward}_\mathbf{W})_X(r, \langle s, x \rangle) = \langle r + s, x \rangle$$

and is induced by the generic effect $(g_\mathbf{W})_{\text{reward}} : \mathbb{R} \rightarrow \mathbf{W}(\mathbb{1})$, where $(g_\mathbf{W})_{\text{reward}}(r) =_{\text{def}} \langle r, * \rangle$. We generally write applications of $(\text{reward}_\mathbf{W})_X$ using an infix operator, $(\cdot_\mathbf{W})_X$, and, in either case, may drop subscripts when they can be understood from the context.

4.3 Operational semantics

While the operational semantics of Section 3 is ordinary and does not address optimization, the *selection operational semantics* selects an optimal choice strategy, as suggested in the Introduction. Below we prove an adequacy result relative to a denotational semantics using the selection monad S_W . We thereby give a compositional account of a global quantity: the optimal reward of a program.

We employ the version of argmax defined in Section 2: given a finite linearly-ordered set S and a reward function $\gamma : S \rightarrow \mathbb{R}$, $\text{argmax}_X(\gamma)$ selects the least $s \in S$ maximizing $\gamma(s)$. So, linearly ordering S by:

$$s \preceq_\gamma s' \iff (\gamma(s) > \gamma(s') \vee (\gamma(s) = \gamma(s') \wedge s \leq s'))$$

the selection is of the least element in this linear order. It will be convenient to use the notation $\text{argmax } s : S. e$ for $\text{argmax}(\lambda s \in S. e)$.

We next define strategies for effect values. The idea is to view an effect value $E : \sigma$ as a one-player game for Player. The subterms E' of E are the positions of the game. In particular:

- if E' is a value, then E is a final position and the reward is 0;
- if $E' = E'_0 \text{ or } E'_1$ then Player can choose whether to move to the position E'_0 or the position E'_1 ; and
- if $E' = c \cdot E'' : \sigma$ then Player moves to E'' and c is added to the final reward.

The finite set $\text{Str}(E)$ of strategies of an effect value E is defined by the following rules, writing $s : E$ for $s \in \text{Str}(E)$:

$$* : V \quad \frac{s : E_1}{1s : E_1 \text{ or } E_2} \quad \frac{s : E_2}{1s : E_1 \text{ or } E_2} \quad \frac{s : E}{s : c \cdot E}$$

These strategies can be reformulated as boolean functions on choice subterms; though standard, this is less convenient. Equivalently, one could work with boolean functions on choice nodes (terms) of the tree naturally associated to a term by the big-step reduction relation, noting that this tree is isomorphic to effect values considered as trees (as we see from equivalences 5,6). There are also probabilistic strategies, although, as is generally true for MDPs [4], they would not change the optimal expected reward.

For any effect value $E : \sigma$, the outcome $\text{Out}(s, E) \in \mathbb{R} \times \text{Val}_\sigma = W(\text{Val}_\sigma)$ of a strategy $s : E$ is defined by:

$$\begin{aligned} \text{Out}(*, V) &= \langle 0, V \rangle \quad (= \eta_W(V)) \\ \text{Out}(1s, E_1 \text{ or } E_2) &= \text{Out}(s, E_1) \\ \text{Out}(2s, E_1 \text{ or } E_2) &= \text{Out}(s, E_2) \\ \text{Out}(s, c \cdot E) &= \text{reward}_W(\llbracket c \rrbracket, \text{Out}(s, E)) \end{aligned}$$

We can then define the reward of such a strategy by:

$$\text{Rew}(s, E) = \pi_1(\text{Out}(s, E))$$

Note that $\pi_1 : \mathbb{R} \times X \rightarrow \mathbb{R}$ can be written as $\alpha_W \circ W(0)$, with 0 the constantly 0 reward function.

As there can be more than one strategy maximizing the reward on a game, we need a way of choosing between them. We therefore define a linear order \leq_E on the strategies of a game $E : \sigma$:

- Game is V :

$$* \leq_V *$$

- Game is $E_1 + E_2$:

$$(i, s) \leq_{E_1+E_2} (j, s') \iff \begin{array}{l} i < j \quad \vee \\ i = j = 1 \wedge s \leq_{E_1} s' \quad \vee \\ i = j = 2 \wedge s \leq_{E_2} s' \end{array}$$

- Game is $c \cdot E$:

$$s \leq_{c \cdot E} s' \iff s \leq_E s'$$

We can now give our selection operational semantics. We define $\text{Op}_s(M) \in \mathbb{R} \times \text{Val}_\sigma$ for $M : \sigma$ by:

$$\text{Op}_s(M) = \text{Out}(\text{argmax } s : \text{Op}(M). \text{Rew}(s, \text{Op}(M)), \text{Op}(M))$$

meaning that we take the $\text{Op}(M)$ -strategy maximizing the reward, and if there is more than one such strategy, we take the least, according to the $\text{Op}(M)$ -strategy linear order $\leq_{\text{Op}(M)}$. That is, we take

$$s_{\text{opt}} =_{\text{def}} \text{argmax } s : \text{Op}(M). \text{Rew}(s, \text{Op}(M))$$

and then

$$\text{Op}_s(M) = \text{Out}(s_{\text{opt}}, \text{Op}(M))$$

Note that $\text{Op}_s(M) = \text{Op}_s(\text{Op}(M))$ (this follows from the form of the definition of the optimizing operational semantics and the fact that $\text{Op}_s^2(M) = \text{Op}_s(M)$).

While the operational semantics has been defined by a global optimization over all strategies, it can be equivalently given locally without reference to any strategies. We need two lemmas, whose straightforward proofs we omit:

Lemma 6. *Given functions $X \xrightarrow{g} Y \xrightarrow{\gamma} \mathbb{R}$, for all $u, v \in X$ we have*

$$g(u \max_{\gamma \circ g} v) = g(u) \max_{\gamma} g(v)$$

Lemma 7. (*First argmax lemma*) Let $S_1 \cup S_2$ split a finite linear order $\langle S, \leq \rangle$ into two with $S_1 < S_2$ (the latter in the sense that $s_1 < s_2$ for all $s_1 \in S_1$ and $s_2 \in S_2$). Then, for all $\gamma : S \rightarrow \mathbb{R}$, we have:

$$\operatorname{argmax} \gamma = \operatorname{argmax}(\gamma|_{S_1}) \max_{\gamma} (\operatorname{argmax} \gamma|_{S_2})$$

We now have our local characterization of the operational semantics:

Theorem 4. For well-typed effect values we have:

1. $\operatorname{Op}_s(V) = \langle 0, V \rangle (= \eta_W(V))$
2. $\operatorname{Op}_s(E_1 \text{ or } E_2) = \operatorname{Op}_s(E_1) \max_{\pi_1} \operatorname{Op}_s(E_2) (= \operatorname{Op}_s(E_1) \max_{\alpha_W \circ W(0)} \operatorname{Op}_s(E_2))$.
3. $\operatorname{Op}_s(c \cdot E) = \text{reward}_W(\llbracket c \rrbracket, \operatorname{Op}_s(E))$

Proof. We calculate:

$$\begin{aligned} \operatorname{Op}_s(V) &= \operatorname{Out}(\operatorname{argmax} s : V. \operatorname{Rew}(s, V), V) \\ &= \operatorname{Out}(*, V) \\ &= \langle 0, V \rangle \end{aligned}$$

The first equality is as $\operatorname{Op}(V) = V$; the second is as values V have only one strategy, $*$.

We just prove parts 2 and 3. For part 2, we calculate:

$$\begin{aligned} \operatorname{Op}_s(E_1 \text{ or } E_2) &= \operatorname{Out}(\operatorname{argmax} s : E_1 \text{ or } E_2. \operatorname{Rew}(s, E_1 \text{ or } E_2), E_1 \text{ or } E_2) \\ &= \operatorname{Out} \left(\left(\begin{array}{c} \operatorname{argmax} 1s : E_1 \text{ or } E_2. \operatorname{Rew}(1s, E_1 \text{ or } E_2) \\ \max_{\operatorname{Rew}(-, E_1 \text{ or } E_2)} \\ \operatorname{argmax} 2s : E_1 \text{ or } E_2. \operatorname{Rew}(2s, E_1 \text{ or } E_2) \end{array} \right), E_1 \text{ or } E_2 \right) \\ &\quad \text{(by the first argmax lemma (Lemma 7))} \\ &= \operatorname{Out} \left(\left(\begin{array}{c} \operatorname{argmax} 1s : E_1 \text{ or } E_2. \operatorname{Rew}(s, E_1) \\ \max_{\pi_1 \circ \operatorname{Out}(-, E_1 \text{ or } E_2)} \\ \operatorname{argmax} 2s : E_1 \text{ or } E_2. \operatorname{Rew}(s, E_2) \end{array} \right), E_1 \text{ or } E_2 \right) \\ &= \begin{array}{c} \operatorname{Out}(\operatorname{argmax} 1s : E_1 \text{ or } E_2. \operatorname{Rew}(s, E_1), E_1 \text{ or } E_2) \\ \max_{\pi_1} \\ \operatorname{Out}(\operatorname{argmax} 2s : E_1 \text{ or } E_2. \operatorname{Rew}(s, E_2), E_1 \text{ or } E_2) \end{array} \\ &\quad \text{(by Lemma 6)} \\ &= \begin{array}{c} \operatorname{Out}(\operatorname{argmax} s : E_1. \operatorname{Rew}(s, E_1), E_1) \\ \max_{\pi_1} \\ \operatorname{Out}(\operatorname{argmax} s : E_2. \operatorname{Rew}(s, E_2), E_2) \end{array} \\ &= \operatorname{Op}_s(E_1) \max_{\pi_1} \operatorname{Op}_s(E_2) \end{aligned}$$

And for part 3 we calculate:

$$\begin{aligned}
\text{Op}_s(c \cdot E) &= \text{Out}(\text{argmax } s : c \cdot E. \text{Rew}(s, c \cdot E), c \cdot E) \\
&= \text{reward}_W(\llbracket c \rrbracket, \text{Out}(\text{argmax } s : c \cdot E. \llbracket c \rrbracket + \text{Rew}(s, E), E)) \\
&= \text{reward}_W(\llbracket c \rrbracket, \text{Out}(\text{argmax } s : c \cdot E. \text{Rew}(s, E), E)) \\
&= \text{reward}_W(\llbracket c \rrbracket, \text{Out}(\text{argmax } s : E. \text{Rew}(s, E), E)) \\
&= \text{reward}_W(\llbracket c \rrbracket, \text{Op}_s(E))
\end{aligned}$$

where the fourth equality holds as the monoid preserves and reflects the ordering of R . \square

A lemma will prove useful later, that substitutions of constants for constants can equivalently be done via W :

Lemma 8. *Suppose $E : b$ is an effect value, and that $f : \text{Val}_b \rightarrow \text{Val}_{b'}$. Let $g : u \subseteq \text{Con}_b$ be the restriction of f to a finite set that includes all the constants of type b in E . Then:*

$$\text{Op}_s(E[g]) = W(f)(\text{Op}_s(E))$$

Proof. The proof is by structural induction. In case E is a constant we have:

$$W(f)(\text{Op}_s(c)) = W(f)(\langle 0, c \rangle) = \langle 0, f(c) \rangle = \text{Op}_s(c[g])$$

In case E has the form $E_1 \text{ or } E_2$ we have:

$$\begin{aligned}
W(f)(\text{Op}_s(E_1 \text{ or } E_2)) &= W(f)(\text{Op}_s(E_1) \max_{\pi_1} \text{Op}_s(E_2)) && \text{(by Theorem 4.2)} \\
&= W(f)(\text{Op}_s(E_1)) \max_{\pi_1} W(f)(\text{Op}_s(E_2)) && \text{(as } \pi_1 \circ W(f) = \pi_1) \\
&= \text{Op}_s(E_1[g]) \max_{\pi_1} \text{Op}_s(E_2[g]) \\
&= \text{Op}_s(E_1[g] \text{ or } E_2[g]) && \text{(by Theorem 4.2)} \\
&= \text{Op}_s((E_1 \text{ or } E_2)[g])
\end{aligned}$$

In case E has the form $c \cdot E_1$ we have:

$$\begin{aligned}
W(f)(\text{Op}_s(c \cdot E_1)) &= W(f)(\llbracket c \rrbracket \cdot \text{Op}_s(E_1)) && \text{(by Theorem 4.3)} \\
&= \llbracket c \rrbracket \cdot (W(f)(\text{Op}_s(E_1))) && \text{(as } W(f) \text{ is homomorphic)} \\
&= \llbracket c \rrbracket \cdot \text{Op}_s(E_1[g]) \\
&= \text{Op}_s(c \cdot (E_1[g])) && \text{(by Theorem 4.3)} \\
&= \text{Op}_s((c \cdot E_1)[g])
\end{aligned}$$

\square

4.4 Denotational semantics

For the denotational semantics, as discussed in Section 2.1, we need an auxiliary monad T , here to handle the reward effect. and we take T to be $W = R \times -$, the writer monad. We

also need a W -algebra $\alpha_W : W(R) \rightarrow R$. As R is itself an R -action (setting $r \cdot s = r + s$), we obtain a W -algebra as described in Section 2.2, finding that $\alpha_T = +$.

We then have a strong monad

$$S(X) = (X \rightarrow R) \rightarrow R \times X$$

and use this monad to give the denotational semantics

$$\mathcal{S}_T \llbracket M \rrbracket : \mathcal{S}_T \llbracket \Gamma \rrbracket \rightarrow S(\mathcal{S}_T \llbracket \sigma \rrbracket) \quad (\text{for } \Gamma \vdash M : \sigma)$$

of our language, following the pattern explained in the previous section (we often drop the subscript on \mathcal{S}_T below). For the semantics of basic types, constants, and function symbols, we assume: $\llbracket \text{Rew} \rrbracket = R$; $\llbracket c \rrbracket$ is as before, for $c : \text{Rew}$; and $+$ and \leq are the monoid operation and ordering on R . Turning to the algebraic operation symbols, for **or** we use the algebraic operation or_X given in Section 2, and for **reward** we take the algebraic operation $(\text{reward}_S)_X$ induced by the $(\text{reward}_W)_X$, so, by Equation 3:

$$(\text{reward}_S)_X(r, G)(\gamma) = (\text{reward}_W)_X(r, G\gamma) (= \langle r + \pi_1(G\gamma), \pi_2(G\gamma) \rangle)$$

4.5 Adequacy

We now aim to prove that the selection operational semantics essentially coincides with its denotational semantics. This will be our *selection adequacy theorem*.

We need some notation to connect the operational semantics of programs with their denotations. We set $(\llbracket - \rrbracket_W)_\sigma = W(\llbracket - \rrbracket_\sigma) : W(\text{Val}_\sigma) \rightarrow W(\llbracket \sigma \rrbracket)$. So for $u = \langle r, V \rangle \in R \times \text{Val}_\sigma = W(\text{Val}_\sigma)$ we have

$$(\llbracket \langle r, V \rangle \rrbracket_W)_\sigma = \langle r, \llbracket V \rrbracket_\sigma \rangle$$

Lemma 9. *For any effect value $E : \sigma$ we have:*

$$\mathcal{S}_W \llbracket E \rrbracket(0) = \llbracket \text{Op}_s(E) \rrbracket$$

Proof. We proceed by structural induction on E , and cases according to its form.

