

# Towards a *UTP*-style framework to deal with probabilities

Riccardo Bresciani, Andrew Butterfield \*



**Abstract** We present an encoding of the semantics of the probabilistic guarded command language (*pGCL*) in the Unifying Theories of Programming (*UTP*) framework. Our contribution is a *UTP* encoding that captures *pGCL* programs as predicate-transformers, on predicates over probability distributions on before- and after-states: these predicates capture the same information as the models traditionally used to give semantics to *pGCL*; in addition our formulation allows us to define a generic choice construct, that covers conditional, probabilistic and non-deterministic choice. We introduce the concept of probabilistic refinement in this framework. This technical report gives a rigorous presentation of our framework, along with a variety of proofs and examples (including the well-known Monty Hall problem), that help to explain it.

## Contents

1	Introduction . . . . .	2
1.1	<i>UTP</i> : general principles . . . . .	2
1.2	<i>pGCL</i> . . . . .	3
1.3	Probabilistic <i>UTP</i> . . . . .	6
1.4	Other background material . . . . .	6
2	States and distributions, informally . . . . .	9
3	Programs . . . . .	11
3.1	Program constructs . . . . .	11
3.2	Deriving pre-expectations . . . . .	13
3.3	More on choice constructs . . . . .	13
4	Refinement . . . . .	15
4.1	Probabilistic refinement . . . . .	16
5	States and distributions, formally . . . . .	17
5.1	States . . . . .	17
5.2	Distributions . . . . .	18
6	Assignments . . . . .	21
6.1	The inverse-image set . . . . .	23
6.2	The remap operator . . . . .	23
7	Conclusion and future work . . . . .	24
A	Examples . . . . .	25
A.1	Definitions . . . . .	25
A.2	Interaction of probabilistic and non-deterministic choice . . . . .	27
A.3	The Monty Hall program . . . . .	29
B	Proofs . . . . .	35
C	Notation . . . . .	44

\*The present work has emanated from research supported by Science Foundation Ireland grant 08/RFP/CMS1277 and, in part, by Science Foundation Ireland grant 03/CE2/I303\_1 to Lero — the Irish Software Engineering Research Centre.

\*Foundations and Methods Group, Trinity College Dublin & Lero@TCD — {bresciar, butrfield}@scss.tcd.ie

## 1 Introduction

Formal program verification allows us to prove that a program complies with its specification and that it does not generate faulty behaviour of any kind, and in general that certain properties hold when a given program is run.

This aim is achieved by writing a model of the program and subsequently verifying the model.

Nonetheless this provides no information regarding the probability that a property will hold: sometimes it is more useful to know what are the odds that a property holds, rather than “simply” assert that it does not always hold.

The purpose of this work is to develop a *UTP*<sup>1</sup>-style framework where we can express probabilistic programs, featuring both probabilistic choice and non-determinism: we aim at being able to do in *UTP* all of the things that are feasible in *pGCL*<sup>2</sup>. *UTP* is based on (state-)predicate transformers, whereas probabilistic models typically involve distributions over states, and so the best way to integrate probability into the *UTP* framework is not obvious.

We aim at constructing a theory of probabilistic programs that is expressed using predicate-transformers<sup>3</sup>.

### 1.1 *UTP*: general principles

The Unifying Theories of Programming (*UTP*) research activity seeks to bring models of a wide range of programming and specification languages under a single semantic framework in order to be able to reason formally about their integration [HJ98; DS06; But10; Qin10]. A success in this area has been the development of the *Circus* language [OCW09], which is a fusion of Z and CSP, with a *UTP* semantics, providing specifications using a “state-rich” process algebra along with a refinement calculus; recent extensions to *Circus* have included timed [SH03] and synchronous [GB09] variants. Recent interest in aspects of the POSIX filestore case study in the Verification Grand Challenge [FWB08] has led us to consider integrating probability into *UTP*, with a view to eventually having a probabilistic variant of *Circus*.

Theories in *UTP* are expressed as second-order predicates<sup>4</sup> over a pre-defined collection of free observation variables, referred to as the *alphabet* of the theory. The predicates are generally used to describe a relation between a before-state and an after-state, the latter typically characterised by dashed versions of the observation variables. For example, a program using two variables  $x$  and  $y$  might be characterised by having the set  $\{x, x', y, y'\}$  as an alphabet, and the meaning of the assignment  $x := y + 4$  would be described by the predicate

$$x' = y + 4 \wedge y' = y.$$

In effect *UTP* uses predicate calculus in a disciplined way to build up a relational calculus for reasoning about programs.

In addition to observations of the values of program variables, often we need to introduce observations of other aspects of program execution via so-called auxiliary variables. So, for example, in order to reason about total correctness, we need to introduce boolean observations that record the starting ( $ok$ ) and termination ( $ok'$ ) of a program, resulting in the above assignment having the following semantics:

$$ok \Rightarrow ok' \wedge x' = y + 4 \wedge y' = y$$

(if started, it will terminate, and the final value of  $x$  will equal the initial value of  $y$  plus four, with  $y$  unchanged).

A problem with allowing arbitrary predicate calculus statements to give semantics is that it is possible to write unhelpful predicates such as  $\neg ok \Rightarrow ok'$ , which describes a “program” that must terminate when not started. In order to avoid assertions that are either nonsense or infeasible, *UTP*

<sup>1</sup>Unifying Theories of Programming. [HJ98; Heh06]

<sup>2</sup>probabilistic Guarded Command Language. [MM04]

<sup>3</sup>So probabilistic programs are predicates too (with apologies to C.A.R. Hoare [Hoa85a])!

<sup>4</sup> Most definitions are in fact first-order, but we need second-order in order to handle the notion of “healthiness”, and recursion.

$abort$	$\hat{=} true$	failure/chaos
$skip$	$\hat{=} ok \Rightarrow ok' \wedge v' = v$	do nothing
$x := e$	$\hat{=} ok \wedge e \text{ is defined} \Rightarrow ok' \wedge x' = e \wedge v' = v$	assignment
$P_1 ; P_2$	$\hat{=} \exists ok_m, v_m \bullet P_1[ok_m, v_m / ok', v'] \wedge P_2[ok_m, v_m / ok, v]$	seq. comp.
$P_1 \triangleleft c \triangleright P_2$	$\hat{=} c \wedge P_1 \vee \neg c \wedge P_2$	conditional
$P_1 \sqcap P_2$	$\hat{=} P_1 \vee P_2$	non-det. choice
$c * P$	$\hat{=} \mu X \bullet (P; X) \triangleleft c \triangleright skip$	while

Figure 1: *UTP* Design semantics of simplified *GCL*

adopts the notion of “healthiness conditions” which are monotonic idempotent predicate transformers whose fixpoints characterise sensible (healthy) predicates. Collections of healthy predicates typically form a sub-lattice of the original predicate lattice under the reverse implication ordering [HJ98, Chp. 3]. Key in *UTP* is a general notion of program refinement as the universal closure of reverse implication<sup>5</sup>:

$$S \sqsubseteq P \hat{=} [P \Rightarrow S]$$

Program  $P$  refines  $S$  if for all observations (free variables),  $S$  holds whenever  $P$  does. The *UTP* framework also uses Galois connections to link different languages/theories with different alphabets [HJ98, Chp. 4], and often these manifest themselves as further modes of refinement.

Of interest to use here is the theory of “Designs” which characterises total correctness for imperative programs. A *UTP* Design semantics of a variant of Dijkstra’s guarded command language (*GCL*, [Dij76]) is shown in Figure 1.

We note in passing that *UTP* follows the key principle that “programs are predicates” [Hoa85a] and so does not distinguish between the syntax of some language and its semantics as alphabetised predicates.

## 1.2 *pGCL*

The approach to probabilistic systems that is presented in [MM97] and later in [MM05] (and more extensively in the book [MM04]) is the one of using expectation transformers of *pGCL* to reason about probabilistic programs: this subsection is dedicated to briefly introduce this, as *pGCL* is the most important reference for our work.

In Dijkstra’s *GCL* the *weakest precondition* is a predicate  $wp.prog.Post$  that is true in those *initial states* that guarantee that the postcondition  $Post$  will be reached after running  $prog$ . [Dij76] *pGCL* is given a semantics that generalises this concept to what they term a *weakest pre-expectation semantics* [MM97; MM04; MM05; NM10].

An expectation is a *function describing how much each program state is “worth”* [MM04] and assigns a weight (a non-negative real number) to program states: it is therefore a random variable. An expectation corresponding to a predicate can be defined as a random variable that maps a state to 1 if it satisfies the predicate and to 0 otherwise. Arithmetic operators and relations are extended pointwise to expectations, as is multiplication by a scalar.

If  $PostE$  is a (post-)expectation after running program  $prog$ , then  $wp.prog.PostE$  is the correspondent weakest<sup>6</sup> (pre-)expectation before the program runs: for each state it returns the minimum expected final weight.

<sup>5</sup>Square brackets denote universal closure —  $[P]$  asserts that  $P$  is true for all values of its free variables.

<sup>6</sup>One expectation is weaker than another if for all states it returns at most the same weight — it is the  $\leq$  relation lifted pointwise.

$$\begin{aligned}
wp.abort.PostE &\triangleq 0 \\
wp.skip.PostE &\triangleq PostE \\
wp.(x := e).PostE &\triangleq PostE\{e/x\} \\
wp.(prog_1; prog_2).PostE &\triangleq wp.prog_1.(wp.prog_2.PostE) \\
wp.(prog_1 \sqcap prog_2).PostE &\triangleq \min\{wp.prog_1.PostE, wp.prog_2.PostE\} \\
wp.(prog_1 \oplus_p prog_2).PostE &\triangleq p \cdot wp.prog_1.PostE + (1 - p) \cdot wp.prog_2.PostE \\
wp.(\mu xxx \bullet C).PostE &\text{ is to be defined with usual concepts from the least-fixpoint theory}
\end{aligned}$$

Figure 2:  $wp$ -semantics of  $pGCL$ , adapted from [MM04, p. 26]. The notation  $PostE\{e/x\}$  denotes the expression describing  $PostE$  with all free occurrences of  $x$  replaced by  $e$ .

Here is the syntax of  $pGCL$ :

$$\begin{aligned}
prog ::= & \text{abort} \\
& | \text{skip} \\
& | x := e \\
& | prog_1; prog_2 \\
& | prog_1 \oplus_p prog_2 \\
& | prog_1 \sqcap prog_2 \\
& | (\mu xxx \bullet C)
\end{aligned}$$

The probabilistic choice operator is the only one that is not present in Dijkstra's original  $GCL$ : it denotes a statement that executes  $prog_1$  with probability  $p$ , and  $prog_2$  with probability  $(1 - p)$ .

Two models can be found in McIver and Morgan's book [MM04]: the first one is a *probabilistic predicate-transformer model*, that uses the weakest pre-expectation semantics shown in Figure 2.

From the assignment semantics, we can see that in some sense when computing the weakest pre-expectation we are going backwards, as we are "translating" the meaning of a  $PostE$  in terms of the states we have before it.

The key features to note in this semantics are that probabilistic choice is the obvious weighting of its alternatives' expectations, whereas demonic choice returns the pointwise minimum.

Non-determinism is crucial in order to define a sensible refinement relation:

$$spec \sqsubseteq prog \triangleq \forall PostE \bullet wp.spec.PostE \leq wp.prog.PostE$$

A program  $prog$  refines a specification  $spec$  if the minimum expected weight for each state after  $prog$  has run is at least as much as we would get after  $spec$  has run.

More formally, in  $wp$ -semantics:

- when we talk about expectations, we talk about elements from the *expectation space*<sup>7</sup> over the state space  $S$ :

$$\mathbb{E}S \triangleq (S \rightarrow \mathbb{R}^+, \geq)$$

- the *expectation transformer model* for programs is:

$$\mathbb{T}S \triangleq (\mathbb{E}S \rightarrow \mathbb{E}S, \sqsubseteq)$$

- we can think of  $wp$  as a function that transforms a program into an expectation transformer:

$$wp : Programs \rightarrow \mathbb{T}S$$

<sup>7</sup> $\geq$  is the ordering in  $\mathbb{R}$  lifted pointwise and  $\mathbb{R}^+ = [0, +\infty)$ .

So we have that:

- $wp.prog \in \mathbb{TS}$
- $PostE \in \mathbb{ES}$
- $wp.prog.PostE \in \mathbb{ES}$

An alternative is the *probabilistic relational model* [HSM97; MM04], which sees a program as a relation from states to up-, convex- and Cauchy-closed sets of probability distributions over the state space. It is possible to see programs as relations from probability distributions to sets of probability distributions via the Kleisli composition of programs [MM04, Chp. 5]: we are going to use a similar approach to port this work to the UTP framework.

A (demonic) probabilistic program takes an initial state to a (set of) fixed final probability distributions over  $S$ :

- the set of *sub-distributions* over  $S$  is:

$$\bar{S} \triangleq \{\Delta : S \rightarrow [0, 1] \mid \Sigma \Delta \leq 1\}$$

- the space of deterministic probabilistic programs over  $S$  is defined:

$$\mathbb{DS} \triangleq (S \rightarrow \bar{S}, \sqsubseteq)$$

- set of up-, convex- and Cauchy-closed sets of discrete distributions over the state space, *i.e.* those which comply with some healthiness criteria:

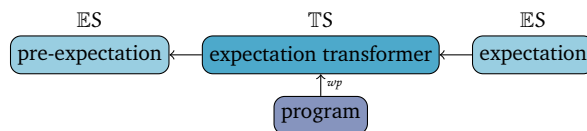
$$\mathbb{CS} \subseteq \wp \bar{S}$$

- complete partial order of demonic probabilistic programs:

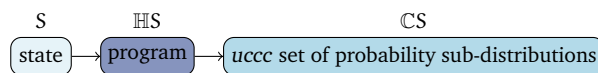
$$\mathbb{HS} \triangleq (S \rightarrow \mathbb{CS}, \sqsubseteq)$$

Summarizing in *pGCL* we have:

- a *probabilistic predicate-transformer model*, that takes a program and turns it into an expectation transformer. This can be applied to an expectation to derive the corresponding pre-expectation:



- a *probabilistic relational model*, that relates a state to a up-, convex- and Cauchy-closed set of probability sub-distributions:



To conclude this brief presentation of *pGCL*, here is a representative sample of laws about probabilistic programs, that it is possible to prove in this framework:

$$\begin{aligned}
 A \sqcap B &\sqsubseteq A \text{ }_p\oplus B \\
 (A \sqcap B) \text{ }_p\oplus C &= (A \text{ }_p\oplus C) \sqcap (B \text{ }_p\oplus C) \\
 (A \sqcap C) \text{ }_p\oplus (B \sqcap C) &\sqsubseteq (A \text{ }_p\oplus B) \sqcap C \\
 (A \sqcap B); C &= (A; C) \sqcap (B; C) \\
 A; (B \sqcap C) &\sqsubseteq (A; B) \sqcap (A; C)
 \end{aligned}$$

In §3.3 we will see that we are able to state similar laws in our framework — they will have a broader scope, as they will be covering a generic choice construct.

### 1.3 Probabilistic UTP

There has already been a certain amount of work looking at encoding probability in a UTP setting. He and Sanders have presented an approach unifying probabilistic choice with standard constructs [HS06], and this work provides an example of how the laws of pGCL could be captured in UTP as predicates about program equivalence and refinement. However only an axiomatic semantics was presented, and the laws were justified via a Galois connection to an expectation-based semantic model.

Sanders and Chen then explored an approach that decomposed demonic choice into a combination of pure probabilistic choice and a unary operator that accounted for demonic behaviour [CS09]. There they commented on the lack of a satisfactory UTP theory, where probabilistic and demonic choice coexist.

A probabilistic BPEL-like language has recently been described by He [He10] that gives a UTP-style semantics for a web-based business semantics language. This language is GCL with extra constructs to handle probabilistic choice and compensations and coordination operators, including exception handling. The UTP model that is developed does not relate before- and after-variables of the same type, but instead uses predicates to encode a relationship between an initial state and a final probability distribution over states.

What is still missing is a presentation of pGCL in UTP that is defined in terms of a before/after relation over the same observation space. We believe the ideal such presentation would use observations that corresponded to program variables and to other aspects of behaviour such as termination, in a manner analogous to our brief earlier presentation of GCL in UTP: here we present a UTP encoding of pGCL semantics based on probability distributions over the set of possible states, relating a before-distribution ( $\delta$ ) to an after-one ( $\delta'$ ), effectively making use of one observation. The key contributions here are the fact that we provide a means by which reasoning can still be carried out at program variable level, and we have uncovered a generic notion of choice that subsumes probabilistic, demonic and conditional choices.

### 1.4 Other background material

Besides the works we have mentioned so far, which are our main reference, the foundation of this work is also represented by all that has been done on probability and logic, in particular that part concerning the interaction between these two topics.