1. Suppose E is a value V . Using Theorem 4.1, we calculate:

$$\llbracket \text{Op}_s(V) \rrbracket = \llbracket \langle 0, V \rangle \rrbracket = \langle 0, \llbracket V \rrbracket \rangle = \eta_W(\llbracket V \rrbracket) = \eta_S(\llbracket V \rrbracket)(0) = \mathcal{S} \llbracket V \rrbracket(0)$$

2. Suppose next that $E = E_1 \text{ or } E_2$. Then:

$$\begin{aligned} \llbracket \text{Op}_s(E_1 \text{ or } E_2) \rrbracket_W &= \llbracket \text{Op}_s(E_1) \max_{\alpha_W \circ W(0)} \text{Op}_s(E_2) \rrbracket_W && (\text{by Theorem 4.2}) \\ &= \llbracket \text{Op}_s(E_1) \max_{\alpha_W \circ W(0) \circ W(\llbracket \cdot \rrbracket_\sigma)} \text{Op}_s(E_2) \rrbracket_W \\ &= \llbracket \text{Op}_s(E_1) \rrbracket_W \max_{\alpha_W \circ W(0)} \llbracket \text{Op}_s(E_2) \rrbracket_W && (\text{using Lemma 6}) \\ &= \mathcal{S} \llbracket E_1 \rrbracket(0) \max_{\alpha_W \circ W(0)} \mathcal{S} \llbracket E_2 \rrbracket(0) && (\text{by induction hypothesis}) \\ &= \text{or}_{\llbracket \sigma \rrbracket}(\mathcal{S} \llbracket E_1 \rrbracket, \mathcal{S} \llbracket E_2 \rrbracket)(0) \\ &= \mathcal{S} \llbracket E_1 \text{ or } E_2 \rrbracket(0) \end{aligned}$$

3. Suppose instead that $E = c \cdot E'$. Then:

$$\begin{aligned}
\llbracket \text{Op}_s(c \cdot E') \rrbracket &= W(\llbracket - \rrbracket_\sigma)(\llbracket c \rrbracket \cdot_W \text{Op}_s(E')) \quad (\text{by Theorem 4.3}) \\
&= \llbracket c \rrbracket \cdot_W W(\llbracket - \rrbracket_\sigma)(\text{Op}_s(E')) \quad (\text{as } W(\llbracket - \rrbracket_\sigma) \text{ is a homomorphism}) \\
&= \llbracket c \rrbracket \cdot_W \mathcal{S}\llbracket E' \rrbracket(0) \quad (\text{by induction hypothesis}) \\
&= (\llbracket c \rrbracket \cdot_S \mathcal{S}\llbracket E' \rrbracket)(0) \\
&= \mathcal{S}\llbracket c \cdot E' \rrbracket(0)
\end{aligned}$$

□

Theorem 5 (Selection adequacy). *For any program $M : \sigma$ we have:*

$$\mathcal{S}_W\llbracket M \rrbracket(0) = \llbracket \text{Op}_s(M) \rrbracket_W$$

Proof. We have:

$$\begin{aligned}
\mathcal{S}_W\llbracket M \rrbracket(0) &= \mathcal{S}_W\llbracket \text{Op}(M) \rrbracket(0) \quad (\text{by Theorem 3}) \\
&= \llbracket \text{Op}_s(\text{Op}(M)) \rrbracket_W \quad (\text{by Lemma 9}) \\
&= \llbracket \text{Op}_s(M) \rrbracket_W \quad (\text{by the definition of } \text{Op}_s, \\
&\quad \text{and as } \text{Op}(\text{Op}(M)) = \text{Op}(M))
\end{aligned}$$

□

The theorem relates the compositional denotational semantics to the globally optimizing operational semantics; in particular, the latter determines the former at the zero-reward continuation. Whereas the denotational semantics optimizes only locally, as witnessed by the semantics of `or`, the latter optimizes over all possible Player strategies. The use of the zero-reward continuation is reasonable as the operational semantics of a program does not consider any continuation, and so, as rewards mount up additively, the zero-reward continuation is appropriate at the top level.

In more detail, writing $\langle r, V \rangle$ for $\text{Op}_s(M)$, the theorem states that $\mathcal{S}_T\llbracket M \rrbracket(0) = \langle r, \llbracket V \rrbracket \rangle$. So the rewards according to both semantics agree, and the denotation of the value returned by the globally optimizing operational semantics is given by the denotational semantics. In the case of basic types, the globally optimizing operational semantics is determined by the denotational semantics as the denotations of values of basic types determine the values (see Section 3), and so, in that case, there is complete agreement between the operational semantics and the denotational semantics at the zero-reward continuation.

4.6 Full abstraction, program equivalences, and purity

A general notion of semantic equivalence was given in Section 3.5. Given a (generally syntactic) notion of observation of some set of programs of basic type derived from their operational semantics, one can also define a notion of *observational (or behavioural) equivalence* $M \approx_\sigma N$ between programs of the same type σ , yielding a purely syntactic criterion for when two programs should be considered equal (assuming a syntactic notion of observation). Observational equivalence is generally robust against variations in the notion of observation, and we explore this in the context of our decision-making languages.

Given a notion of observations $\text{Ob}(M)$ of programs $M : b$ of basic type b we can define a notion of observational equivalence $\approx_{b, \text{Ob}}$ in a standard contextual manner relative to contexts:

$$M(\approx_{b, \text{Ob}})_\sigma N \iff \forall C[\] : b \rightarrow \text{Bool}. \text{Ob}(C[M]) = \text{Ob}(C[N])$$

where M and N are closed terms of the same type σ , and $C[M] : \text{Bool}$ and $C[N] : \text{Bool}$. We generally drop the type subscript σ below. Observational equivalence is an equivalence relation and it is closed under contexts, in the sense that for all programs $M : \sigma$, $N : \sigma$ and contexts $C[\] : \sigma \rightarrow \tau$ we have:

$$M \approx_{b, \text{Ob}} N \implies C[M] \approx_{b, \text{Ob}} C[N]$$

Operational adequacy generally yields the implication:

$$\models_{\mathcal{M}} M = N : b \implies \text{Ob}(M) = \text{Ob}(N) \quad (10)$$

It then follows that

$$\models_{\mathcal{M}} M = N : \sigma \implies M \approx_{b, \text{Ob}} N \quad (11)$$

and the converse is *full abstraction* (of \mathcal{M} with respect to $\approx_{b, \text{Ob}}$) at type σ . As a particular case of this implication we have $M \approx_{b, \text{Ob}} \text{Op}(M)$ for programs $M : \sigma$.

In the case of our language of choice and rewards, we work with observational equivalence at boolean type, and take the notion of observation to be simply the optimizing operational semantics Op_s , and write \approx_b for \approx_{b, Op_s} , and \approx for \approx_{Bool} . Note that the selection adequacy theorem (Theorem 5) immediately implies Condition 10 for \mathcal{S}_W and Op_s , as expected. We next see that, with this notion of observation, observational equivalence is robust against changes in choice of basic type (Proposition 5). We consider weakenings of the notion of observation later, observing either only values (Proposition 6) or only rewards (Corollary 2).

Lemma 10. *Suppose that b is a base type with at least two constants. Then for any base type b' and programs $M_1, M_2 : b'$ we have:*

$$M_1 \approx_b M_2 \implies \text{Op}_s(M_1) = \text{Op}_s(M_2)$$

Proof. Let E_i be $\text{Op}(M_i)$ for $i = 1, 2$. Then $E_1 \approx_b E_2$ and it suffices to prove that $\text{Op}_s(E_1) = \text{Op}_s(E_2)$. Suppose that $\text{Op}_s(E_i) = \langle t_i, c_i \rangle$ for $i = 1, 2$. Let $f : \text{Val}_{b'} \rightarrow \text{Val}_b$ be such that $f(c_1)$ and $f(c_2)$ are distinct, in case c_1 and c_2 are, and let g be its restriction to the constants of the E_i of type b' . For $i = 1, 2$, we have:

$$\begin{aligned} \text{Op}_s(F_g E_i) &= \text{Op}_s(E_i[g]) && \text{(by Lemma 4)} \\ &= W(f)(\text{Op}_s(E_i)) && \text{(by Lemma 8)} \\ &= \langle t_i, f(c_i) \rangle \end{aligned}$$

As $E_1 \approx_b E_2$, we have $F_g E_1 \approx_b F_g E_2$ and so $\text{Op}_s(F_g E_1) = \text{Op}_s(F_g E_2)$ and so, from the above equations, that $\langle t_1, f(c_1) \rangle = \langle t_2, f(c_2) \rangle$. So, as f is 1-1 on $\{c_1, c_2\}$, $\langle t_1, c_1 \rangle = \langle t_2, c_2 \rangle$ as required. \square

As an immediate consequence of this lemma we have:

Proposition 5. *For all base types b and programs $M, N : \sigma$, we have*

$$M \approx N \implies M \approx_b N$$

with the converse holding if there are at least two constants of type b .

Because the denotational semantics is compositional, it facilitates proofs of program equivalences, including ones that justify program transformations, and more broadly can be convenient for certain arguments about programs. For this purpose, we rely on the equivalence relation $\Gamma \vdash_{\text{Ax}} M = N : \sigma$ described in Section 3.5. As remarked there, our general semantics is equationally consistent. We interest ourselves in a limited converse, where σ is a basic type and M and N are programs; we call this *program completeness for basic types*.

Our system of axioms, Ax, is given in Figure 3. As shown in Theorem 2, the choice operation is associative and idempotent; by the discussion after Equation 3, the reward operation is an R-action on the $S(X)$ as it is on the $W(X)$; and we see from Proposition 3 that the reward operation commutes with the choice operation as the monoid addition preserves and reflects the order. This justifies the first five of our axioms. A pointwise argument shows that the following equality holds for $r, s \in \mathbb{R}$ and $F, G \in S(X)$, for any set X :

$$r \cdot F \text{ or } s \cdot F = t \cdot F \quad (t = \max(r, s)) \quad (12)$$

Using this equality, the left-bias of the choice operation (shown in Theorem 2) and associativity, we have:

$$(r \cdot F \text{ or } G) \text{ or } s \cdot F = r \cdot F \text{ or } G \quad (r \geq s) \quad (13)$$

and another pointwise argument establishes this equation:

$$(r \cdot F \text{ or } G) \text{ or } s \cdot F = G \text{ or } s \cdot F \quad (r < s) \quad (14)$$

$$\begin{aligned}
(L \text{ or } M) \text{ or } N &= L \text{ or } (M \text{ or } N) & M \text{ or } M &= M \\
0 \cdot N &= N & x \cdot (y \cdot N) &= (x + y) \cdot N \\
x \cdot (M \text{ or } N) &= (x \cdot M) \text{ or } (x \cdot N) \\
\text{if } x \geq y \text{ then } x \cdot M \text{ else } y \cdot M &= x \cdot M \text{ or } y \cdot M \\
\text{if } x \geq z \text{ then } (x \cdot M \text{ or } N) \text{ else } (N \text{ or } z \cdot M) &= (x \cdot M \text{ or } N) \text{ or } z \cdot M
\end{aligned}$$

Figure 3: Equations for choices and rewards

These remarks justify our last two axioms.

Some useful consequences of these equations, mirroring the equalities 12–14, are:

$$c \cdot M \text{ or } c' \cdot M = c'' \cdot M \quad (\text{where } \llbracket c'' \rrbracket = \max(\llbracket c \rrbracket, \llbracket c' \rrbracket)) \quad (\text{R}_1)$$

$$(c \cdot M \text{ or } N) \text{ or } c' \cdot M = c \cdot M \text{ or } N \quad (\text{if } \llbracket c \rrbracket \geq \llbracket c' \rrbracket) \quad (\text{R}_2)$$

$$(c \cdot M \text{ or } N) \text{ or } c' \cdot M = N \text{ or } c' \cdot M \quad (\text{if } \llbracket c \rrbracket < \llbracket c' \rrbracket) \quad (\text{R}_3)$$

Our equational system allows us to put programs of basic type into a normal form. We say that such a term is in *normal form* if (ignoring bracketing of **or**) it is an effect value of the form

$$(c_1 \cdot d_1) \text{ or } \dots \text{ or } (c_n \cdot d_n)$$

with $n > 0$ and no d_i occurring twice.

Lemma 11. *Every program M of basic type is provably equal to a normal form $\text{NF}(M)$.*

Proof. By the ordinary adequacy theorem (Theorem 3), M can be proved equal to an effect value E . Using the associativity equations, the fact that **reward** and **or** commute, and the R-action equations, E can be proved equal to a term of the form $c_1 \cdot d_1 \text{ or } \dots \text{ or } c_n \cdot d_n$, possibly with some d 's occurring more than once. Such duplications can be removed using equations **R**₁, **R**₂, and **R**₃ and associativity. \square

The next theorem shows that four equivalence relations coincide, and thereby simultaneously establishes: a normal form for provable equality; completeness of our proof system for equations between programs; and full abstraction for programs of basic type.

Theorem 6. *For any two programs $M : b$ and $N : b$ of basic type, the following equivalences hold:*

$$\text{NF}(M) = \text{NF}(N) \iff \vdash_{\text{Ax}} M = N : b \iff \models_{\mathcal{S}} M = N : b \iff M \approx N$$

Proof. We already know the implications from left-to-right hold (using Lemma 11). So it suffices to show that:

$$M \approx N \implies \text{NF}(M) = \text{NF}(N)$$

First fix constants $l, r : \text{Rew}$ with $l < r$ (possible as R is expressively nontrivial). We remark that, in general, to prove $A \not\approx B$ for $A, B : b$ it suffices to prove $A' \not\approx B'$ if we have $\vdash_{\text{Ax}} A = A' : b$ and $\vdash_{\text{Ax}} B = B' : b$. We use this fact freely below. We also find it convenient to confuse sums of Rew -constants with their denotations.

Let the normal forms of $\text{NF}(M)$ and $\text{NF}(N)$ be

$$A = (c_1 \cdot d_1) \text{ or } \dots \text{ or } (c_n \cdot d_n) \quad \text{and} \quad B = (c'_1 \cdot d'_1) \text{ or } \dots \text{ or } (c'_{n'} \cdot d'_{n'})$$

Suppose, first, that for some i_0 , d_{i_0} is no d'_j . Choose c to be the maximum of the c_i and the c'_j , other than c_{i_0} . Consider the context:

$$C_1[-] =_{\text{def}} \text{if } [-] = d_{i_0} \text{ then } (c + r) \cdot \text{tt} \text{ else } (c_{i_0} + l) \cdot \text{tt}$$

As $c_i + c_{i_0} + l < c_{i_0} + c + r$, we have $\pi_1(\text{Op}_s(C_1[M])) = c_{i_0} + c + r$. Further $\pi_1(\text{Op}_s(C_1[N]))$ will be the maximum of the $c'_j + c_{i_0} + l$ and so $< c_{i_0} + c + r = \text{Op}_s(C_1[M])$. So we see that $M \not\approx N$, contradicting our assumption, and so this case cannot arise.

Suppose, instead, that for some i_0 , d_{i_0} is d'_{j_0} for some j_0 but that $c_{i_0} \neq c'_{j_0}$. Then we find that $\pi_1(\text{Op}_s(C_1[M])) = c_{i_0} + c + r$ and $\pi_1(\text{Op}_s(C_1[N])) = c'_{j_0} + c + r$ and we have again distinguished M and N , and so this case cannot arise either.

So we now have that $n = n'$ and that $(c_1 \cdot d_1) \dots (c_n \cdot d_n)$ and $(c'_1 \cdot d'_1) \dots (c'_n \cdot d'_n)$ are distinct and are permutations of each other. Suppose there is a first point i_0 at which they differ. We can then write A and B as:

$$A = A_0 \text{ or } c_{i_0} \cdot d_{i_0} \text{ or } A_1 \text{ or } c_{i_1} \cdot d_{i_1} \text{ or } A_2$$

and

$$B = A_0 \text{ or } c'_{i_0} \cdot d'_{i_0} \text{ or } B_1 \text{ or } c'_{i_2} \cdot d'_{i_2} \text{ or } B_2$$

with $d_{i_0} \neq d'_{i_0}$, $c_{i_1} \cdot d_{i_1} = c'_{i_0} \cdot d'_{i_0}$, and $c'_{i_2} \cdot d'_{i_2} = c_{i_0} \cdot d_{i_0}$.

Let c be the maximum of the c_i and the c'_i , except for c_{i_0} and c'_{i_0} , and consider the context

$$C_2[-] =_{\text{def}} \begin{array}{l} \text{let } x : b \text{ be } [-] \text{ in} \\ \text{if } x = d_{i_0} \text{ then } (c + c'_{i_0} + r) \cdot \text{tt} \text{ else} \\ \text{if } x = d'_{i_0} \text{ then } (c + c_{i_0} + r) \cdot \text{ff} \text{ else} \\ (c_{i_0} + c'_{i_0} + l) \cdot \text{ff} \end{array}$$

Then $C_2[M]$ is provably equal to

$$(\bar{c}_1 \cdot \bar{d}_1) \text{ or } \dots \text{ or } (\bar{c}_n \cdot \bar{d}_n)$$

where

$$\begin{aligned}\bar{c}_{i_0} \cdot \bar{d}_{i_0} &= (c_{i_0} + c + c'_{i_0} + r) \cdot \mathbf{tt} \\ \bar{c}_{i_1} \cdot \bar{d}_{i_1} &= (c'_{i_0} + c + c_{i_0} + r) \cdot \mathbf{ff} \\ \bar{c}_i \cdot \bar{d}_i &= (c_i + c_{i_0} + c'_{i_0} + l) \cdot \mathbf{ff} \quad (i \neq i_0, i_1)\end{aligned}$$

and we see that $\text{Op}_s(C_2[M]) = \langle c_{i_0} + c + c'_{i_0} + r, \mathbf{tt} \rangle$.

Further, $C_2[N]$ is provably equal to

$$(\bar{c}'_1 \cdot \bar{d}'_1) \text{ or } \dots \text{ or } (\bar{c}'_n \cdot \bar{d}'_n)$$

where

$$\begin{aligned}\bar{c}'_{i_0} \cdot \bar{d}'_{i_0} &= (c'_{i_0} + c + c_{i_0} + r) \cdot \mathbf{ff} \\ \bar{c}'_{i_2} \cdot \bar{d}'_{i_2} &= (c_{i_0} + c + c'_{i_0} + r) \cdot \mathbf{tt} \\ \bar{c}'_i \cdot \bar{d}'_i &= (c'_i + c_{i_0} + c'_{i_0} + l) \cdot \mathbf{ff} \quad (i \neq i_0, i_2)\end{aligned}$$

and we see that $\text{Op}_s(C_2[N]) = \langle c'_{i_0} + c + c_{i_0} + r, \mathbf{ff} \rangle$. So $C_2[-]$ distinguishes M and N , and so this final case cannot arise either. \square

Theorem 6 is in the spirit of [33] in giving an axiomatic and denotational accounts of observational equivalence at basic types, though here at the level of terms rather than, as there, only effect values. (A natural axiomatic account of the observational equivalence of effect values can be given by specializing the above axioms to them, including \mathbf{R}_1 , \mathbf{R}_2 , and \mathbf{R}_3 , but deleting the last two in Figure 3.)

As a corollary of this theorem we have completeness for purity (i.e., effect-freeness) assertions at basic types. Indeed we have it in a strong form:

Corollary 1. *For any program $M : b$, we have:*

$$\models_{\mathcal{S}} M \downarrow b \implies \exists c : b \vdash_{\text{Ax}} M = c$$

Proof. Suppose $\models_{\mathcal{S}} M \downarrow b$. That is, for some $x \in \llbracket b \rrbracket$, $\mathcal{S}[M] = \eta_{\text{Sw}}(x) = \lambda\gamma. \langle 0, x \rangle$. For some $r \in \mathbf{R}$ and $c : b$, $\text{Op}_s(M) = \langle r, c \rangle$. So, by adequacy, $\mathcal{S}[M](0) = \llbracket \text{Op}_s(M) \rrbracket = \llbracket \langle r, c \rangle \rrbracket = \langle r, \llbracket c \rrbracket \rangle$. As $\mathcal{S}[M] = \lambda\gamma. \langle 0, x \rangle$ we therefore have $r = 0$ and $\llbracket c \rrbracket = x$ and so $\models_{\text{W}} M = b$. It follows, using Theorem 6, that $\vdash_{\text{Ax}} M = c$. \square

A natural question is whether, instead of using the selection monad S_T , we can treat the choice operator at the same level as the reward one, say using a suitable free-algebra monad. This can be done, to some extent, by making use of the equations we have established for these operations at the term level and Theorem 6. We consider an equational system with a binary (infix) operation symbol or and an \mathbf{R} -indexed family of unary operation symbols $r \cdot -$ ($r \in \mathbf{R}$). We impose Equations 9, associativity and commutativity equations:

$$x \text{ or } (y \text{ or } z) = (x \text{ or } y) \text{ or } z \quad r \cdot (x \text{ or } y) = r \cdot x \text{ or } r \cdot y$$

and equations corresponding to Equations R_1 , R_2 , and R_3 :

$$\begin{aligned} r \cdot x \text{ or } r' \cdot x &= \max(r, r') \cdot x \\ (r \cdot x \text{ or } r' \cdot y) \text{ or } r'' \cdot x &= r \cdot x \text{ or } r' \cdot y \quad (\text{if } r \geq r'') \\ (r \cdot x \text{ or } r' \cdot y) \text{ or } r'' \cdot x &= r' \cdot y \text{ or } r'' \cdot x \quad (\text{if } r < r'') \end{aligned}$$

Let C be the resulting free-algebra monad. This yields a corresponding denotational semantics \mathcal{C} , and one can show that for all effect values $E, E' : b$ of a basic type b we have:

$$\models_{\mathcal{C}} E = E' \iff \vdash_{Ax} E = E' : b$$

Using Theorems 3 and 6 we then obtain a version of Theorem 6 for \mathcal{C} :

$$\vdash_{Ax} M = N : b \iff \models_{\mathcal{C}} M = N : b \iff M \approx N$$

However we do not obtain an adequacy theorem analogous to the adequacy theorem (Theorem 5) which relates the operational semantics to the selection monad semantics at the zero-reward continuation. Consider, for example, the two boolean effect values \mathbf{tt} and $\mathbf{tt} \text{ or } \mathbf{ff}$. Operationally they both evaluate to \mathbf{tt} . But they have different \mathcal{S} -semantics as the second value is sensitive to the choice of reward continuation. They therefore have different \mathcal{C} -semantics, i.e., in this sense the \mathcal{C} -semantics is not sound. An alternative would be to extend the operational semantics of programs to take a reward continuation into account, as done in [33]; however this would be in tension with the idea that programs should be executable without additional information.

Turning to weakening the notion of observation, we may observe either just the reward or just the final value, giving two weakened notions of observation $\text{Ob}_r = \pi_1 \circ \text{Op}_s$, for the first, and $\text{Ob}_v = \pi_2 \circ \text{Op}_s$, for the second. We begin by investigating observing only values.

Lemma 12. *For programs $M, N : \text{Bool}$ we have:*

$$M_1 \approx_{\text{Ob}_v} M_2 \implies \text{Op}_s(M_1) = \text{Op}_s(M_2)$$

Proof. As Ob_v is weaker than Op_s the implication 10 holds for \mathcal{S}_W and it. We can therefore assume w.l.o.g. that M_1 and M_2 are effect values, E_1 and E_2 , say. Suppose $\text{Op}_s(E_i) = \langle r_i, c_i \rangle$ ($i = 1, 2$).

Assume $E_1 \approx_{\text{Ob}_v} E_2$. We then have $v_1 = v_2 = \mathbf{tt}$, say. Suppose, for the sake of contradiction, that $r_1 \neq r_2$, and then, w.l.o.g., that $r_1 < r_2$. Define $f : \text{Val}_{\text{Bool}} \rightarrow \text{Val}_{\text{Bool}}$ to be constantly \mathbf{ff} . Then we have

$$\begin{aligned} \text{Ob}_v(E_1 \text{ or } F_f E_2) &= \pi_2(\text{Op}_s(E_1 \text{ or } F_f E_2)) \\ &= \pi_2(\text{Op}_s(E_1) \max_{\pi_1} \text{Op}_s(F_f E_2)) && (\text{by Theorem 4.2}) \\ &= \pi_2(\langle r_1, \mathbf{tt} \rangle \max_{\pi_1} \text{Op}_s(E_2[f])) && (\text{by Lemma 4}) \\ &= \pi_2(\langle r_1, \mathbf{tt} \rangle \max_{\pi_1} W(f)(\text{Op}_s(E_2))) && (\text{by Lemma 8}) \\ &= \pi_2(\langle r_1, \mathbf{tt} \rangle \max_{\pi_1} \langle r_2, \mathbf{ff} \rangle) \\ &= \mathbf{ff} \end{aligned}$$

and, similarly,

$$\begin{aligned}\text{Ob}_v(E_2 \text{ or } F_f E_2) &= \pi_2(\langle r_2, \text{tt} \rangle \max_{\pi_1} \langle r_2, \text{ff} \rangle) \\ &= \text{tt}\end{aligned}$$

yielding the required contradiction. \square

It immediately follows that observing only values does not weaken the notion of observational equivalence.

Proposition 6. *For programs $M, N : \text{Bool}$ we have:*

$$M \approx N \iff M \approx_{\text{Ob}_v} N$$

To investigate observing only rewards, we consider another free algebra monad, \mathcal{M}_r . It is the free algebra monad for the equational system with a binary (infix) associative, commutative, absorptive binary operation $- \text{ or } -$ which forms a module relative to the max-plus structure of \mathbb{R} , meaning that there is an \mathbb{R} -indexed family of unary operation symbols $r \cdot -$ ($r \in \mathbb{R}$) forming an \mathbb{R} -action and with the following two equations holding:

$$r \cdot (x \text{ or } y) = r \cdot x \text{ or } r \cdot y \quad r \cdot x \text{ or } r' \cdot x = \max(r, r') \cdot x$$

We write \mathcal{M}_r for the associated denotational semantics of our language with rewards.

Lemma 13. *For any programs $M, N : b$ of base type, we have:*

$$\mathcal{M}_r(M) = \mathcal{M}_r(N) \implies \text{Ob}_r(M) = \text{Ob}_r(N)$$

Proof. Assume $\mathcal{M}_r(M) = \mathcal{M}_r(N)$. We can assume M and N are effect values, say E_1 and E_2 . These effect values take their denotations in the free algebra $\mathcal{M}_r(\llbracket b \rrbracket)$. They can be considered as algebra terms if we add the constants $c : b$ in E_1 and E_2 to the signature and identify the constants $c : \text{Rew}$ occurring in subterms of the form $c \cdot E$ with their denotations. With that, their denotations are the same as their denotations in the free algebra extended so that the two denotations of the constants agree. So, as their denotations are equal, they can be proved equal in equational logic using closed instances of the axioms. We show by induction on the size of proof that if $E = E'$ is so provable, then $\text{Ob}_r(E) = \text{Ob}_r(E')$.

Other than commutativity, all closed instances $E = E'$ of the axioms hold in \mathcal{S} and so $\text{Op}_s(E) = \text{Op}_s(E')$ for such instances. By Theorem 4.2, for any effect values $E, E' : b$ we have $\text{Ob}_r(E \text{ or } E') = \text{Ob}_r(E) \max \text{Ob}_r(E')$, and so all closed instances of commutativity also hold. The only remaining non-trivial cases are the congruence rules. For that for choice we again use Theorem 4.2; for that for rewards we use Theorem 4.3, which implies $\text{Ob}_r(r \cdot E) = r + \text{Ob}_r(E)$, for any effect value $E : b$. \square

Theorem 7. *For any programs $M, N : b$ of base type, we have:*

$$\mathcal{M}_r(M) = \mathcal{M}_r(N) \iff M \approx_{\text{Ob}_r} N$$

Proof. The implication from left to right follows immediately from Lemma 13. For the converse, suppose that $M \approx_{\text{Ob}_r} N$. We can assume M and N are effect values, say E_1 and E_2 . Let A_1 and A_2 be their normal forms. As the program equivalences used to put effect values of basic type in normal form follow from those true in \mathcal{M}_r , we have $\mathcal{M}_r(E_i) = \mathcal{M}_r(A_i)$ ($i = 1, 2$). So, as $E_1 \approx_{\text{Ob}_r} E_2$ we have $A_1 \approx_{\text{Ob}_r} A_2$, using the implication from left to right. Writing the normal forms out as

$$A_1 = (c_1 \cdot d_1) \text{ or } \dots \text{ or } (c_n \cdot d_n) \quad \text{and} \quad A_2 = (c'_1 \cdot d'_1) \text{ or } \dots \text{ or } (c'_{n'} \cdot d'_{n'})$$

we see from the first part of the proof of Theorem 6 that $(c_1 \cdot d_1) \dots (c_n \cdot d_n)$ is a permutation of $(c'_1 \cdot d'_1) \dots (c'_{n'} \cdot d'_{n'})$. As the commutativity program equivalence holds in \mathcal{M}_r , we therefore have $\mathcal{M}_r(A_1) = \mathcal{M}_r(A_2)$ and the proof concludes as we also have $\mathcal{M}_r(E_i) = \mathcal{M}_r(A_i)$ for $i = 1, 2$. \square

Corollary 2. *The selection monad semantics with auxiliary monad the writer monad W is not fully abstract at basic types for \approx_{Ob_r} (and so \approx_{Ob_r} is strictly weaker than \approx). Indeed for programs $M, N : \sigma$ of any type we have:*

$$M \text{ or } N \approx_{\text{Ob}_r} N \text{ or } M$$

So, if we only care about optimizing rewards, we may even assume that **or** is commutative.

5 Adding probabilities

We next extend the language of choices and rewards by probabilistic nondeterminism. Thus, we have the three main ingredients of MDPs, though in the setting of a higher-order λ -calculus rather than the more usual state machines. We proceed as in the previous section, often reusing notation.

5.1 Syntax

To the language of Section 4.1 we add function symbols $\text{con}_p : \text{Rew Rew} \rightarrow \text{Rew}$ ($p \in [0, 1]$) to the basic vocabulary, and $+_p : \varepsilon, 2$ ($p \in [0, 1]$) to the algebraic operation symbols. We use infix notation for both the con_p and the $+_p$; the former are for a convex combination of rewards; the latter is for binary probabilistic choice.

For example (continuing an example from Section 4.1), we may write the tiny program:

$$(5 \cdot \text{tt}) \text{ or } ((5 \cdot \text{tt}) +_{.5} (6 \cdot \text{ff}))$$

Intuitively, like the program of Section 4.1, this program could return either **tt** or **ff**, with respective rewards 5 and 6. Both outcomes are possible on the right branch of its choice,

each with probability .5. The intended semantics aims to maximize expected rewards, so that branch is selected.

This example illustrates how the language can express MDP-like transitions. In MDPs, at each time step, the decision-maker chooses an action, and the process randomly moves to a new state and yields rewards; the distribution over the new states depends on the current state and the action. In our language, all decisions are binary, but bigger decisions can be programmed from them. Moreover, the decisions are separate from the probabilistic choices and the rewards, but as in this example it is a simple matter of programming to combine them. A more complete encoding of MDPs can be done by adding primitive recursion to the language, as suggested in the Introduction.