There is a work dating back to 1990 by Fagin, Halpern, and Megiddo [FHM90] where the authors present a logic to reason about probabilities (but still not to reason about formulas that can have a value which is probabilistically true or false). This paper sets ideas that can be found in different other papers, and among such ideas we can find the *Dempster-Shafer belief theory* and *Bayesian networks*, which are recurring topics in the literature of logic and probability — in particular Bayesian networks are seen as an area with great potential for the development of probabilistic logic, at least according to Williamson [WIL02].

Halpern and Pucella [HP07] have presented an axiomatization of probabilistic logic, characterizing probabilistic and non-probabilistic expectation, and discussing about expressiveness and satisfiability of such a system.

Argumentation is a technique closely related to logic, that aims at deducting facts starting from given premises: a logic system for probabilistic argumentation, inspired by the Dempster-Shafer belief theory can be found in [Kra<sup>+</sup>95] — a probability is assigned to each proposition and the purpose of the argumentation system is to aggregate these probabilities.

A survey on probabilistic argumentation can be found in [Hae<sup>+</sup>01] and [Koh03]: there have been different developments of probabilistic argumentation systems and the one presented in these papers is based both on logic and probability theory, where probability is used to weight arguments for and against a particular conclusion. Haenni et al. [Hae<sup>+</sup>01] state: “the strength of our method comes certainly from this simple way of combining logic and probability theory”, and this results in the existence of efficient computational methods.

An application of probabilistic argumentation can be seen in [KJH08], where it is used to make trust evaluations.

The majority of approaches to merge logic and probability try to accomplish this task either by defining a probability function on the sentences of logic, or by incorporating probabilities in logic itself: a common framework to link probabilistic argumentation theory and other probabilistic logics is proposed in [Hae<sup>+</sup>08].

An interesting approach is presented in [Jøs01], which borrows ideas from the Dempster-Shafer belief theory. It uses a belief functions to evaluate the probability of a state and makes use of sets of substates to define elementary probabilities; it is somehow a three-value logic, as for each uncertain predicate (*opinion*) there is a *belief function*, a *disbelief function* and an *uncertainty function*, which sum up to 1.

Another example of a subject using probability and logic is *probabilistic logic learning*, that adds also machine learning to the picture: in [DK03] the authors present a probabilistic logic, by adding probability to first order logic through Bayesian networks. The authors mention also the possibility of modelling relations among objects: for this purpose Bayesian networks are not enough, and logical/relational Markov models have to be exploited (eventually with some extension, as proposed by Jain, Kirchlechner, and Beetz [JKB07]).

On the side of process algebras, probabilistic CSP [Mor<sup>+</sup>96] is obtained by adding probability to Hoare's CSP [Hoa85b]. Probability is defined in such a way that it distributes through all operators. A refinement operator is also defined in this same paper: we have a definition of a probabilistic refinement calculus.

We deal with a *timed specification*, that has a limited validity: this is in line with real-world systems, as they cannot possibly work forever (we simply have to wait long enough for their failure probability to raise), and for this reason we can specify a time limit for which a specification has to be satisfied.

There is a problem regarding the compositionality of probabilistic CSP, which is not straightforward: Morgan [Mor04] explains this using the metaphor of the colour of a child's eye, knowing the colour of the parents' — too much information has to be brought forward if we want accurate information, but simply a phenotypical description is unreliable and not sufficient, as what is enough is to know colour and whether the allele is predominant or recessive. This same kind of information is the one that has to be sought to have an accurate probabilistic compositionality: in fact if we observe an event, we would want to be able to identify the facts that have led to that event.

For example if we observe a failure (*i.e.* a composite event) during the run of a program, we want to track down the reasons of this failure and to identify what factors (*i.e.* base events) have been responsible for the happening.

Another ingredient of probabilistic systems is the choice operator, that can be instantiated in three different ways:

- *demonic choice*, that picks the “worst-case” scenario for that choice;
- *angelic choice*, that picks the “best-case” scenario for that choice;
- *probabilistic choice*, that picks one of the two options with a given probability.

Interactions among demonic, angelic and probabilistic choices may be subtle. In fact a deterministic (although probabilistic) program is characterised by monotonicity, conjunctivity and disjunctivity, when introducing demonic choice we drop disjunctivity; if demonic choice and angelic choice coexist in the same program, we lose also conjunctivity and we remain only with monotonicity. [MM98]

Back in the early eighties Kozen [Koz81; Koz85] had proposed a model for probabilistic programs, that featured probabilistic choice, but left out demonic choice.

Nonetheless it is crucial to retain demonic choice, as it is the basis of refinement calculus and abstraction of programs. [MM04]

When composing processes we must be careful about the issue of *duplication*, which in presence of probabilistic and non-deterministic choice may lead to incorrect results. [Mor<sup>+</sup>95]

An example is given by the issue of the idempotency of the demonic choice operator, which depends on its definition: if the demonic choice operator can distribute through probabilistic choice operators we can have the following behaviour [Mis00]:

$$(A \frac{1}{2} \oplus B) \sqcap (A \frac{1}{2} \oplus B) = A \frac{1}{4} \oplus ((A \sqcap B) \frac{1}{3} \oplus B)$$

The reason for this is that two instances of the same program containing a demonic choice are actually two different programs because of it, as every demonic choice is a unique element.

Another way of seeing this is that it is crucial to know when a choice is made, thus we have to be very careful when we distribute choice operators.

We also need a probabilistic version of healthiness conditions — this is another contribution of this same paper [Mis00]: a probabilistic semantic link between action systems (these are sets of guarded commands, here expressed in *pGCL*) and CSP is provided, and this induces probabilistic versions of the healthiness conditions.

The problem of automatically checking properties involving real numbers (such as probabilities) is a difficult one. A simplifying approach to the problem has been by McIver and Weber [MW05] via a generalization of Kleene algebra: a probabilistic Kleene algebra treats probability implicitly, as it is contained in the fragment of probabilistic programs on which it operates.

The idea is to leave an underlying probabilistic level to be examined once the problem has been simplified through higher level proofs — this takes away non-necessary probabilistic reasoning which is computationally very expensive.

Thus the ideal approach sees a qualitative proof as a first step in the verification process, and subsequently the quantitative model-checking technique is applied [MCM06]: this reduces the problem of the state space explosion — which is a typical problem, especially in a larger setting, such as the one of distributed probabilistic systems.

A model checker that has been tested for verification of probabilistic systems is PRISM [McI06]: this paper bases its approach to the analysis of probabilistic systems using *pGCL* and on the formalism of probabilistic action systems, used to describe an example of wireless communication; the comparison of a process with a more refined one (in terms of details added to the description) shows that there is a consistent increase in the number of transitions and states.

Another application of a probabilistic method to a real algorithm can be found in [MV04] and is the verification of the Miller-Rabin algorithm through *pGCL* and through a probabilistic extension of Hoare logic by den Hartog (also relying on a variation of *GCL*, usually referred to as *pH* — *pGCL* is easier to use, according to the authors).

The theory of probabilistic sequential programs is developed further by Ying [Yin03]: the underlying logic is changed, *i.e.* a probabilistic logic is used instead of ordinary two-valued logic. The paper discusses a different concept of refinement, that is obviously probabilistic, and gives semantics for the language used. One remark about this paper: angelic and demonic choices are taken into account, but it does not handle probabilistic choice.

On the topic of quantum computation, which is obviously closely related to probability issues, an *ad hoc* language has been proposed, namely *qGCL*, which is a variation of *pGCL*. [SZ99]

It features demonic non-determinism and probabilistic choice and has an associated refinement calculus that enables proof of algorithm correctness by formal reasoning. The difference with respect to *pGCL* is the presence of three quantum procedures, namely *initialisation*, *evolution* and *finalization*.

A different approach to reasoning about distributed probabilistic systems can be found in [NS09]: the authors use PTSC, which is a language to describe systems from a perspective that merges probability, time and shared-variable-concurrency, that features a delay operator, guarded assignments, a probabilistic choice operator and also a probabilistic parallel composition.



This work contributes to the theory by extending this language with constructs for interleaving and handling of local scopes.

In recent times a paper by Jun Sun *et al.* [SSL10] has described a probabilistic analysis of the likelihood of a program in a medical device satisfying a safety specification, given that random, but hopefully unlikely events, can prevent the correct behaviour, even if the program is the best one possible. Their probabilistic model checking directly corresponds to the probabilistic refinement we are going to present in §4.1.

Finally it is worth mentioning also [MM02], where the authors give a probabilistic extension of the  $\mu$ -calculus, with an added probabilistic choice operator, and an interesting game interpretation of this calculus is presented.