5.2 Rewards and additional effects

We further assume that \mathbf{R} is equipped with a *convex* algebra (a.k.a. a *barycentric* algebra) structure compatible with the order and monoid structure. That is: there are binary probabilistic choice functions $+_p : \mathbf{R}^2 \rightarrow \mathbf{R}$ ($p \in [0, 1]$) such that the following four equations hold:

$$\begin{aligned} x +_1 y &= x \\ x +_p x &= x \\ x +_p y &= y +_{1-p} x \\ (x +_p y) +_q z &= x +_{pq} (y +_{\frac{1-pq}{1-pq}} z) \quad (p, q < 1) \end{aligned}$$

and probabilistic choice respects and preserves the order in its first argument (and so, too, in its second) in the sense that

$$r \leq r' \iff r +_p s \leq r' +_p s \quad (p \neq 0)$$

and probabilistic choice is homogeneous, in the sense that

$$r + (s +_p s') = (r + s) +_p (r + s')$$

Continuing the three examples of Section 4.2, in all cases probabilistic choice can be defined using the usual convex combination: $r +_p s = pr + (1 - p)s$ of real numbers. Convex algebras provide a suitable algebraic structure for probability. They were introduced by Stone in [41], and have an extensive history, briefly surveyed in [30]. They are equivalent to *convex spaces* which are algebras equipped with operations $\sum_{i=1}^n p_i x_i$ (where $p_i \in [0, 1]$ and $\sum_{i=1}^n p_i = 1$), subject to natural axioms [39]; we will use the two notations interchangeably.

We use the combination

$$\text{DW}(X) =_{\text{def}} \mathcal{D}_f(\mathbf{R} \times X)$$

of the finite probability distribution monad with the writer monad for both operational and denotational semantics. The operational semantics we will define evaluates programs M of type σ to finite distributions of pairs $\langle r, V \rangle$, with $r \in \mathbf{R}$ and $V : \sigma$, that is to elements

of $\text{DW}(\text{Val}_\sigma)$. The monad is the free-algebra monad for *barycentric R-modules*. These are algebras with: an \mathbf{R} -indexed family of unary operations, written as $\mathbf{reward}(r, -)$ or $r \cdot -$, forming an \mathbf{R} -action (Equation 9); and a $[0, 1]$ -indexed family $- +_p -$ of binary operations forming a barycentric algebra over which the \mathbf{R} -action distributes, i.e., with the following equation holding:

$$r \cdot (x +_p y) = r \cdot x +_p r \cdot y \quad (15)$$

The resulting monad has unit $(\eta_{T_1})_X(x) = \langle 0, x \rangle$; the extension to $\text{DW}(X)$ of a map $f : X \rightarrow A$ to an algebra A is given by

$$f^{\dagger T_1}(\sum p_i \langle r_i, x_i \rangle) = \sum_i p_i (r_i \cdot_A f(x_i))$$

With the assumptions made on \mathbf{R} , it forms a barycentric \mathbf{R} -module. Viewing \mathbf{R} as a DW -algebra, $\alpha_{\text{DW}} : T_1(\mathbf{R}) \rightarrow \mathbf{R}$, we have $\alpha_{\text{DW}} = (\text{id}_{\mathbf{R}})^{\dagger \text{DW}}$; explicitly:

$$\alpha_{\text{DW}}(\sum p_i \langle r_i, s_i \rangle) = \sum p_i (r_i + s_i)$$

The two algebraic operations are:

$$(\mathbf{reward}_{T_1})_X(r, \sum p_i \langle r_i, x_i \rangle) = \sum p_i (r + r_i, x_i) \quad (+_{p T_1})_X(\mu, \nu) = p\mu + (1 - p)\nu$$

They are induced by the generic effects $(g_{T_1})_{\mathbf{reward}} : \mathbf{R} \rightarrow T_1(\mathbb{1})$ and $(g_{T_1})_+ : [0, 1] \rightarrow T_1(\mathbb{B})$, where $(g_{T_1})_{\mathbf{reward}}(r) = \langle r, * \rangle$ and $(g_{T_1})_+(p) = p\langle 0, 0 \rangle + (1 - p)\langle 0, 1 \rangle$. We generally write $(\mathbf{reward}_{T_1})_X$ using an infix operator $(\cdot_{T_1})_X$, as in Section 4.2.

5.3 Operational semantics

We again take a game-theoretic point of view, with Player now playing a game against Nature, assumed to make probabilistic choices. Player therefore seeks to optimize their expected rewards. Effect values $E : \sigma$ are regarded as games but with one additional clause:

- if $E = E_1 +_p E_2$, it is Nature's turn to move. Nature picks E_1 with probability p , and E_2 with probability $1 - p$.

To account for probabilistic choice we add a rule to the definition of strategies:

$$\frac{s_1 : E_1 \quad s_2 : E_2}{(s_1, s_2) : E_1 +_p E_2}$$

and a case to the definition of the linear orders on strategies:

- Game is $E_1 +_p E_2$:

$$(s_1, s_2) \leq_{E_1 +_p E_2} (s'_1, s'_2) \iff s_1 <_{E_1} s'_1 \quad \vee \quad (s_1 = s'_1 \wedge s_2 \leq_{E_2} s'_2)$$

For any effect value $E : \sigma$, the outcome $\text{Out}(s, E)$ of a strategy $s : E$ is a finite probability distribution over $\mathbb{R} \times \text{Val}_\sigma$, i.e., an element of $\mathcal{D}_f(\mathbb{R} \times \text{Val}_\sigma)$. It is defined by:

$$\begin{aligned} \text{Out}(*, V) &= \delta_{\langle 0, V \rangle} \\ \text{Out}(1s, E_1 \text{ or } E_2) &= \text{Out}(s, E_1) \\ \text{Out}(2s, E_1 \text{ or } E_2) &= \text{Out}(s, E_2) \\ \text{Out}(s, c \cdot E) &= \llbracket c \rrbracket \cdot_{\text{Val}_\sigma} \text{Out}(s, E) \\ \text{Out}((s_1, s_2), E_1 +_p E_2) &= p \text{Out}(s_1, E_1) + (1 - p) \text{Out}(s_2, E_2) \end{aligned}$$

We used the Dirac distribution δ_x here; below, as is common, we just write x .

The expected reward of a finite probability distribution on $\mathbb{R} \times X$, for a set X , is:

$$\mathbf{E}(\sum p_i(r_i, x_i)) =_{\text{def}} \sum p_i r_i$$

Note that $\mathbf{E} : \mathcal{D}_f(\mathbb{R} \times X) \rightarrow \mathbb{R}$ can be written as $\alpha_{T_1} \circ T_1(0)$ (as $\pi_1 : \mathbb{R} \times X \rightarrow \mathbb{R}$ in the previous section could be) and is an algebra homomorphism, being a composition of such. The expected reward of a strategy is then:

$$\text{Rew}(s, E) = \mathbf{E}(\text{Out}(s, E))$$

Our selection operational semantics, $\text{Op}_s(M) \in \mathbb{R} \times \text{Val}_\sigma$ for $M : \sigma$, is defined as before by:

$$\text{Op}_s(M) = \text{Out}(\text{argmax } s : \text{Op}(M). \text{Rew}(s, \text{Op}(M)), \text{Op}(M))$$

where we are now, as anticipated, maximizing expected rewards.

We remark that, now that probabilistic choice is available, we could change our strategies to make a probabilistic choice for effect values $E_1 \text{ or } E_2$. However, as with Markov decision processes [21], that would make no change to the optimal expected reward. It would, however, make a difference to the equational logic of choice if we chose with equal probability between effect values with equal expected reward: choice would then be commutative, but not associative.

Much as in Section 4, we now develop a local characterization of the globally optimizing selection operational semantics. We give this characterization in Theorem 8, below; it is analogous to Theorem 4 in Section 4. Some auxiliary lemmas are required. The first of them is another argmax lemma that will enable us to deal with strategies for probabilistic choice:

Lemma 14. *(Second argmax lemma) Let P and Q be finite linear orders, let $P \times Q$ be given the lexicographic ordering, and suppose $\gamma : P \times Q \rightarrow \mathbb{R}$. Define $g : P \rightarrow Q$, $\underline{u} \in P$ and $\underline{v} \in Q$ by:*

$$\begin{aligned} g(u) &= \text{argmax } v : Q. \gamma(u, v) \\ \underline{u} &= \text{argmax } u : P. \gamma(u, g(u)) \\ \underline{v} &= g(\underline{u}) \end{aligned}$$

Then:

$$\operatorname{argmax} \gamma = (\underline{u}, \underline{v})$$

$$(\underline{u}, \underline{v}) = \operatorname{argmax} (u, v) : P \times Q. \gamma(u, v)$$

Proof. Consider any pair (u_0, v_0) . By the definition of g we have $g(u_0) \preceq u_1$ in the sense that:

$$(\gamma(u_0, g(u_0)) > \gamma(u_0, v_0) \vee (\gamma(u_0, g(u_0)) = \gamma(u_0, v_0) \wedge g(u_0) \leq v_0)$$

and it follows that $(u_0, g(u_0)) \preceq_\gamma (u_0, v_0)$.

Next, by the definition of \underline{u} we have $\underline{u} \preceq u_0$ in the sense that:

$$(\gamma(\underline{u}, g(\underline{u})) > \gamma(u_0, g(u_0)) \vee (\gamma(\underline{u}, g(\underline{u})) = \gamma(u_0, g(u_0)) \wedge \underline{u} \leq u_0)$$

and it follows that $(\underline{u}, g(\underline{u})) \preceq_\gamma (u_0, g(u_0))$. (The only non-obvious point may be that in the case where $\gamma(\underline{u}, g(\underline{u})) = \gamma(u_0, g(u_0))$, we have $\underline{u} \leq u_0$, so either $\underline{u} < u_0$, when $(\underline{u}, g(\underline{u})) <_\gamma (u_0, g(u_0))$ or else $\underline{u} = u_0$, when $(\underline{u}, g(\underline{u})) = (u_0, g(u_0))$.)

So, as $\underline{v} = g(\underline{u})$, we have thereby establishing the required minimality of $(\underline{u}, \underline{v})$. \square

The next two lemmas concern expectations for probability distributions constructed by the \cdot operation and by convex combinations.

Lemma 15.

$$\operatorname{Rew}(s, c \cdot E) = \llbracket c \rrbracket + \operatorname{Rew}(s, E)$$

Proof. Using the fact that \mathbf{E} is a homomorphism, we have:

$$\operatorname{Rew}(s, c \cdot E) = \mathbf{E}(\operatorname{Out}(s, c \cdot E)) = \mathbf{E}(\llbracket c \rrbracket \cdot \operatorname{Out}(s, \cdot E)) = \llbracket c \rrbracket + \mathbf{E}(\operatorname{Out}(s, \cdot E)) = \llbracket c \rrbracket + \operatorname{Rew}(s, E)$$

\square

Lemma 16.

$$\operatorname{Rew}((s_1, s_2), E_1 +_p E_2) = p \operatorname{Rew}(s_1, E_1) + (1 - p) \operatorname{Rew}(s_2, E_2)$$

Proof. We calculate:

$$\begin{aligned} \operatorname{Rew}((s_1, s_2), E_1 +_p E_2) &= \mathbf{E}(\operatorname{Out}((s_1, s_2), E_1 +_p E_2)) \\ &= \mathbf{E}(p \operatorname{Out}(s_1, E_1) + (1 - p) \operatorname{Out}(s_2, E_2)) \\ &= p \mathbf{E}(\operatorname{Out}(s_1, E_1)) + (1 - p) \mathbf{E}(\operatorname{Out}(s_2, E_2)) \\ &= p \operatorname{Rew}(s_1, E_1) + (1 - p) \operatorname{Rew}(s_2, E_2) \end{aligned}$$

\square

Theorem 8. *The following hold for well-typed effect values:*

1. $\text{Op}_s(V) = \langle 0, V \rangle (= \eta_{T_1}(V))$
2. $\text{Op}_s(E_1 \text{ or } E_2) = \text{Op}_s(E_1) \max_{\mathbf{E}} \text{Op}_s(E_2) (= \text{Op}_s(E_1) \max_{\alpha_T \circ T_1(0)} \text{Op}_s(E_2))$
3. $\text{Op}_s(c \cdot E) = \llbracket c \rrbracket \cdot \text{Op}_s(E)$
4. $\text{Op}_s(E_1 +_p E_2) = p\text{Op}_s(E_1) + (1 - p)\text{Op}_s(E_2)$

Proof. 1. The proof here is the same as the corresponding case of Theorem 4.

2. The proof here is the same as that of the corresponding case of Theorem 4, except that π_1 is replaced by \mathbf{E} .

3. The proof here is again the same as that of the corresponding case of Theorem 4, except that we use Lemma 15 to show that $\text{Rew}(s, c \cdot E) = \llbracket c \rrbracket + \text{Rew}(s, E)$.

4. We just consider the fourth case. We have:

$$\begin{aligned} \text{Op}_s(E_1 +_p E_2) = \\ \text{Out}(\text{argmax}(s_1, s_2) : E_1 +_p E_2. \text{Rew}((s_1, s_2), E_1 +_p E_2), E_1 +_p E_2) \end{aligned}$$

So, following the second argmax lemma (Lemma 14), we first consider the function

$$f(s_1, s_2) \stackrel{\text{def}}{=} \text{Rew}((s_1, s_2), E_1 +_p E_2) = p\text{Rew}(s_1, E_1) + (1 - p)\text{Rew}(s_2, E_2)$$

where the second equality holds by Lemma 16. We next consider the function:

$$\begin{aligned} g(s_1) &\stackrel{\text{def}}{=} \text{argmax } s_2 : E_2. f(s_1, s_2) \\ &= \text{argmax } s_2 : E_2. p\text{Rew}(s_1, E_1) + (1 - p)\text{Rew}(s_2, E_2) \\ &= \text{argmax } s_2 : E_2. \text{Rew}(s_2, E_2) \end{aligned}$$

where the second equality holds as convex combinations are order-preserving and reflecting in their second argument. Finally we consider

$$\begin{aligned} \underline{s}_1 &\stackrel{\text{def}}{=} \text{argmax } s_1 : E_1. f(s_1, g(s_1)) \\ &= \text{argmax } s_1 : E_1. p\text{Rew}(s_1, E_1) + (1 - p)\text{Rew}(g(s_1), E_2) \\ &= \text{argmax } s_1 : E_1. \text{Rew}(s_1, E_1) \end{aligned}$$

where the second equality holds as convex combinations are order-preserving and reflecting in their first argument, and as $g(s_1)$ is independent of s_1 .

So setting

$$\underline{s}_2 = g(\underline{s}_1) = \text{argmax } s_2 : E_2. \text{Rew}(s_2, E_2)$$

by the second argmax lemma (Lemma 14) we have:

$$(\underline{s}_1, \underline{s}_2) = \text{argmax}(s_1, s_2) : E_1 +_p E_2. \text{Rew}((s_1, s_2), E_1 +_p E_2)$$

so we finally have:

$$\begin{aligned}
\text{Op}_s(E_1 +_p E_2) &= \text{Out}((\underline{s}_1, \underline{s}_1), E_1 +_p E_2) \\
&= p\text{Out}(\underline{s}_1, E_1) + (1-p)\text{Out}(\underline{s}_2, E_2) \\
&= p\text{Out}(\text{argmax } s_1 : E_1.\text{Rew}(s_1, E_1), E_1) \\
&\quad + (1-p)\text{Out}(\text{argmax } s_2 : E_2.\text{Rew}(s_2, E_2), E_2) \\
&= p\text{Op}_s(E_1) + (1-p)\text{Op}_s(E_2)
\end{aligned}$$

as required. \square

An analogous lemma to Lemma 8 will prove useful later, that substitutions of constants for constants can equivalently be done via DW.

Lemma 17. *Suppose $E : b$ is an effect value, and that $f : \text{Val}_b \rightarrow \text{Val}_{b'}$. Let $g : u \subseteq \text{Con}_b$ be the restriction of f to a finite set that includes all the constants of type b in E . Then:*

1. $\mathbf{E}(E[g]) = \mathbf{E}(E)$
2. $\text{Op}_s(E[g]) = \text{DW}(f)(\text{Op}_s(E))$

Proof. The first part is proved by structural induction on E , as is the second part, using Theorem 8. (The result on expectations is used in the proof of the second part in the case where E has the form E_0 or E_1). \square

5.4 Denotational semantics

For the denotational semantics we consider three auxiliary monads, corresponding to three notions of observation with varying degrees of correlation between possible program values and expected rewards. Consider, for example, the effect value $1 \cdot \mathbf{tt} +_{0.5} (2 \cdot \mathbf{ff} +_{0.4} 3 \cdot \mathbf{tt})$. With probability 0.5 this returns \mathbf{tt} with reward 1, with probability 0.2 it returns \mathbf{ff} with reward 2, and with probability 0.3 it returns \mathbf{tt} with reward 3. This level of detail is recorded using our first monad T_1 . At a much coarser grain, we may simply record that \mathbf{tt} and \mathbf{ff} are returned with respective probabilities 0.8 and 0.2, and that the overall expected reward is 1.8. This level of detail is recorded using our third monad T_3 . At an intermediate level we may record the same outcome distribution and the expected reward given a particular outcome (in the example, the expected reward is 1.75, given outcome \mathbf{tt} , and 2, given outcome \mathbf{ff}). This level of detail is recorded using our second monad T_2 .

We will work with a general auxiliary monad, and then specialize our results to the T_i . Specifically we assume we have: a monad T ; T -generic effects $(g_T)_{\text{reward}} : \mathbb{R} \rightarrow T(\mathbb{1})$ and $(g_T)_{+_p} \in T(\mathbb{B})$, with corresponding algebraic operations

$$(\text{reward}_T)_X : \mathbb{R} \times T(X) \rightarrow T(X) \quad (+_p)_T)_X : T(X)^2 \rightarrow T(X)$$

and a T -algebra $\alpha_T : T(\mathbb{R}) \rightarrow \mathbb{R}$, such that, using evident infix notations:

A1 For any set X , $(\mathbf{reward}_T)_X : R \times T(X) \rightarrow T(X)$ and $(+_p)_X : T(X)^2 \rightarrow T(X)$ form a barycentric R -module.

A2 The algebra map $\alpha_T : T(R) \rightarrow R$ is a barycentric R -module homomorphism, i.e., for $x, y \in T(R)$ we have:

$$\alpha_T(r \cdot x) = r + \alpha_T(x) \quad \alpha_T(x +_p y) = \alpha_T(x) +_p \alpha_T(y)$$

So we have the anticipated strong monad

$$S(X) = (X \rightarrow R) \rightarrow T(X)$$

Extending the semantics of the language of the previous section, the function symbols \mathbf{con}_p denote the convex combination operations on R . As regards the algebraic operation symbols, for \mathbf{or} we use the algebraic operation \mathbf{or} given in Section 2, and for \mathbf{reward} and $+_p$ we take the algebraic operations $(\mathbf{rewards})_X$ and $(+_p)_X$ induced by the $(\mathbf{reward}_T)_X$ and $(+_p)_X$, so:

$$(\mathbf{rewards})_X(r, G)(\gamma) = (\mathbf{reward}_T)_X(r, G\gamma)$$

and

$$(+_{p_S})_X(F, G)(\gamma) = (+_{p_T})_X(F(\gamma), G(\gamma))$$

Our first monad is $T_1 =_{\text{def}} DW$; with its associated generics for reward and probabilistic choice it evidently satisfies the two assumptions A1 and A2.

Writing $\text{supp}(\nu)$ for the support of a probability distribution ν , our second monad is

$$T_2(X) = \{\langle \mu, \rho \rangle \mid \mu \in \mathcal{D}_f(X), \rho : \text{supp}(\mu) \rightarrow R\}$$

It is the free-algebra monad for algebras with an R -indexed family of unary operations, written as $\mathbf{reward}(r, -)$ or $r \cdot -$, and a $[0, 1]$ -indexed family $- +_p -$ of binary operations satisfying the equations for DW together with the equation:

$$r \cdot x +_p s \cdot x = (r +_p s) \cdot x \tag{16}$$

The two algebraic operations on the $T_2(X)$ are:

$$(\mathbf{reward}_{T_2})_X(r, \langle \mu, \rho \rangle) = \langle \mu, x \mapsto r + \rho(x) \rangle$$

and

$$(+_{p_{T_2}})_X(\langle \mu, \rho \rangle, \langle \mu', \rho' \rangle) = \langle p\mu + (1-p)\nu, \rho'' \rangle$$

where:

$$\rho''(x) = \begin{cases} \rho(x) +_q \rho'(x) & (x \in \text{supp}(\mu) \cap \text{supp}(\mu')) \\ \rho(x) & (x \in \text{supp}(\mu) \setminus \text{supp}(\mu')) \\ \rho'(x) & (x \in \text{supp}(\mu') \setminus \text{supp}(\mu)) \end{cases}$$

where $q = p\mu(x)/(\mu(x) +_p \mu'(x))$. One can then show that:

$$\sum_i p_i \langle \mu_i, \rho_i \rangle = \langle \mu, \rho \rangle$$

where $\mu = \sum_i p_i \mu_i$ and:

$$\rho(x) = \sum_{x \in \text{supp}(\mu_i)} \frac{p_i \mu_i(x)}{\mu(x)} \rho_i(x)$$

The resulting monad has unit $(\eta_{T_2})_X(x) = \langle x, \{\langle x, 0 \rangle\} \rangle$; the extension to $T_2(X)$ of a map $f : X \rightarrow A$ to an algebra A is given by

$$f^{\dagger_{T_2}}(\mu, \rho) = \sum_{x \in \text{supp}(\mu)} \mu(x) (\rho(x) \cdot f(x))$$

and for any $f : X \rightarrow Y$ we have:

$$T_2(f)(\mu, \rho) = \langle \mathcal{D}_f(\mu), \rho' \rangle$$

where

$$\rho'(y) = \sum_{f(x)=y} \frac{\mu(x)}{\mathcal{D}_f(\mu)(y)} \rho(x)$$

Equation 16 holds for R , using commutativity and homogeneity, so we can take the algebra map α_{T_2} to be $\text{id}_R^{\dagger_{T_2}}$; explicitly we find:

$$\alpha_{T_2} \left(\sum p_i r_i, \rho \right) = \sum p_i (\rho(r_i) + r_i)$$

Our third monad $T_3(X) = \mathcal{D}_f(X) \times R$ is the free-algebra monad for algebras with an R -indexed family of unary operations, written as **reward**($r, -$) or $r \cdot -$, and a $[0, 1]$ -indexed family $- +_p -$ of binary operations satisfying the equations for DW and the equation:

$$r \cdot x +_p s \cdot y = (r +_p s) \cdot x +_p (r +_p s) \cdot y \quad (17)$$

The two algebraic operations on the $T_3(X)$ are:

$$(\text{reward}_{T_3})_X(r, \langle \sum p_i(x_i), s \rangle) = \langle \sum p_i(x_i), r + s \rangle$$

and

$$(+_{p_{T_3}})_X(\langle \mu, r \rangle, \langle \nu, s \rangle) = \langle p\mu + (1-p)\nu, pr + (1-p)s \rangle$$

The resulting monad has unit

$$(\eta_{T_3})_X(x) = \langle x, 0 \rangle$$

the extension to $T_3(X)$ of a map $f : X \rightarrow A$ to an algebra A is given by

$$f^{\dagger T_3}(\sum p_i x_i, r) = r \cdot \sum p_i f(x_i)$$

and

$$T_3(f)(\langle \sum p_i x_i, r \rangle) = \langle \sum p_i f(x_i), r \rangle$$

Unfortunately Equation 17 need not hold for R with the assumptions made on it so far: while it does hold for the two examples with the reals and addition, it does not hold for the example of the positive reals and multiplication. When dealing with T_3 we therefore assume additionally that R satisfies Equation 17, and so we can take α_{T_3} to be $\text{id}_R^{\dagger T_3}$; explicitly we find:

$$\alpha_{T_3}(\sum p_i r_i, r) = r + (\sum p_i r_i)$$

There is a generally available

monad morphism

$$\theta_{DW, T} : DW \rightarrow T$$

where $(\theta_{DW, T})_X =_{\text{def}} (\eta_T)_X^{\dagger DW}$. This morphism will be useful when discussing the adequacy theorem.

In the case of T_1 , θ_{T_1} is the identity. In the case of T_2 , first, given a distribution $\mu = \sum_{i=1}^n p_i \langle r_i, x_i \rangle \in T_1(X)$, define its *value distribution* $V\text{Dis}(\mu)$ in $\mathcal{D}_f(X)$, and its *value support* $\text{vsupp}(\mu) \subseteq X$ by:

$$V\text{Dis}(\mu) = \sum p_i \eta_{\mathcal{D}_f}(x_i) \quad \text{vsupp}(\mu) = \{x_i\}$$

and then define the *conditional expected reward* of μ given $x \in \text{vsupp}(\mu)$ by:

$$\text{Rew}(\mu|x) = \frac{\sum_{x_i=x} p_i r_i}{\sum_{x_i=x} p_i}$$

We then have:

$$(\theta_{T_2})_X(\mu) = \langle V\text{Dis}(\mu), \text{Rew}(\mu| -) \rangle$$

In the case of T_3 we have:

$$(\theta_{T_3})_X(\mu) = \langle V\text{Dis}(\mu), \mathbf{E}(\mu) \rangle$$

Two properties of the T_i will be useful later when we consider full abstraction. For the first property, say that \mathbb{B} is *characteristic* for T if, for any set X and any two $u, v \in T(X)$ we have:

$$u \neq v \implies \exists f : X \rightarrow \mathbb{B}. T(f)(u) \neq T(f)(v)$$

Lemma 18. \mathbb{B} is characteristic for each of the T_i .

Proof. Fix X , and, for any $x \in X$ let $f_x : X \rightarrow \mathbb{B}$ be the map that sends x to 0 and everything else in X to 1.

For the case of T_1 suppose $\mu = \sum p_i \langle r_i, x_i \rangle$ and $\nu = \sum q_j \langle s_j, y_j \rangle$ are distinct elements of $T_1(X)$. Then there is an $\langle r_i, x_i \rangle$ in the support of (say) μ that is either not in the support of ν or has different probability there. Then $\langle r_i, 0 \rangle$ is in the support of $T_1(f_{x_i})(\mu)$ but is either not in the support of $T_1(f_{x_i})(\nu)$ or has different probability there.

In the case of T_2 suppose we have distinct elements $a = \langle \sum p_i x_i, \rho_0 \rangle$ and $b = \langle \sum q_j y_j, \rho_1 \rangle$ of $T_2(X)$. If $\sum p_i x_i$ and $\sum q_j y_j$ are distinct we proceed as in the case of T_1 . Otherwise there is an x_i , say x_1 , such that $\rho_0(x_1) \neq \rho_1(x_1)$. Let ρ'_0 and ρ'_1 be the second components of $T_2(f_{x_1})(a)$ and $T_2(f_{x_1})(b)$. Then $\rho'_0(0) = \rho_0(x_1)$ and $\rho'_1(0) = \rho_1(x_1)$ and these are different.

In the case of T_3 suppose we have distinct elements $a = \langle \sum p_i x_i, r \rangle$ and $b = \langle \sum q_j y_j, s \rangle$ of $T_3(X)$. If $\sum p_i x_i$ and $\sum q_j y_j$ are distinct we proceed as in the case of T_1 . Otherwise, $r \neq s$ and $T(f)$ will distinguish a and b . \square

For the second property, for any X and $\gamma : X \rightarrow R$, define the *reward addition* function

$$k_\gamma : T(X) \rightarrow T(X)$$

to be $f^{\dagger T}$, where $f(x) =_{\text{def}} \gamma(x) \cdot_T \eta_T(x)$. Then we say that *reward addition is injective for T* if such functions are always injective.

Lemma 19. *Reward addition is injective for each of the T_i .*

Proof. Fix X and $\gamma : X \rightarrow R$. Beginning with T_1 for any $\mu = \sum_i p_i \langle r_i, x_i \rangle$, with no repeated $\langle r_i, x_i \rangle$, we have

$$k_\gamma(\mu) = \sum_{i=1}^m p_i \langle \gamma(x_i) + r_i, x_i \rangle$$

with no repeated $\langle \gamma(x_i) + r_i, x_i \rangle$ (since the monoid addition on R reflects the order). So, for any such $\mu = \sum_{i=1}^m p_i \langle r_i, x_i \rangle$ and $\nu = \sum_{j=1}^n q_j \langle s_j, y_j \rangle$, if $k_\gamma(\mu) = k_\gamma(\nu)$, i.e., if $\sum_{i=1}^m p_i \langle r_i, x_i \rangle = \sum_{j=1}^n q_j \langle s_j, y_j \rangle$, then $m = n$ and, for some permutation π of the indices, we have $p_i \langle \gamma(x_i) + r_i, x_i \rangle = q_{\pi(j)} \langle \gamma(y_{\pi(j)}) + s_{\pi(j)}, y_{\pi(j)} \rangle$, so $p_i \langle r_i, x_i \rangle = q_{\pi(j)} \langle s_{\pi(j)}, y_{\pi(j)} \rangle$, so $\mu = \nu$, as required.