It is doubtless that a quantitative formal analysis offers great advantages compared to a qualitative one: the challenge is to find a computationally feasible way of dealing with this.

## 2 States and distributions, informally

The purpose of this work is to develop a framework that integrates well into *UTP* and, at the same time, offers an effective way to handle probabilistic choice and non-determinism together in probabilistic programs.

More specifically the *UTP* approach implies that we treat program as predicates, that relate the situation before the program is run to the situation after the program is run.

In *UTP* we usually talk about variables and the values they map to, so a naïve (and quite straightforward) generalization to handle probability would simply consist in mapping variables to pairs containing a values and corresponding probabilities; in this case we would be handling objects with the following shape<sup>8</sup>:

$$\underline{v} \rightarrow (\underline{w} \rightarrow [0..1])$$

Although such an easy generalization may look appealing, this yields wrong results: the reason is that we lose the “entanglement” among the variables, and we should rather use objects with this other shape:

$$(\underline{v} \rightarrow \underline{w}) \rightarrow [0..1]$$

To see this let us consider an example: from an initial situation where  $y$  is initialized to 0 with probability  $1/2$  and to 1 with probability  $1/2$ , after running the simple program  $x := y$  we obtain these descriptions of the resulting situation, with obvious meaning of the notation:

$$\begin{aligned} x \mapsto \begin{pmatrix} 0 \mapsto 1/2 \\ 1 \mapsto 1/2 \end{pmatrix}, y \mapsto \begin{pmatrix} 0 \mapsto 1/2 \\ 1 \mapsto 1/2 \end{pmatrix} & \quad \text{(first case — too naïve, indeed)} \\ \begin{pmatrix} x \mapsto 0 \\ y \mapsto 0 \end{pmatrix} \mapsto 1/2, \begin{pmatrix} x \mapsto 1 \\ y \mapsto 1 \end{pmatrix} \mapsto 1/2 & \quad \text{(second case — we keep the entanglement)} \end{aligned}$$

Now, what about the probability that  $x = y$ ? Obviously this probability is 1, but clearly the first kind of object does not provide enough information to give the appropriate answer, while the second does — so this is the way we will be modelling things.

Let us now start with a few informal definitions for the foundational elements of this framework: *states* and *distributions*.

A *state*  $\sigma$  is a memory mapping, that associates a vector of variables  $\underline{v}$  with a corresponding vector  $\underline{w}$ , whose components are the values contained by the variables in  $\underline{v}$ .

$$\sigma \hat{=} \underline{v} \mapsto \underline{w}$$

<sup>8</sup>We underline whenever we talk about vectors or sets of vectors:  $\underline{A}$  stands for a  $n$ -th dimensional vectorial space  $A \times A \times \dots \times A$ , for an appropriate  $n$ .

The components of  $\underline{v}$ , which are the variables mentioned by a state, constitute its *alphabet*: for the moment we will consider only states having a fixed alphabet  $\mathcal{A}$  — let us note the set of all such states as  $\mathcal{S}_{\mathcal{A}}$ .

Generally speaking we can define a distribution as a function  $\chi$  mapping states to real numbers<sup>9</sup>, and if we restrict ourselves to the interval  $[0..1]$  we have a *probability distribution*  $\delta$ , which associates states with probabilities (for the moment let us consider only those distributions which mention every possible state in the finite set  $\mathcal{S}_{\mathcal{A}}$ ):

$$\delta \triangleq \{ \sigma \mapsto p \mid \sigma \in \mathcal{S}_{\mathcal{A}} \}$$

In a probability distribution  $\delta$  the sum of the probabilities of all states (noted as  $\|\delta\|$ ) cannot exceed 1.

It is possible to operate on distributions by pointwise lifting in an obvious way operators such as addition, product and multiplication by a scalar number.

An interesting case is the one when we multiply a probability distribution by what we term a *weighting distribution*, which is a distribution  $\pi$  mapping states to real numbers in the interval  $[0..1]$ , *without* the constraint  $\|\pi\| \leq 1$ . The resulting probability distribution, noted  $\delta\langle\pi\rangle$ , has the property of being pointwise smaller than  $\delta$ , and will have an important role when defining choice constructs:

$$\delta\langle\pi\rangle \triangleq \{ \sigma \mapsto \pi(\sigma) \cdot \delta(\sigma) \mid \sigma \in \text{dom}(\delta) \}$$

Another example is when we want to select the subset of a distribution  $\delta$ , which comprises only states where a condition  $c$  (which is a boolean expression) is satisfied: for reasons that will become clear later on, we have chosen to overload the above notation and note this as  $\delta\langle c \rangle$ .

$$\delta\langle c \rangle \triangleq \{ \sigma \mapsto \delta(\sigma) \mid \sigma \in \text{dom}(\delta) \text{ satisfies } c \}$$

As the probability of a condition  $c$  to be true on a distribution  $\delta$  can be computed by adding up the probabilities relative to all states that satisfy such a condition, we can express this probability using the notation introduced so far as  $\|\delta\langle c \rangle\|$ .

Modifying the probability associated with a state is not the only operation we are interested in: there are cases when we want to replace a pair  $(\sigma \mapsto p)$  with a pair  $(\sigma' \mapsto p)$ .

Such an operation may seem a little unusual, but it is actually what happens when something alters a state, transforming a *before-state*  $\sigma$  into an *after-state*  $\sigma'$ : the probability of the after-state being  $\sigma'$  is the same as the *before-state* being  $\sigma$ .

Given an assignment  $\underline{v} := \underline{e}$ , where  $\underline{e}$  is a vector of expressions, if we perform this operation on every state of a distribution  $\delta$  we obtain the distribution  $\delta\{\underline{e}/\underline{v}\}$ : the postfix operator  $\{\underline{e}/\underline{v}\}$  modifies  $\delta$  to reflect the modifications introduced by the assignment — the intuition behind this, roughly speaking, is that all states  $\sigma$  where the expression  $\underline{e}$  evaluates to the same value  $\underline{w} = \text{eval}_{\sigma}(\underline{e})$  are replaced by a single state  $\sigma' = (\underline{v} \mapsto \underline{w})$  that maps to a probability that is the sum of the probabilities of the states it replaces.

$$\delta\{\underline{e}/\underline{v}\} \triangleq \{ \sigma' \mapsto \sum \delta(\sigma) \mid \sigma \in \text{dom}(\delta) \wedge \text{eval}_{\sigma}(\underline{e}) = \sigma'(\underline{v}) \}$$

Let us see this on the example we had before:

$$\begin{aligned} \delta &= \left\{ \left( \begin{array}{l} x \mapsto 0 \\ y \mapsto 0 \end{array} \right) \mapsto 1/4, \left( \begin{array}{l} x \mapsto 0 \\ y \mapsto 1 \end{array} \right) \mapsto 1/4, \left( \begin{array}{l} x \mapsto 1 \\ y \mapsto 1 \end{array} \right) \mapsto 1/4, \left( \begin{array}{l} x \mapsto 1 \\ y \mapsto 0 \end{array} \right) \mapsto 1/4 \right\} \\ \delta\{y/x\} &= \left\{ \left( \begin{array}{l} x \mapsto 0 \\ y \mapsto 0 \end{array} \right) \mapsto 1/2, \left( \begin{array}{l} x \mapsto 1 \\ y \mapsto 1 \end{array} \right) \mapsto 1/2 \right\} \end{aligned}$$

This is shown in Figure 3.

<sup>9</sup>In other words, it is a real-valued random variable — *pGCL* expectations are therefore distributions with the additional constraint of having only non-negative values.

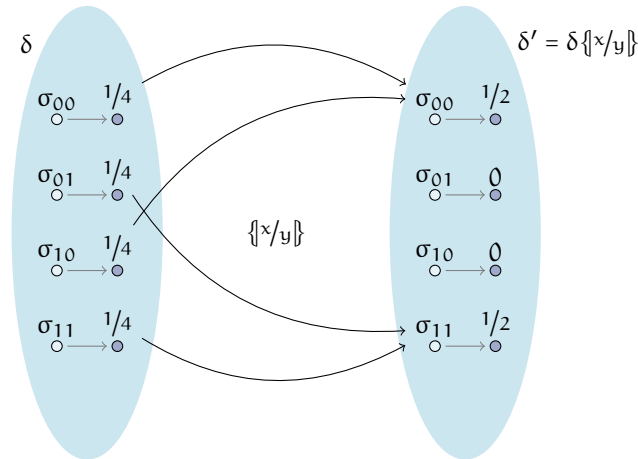


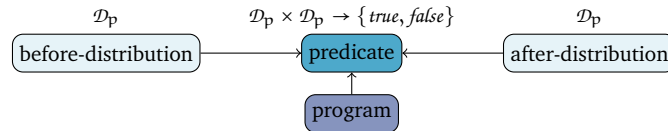
Figure 3: The assignment in the example.