The proofs for T_2 and T_3 are similar, using the respective formulas

$$k_\gamma(\langle \sum_i p_i x_i, \rho \rangle) = \langle \sum_i p_i x_i, x_i \mapsto \rho(x_i) + \gamma(x_i) \rangle$$

and

$$k_\gamma(\langle \sum_i p_i x_i, r \rangle) = \langle \sum_i p_i x_i, r + \sum_i p_i \gamma(x_i) \rangle$$

\square

5.5 Adequacy

As in Section 4.5, we aim to prove a selection adequacy theorem connecting the globally defined selection operational semantics with the denotational semantics. We again need some notation. Using assumption A1, we set

$$(\llbracket - \rrbracket_T)_\sigma = ((\eta_T)_{\llbracket \sigma \rrbracket} \circ \llbracket - \rrbracket_\sigma)^{\dagger_{\text{DW}}} : \text{DW}(\text{Val}_\sigma) \rightarrow T(\llbracket \sigma \rrbracket)$$

So, for $\mu = \sum p_i \langle r_i, V_i \rangle \in \text{DW}(\text{Val}_\sigma)$ we have:

$$\llbracket \mu \rrbracket_T = \sum p_i (r_i \cdot_T \eta_T(\llbracket V_i \rrbracket))$$

Lemma 20. *For any $\mu \in \text{DW}(\text{Val}_\sigma)$ we have:*

$$\alpha_T(T(0)(\llbracket \mu \rrbracket_T)) = \alpha_{\text{DW}}(\text{DW}(0)(u))$$

Proof. Suppose $\mu = \sum p_i \langle r_i, V_i \rangle$. We calculate:

$$\begin{aligned} \alpha_T(T(0)(\llbracket \sum p_i \langle r_i, V_i \rangle \rrbracket_T)) &= \alpha_T(T(0)(\sum p_i (r_i \cdot_T \eta_T(\llbracket V_i \rrbracket)))) \\ &= \alpha_T(\sum p_i (r_i \cdot_T T(0)(\eta_T(\llbracket V_i \rrbracket)))) \\ &= \alpha_T(\sum p_i (r_i \cdot_T \eta_T(0))) \\ &= \sum p_i (r_i + \alpha_T(\eta_T(0))) && \text{(by assumption A2)} \\ &= \sum p_i r_i \\ &= \alpha_{\text{DW}}(T_1(0)(\sum p_i \langle r_i, V_i \rangle)) \end{aligned}$$

□

Lemma 21. *For any effect value $E : \sigma$ we have:*

$$\mathcal{S}[\llbracket E \rrbracket](0) = \llbracket \text{Op}_s(E) \rrbracket_T$$

Proof. We use structural induction on E . The proofs in the cases where E has one of the forms V , $E_1 \text{ or } E_2$, or $c \cdot E'$ are similar to those in the proof of Lemma 9, now making use of Theorem 8 and Lemma 20. In the case where $E = E_1 +_p E_2$, we calculate:

$$\begin{aligned} \llbracket \text{Op}_s(E_1 +_p E_2) \rrbracket_T &= \llbracket \text{Op}_s(E_1) +_p \text{Op}_s(E_2) \rrbracket_T && \text{(by Theorem 8.4)} \\ &= \llbracket \text{Op}_s(E_1) \rrbracket_T +_p \llbracket \text{Op}_s(E_2) \rrbracket_T && \text{(as } \llbracket - \rrbracket_\sigma \text{ is a homomorphism)} \\ &= \mathcal{S}[\llbracket E_1 \rrbracket](0) +_p \mathcal{S}[\llbracket E_2 \rrbracket](0) && \text{(by induction hypothesis)} \\ &= (+_{p\mathcal{S}})_{\llbracket \sigma \rrbracket}(\mathcal{S}[\llbracket E_1 \rrbracket], \mathcal{S}[\llbracket E_2 \rrbracket])(0) \\ &= \mathcal{S}[\llbracket E_1 +_p E_2 \rrbracket](0) \end{aligned}$$

□

Selection adequacy for our language with probabilities then follows:

Theorem 9 (Selection adequacy). *For any program $M : \sigma$ we have:*

$$\mathcal{S}[\![M]\!](0) = \llbracket \text{Op}_s(M) \rrbracket_T$$

The proof of this theorem is the same as that of Theorem 5. As before, the adequacy theorem implies that the globally optimizing operational semantics determines the denotational semantics at the zero-reward continuation.

To discuss the converse direction, noting that

$$\begin{aligned} (\llbracket - \rrbracket_T)_\sigma &= ((\eta_T)_{\llbracket - \rrbracket_\sigma} \circ \llbracket - \rrbracket_\sigma)^{\dagger_{\text{DW}}} \\ &= (T(\llbracket - \rrbracket_\sigma) \circ (\eta_T)_{\text{Val}_\sigma})^{\dagger_{\text{DW}}} \quad (\eta \text{ is natural}) \\ &= T(\llbracket - \rrbracket_\sigma) \circ ((\eta_T)_{\text{Val}_\sigma})^{\dagger_{\text{DW}}} \\ &= T(\llbracket - \rrbracket_\sigma) \circ (\theta_{\text{DW}, T})_{\text{Val}_\sigma} \end{aligned}$$

we see from the adequacy theorem that, for $M : \sigma$, the denotational semantics determines $(\theta_{\text{DW}, T})_{\text{Val}_\sigma}(\text{Op}_s(M)) \in T(\text{Val}_\sigma)$ up to $T(\llbracket - \rrbracket_\sigma)$. In the case where σ is a basic type b , $T(\llbracket - \rrbracket_b)$ is an injection (as $\llbracket - \rrbracket_b$ is and as T preserves injections, if their domain is nonempty—as do all functors on sets). So, in this case, the denotational semantics determines the operational semantics up to $(\theta_{\text{DW}, T})_{\text{Val}_\sigma}$.

It is therefore reasonable to define a general notion of observation at basic types by setting

$$\text{Ob}_{b, T}(M) = (\theta_{\text{DW}, T})_{\text{Val}_{\text{Bool}}}(\text{Op}_s(M)) \in T(\text{Val}_b)$$

We therefore have notions of observation determined by the denotational semantics for each of the T_i . These notions of observation provide corresponding amounts of information about the operational semantics. With the aid of the above discussion of the monad morphism $\theta_{\text{DW}, T}$ we find for $M : b$ that:

$$\begin{aligned} \text{Ob}_{b, T_1}(M) &= \text{Op}_s(M) \\ \text{Ob}_{b, T_2}(M) &= \langle \text{VDis}(\text{Op}_s(M)), \text{Rew}(\text{Op}_s(M) | -) \rangle \\ \text{Ob}_{b, T_3}(M) &= \langle \text{VDis}(\text{Op}_s(M)), \mathbf{E}(\text{Op}_s(M)) \rangle \end{aligned}$$

We write Ob_T for $\text{Ob}_{\text{Bool}, T}$, and similarly for the T_i .

5.6 Full abstraction, program equivalences, and purity

With probability, there are several natural notions of observation of boolean programs which result in different equivalences. We define three such notions corresponding to our three monads. We continue to proceed abstractly in terms of an auxiliary monad T and algebra $\alpha_T : T(R) \rightarrow R$, as above. Having general notions of observation $\text{Ob}_{b, T}$ at base types, we have corresponding general observational equivalence relations $\approx_{b, T}$, and so, instantiating, observational equivalence relations \approx_{b, T_i} for the T_i . We write \approx_T for $\approx_{\text{Bool}, T}$, and similarly for the T_i .

We next consider, as we did for our first language whether observing at different base types makes a difference to contextual equivalence.

Lemma 22. *Suppose that \mathbb{B} is characteristic for T and that b is a base type with at least two constants. Then for any base type b' and programs $M_1, M_2 : b'$ we have:*

$$M_1 \approx_{b,T} M_2 \implies \text{Ob}_{b',T}(M_1) = \text{Ob}_{b',T}(M_2)$$

Proof. We can assume w.l.o.g. that the M_i are effect values, and write E_1 for them. Assume $E_1 \approx_{b,T} E_2$ and suppose, for the sake of contradiction, that $\text{Ob}_{b',T}(E_1) \neq \text{Ob}_{b',T}(E_2)$. As \mathbb{B} is characteristic for T , there is a map $f : \text{Val}_{b'} \hookrightarrow \text{Val}_{\text{Bool}}$ st. $T(f)(\text{Ob}_{b',T}(E_1)) \neq T(f)(\text{Ob}_{b',T}(E_2))$. As b has at least two constants, there is an injection $\iota : \text{Val}_{\text{Bool}} \rightarrow \text{Val}_b$. Set $f' = f \circ \iota : \text{Val}_{b'} \rightarrow \text{Val}_b$. As T preserves injections with nonempty domain we have $T(f')(\text{Ob}_{b',T}(E_1)) \neq T(f')(\text{Ob}_{b',T}(E_2))$. Let g be the restriction of f' to $g : u \rightarrow \text{Val}_b$, where u is the set of constants of type b occurring in E_1 or E_2 .

As $E_1 \approx_{b,T} E_2$ we have $F_g E_1 \approx_{b,T} F_g E_2$, so $\text{Ob}_{b,T}(F_g E_1) = \text{Ob}_{b,T}(F_g E_2)$, and so, by adequacy, $\mathcal{S}[F_g E_1](0) = \mathcal{S}[F_g E_2](0)$. For $i = 1, 2$, we calculate:

$$\begin{aligned} \mathcal{S}[F_g E_i](0) &= \mathcal{S}[E_i[g]](0) && \text{(by Lemma 4)} \\ &= \llbracket \text{Op}_s(E_i[g]) \rrbracket && \text{(by adequacy)} \\ &= \llbracket \text{DW}(f') \text{Op}_s(E_i) \rrbracket && \text{(by Lemma 17.2)} \\ &= T(\llbracket - \rrbracket)((\theta_{\text{DW},T})_{\text{Val}_b}(\text{DW}(f')(\text{Op}_s(E_i)))) \\ &= T(\llbracket - \rrbracket)(T(f')(\theta_{\text{Val}_b'}(\text{Op}_s(E_i)))) && (\theta_{\text{DW},T} \text{ is natural}) \\ &= T(\llbracket - \rrbracket)(T(f')(\text{Ob}_{b',T}(E_i))) \end{aligned}$$

So, as $T(\llbracket - \rrbracket)$ is injective, $T(f')(\text{Ob}_{b,T}(E_1)) = T(f')(\text{Ob}_{b,T}(E_2))$, yielding the required contradiction. \square

We then have the following analogue of Proposition 5:

Proposition 7. *Suppose that \mathbb{B} is characteristic for T . Then, for all base types b and programs $M, N : \sigma$, we have*

$$M \approx_T N \implies M \approx_{b,T} N$$

with the converse holding if there are at least two constants of type b .

As \mathbb{B} is characteristic for the T_i (Lemma 18), we have invariance of the observational equivalences \approx_{b,T_i} under changes of base type with at least two constants.

Modulo a reasonable definability assumption, each of our three semantics is fully abstract at basic types with respect to their corresponding notion of observational equivalence. We establish this via general results for T and $\alpha_T : T(R) \rightarrow R$, as above.

We first need a general result on reward continuations. Suppose $u \subseteq \text{Con}_b$, with the c_i distinct and suppose $\gamma : \llbracket b \rrbracket \rightarrow R$ is *definable on u* in the sense that there is a (necessarily unique) $g : \text{Con}_b \rightarrow \text{Con}_{\text{Rew}}$ such that $\gamma(\llbracket c \rrbracket) = \llbracket g(c) \rrbracket$, for $c \in u$. Set $K_{u,\gamma} = F_h : b \rightarrow b$ where $h(c) = g(c) \cdot c$ ($c \in u$). We have:

$$\mathcal{S}[K_{u,\gamma} c](0) = \gamma(\llbracket c \rrbracket) \cdot_T \eta_T(\llbracket c \rrbracket) \quad (c \in u)$$

This program $K_{u,\gamma}$ can be used to reduce calling definable reward continuations to calling the zero-reward continuation, modulo reward addition:

Lemma 23. *Suppose $E : b$ is an effect value, u a finite set of constants of type b including all those occurring in E , and $\gamma : \llbracket b \rrbracket \rightarrow \mathbf{R}$ is a reward function definable on u . Then we have:*

$$k_\gamma(\mathcal{S}[\llbracket E \rrbracket](\gamma)) = \mathcal{S}[\llbracket K_{u,\gamma} E \rrbracket](0)$$

Proof. The proof is by structural induction on E . If E is a constant c , than

$$k_\gamma(\mathcal{S}[\llbracket c \rrbracket](\gamma)) = k_\gamma(\eta_T(\llbracket c \rrbracket)) = \gamma(\llbracket c \rrbracket) \cdot_T \eta_T(\llbracket c \rrbracket) = \mathcal{S}[\llbracket K_{u,\gamma} c \rrbracket](0)$$

Suppose next that E has the form E_1 or E_2 . We first show that

$$\alpha_T \circ T(\gamma) = \alpha_T \circ T(0) \circ k_\gamma \quad (*)$$

We have $k_\gamma = f^{\dagger_T}$, where $f(x) =_{\text{def}} \gamma(x) \cdot_T \eta_T(x)$. We then see that $T(0) \circ k_\gamma = g^{\dagger_T}$, setting $g(x) =_{\text{def}} T(0)(\gamma(x) \cdot_T \eta_T(x))$. Making use of assumption A2, we then see that $\alpha_T(g(x)) = \alpha_T(\gamma(x) \cdot_T T(0)(\eta_T(x))) = \alpha_T(\gamma(x) \cdot_T \eta_T(0)) = \gamma(x) + 0 = \gamma(x)$. This, in turn, yields $\alpha_T \circ T(0) \circ k_\gamma = \alpha_T \circ g^{\dagger_T} = \alpha_T \circ T(\alpha_T \circ g) = \alpha_T \circ T(\gamma)$ as required (the second equation is one holding generally for monad algebras).

We then calculate:

$$\begin{aligned}
& k_\gamma(\mathcal{S}[\![E_1 \text{ or } E_2]\!](\gamma)) \\
&= k_\gamma(\mathcal{S}[\![E_1]\!](\gamma) \max_{\alpha_T \circ T(\gamma)} \mathcal{S}[\![E_2]\!](\gamma)) \\
&= \begin{cases} k_\gamma(\mathcal{S}[\![E_1]\!](\gamma)) & \text{(if } (\alpha_T \circ T(\gamma))\mathcal{S}[\![E_1]\!](\gamma) \geq (\alpha_T \circ T(\gamma))\mathcal{S}[\![E_2]\!](\gamma)) \\ k_\gamma(\mathcal{S}[\![E_2]\!](\gamma)) & \text{(otherwise)} \end{cases} \\
&= \begin{cases} \mathcal{S}[\![K_{u,\gamma}(E_1)]\!](0) & \text{(if } (\alpha_T \circ T(0) \circ k_\gamma)\mathcal{S}[\![E_1]\!](\gamma) \geq (\alpha_T \circ T(0) \circ k_\gamma)\mathcal{S}[\![E_2]\!](\gamma)) \\ \mathcal{S}[\![K_{u,\gamma}(E_2)]\!](0) & \text{(otherwise)} \end{cases} \\
&\quad \text{(by induction hypothesis and (*))} \\
&= \begin{cases} \mathcal{S}[\![K_{u,\gamma}(E_1)]\!](0) & \text{(if } (\alpha_T \circ T(0))k_\gamma(\mathcal{S}[\![E_1]\!](\gamma)) \geq (\alpha_T \circ T(0))k_\gamma(\mathcal{S}[\![E_2]\!](\gamma)) \\ \mathcal{S}[\![K_{u,\gamma}(E_2)]\!](0) & \text{(otherwise)} \end{cases} \\
&= \begin{cases} \mathcal{S}[\![K_{u,\gamma}(E_1)]\!](0) & \text{(if } (\alpha_T \circ T(0))\mathcal{S}[\![K_{u,\gamma}(E_1)]\!](0) \geq (\alpha_T \circ T(0))\mathcal{S}[\![K_{u,\gamma}(E_2)]\!](0)) \\ \mathcal{S}[\![K_{u,\gamma}(E_2)]\!](0) & \text{(otherwise)} \end{cases} \\
&\quad \text{(by induction hypothesis and (*))} \\
&= \mathcal{S}[\![(K_{u,\gamma}E_1) \text{ or } (K_{u,\gamma}E_2)]\!](0) \\
&= \mathcal{S}[\![K_{u,\gamma}(E_1 \text{ or } E_2)]\!](0) \\
&\quad \text{(using Equation 7)}
\end{aligned}$$

Suppose next that E has the form $E_1 +_p E_2$. Then we calculate:

$$\begin{aligned}
k_\gamma(\mathcal{S}[\![E_1 +_p E_2]\!](\gamma)) &= k_\gamma(\mathcal{S}[\![E_1]\!](\gamma) +_p \mathcal{S}[\![E_2]\!](\gamma)) \\
&= k_\gamma(\mathcal{S}[\![E_1]\!](\gamma)) +_p k_\gamma(\mathcal{S}[\![E_2]\!](\gamma)) \\
&= \mathcal{S}[\![K_{u,\gamma}E_1]\!](0) +_p \mathcal{S}[\![K_{u,\gamma}E_2]\!](0) \\
&= \mathcal{S}[\![K_{u,\gamma}E_1 +_p K_{u,\gamma}E_2]\!](0) \\
&= \mathcal{S}[\![K_{u,\gamma}(E_1 +_p E_2)]\!](0)
\end{aligned}$$

Finally, suppose that E has the form $c \cdot E'$. This case is handled similarly to the previous one:

$$\begin{aligned}
k_\gamma(\mathcal{S}[\![c \cdot E']\!](\gamma)) &= k_\gamma(\llbracket c \rrbracket \cdot \mathcal{S}[\![E']\!](\gamma)) \\
&= \llbracket c \rrbracket \cdot k_\gamma(\mathcal{S}[\![E']\!](\gamma)) \\
&= \llbracket c \rrbracket \cdot \mathcal{S}[\![K_{u,\gamma}E']\!](0) \\
&= \mathcal{S}[\![c \cdot K_{u,\gamma}E']\!](0) \\
&= \mathcal{S}[\![K_{u,\gamma}(c \cdot E')]\!](0)
\end{aligned}$$

□

We can now demonstrate full abstraction for general T , subject to three assumptions, Say that a type b is *numerable* if all elements of $\llbracket b \rrbracket$ are definable by a constant.

Theorem 10. *Suppose \mathbb{B} is characteristic for T , reward addition is injective for T , and Rew is numerable. Then \mathcal{S} is fully abstract with respect to \approx_T at b .*

Proof. Suppose $M_1(\approx_T)_b M_2$. We wish to show that $\mathcal{S}\llbracket M_1 \rrbracket = \mathcal{S}\llbracket M_2 \rrbracket$. By the ordinary adequacy theorem there are effect values E_1, E_2 with $\mathcal{S}\llbracket M_1 \rrbracket = \mathcal{S}\llbracket E_1 \rrbracket$ and $\mathcal{S}\llbracket M_2 \rrbracket = \mathcal{S}\llbracket E_2 \rrbracket$.

Let u be the set of constants of type b appearing in any one of these effect values. Let $\gamma : \llbracket b \rrbracket \rightarrow \mathbb{R}$ be a reward function. It is definable on u by the numerability assumption. Using Lemma 23 we see that

$$k_\gamma(\mathcal{S}\llbracket M_i \rrbracket(\gamma)) = k_\gamma(\mathcal{S}\llbracket E_i \rrbracket(\gamma)) = \mathcal{S}\llbracket K_{u,\gamma} E_i \rrbracket(0) \quad (*)$$

As $M_1(\approx_T)_b M_2$, we have $E_1(\approx_T)_b E_2$ so $K_{u,\gamma} E_1(\approx_T)_b K_{u,\gamma} E_2$. As \mathbb{B} is characteristic for T , we then have $\text{Ob}_T(K_{u,\gamma} E_1) = \text{Ob}_T(K_{u,\gamma} E_2)$ by Lemma 22, so, by adequacy, $\mathcal{S}\llbracket K_{u,\gamma} E_1 \rrbracket(0) = \mathcal{S}\llbracket K_{u,\gamma} E_2 \rrbracket(0)$. We then see, using (*), that $k_\gamma(\mathcal{S}\llbracket M_1 \rrbracket(\gamma)) = k_\gamma(\mathcal{S}\llbracket M_2 \rrbracket(\gamma))$, so, as reward addition is injective for T , that $\mathcal{S}\llbracket M_1 \rrbracket(\gamma) = \mathcal{S}\llbracket M_2 \rrbracket(\gamma)$.

As γ is any reward function, we finally have $\mathcal{S}\llbracket M_1 \rrbracket = \mathcal{S}\llbracket M_2 \rrbracket$ as required. □

As, by Lemmas 18 and 19, \mathbb{B} is characteristic for all of the T_i and reward addition is injective for all of them, we immediately obtain:

Corollary 3. *Suppose that Rew is numerable. Then \mathcal{S}_{T_i} is fully abstract with respect to \approx_{T_i} at basic types for $i = 1, 3$.*

Analogously to Section 4 we could forget all reward information, by taking our notion of observation Ob_{vdis} to be $\text{VDis} \circ \text{Op}_s$. However, as we next show, the observational equivalence $\approx_{\text{Ob}_{\text{vdis}}}$ resulting from this notion coincides with \approx_{T_3} .

Lemma 24. *For programs $M, N : \text{Bool}$ we have*

$$M_1 \approx_{\text{Ob}_{\text{vdis}}} M_2 \implies \text{Ob}_{T_3}(M_1) = \text{Ob}_{T_3}(M_2)$$

Proof. As Ob_{vdis} is weaker than Ob_{T_3} , and adequacy holds for \mathcal{S}_{T_3} , implication 10 holds for \mathcal{S}_{T_3} and it. We can therefore assume w.l.o.g. that M_1 and M_2 are effect values, E_1 and E_2 , say. So suppose that $E_1 \approx_{\text{Ob}_{\text{vdis}}} E_2$ and, for the sake of contradiction, that, for example, $\mathbf{E}(\text{Op}_s(E_1)) < \mathbf{E}(\text{Op}_s(E_2))$.

Since $E_1 \approx_{\text{Ob}_{\text{vdis}}} E_2$, they return the same probability distribution $\text{tt} +_p \text{ff}$ on boolean values. Suppose, w.l.o.g., that this distribution is not ff (if it is, we can work with tt instead). Define $f : \text{Val}_{\text{Bool}} \rightarrow \text{Val}_{\text{Bool}}$ to be constantly ff . Then we have

$$\begin{aligned}
\text{Ob}_{\text{vdis}}(E_1 \text{ or } F_f E_2) &= \mathbf{E}(\text{Op}_s(E_1 \text{ or } F_f E_2)) \\
&= \text{VDis}(\text{Op}_s(E_1) \max_{\mathbf{E}} \text{Op}_s(F_f E_2)) && \text{(by Theorem 8.2)} \\
&= \text{VDis}(\text{Op}_s(E_1) \max_{\mathbf{E}} \text{Op}_s(E_2[f])) && \text{(by Lemma 4)} \\
&= \text{VDis}(\text{Op}_s(E_1) \max_{\mathbf{E}} W(f)(\text{Op}_s(E_2))) && \text{(by Lemma 17.2)} \\
&= \text{VDis}(W(f)(\text{Op}_s(E_2))) \\
&= \text{ff}
\end{aligned}$$

where the second last equality holds as we have $\mathbf{E}(\text{Op}_s(E_1)) < \mathbf{E}(\text{Op}_s(E_2)) = \mathbf{E}(W(f)(\text{Op}_s(E_1)))$, using Lemma 17.1.

Similarly,

$$\begin{aligned}
\text{Ob}_{\text{vdis}}(E_2 \text{ or } F_f E_2) &= \text{VDis}(\text{Op}_s(E_2) \max_{\mathbf{E}} W(f)(\text{Op}_s(E_2))) \\
&= \text{VDis}(\text{Op}_s(E_2)) \\
&= \text{tt} +_p \text{ff}
\end{aligned}$$

yielding the required contradiction. \square

We then have the following analogue to Proposition 6:

Proposition 8. *For any programs $M, N : \sigma$, we have*

$$M \approx_{T_3} N \iff M \approx_{\text{VDis}} N$$

Turning to equations, we consider those which hold in \mathcal{S}_T for general T as above. We need some terminology and notation. Say that a term $M : \sigma$ is in *expectation PR-form* (over terms L_1, \dots, L_n) if it has the form

$$\sum_{i=1}^m p_i \sum_{j=1}^{n_i} q_{ij} (M_{ij} \cdot L_i)$$

where the M_{ij} are either variables or constants, and we say M is a *expectation PR-value* if the M_{ij} and L_i are all constants. For such a term we write $\mathbf{E}_s(M) : \text{Rew}$ for the term:

$$\bigoplus_{i=1}^m p_i \bigoplus_{j=1}^{n_i} q_{ij} M_{ij}$$

(We write $\bigoplus M_i$ for iterated uses of the con_p to avoid confusion with iterated uses of the $+_p$.) In case the M_{ij} are constants d_{ij} , we set:

$$\left[\bigoplus_{i=1}^m p_i \bigoplus_{j=1}^{n_i} q_{ij} d_{ij} \right] = \sum_{i=1}^m p_i \sum_{j=1}^{n_i} q_{ij} \llbracket d_{ij} \rrbracket$$

Our system Ax_1 of equations is given in Figure 4 (we omit type information). In the last two equations it is assumed that M and N are in expectation PR-form over the same L_1, \dots, L_n . The equations express at the term level, that: choice is idempotent and associative; rewards form an action for the commutative monoid structure on \mathbf{R} ; probabilistic choice forms a convex algebra; the \mathbf{R} -action acts on both forms of choice, and probabilistic choice distributes over choice; and, for a number of cases where this can be seen from the syntax, that choice is made according to the highest reward, with priority to the left for ties.

$$\begin{aligned}
M \text{ or } M &= M & (L \text{ or } M) \text{ or } N &= L \text{ or } (M \text{ or } N) \\
0 \cdot N &= N & x \cdot (y \cdot N) &= (x + y) \cdot N \\
M +_1 N &= M & M +_p N &= N +_{1-p} M \\
(M +_p N) +_q P &= M +_{pq} (N +_{\frac{r-pq}{1-pq}} P) & (p, q < 1) \\
x \cdot (M \text{ or } N) &= (x \cdot M) \text{ or } (x \cdot N) \\
x \cdot (M +_p N) &= x \cdot M +_p x \cdot N \\
L +_p (M \text{ or } N) &= (L +_p M) \text{ or } (L +_p N) \\
(L \text{ or } M) +_p N &= (L +_p N) \text{ or } (M +_p N) \\
\text{if } \mathbf{E}_s(M) \geq \mathbf{E}_s(N) \text{ then } M \text{ else } N &= M \text{ or } N \\
\text{if } \mathbf{E}_s(M) \geq \mathbf{E}_s(N) \text{ then } (M \text{ or } P) \text{ else } (P \text{ or } N) &= (M \text{ or } P) \text{ or } N
\end{aligned}$$

Figure 4: Equations for choices, probability, and rewards

Below, for $u \in \mathbf{T}(X)$ and $\gamma : X \rightarrow \mathbf{R}$, we write $\mathbf{E}(u|\gamma)$ for $\alpha_T(\mathbf{T}(\gamma)(u))$; this is the expected reward of u , given γ .

Proposition 9. *The axioms hold for general \mathcal{S}_T .*

Proof. Other than the last two axiom schemas, this follows from Theorem 2 and Proposition 3. The last two cases are straightforward pointwise arguments, although we need an observation. We calculate that for a PR-term $M = \sum_{i=1}^m p_i \left(\sum_{j=1}^{n_i} q_{ij} (d_{ij} \cdot L_i) \right)$ of type σ and a reward function $\gamma : \llbracket \sigma \rrbracket \rightarrow \mathbf{R}$ we have:

$$\begin{aligned}
\mathbf{E}(\mathcal{S}[\sum_{i=1}^m p_i \sum_{j=1}^{n_i} q_{ij} (d_{ij} \cdot L_i)] \gamma \mid \gamma) &= \sum_{i=1}^m p_i (\sum_{j=1}^{n_i} q_{ij} (\llbracket d_{ij} \rrbracket + \mathbf{E}(\mathcal{S}[\llbracket L_i \rrbracket] \gamma \mid \gamma))) \\
&= \sum_{i=1}^m p_i (\sum_{j=1}^{n_i} q_{ij} \llbracket d_{ij} \rrbracket) + \\
&\quad \sum_{i=1}^m p_i (\sum_{j=1}^{n_i} q_{ij} \mathbf{E}(\mathcal{S}[\llbracket L_i \rrbracket] \gamma \mid \gamma)) \\
&= \sum_{i=1}^m p_i (\sum_{j=1}^{n_i} q_{ij} (\llbracket d_{ij} \rrbracket)) + \sum_{i=1}^m p_i \mathbf{E}(\mathcal{S}[\llbracket L_i \rrbracket] \gamma \mid \gamma)
\end{aligned}$$

and we further have:

$$\mathcal{S} \left[\bigoplus_{i=1}^m p_i \bigoplus_{j=1}^{n_i} q_{ij} d_{ij} \right] \gamma = \eta_T \left(\sum_{i=1}^m p_i \sum_{j=1}^{n_i} q_{ij} \llbracket d_{ij} \rrbracket \right)$$

So if $M : \sigma$ and $N : \sigma$ are in expectation PR-form over the same L_1, \dots, L_n then, for $\gamma : \llbracket \sigma \rrbracket \rightarrow \mathbf{R}$, we have:

$$\mathbf{E}(\mathcal{S}\llbracket M \rrbracket \gamma \mid \gamma) \geq \mathbf{E}(\mathcal{S}\llbracket N \rrbracket \gamma \mid \gamma) \iff \mathcal{S}\llbracket \mathbf{E}_s(M) \geq \mathbf{E}_s(N) \rrbracket \gamma = \eta_{\mathbf{T}}(0)$$

(recall that $\llbracket \mathbf{tt} \rrbracket = 0$). With this observation, the pointwise argument for the last two equation schemas goes through. \square

In the case of $\mathcal{S}_{\mathbf{T}_2}$ we inherit Equation 16 from \mathbf{T}_2 so we additionally have:

$$x \cdot M +_p y \cdot M = (x +_p y) \cdot M$$

Let \mathbf{Ax}_2 be \mathbf{Ax}_1 extended with this equation. In the case of $\mathcal{S}_{\mathbf{T}_3}$ we inherit Equation 17 from \mathbf{T}_3 so we have the stronger:

$$x \cdot M +_p y \cdot N = (x +_p y) \cdot M +_p (x +_p y) \cdot N$$

Let \mathbf{Ax}_3 be \mathbf{Ax}_1 extended with this equation.

Unfortunately, we do not have any results analogous to Theorem 6 for any of the above three axiom systems for the probabilistic case, and further axioms may well be needed to obtain completeness for program equivalence at basic types holds. We do, however have a completeness result for purity at basic types.

First, some useful consequences of these equations, are the following, where M and N are expectation PR-values:

$$M \text{ or } N = M \quad (\text{if } \mathbf{E}_s(M) \geq \mathbf{E}_s(N)) \quad (\text{PR}_1)$$

$$M \text{ or } N = N \quad (\text{if } \mathbf{E}_s(M) < \mathbf{E}_s(N)) \quad (\text{PR}_2)$$

$$(M \text{ or } L) \text{ or } N = M \text{ or } L \quad (\text{if } \mathbf{E}_s(M) \geq \mathbf{E}_s(N)) \quad (\text{PR}_3)$$

$$(M \text{ or } L) \text{ or } N = L \text{ or } c' \cdot N \quad (\text{if } \mathbf{E}_s(M) < \mathbf{E}_s(N)) \quad (\text{PR}_4)$$

Next, our equational system \mathbf{Ax}_1 allows us to put programs of basic type into a weak canonical form. First consider those which are *PR-effect values*, i.e., programs obtained by probabilistic and reward combinations of constants. Every such term is provably equivalent to one of the form $\sum_{j=1}^n p_j(d_j \cdot c_j)$ where, $n > 0$, the d_j and the c_j are constants and no pair $\langle d_j, c_j \rangle$ is repeated. We call such terms *canonical PR-effect values*, and do not distinguish any two such if they are identical apart from the ordering of the $d_j \cdot c_j$.