If we had had the assignment  $x := 4$  instead of  $x := y$ , the result would have been:

$$\delta \{4/x\} = \left\{ \left( \begin{array}{l} x \mapsto 4 \\ y \mapsto 0 \end{array} \right) \mapsto 1/2, \left( \begin{array}{l} x \mapsto 4 \\ y \mapsto 1 \end{array} \right) \mapsto 1/2 \right\}$$

### 3 Programs

Once we have these foundational elements, we can build predicates<sup>10</sup> that talk about probability distributions.



A *program*  $A(\delta, \delta')$  is a particular predicate that links a *before-distribution*  $\delta$ , that describes the situation before running the program (in terms of the probabilities associated with the possible before-states), to an *after-distribution*  $\delta'$ , that describes the situation after running the program (in terms of the probabilities associated with the possible after-states).

The *program image*  $A(\delta)$ <sup>11</sup> is the set of all possible after-distributions  $\delta'$  that satisfy the program  $A(\delta, \delta')$ .

The *probability*  $p$  of the condition  $c$  being satisfied by the program image  $A(\delta)$  lies in the set  $\|A(\delta)\langle c \rangle\|$ , where<sup>12</sup>:

$$\|A(\delta)\langle c \rangle\| = \{\|\delta'\langle c \rangle\| \mid \delta' \in A(\delta)\}$$

We refer to this set as the *weight* of the program  $A$  with respect to the condition  $c$ .

#### 3.1 Program constructs

We define the following program constructs:

- *skip*  $\hat{=} \delta' = \delta$

<sup>10</sup>Ordinary logic predicates, featuring all standard logic operators and quantifiers.

<sup>11</sup>When we have that  $\forall \delta_1, \delta_2 \bullet A(\delta_1) = A(\delta_2)$ , we may write simply  $A$  instead of  $A(\delta)$ .

<sup>12</sup>Note that the operators  $\| \_ \|$  and  $\langle \_ \rangle$  have been lifted to sets of probability distributions: in both cases the result is a set, whose elements are the results of applying the operator to each element of the set of probability distributions.

- $abort \triangleq true$
- $\underline{v} := e \triangleq \delta' = \delta\{\{e/v\}\}$
- $v_i := e \triangleq \delta' = \delta\{\{e/v_i\}\}$
- $A; B \triangleq \exists \delta_m \bullet A(\delta, \delta_m) \wedge B(\delta_m, \delta')$
- $A \triangleleft c \triangleright B \triangleq \exists \delta_A, \delta_B \bullet A(\delta\{c\}, \delta_A) \wedge B(\delta\{-c\}, \delta_B) \wedge \delta' = \delta_A + \delta_B$
- $A \text{ }_p\oplus B \triangleq \exists \delta_A, \delta_B \bullet A(p \cdot \delta, \delta_A) \wedge B((1-p) \cdot \delta, \delta_B) \wedge \delta' = \delta_A + \delta_B$
- $A \sqcap B \triangleq \exists \pi, \delta_A, \delta_B \bullet A(\delta\{\pi\}, \delta_A) \wedge B(\delta\{\bar{\pi}\}, \delta_B) \wedge \delta' = \delta_A + \delta_B$
- $c * A \triangleq \mu X \bullet (A; X) \triangleleft c \triangleright skip$

d:P:Abtr

d:P:A

d:P:A:Sng

d:P:Seq

d:P:Ch:Cnd

d:P:Ch:Prb

d:P:Ch:Dmn

d:P:Loop

The failing program *abort* is represented by the predicate *true*, which captures the fact that it is maximally unpredictable.

Program *skip* makes no changes and immediately terminates. Assignment remaps the distribution as has already been discussed in the previous §2.

Sequential composition is characterised by the existence of a “mid-point” distribution that is the outcome of the first program, and is then fed into the second.

We characterise conditional choice by using the condition (and its negation) to filter the left- and right-hand programs appropriately, and we simply sum the (now effectively disjoint) distributions.

Probabilistic choice simply uses the probability and its complement to scale the distributions for merge — this definition preserves all usual properties.

Non-deterministic choice in UTP is obtained by existentially quantifying over all possible weighting distributions, used to weight both sides<sup>13</sup>. In effect the predicate is only satisfied by any combination of left and right distributions that is pointwise larger than the minimum of both.

The definition of loop that we are proposing is somehow still incomplete, as we still have not provided an ordering relation among programs: we will fix this in the following §4, where we define the *refinement* relation between two programs — so in a loop we take the least fixpoint with respect to the ordering introduced by refinement.

We can see that all programs that mention  $\delta$  and  $\delta'$  can be written as a predicate of the following shape<sup>14</sup>:

$$\exists \text{QuantOf}(A) \bullet \delta' = \text{BodyOf}(A) \circ \delta \wedge \text{OtherCndOf}(A)$$

where:

- $\text{BodyOf}(A)$  is a sequence of modifications (*i.e.* interleaved restrictions and remapping operations) that are applied to  $\delta$  in order to obtain the corresponding  $\delta'$ ;
- $\text{QuantOf}(A)$  is a list of weighting distributions — all of the quantified probability distributions can be eliminated via the one-point rule, so that  $\delta'$  can be expressed as  $\text{BodyOf}(A) \circ \delta$ ;
- $\text{OtherCndOf}(A)$  is a list of any other conditions that are asserted by the program — no program constructs features other conditions so far, but we will have an extra condition in the generic choice operator, which we will define later on.

In the next subsection we relate *pGCL* expectations to this framework; thereafter we discuss some considerations on the topic of choice constructs.

<sup>13</sup>If the weight of one side depends on  $\pi$ , the weight of the other side will depend on  $(1 - \pi)$ , here and later on noted as  $\bar{\pi}$ .

<sup>14</sup>We can prove this by structural induction of the language syntax.

d:P:Structure

### 3.2 Deriving pre-expectations

As an expectation is a random variable (with non-negative real values), that can be represented by a distribution in our framework. Then if  $\chi'$  represents a post-expectation and  $A$  is a program, we can define the corresponding pre-expectation  $\chi$  by computing the expected final weight of each state before  $A$  is run:

$$\chi(\sigma) = \min(\{\|\chi' \cdot \delta'\| \mid \delta' \in A(\eta_\sigma)\})$$

where  $\_ \cdot \_$  is the pointwise multiplication of two distribution and we have used the notation  $\eta_\sigma$  to represent the distribution null everywhere with the exception of a single state  $\sigma$  mapping to 1:

$$\eta_\sigma \triangleq \epsilon \dagger \{\sigma \mapsto 1\}$$

### 3.3 More on choice constructs

Choice constructs deserve a bit more attention: we are now going to discuss some of the properties of probabilistic and non-deterministic choice; later on we will define a generic choice construct that covers conditional, probabilistic and non-deterministic choice (and more).

#### On probabilistic choice

First of all it is worth noticing that, from the above definition of probabilistic choice, we have the following equivalence:

$$A \text{ }_p\oplus B \equiv B \text{ }_{(1-p)}\oplus A$$

Moreover we have the following property:

$$A \text{ }_p\oplus (B \text{ }_q\oplus C) \equiv (A \text{ }_r\oplus B) \text{ }_s\oplus C \wedge p = rs \wedge (1-s) = (1-p)(1-q)$$

A few words on the probability  $p$ , that parametrises this operator: this may be a number in the range  $[0, 1]$  in the simplest setting, but in a more general case it is one of the possible values of a stochastic variable  $P$  that follows a probability distribution, whose probability density function  $f_P$  has the property of being compact in the range  $[0, 1]$ :

$$\int_{-\infty}^{+\infty} f_P(p) dp = \int_0^1 f_P(p) dp = 1$$

The distribution of this stochastic variable does not depend on the program variables, but in an even more general case may depend on other parameters  $q_1, q_2, \dots, q_n$ :

$$\int_{-\infty}^{+\infty} f_{PQ}(p, q_1, q_2, \dots, q_n) dp = \int_0^1 f_{PQ}(p, q_1, q_2, \dots, q_n) dp = f_Q(q_1, q_2, \dots, q_n)$$

#### On non-deterministic choice

Usually we talk about demonic non-determinism when we are expecting the worst-case behaviour, to model something that behaves as bad as it can for any desired outcome.