We say that an effect value of basic type is in *weak canonical form* if (ignoring bracketing of *or*) it is an effect value of the form

$$E_1 \text{ or } \dots \text{ or } E_n$$

where $n > 0$, the E_i are canonical PR-effect values, and no E_i occurs twice. (We could have simplified canonical forms further by applying the PR_i , obtaining a stronger canonical form, but did not do so as, in any case, we do not have an equational completeness result.)

Lemma 25. *Every program M of basic type is provably equal to a weak canonical form $\text{CF}(M)$.*

Proof. By Proposition 4, M can be proved equal to an effect value E . Using the associativity equation and the fact that **reward** and $+_p$ distribute over **or**, the effect value E can be proved equal to a term of the form $E_1 \text{ or } \dots \text{ or } E_n$ where each E_i is a PR-effect term. \square

Say that a theory Ax , valid in S_T , is *strongly purity complete for basic PR-effect values*, if for all PR-effect values $E : b$ we have:

$$\models_{\text{S}_T} E \downarrow_b \implies \exists c : b. \vdash_{\text{Ax}} E = c : b$$

Lemma 26. T_i is strongly purity complete for basic PR-effect values wrt. Ax_i ($i = 1, 3$).

Proof. For T_1 we have already noted that every PR-effect value is provably equal using Ax_1 to a term of the form $\sum_{i=1}^n p_i(d_i \cdot c_i)$ with no $(d_i \cdot c_i)$ repeated. For such a term $\models_{\text{S}_T} E \downarrow_b$ holds iff $n = 1$ and $c_i = 0$. In that case the term is provably equal, using Ax_1 , to c_1 and so $\vdash_{\text{Ax}} E \downarrow_b$. The other two cases are similar: for T_2 we note that every PR-effect value is provably equal using Ax_2 to a term of the form $\sum_{i=1}^n p_i(d_i \cdot c_i)$ with no c_i repeated, and for T_3 we note that every PR-effect value is provably equal using Ax_3 to a term of the form $\sum_{i=1}^n p_i(d \cdot c_i)$ with no c_i repeated. \square

Theorem 11 (General purity completeness). *Let Ax be a theory extending Ax_1 that is valid in S_T . If Ax is strongly purity complete for basic PR-effect values, then it is strongly purity complete at basic types, i.e., for all programs $M : b$ we have:*

$$\models_{\text{S}_T} M \downarrow_b \implies \exists c : b. \vdash_{\text{Ax}} M = c : b$$

Proof. Note first that, in general, for any term $N : b$ and any PR-effect value $E : b$, if $\models_{\text{S}_T} N \downarrow_b$ and $\text{S}_T[N](\gamma) = \text{S}_T[E](\gamma)$ for some γ then $\text{S}_T[N] = \text{S}_T[E]$ (and so $\models_{\text{S}_T} E \downarrow_b$). Using the strong purity completeness assumption, it follows that, for some c , $\vdash_{\text{Ax}} E = c$.

Suppose T is strongly purity complete for basic PR-effect values wrt. a theory Ax extending Ax_1 and valid in S_T . It suffices to prove the claim for terms M in weak canonical form, i.e., of the form

$$E_1 \text{ or } \dots \text{ or } E_n$$

where $n > 0$, and the E_i are canonical PR-effect values. We proceed by induction on n .

So suppose that $\models_{\text{S}_T} M \downarrow_b$. For some E_{i_0} we have $\text{S}_T[M](0) = \text{S}_T[E_{i_0}](0)$, and so $\text{S}_T[N] = \text{S}_T[E_{i_0}]$ and there is a $c_{i_0} : b$ such that $\vdash_{\text{Ax}} E_{i_0} = c_{i_0}$. In case $n = 1$ we have shown that $\vdash_{\text{Ax}} M = c$ for some $c : b$, as required. Otherwise consider $E_{i_1} = \sum_{j=1}^n p_j(d_j \cdot c_j)$

for an $i_1 \neq i_0$. If every c_j is c_{i_0} then both E_{i_0} and E_{i_1} can be put into expectation PR-value form over c_{i_0} , one of the equations **PR₁**-**PR₄** can be used to reduce the size of M , and the induction hypothesis can be applied. Otherwise, some c_{j_1} is not c_{i_0} . Choose $l < r \in \mathbb{R}$ and define $\gamma : \llbracket b \rrbracket \rightarrow \mathbb{R}$ by setting $\gamma(c) = r$ for $c \neq c_{i_0}$, and $\gamma(c_{i_0}) = r_0 + l$, where r_0 is the least of the $\llbracket d_j \rrbracket$. Then we have:

$$\begin{aligned} \mathbf{E}(\mathbf{S}_T \llbracket E_{i_1} \rrbracket \gamma \mid \gamma) &= \sum_{j=1}^n p_j (\llbracket d_j \rrbracket + \gamma(c_j)) \\ &\geq \sum_{j=1}^n p_j (r_0 + \gamma(c_j)) \\ &= r_0 + \sum_{j=1}^n p_j \gamma(c_j) \\ &> r_0 + \gamma(c_{i_0}) \\ &= \mathbf{E}(\mathbf{S}_T \llbracket E_{i_0} \rrbracket \gamma \mid \gamma) \end{aligned}$$

with the strict inequality holding as all the $\gamma(c_j)$ are $\geq \gamma(c_{i_0})$ and $\gamma(c_{j_1}) = r > l = \gamma(c_{i_0})$.

So for some E_{i_2} , with $i_2 \neq i_0$ we have $\mathbf{S}_T \llbracket M \rrbracket (\gamma) = \mathbf{S}_T \llbracket E_{i_2} \rrbracket (\gamma)$, and so $\mathbf{S}_T \llbracket M \rrbracket = \mathbf{S}_T \llbracket E_{i_2} \rrbracket$ and there is a $c_{i_2} : b$ such that $\vdash_{\text{Ax}} E_{i_2} = c_{i_2}$. As Ax is valid in \mathbf{S}_T we have $\mathbf{S}_T \llbracket c_{i_0} \rrbracket = \mathbf{S}_T \llbracket c_{i_2} \rrbracket$ and so $c_{i_0} = c_{i_2}$ (monad units are always injective and so is $\llbracket - \rrbracket_b$). We can therefore replace E_{i_0} and E_{i_2} by c_{i_0} , apply **PR₁** or **PR₃** to obtain a shorter canonical form, and apply the induction hypothesis, concluding the proof. \square

So we see that purity completeness at basic types holds for T_i wrt. Ax_i ($i = 1, 3$).

There is a “cheap” version of the free-algebra monad \mathbf{C} discussed at the end of Section 4.6 for general auxiliary monads T . Take Ax to be the set of equations between effect values that hold in \mathcal{S}_T , and take \mathbf{D} to be the corresponding free algebra monad, yielding a corresponding denotational semantics \mathcal{D} . Then we have:

$$\models_{\mathcal{D}} E = E' : b \iff \vdash_{\text{Ax}} E = E' : b \iff \models_{\mathcal{S}_T} E = E' : b$$

Assuming \mathbb{B} characteristic for T and reward addition injective for T , using Theorems 3 and 10 we then obtain a version of Theorem 6 for \mathcal{D} for numerable b :

$$\vdash_{\text{Ax}} M = N : b \iff \models_{\mathcal{C}} M = N : b \iff M \approx_b N$$

6 Conclusion

This paper studies decision-making abstractions in the context of simple higher-order programming languages, focusing on their semantics, treating them operationally and denotationally. The denotational semantics are compositional. They are based on the selection monad, which has rich connections with logic and game theory. Unlike other programming-language research (e.g., [2, 27]), the treatment of games in this paper is extensional, focusing on choices but ignoring other aspects of computation, such as function calls and returns. Moreover, the games are one-player games. Going further, we have started to explore extensions of our languages with multiple players, where each choice and each reward is

associated with one player. For example, writing **A** and **E** for the players, we can program a version of the classic prisoners’s dilemma:

```
let silentA, silentE : Bool be (tt orA ff), (tt orE ff) in
if silentA and silentE then - 1 ·A - 1 ·E *
else if silentA then - 3 ·A *
else if silentE then - 3 ·E *
else - 2 ·A - 2 ·E *
```

Here, **silent_A** and **silent_E** indicate whether the players remain silent, and the rewards, which are negative, correspond to years of prison. Many of our techniques carry over to languages with multiple players, which give rise to interesting semantic questions (e.g., should we favor some players over others? require Nash equilibria?) and may also be useful in practice.

In describing Software 2.0, Karpathy suggested specifying some goal on the behavior of a desirable program, writing a “rough skeleton” of the code, and using the computational resources at our disposal to search for a program that works [29]. While this vision may be attractive, realizing it requires developing not only search techniques but also the linguistic constructs to express goals and code skeletons. In the variant of this vision embodied in SmartChoices, the skeleton is actually a complete program, albeit in an extended language with decision-making abstractions. Thus, in the brave new world of Software 2.0 and its relatives, programming languages still have an important role to play, and their study should be part of their development. Our paper aims to contribute to one aspect of this project; much work remains.

In comparison with recent theoretical work on languages with differentiation (e.g., [23, 1, 3, 8, 13, 26]), our languages are higher-level: they focus on how optimization or machine learning may be made available to a programmer rather than on how they would be implemented. However, a convergence of these research lines is possible, and perhaps desirable. One thought is to extend our languages with differentiation primitives to construct selection functions that use gradient descent. These would be alternatives to argmax as discussed in the Introduction. Monadic reflection and reification, in the sense of Filinski [22], could support the use of such alternatives, and more generally enhance programming flexibility. Similarly, it would be attractive to deepen the connections between our languages and probabilistic ones (e.g., [24]). It may also be interesting to connect our semantics with particular techniques from the literature on MDPs and RL, and further to explore whether monadic ideas can contribute to implementations that include such techniques. Finally, at the type level, the monadic approach distinguishes “selected” values and “ordinary” ones; the “selected” values are reminiscent of the “uncertain” values of type `Uncertain <T>` [6], and the distinction may be useful as in that setting.

Acknowledgements

We are grateful to Craig Boutilier, Eugene Brevdo, Daniel Golovin, Michael Isard, Eugene Kirpichov, Ohad Kammar, Matt Johnson, Dougal Maclaurin, Martin Mladenov, Adam Paszke, Sam Staton, Dimitrios Vytiniotis, and Jay Yagnik for discussions.

References

- [1] Martín Abadi and Gordon D. Plotkin. A simple differentiable programming language. *Proc. ACM Program. Lang.*, 4(POPL):38:1–38:28, 2020.
- [2] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.
- [3] Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. On the versatility of open logical relations - continuity, automatic differentiation, and a containment theorem. In Peter Müller, editor, *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12075 of *Lecture Notes in Computer Science*, pages 56–83. Springer, 2020.
- [4] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.
- [5] Joe Bolt, Jules Hedges, and Philipp Zahn. Sequential games and nondeterministic selection functions. *CoRR*, abs/1811.06810, 2018.
- [6] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. Uncertain_{ij}: a first-order type for uncertain data. In Rajeev Balasubramonian, Al Davis, and Sarita V. Adve, editors, *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14*, pages 51–66. ACM, 2014.
- [7] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level robot programming in the situation calculus. In *Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2000.
- [8] Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. *Proc. ACM Program. Lang.*, 4(POPL):64:1–64:27, 2020.
- [9] David Budden, Matteo Hessel, John Quan, and Steven Kapturowski. RLax: Reinforcement Learning in JAX, 2020.

- [10] Vladimir Bychkovsky. Spiral: Self-tuning services via real-time machine learning, 2018. Blog post [here](#).
- [11] Victor Carbune, Thierry Coppey, Alexander N. Daryin, Thomas Deselaers, Nikhil Sarda, and Jay Yagnik. Predicted variables in programming. *CoRR*, abs/1810.00619, 2018.
- [12] Kai-Wei Chang, He He, Stéphane Ross, Hal Daumé III, and John Langford. A credit assignment compiler for joint prediction. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1705–1713, 2016.
- [13] Geoff Cruttwell, Jonathan Gallagher, and Ben MacAdam. Towards formalizing and extending differential programming using tangent categories. *Proc. ACT*, 2019.
- [14] Martín Escardó. Constructive decidability of classical continuity. *Math. Struct. Comput. Sci.*, 25(7):1578–1589, 2015.
- [15] Martin Escardó and Paulo Oliva. Sequential games and optimal strategies. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2130):1519–1545, 2011.
- [16] Martín Escardó and Paulo Oliva. The Herbrand functional interpretation of the double negation shift. *J. Symb. Log.*, 82(2):590–607, 2017.
- [17] Martín Hötzel Escardó and Paulo Oliva. Selection functions, bar recursion and backward induction. *Math. Struct. Comput. Sci.*, 20(2):127–168, 2010.
- [18] Martín Hötzel Escardó and Paulo Oliva. The Peirce translation. *Ann. Pure Appl. Log.*, 163(6):681–692, 2012.
- [19] Martín Hötzel Escardó, Paulo Oliva, and Thomas Powell. System T and the product of selection functions. In Marc Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 233–247. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.
- [20] Matthias Felleisen and Daniel P. Friedman. Control operators, the secd-machine, and the λ -calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.

- [21] William Feller. *An introduction to probability theory and its applications, vol 2*. John Wiley & Sons, 2008.
- [22] Andrzej Filinski. Representing monads. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '94*, page 446–457. Association for Computing Machinery, 1994.
- [23] Brendan Fong, David I. Spivak, and Rémy Tuyéras. Backprop as functor: A compositional perspective on supervised learning. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.
- [24] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. *CoRR*, abs/1206.3255, 2012.
- [25] Jules Hedges. The selection monad as a CPS transformation. *CoRR*, abs/1503.06061, 2015.
- [26] Mathieu Huot, Sam Staton, and Matthijs Vákár. Correctness of automatic differentiation via diffeologies and categorical gluing. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2020.
- [27] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.
- [28] Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1-3):70–99, 2006.
- [29] Andrej Karpathy. Software 2.0, 2017. Blog post [here](#).
- [30] Klaus Keimel and Gordon D. Plotkin. Mixed powerdomains for probability and non-determinism. *Log. Methods Comput. Sci.*, 13(1), 2017.
- [31] Anders Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.
- [32] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003.

- [33] Aliaume Lopez and Alex Simpson. Basic operational preorders for algebraic effects in general, and for combined probability and nondeterminism in particular. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPICs*, pages 29:1–29:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- [34] John McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 33 – 70. Elsevier, 1963.
- [35] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.
- [36] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989.
- [37] Gordon Plotkin and John Power. Adequacy for algebraic effects. In Furio Honsell and Marino Miculan, editors, *Foundations of Software Science and Computation Structures*, pages 1–24, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [38] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.
- [39] Dieter Pumplün and Helmut Röhr. Convexity theories iv. klein-hilbert parts in convex modules. *Applied Categorical Structures*, 3(2):173–200, 1995.
- [40] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description, 2011. Official language of the uncertainty track of the Seventh International Planning Competition.
- [41] Marshall Harvey Stone. Postulates for the barycentric calculus. *Annali di Matematica Pura ed Applicata*, 29(1):25–30, 1949.
- [42] Tim Vieira, Matthew Francis-Landau, Nathaniel Wesley Filardo, Farzad Khorasani, and Jason Eisner. Dyna: Toward a self-optimizing declarative language for machine learning applications. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2017, page 8–17. Association for Computing Machinery, 2017.
- [43] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees: representing recursive and impure programs in coq. *Proc. ACM Program. Lang.*, 4(POPL):51:1–51:32, 2020.