Our definition of non-deterministic choice *per se* has no such behaviour, but it will show up with the definition of refinement that we give in §4 or, more in general, whenever we explicitly choose to focus on the worst-case scenario: for this reason we prefer to use the more refer it as to the non-deterministic choice, rather than to the demonic choice.

The non-deterministic choice operator is idempotent according to the above definition: although some definitions in the literature have this property, there are some other where this property does not hold.

For example if on both sides we have the same program containing a probabilistic choice and this choice is resolved independently on each side *before* the non-deterministic choice is performed, then idempotency does not hold. Nonetheless idempotency does hold if the probabilistic choice is triggered *after* the non-deterministic choice is made — this is the behaviour that we can find in our framework.

*p:P:Ch:Prb:Comm*

*p:P:Ch:Prb:Assoc*  
See proof B.23

We can reproduce the other behaviour if we run the program twice with probabilistic choice on local variables, and then we merge the outputs by means of a non-deterministic choice: this is a behaviour that has nothing to do with idempotency — we keep the actions of one program separate from the other's, so we are actually dealing with two *different* programs that share the same specification.

### On a generic choice construct

Another remark that is worth making, is that we can see how all choice constructs look quite similar, or at least they follow a common pattern. The reason is that all choice constructs can be seen as a specific instance of a generic choice construct:

$$\mathit{choice}(A, B, \mathcal{X}) \triangleq \exists \pi, \delta_A, \delta_B \bullet \pi \in \mathcal{X} \wedge A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B$$

where  $\mathcal{X} \subseteq \mathcal{D}_w$  and  $\mathcal{D}_w$  is the set of all weighting distributions.

In fact we have that:

- for  $\mathcal{X} = \{\iota(c)\}$  we have conditional choice:

$$A \triangleleft c \triangleright B = \mathit{choice}(A, B, \{\iota(c)\})$$

- for  $\mathcal{X} = \{p \cdot \iota\}$  we have probabilistic choice:

$$A \text{ }_p\oplus B = \mathit{choice}(A, B, \{p \cdot \iota\})$$

- for  $\mathcal{X} = \mathcal{D}_w$  we have non-deterministic choice:

$$A \sqcap B = \mathit{choice}(A, B, \mathcal{D}_w)$$

Moreover we can see the disjunction of two programs as another kind of choice, where  $\mathcal{X} = \{\epsilon, \iota\}$ :

$$A \vee B = \mathit{choice}(A, B, \{\epsilon, \iota\})$$

Finally we can also use this generic construct to create new kinds of choices, other than the more traditional ones:

- for  $\mathcal{X} = \{p \cdot \iota(c)\}$  we have the *conditional probabilistic choice*, which behaves like A with probability p and like B with probability (1 - p) in the case when c holds, but it behaves like B if c does not hold:

$$A \triangleleft \text{ }_p c \triangleright B = \mathit{choice}(A, B, \{p \cdot \iota(c)\})$$

- for  $\mathcal{X} = \{p \cdot \iota(c) + q \cdot \iota(\neg c)\}$  we have the *switching probabilistic choice*, which is equivalent to a probabilistic choice with parameter p if c holds, with parameter q if c does not hold:

$$A \text{ }_{p \triangleleft c \triangleright q} \oplus B = \mathit{choice}(A, B, \{p \cdot \iota(c) + q \cdot \iota(\neg c)\})$$

- for  $\mathcal{X} = \mathcal{D}_w(c)$  we have the *conditional non-deterministic choice*, which behaves like  $A \sqcap B$  if c holds, but it behaves like B if c does not hold:

$$A_c \sqcap B = \mathit{choice}(A, B, \mathcal{D}_w(c))$$

- for  $\mathcal{X} = \{\pi \mid \forall \sigma \bullet p \leq \pi(\sigma) \leq 1 - q\}$ , where  $p + q \leq 1$ , we have the *non-deterministic probabilistic choice*, which guarantees a probability p to behave like A and a probability q to behave like B:

$$A \text{ }_p \oplus_q B = \mathit{choice}(A, B, \{\pi \mid \forall \sigma \bullet p \leq \pi(\sigma) \leq 1 - q\})$$

d:P:Ch

p:P:Ch:Cnd:Alt

p:P:Ch:Prb:Alt

p:P:Ch:Dmn:Alt

p:P:Ch:Or:Alt

d:P:Ch:CndPrb

p:P:Ch:SwPrb

d:P:Ch:CndDmn

p:P:Ch:DmnPrb

- for  $X = \{p \cdot \iota \mid p \in [0..1]\}$  we have the *fair non-deterministic choice*:

$$A \overset{\text{fair}}{\sqcap} B = \text{choice}(A, B, X = \{p \cdot \iota \mid p \in [0..1]\}) = \exists p \bullet A \oplus p B$$

It is worth noticing that this kind of choice is different from non-deterministic choice (we can view it as a less general form of it), in fact from this definition we have that:

$$\forall \delta \bullet (A \overset{\text{fair}}{\sqcap} B)(\delta) \subseteq (A \sqcap B)(\delta)$$

These possibilities have to be explored further, as there can be many more — and potentially more useful than these ones.

### A few laws on choice operators

Here is a non-comprehensive list of interesting laws on choice operators, that hold in our framework and that can also be found in *pGCL*:

*Idempotency of choice operators* :  $\forall X \bullet \text{choice}(A, A, X) \equiv A$

*Discarding right-hand option* :  $\text{choice}(A, B, \{\iota\}) \equiv A$

*Distributivity of choice operators* :

$$\text{choice}(A, (\text{choice}(B, C, X_2)), X_1) \equiv \text{choice}\left(\left(\text{choice}(A, B, X_1)\right), \left(\text{choice}(A, C, X_1)\right), X_2\right)$$

*Sequential composition* :  $\text{choice}(A, B, X); C \equiv \text{choice}(A; C, B; C, X)$

*Choice flipping* :  $\forall X \bullet \text{choice}(A, B, X) \equiv \text{choice}(B, A, \bar{X}) \wedge \bar{X} = \bigcup_{\pi \in X} \bar{\pi}$

*Monotonicity of generic choice* :  $\forall \delta \bullet X_1 \subseteq X_2 \Rightarrow \text{choice}(A, B, X_1)(\delta) \subseteq \text{choice}(A, B, X_2)(\delta)$

## 4 Refinement

We are going to define the refinement relation between two programs through a relation between the corresponding program images: we say that a program  $A$  is refined by a program  $B$  when for all conditions and (before-)distributions, the minimal probability that an (after-)distribution from  $A(\delta)$  satisfies a condition is less than that for  $B(\delta)$ :

$$A \sqsubseteq B \triangleq \forall z, \delta \bullet \min(\|A(\delta)\{z\}\|) \leq \min(\|B(\delta)\{z\}\|)$$

Informally, whatever condition  $z$  we are expecting, program  $B$  refines program  $A$  if it is at least “as good” when it comes to the probability of satisfying it.

The use of  $\min$  here matches the use in *pGCL* of it to define demonic choice, so we can see how this notion of refinement creates an order relation that is exactly the one created by the refinement relation used for *pGCL*. [MM04]

The whole point of defining refinement this way was to show the similarity with *pGCL*; moving further and taking advantage of the structure of our framework, we can give an alternative definition:

$$A \sqsubseteq B \triangleq \forall \delta \bullet B(\delta) \subseteq (A(\delta))^\Delta$$

where the *refinement set*  $(A(\delta))^\Delta$  is the (up-, convex- and Cauchy-closed) set defined as:

$$(A(\delta))^\Delta \triangleq \{\delta_\Delta \mid \delta' \leq \delta_\Delta \leq \iota \wedge \delta' = \sum_{\delta'_i \in A(\delta\{\pi_i\})} \delta'_i \wedge \sum \pi_i = \iota\}$$

p:P:Ch:FDmn

p:P:Ch:Idem  
See proof B.24

p:P:Ch:DscrD  
See proof B.25

p:P:Ch:Dst  
See proof B.26

p:P:Ch:Seq  
See proof B.27

p:P:Ch:Flip  
See proof B.28

p:P:Ch:Mntn  
See proof B.29

d:P:Rfn

d:P:Rfn:Alt

d:P:RfnSet

This set includes all after-distributions that are at least as great as those obtainable because of the non-determinism in the behaviour of  $A$ : a program whose image lies in this set for all  $\delta$  is a refinement of  $A$ , and hence the term “refinement set”.

From the above definition(s) we can easily demonstrate familiar refinement relations:

$$\begin{aligned} A \sqcap B &\sqsubseteq A \\ A \sqcap B &\sqsubseteq B \\ A \sqcap B &\sqsubseteq A \text{ }_{p\oplus} B \\ A \sqcap B &\sqsubseteq A \triangleleft c \triangleright B \end{aligned}$$

This comes as no surprise, in fact:

$$\begin{aligned} A \text{ }_{p\oplus} B &= \exists \pi, \delta_A, \delta_B \bullet A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B \wedge \pi = p \cdot \iota \\ A \triangleleft c \triangleright B &= \exists \pi, \delta_A, \delta_B \bullet A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B \wedge \pi = \iota(c) \end{aligned}$$

Concerning disjunction, we have that refinement fails to distinguish it from non-deterministic choice, as their refinement sets are the same:

$$A \sqcap B \sqsubseteq A \vee B \wedge A \vee B \sqsubseteq A \sqcap B$$

or, for short,  $A \sqcap B \Leftrightarrow A \vee B$ .

This result is due to the definition we have used for refinement, as we have used the traditional view of non-determinism as *demonic* non-determinism, *i.e.* that returning the worst possible result for any desired outcome: this is in line with the traditional use of disjunction as a definition for demonic choice.

Alternative definitions of refinement may take advantage of the possibility to distinguish between the operators  $\sqcap$  and  $\vee$  — this is left for future work.

In general, from the definition of refinement and the monotonicity of generic choice, we can show that:

$$\mathcal{X}_2 \sqsubseteq \mathcal{X}_1 \Rightarrow \text{choice}(A, B, \mathcal{X}_1) \sqsubseteq \text{choice}(A, B, \mathcal{X}_2)$$

It is worth stressing that the reverse implication is false — a counterexample is given by the case of the disjunction operator, where we have that:

$$\begin{aligned} A \vee B &\sqsubseteq A \text{ }_{p\oplus} B \\ A \vee B &\sqsubseteq A \triangleleft c \triangleright B \end{aligned}$$

$p:P:Rfn:Ch$   
See proof B.30

$p:P:Rfn:Dsj$   
See proof B.31

$p:P:Rfn:Dsj2$   
See proof B.32

## 4.1 Probabilistic refinement

We want to generalise things even further, and introduce a notion of *probabilistic refinement*:

$$A \stackrel{p}{\sqsubseteq} B \triangleq \forall z, \delta \bullet p \cdot \min(\|A(\delta)\{z\}\|) \leq \min(\|B(\delta)\{z\}\|)$$

We call this a  $p$ -accurate refinement, meaning that the refinement relation  $\sqsubseteq$  is true in a fraction  $p$  of the possible cases.

We can give this alternative definition as well, similarly as we did above:

$$A \stackrel{p}{\sqsubseteq} B \triangleq \forall \delta \bullet B(\delta) \sqsubseteq (p \cdot A(\delta))^\Delta$$

where  $p \cdot A(\delta)$  is the set made of all elements of  $A(\delta)$  multiplied by  $p$ .

Let  $p^*$  be the highest positive real number such that  $A \stackrel{p^*}{\sqsubseteq} B$ : this is the *accuracy* with which  $B$  refines  $A$  and is a measure of how “better”  $B$  is when compared to  $A$  in any possible case — and of course  $p < 1$  implies that  $B$  is not as “good” as  $A$ .

$d:P:PRfn$

$d:P:PRfn:Alt$



It is immediate to see that the refinement relation we have defined before is a special case of this more generic operator for  $p = 1$ , *i.e.* it is a 1-accurate refinement<sup>15</sup>:

$$A \sqsubseteq B = A \stackrel{1}{\sqsubseteq} B$$

This definition makes it much more meaningful to have a deterministic program on the left-hand side of the refinement relation<sup>16</sup>: the utility of such a thing is for example that a deterministic specification can be refined probabilistically by a (potentially) non-deterministic implementation, and the implementation accuracy is a piece of information of great value.

This notion of refinement may seem like generalisation for its own sake, but it has useful real-world applications — an example on medical devices can be found in [SSL10].

## 5 States and distributions, formally

Now that we have presented the whole idea, let us go down to the details which have been explained informally or in a less general way (when not skipped) in §2.

Examples relative to this part can be found in §A.1.

### 5.1 States

A state  $\sigma$  is a total function  $\sigma : \mathcal{V} \rightarrow \mathcal{W}$  that maps each variable  $v_i$  in the memory to a value  $w_i$ . We use  $S$  to note the *state space*, which is the set of all possible states.

d:S

According to this definition we have:

$$\text{dom}(\sigma) = \mathcal{V} = \{v_1, v_2, \dots, v_n, \dots\}$$

We refer to the domain of the state function also as the *alphabet* of that state:

d:S:Alph

$$\text{alph}(\sigma) \triangleq \text{dom}(\sigma)$$

$\mathcal{W}_i$  is the *type* of the variable  $v_i$ , *i.e.*  $v_i : \mathcal{W}_i$ . We note this also as:

d:T

$$\text{type}(v_i) \triangleq \mathcal{W}_i \subset \mathcal{W}$$

where  $\mathcal{W}$  is the set containing all types. For now we assume we have only booleans and integers as types.

An *expression* on variables is a combination of constants and variables, combined by operators. The set of all expressions is  $\mathcal{E}$ .

d:E

An expression  $e$  can be evaluated in a state  $\sigma$  by replacing each variable  $v_i$  it mentions with the value  $\sigma(v_i)$  that is contained by that variable in that state: doing the calculations with these values returns the *evaluation of the expression  $e$  on the state  $\sigma$* , which is the value  $\text{eval}_\sigma(e)$ .

d:E:Ev

Here is a recursive definition, where  $k$  is a constant,  $\mathcal{F}$  a  $n$ -ary function and  $x_i$  an expression:

$$\begin{aligned} \text{eval}_\sigma(k) &\triangleq k \\ \text{eval}_\sigma(v_i) &\triangleq \sigma(v_i) \\ \text{eval}_\sigma(\mathcal{F}(x_1, x_2, \dots, x_n)) &\triangleq \mathcal{F}(\text{eval}_\sigma(x_1), \text{eval}_\sigma(x_2), \dots, \text{eval}_\sigma(x_n)) \end{aligned}$$

<sup>15</sup>Or a 100%-accurate refinement, in case we prefer expressing  $p$  as a percentage.

<sup>16</sup>It is immediate to prove that a deterministic program  $A$  can be refined only by another program  $B$ , which has to be equivalent to  $A$ , *i.e.* such that  $A \sqsubseteq B \Leftrightarrow B \sqsubseteq A$ .

As a shorthand notation for the evaluation function, we overload the function state:

$$\sigma(e) \triangleq \text{eval}_\sigma(e)$$

When an expression  $e$  contains only values and operators, we have that its evaluation is the same on any state, thus when the notation is clear from the context we will simply write  $e$  instead of  $\text{eval}_\sigma(e)$  (or  $\sigma(e)$ , using the shorthand notation).

Using this, we can write that:

$$\sigma(e) = \text{eval}_\sigma(e\{\sigma(v_i)/v_i\}) = \sigma(e\{\sigma(v_i)/v_i\}) = e\{\sigma(v_i)/v_i\}$$

A *condition* is a boolean expression: we say that a state satisfies a condition  $c$  when it evaluates to *true* in that state.

An *abstract state*  $\alpha \subseteq \mathcal{S}$  is a set of states:

$$\alpha \triangleq \{\sigma_1, \sigma_2, \dots, \sigma_n, \dots\}$$

The alphabet of an abstract state is defined as the set of all the different alphabets that appear in the abstract state:

$$\text{alph}(\alpha) \triangleq \{\mathcal{A} \mid \mathcal{A} = \text{alph}(\sigma) \wedge \sigma \in \alpha\}$$

We use  $\mathcal{S}_\mathcal{A}$  to note the largest abstract state such that  $\text{alph}(\alpha) = \{\mathcal{A}\}$ :

$$\mathcal{S}_\mathcal{A} \triangleq \{\sigma \mid \text{alph}(\sigma) = \mathcal{A}\}$$

We write it this way as it is the largest subset of  $\mathcal{S}$ , whose elements are all those states with alphabet  $\mathcal{A}$ .

We say that an abstract state satisfies a condition  $c$  when all its elements do.

We define the *restriction* of an abstract state through a condition  $c$  as a total function  $\alpha\langle c \rangle : (\wp\mathcal{S} \times \mathcal{E}) \rightarrow \wp\mathcal{S}$ , defined as follows:

$$\alpha\langle c \rangle \triangleq \{\sigma \mid \sigma \in \alpha \wedge \sigma(c) = \text{true}\}$$

We have that:

$$\alpha\langle c \rangle = \mathcal{S}\langle c \rangle \cap \alpha$$

Clearly if the condition is *true* we have:

$$\alpha\langle \text{true} \rangle = \alpha$$

And obviously if the condition is *false* we have:

$$\alpha\langle \text{false} \rangle = \emptyset$$

## 5.2 Distributions

A *distribution*  $\chi$  is a partial function  $\chi : \mathcal{S} \rightarrow \mathbb{R}$ , that maps some states to real numbers.

$$\chi \triangleq \{\sigma_1 \mapsto x_1, \sigma_2 \mapsto x_2, \dots, \sigma_n \mapsto x_n, \dots\}$$

We refer to  $x_i$  as the *weight* of that state  $\sigma_i$  and we use  $\mathcal{D}$  to note the set of all possible distributions.

The weight of a distribution  $\chi$  is the sum over its domain of all the state weights:

$$\|\chi\| \triangleq \sum_{\sigma \in \text{dom}(\chi)} \chi(\sigma)$$

d:E:Ev:SH

p:E:Ev:VS

d:S:Sat

d:A

d:A:Alph

d:A:LAA

d:A:Sat

d:A:Rst

p:A:Rst:Alt  
See proof B.1

p:A:Rst:T

p:A:Rst:F

d:D

d:D:Wt

This can be lifted to a set  $\mathcal{X} \subseteq \mathcal{D}$  of distributions in an obvious way:

$$\|\mathcal{X}\| \triangleq \{\|\chi\| \mid \chi \in \mathcal{X}\}$$

d:D:Wt:Lift

The alphabet of a distribution is defined as the set of all the different alphabets that appear in the distribution domain:

$$\text{alph}(\chi) \triangleq \text{alph}(\text{dom}(\chi))$$

d:D:Alph

A particular distribution is the *empty distribution*  $\epsilon_\alpha : \mathcal{S} \rightarrow \mathbb{R}$ , which is a distribution such that  $\text{img}(\epsilon_\alpha) = \{0\}$ , i.e. it maps each state in the abstract state  $\alpha$  to 0:

$$\epsilon_\alpha \triangleq \{\sigma \mapsto 0 \mid \sigma \in \alpha\}$$

d:D:ED

Another particular distribution is the *unity distribution*  $\iota_\alpha : \mathcal{S} \rightarrow \mathbb{R}$ , which is a distribution such that  $\text{img}(\iota_\alpha) = \{1\}$ , i.e. it maps each state in the abstract state  $\alpha$  to 1:

$$\iota_\alpha \triangleq \{\sigma \mapsto 1 \mid \sigma \in \alpha\}$$

d:D:UD

We define the following shortcuts:

$$\begin{array}{ll} \epsilon_{\mathcal{A}} \triangleq \epsilon_{\mathcal{S}_{\mathcal{A}}} & \iota_{\mathcal{A}} \triangleq \iota_{\mathcal{S}_{\mathcal{A}}} \\ \epsilon_{\mathcal{X}} \triangleq \epsilon_{\text{dom}(\mathcal{X})} & \iota_{\mathcal{X}} \triangleq \iota_{\text{dom}(\mathcal{X})} \\ \epsilon \triangleq \emptyset & \iota \triangleq \iota_{\mathcal{S}} \end{array}$$

d:D:E:SH / d:D:U:SH

We define the *restriction of a distribution through a condition*  $c$  as follows:

$$\chi\langle c \rangle \triangleq \{\sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi)\langle c \rangle\}$$

d:D:Rst

From this definition we have:

$$\chi\langle c_1 \wedge c_2 \rangle = \chi\langle c_1 \rangle\langle c_2 \rangle = \chi\langle c_2 \rangle\langle c_1 \rangle$$

p:D:Rst:Cnj

Moreover:

$$\text{Restriction through equivalent condition} : (c_1 \Leftrightarrow c_2) \Rightarrow \chi\langle c_1 \rangle = \chi\langle c_2 \rangle$$

p:D:Rst:EqC  
See proof B.2

$$\text{Restriction through implied condition (I)} : (c_2 \Rightarrow c_1) \Leftrightarrow \chi\langle c_1 \rangle\langle c_2 \rangle = \chi\langle c_1 \rangle$$

p:D:Rst:ImC1  
See proof B.3

$$\text{Restriction through implied condition (II)} : (c_1 \Rightarrow \neg c_2) \Rightarrow \chi\langle c_1 \rangle\langle c_2 \rangle = \epsilon$$

p:D:Rst:ImC2  
See proof B.4

In case we have conditions  $c_\sigma$  and  $c_\alpha$  selecting respectively a single state  $\sigma$  and an abstract state  $\alpha$ , we simplify the notation as follows:

- $\delta\langle \sigma \rangle \triangleq \delta\langle c_\sigma \rangle$
- $\delta\langle \alpha \rangle \triangleq \delta\langle c_\alpha \rangle$

d:D:RstS

d:D:RstA

We define the *point distribution* (with domain  $\alpha$ ) as the restriction of a unity distribution to a single state:

$$\eta_{\sigma,\alpha} \triangleq \iota_\alpha\langle \sigma \rangle$$

d:D:Pnt

Concerning the weights of the restricted distributions above we have that:

- $\|\delta\langle c \rangle\| = \sum_{\sigma \in \text{dom}(\chi)\langle c \rangle} \delta(\sigma)$

p:D:Rst:Wt

- $\|\delta(\sigma)\| = \delta(\sigma)$
- $\|\delta(\alpha)\| = \sum_{\sigma \in \alpha} \delta(\sigma)$
- $\|\eta_{\sigma, \alpha}\| = 1$

p:D:RstS:Wt

p:D:RstA:Wt

p:D:Pnt:Wt

We also define the *restriction of a distribution through another distribution* as follows:

d:D:RstD

$$\chi_1 \langle \chi_2 \rangle \triangleq \left\{ \sigma \mapsto \chi_1(\sigma) \cdot \chi_2(\sigma) \mid \sigma \in \text{dom}(\chi_1) \cap \text{dom}(\chi_2) \right\}$$

From this definition we have:

$$\chi_1 \langle \chi_2 \rangle = \chi_2 \langle \chi_1 \rangle$$

p:D:RstD:Cmm

The reason why we call these operations in a similar way is that if we can see that the restriction of a distribution through a condition as a generalization to distributions of the restriction of abstract states through a condition, the restriction of a distribution through a distribution can be seen as a further generalization:

p:D:Rst:Alt

See proof B.5

$$\chi \langle c \rangle = \chi \langle \iota_\chi \langle c \rangle \rangle$$

All of this can be lifted to a set  $\mathcal{X} \subseteq \mathcal{D}$  of distributions in an obvious way:

- $\mathcal{X} \langle c \rangle = \{ \chi \langle c \rangle \mid \chi \in \mathcal{X} \}$
- $\mathcal{X} \langle \chi \rangle = \{ \xi \langle \chi \rangle \mid \xi \in \mathcal{X} \}$

d:D:Rst:Lift

d:D:RstD:Lift

### Operations on distributions

The *sum* of distributions is defined as:

d:D:Sum

$$\chi_1 + \chi_2 \triangleq \left\{ \sigma \mapsto (\chi_1(\sigma) + \chi_2(\sigma)) \right\}$$

From this definition we have that:

- $\|\chi_1 + \chi_2\| = \|\chi_1\| + \|\chi_2\|$
- $(\chi_1 + \chi_2) \langle \pi \rangle = \chi_1 \langle \pi \rangle + \chi_2 \langle \pi \rangle$

p:D:Sum:Wt

p:D:Sum:Rst

Thanks to the latter property we can split a distribution into two other distributions, where all the elements of one satisfy a given condition  $c$ , while the elements of the other do not:

p:D:Sum:CS

See proof B.6

$$\chi = \chi \langle c \rangle + \chi \langle -c \rangle$$

The *multiplication by a scalar number* is defined as:

d:D:Mul

$$n \cdot \chi \triangleq \left\{ \sigma \mapsto (n \cdot \chi(\sigma)) \right\}$$

The restriction of a distribution through another distribution may be regarded as a kind of product, and sometimes it is easier to think of it in these terms, so we define the *product* of two distribution as:

d:D:Prod

$$\chi_1 \cdot \chi_2 \triangleq \chi_1 \langle \chi_2 \rangle$$

From this definition we have that:

$$\chi_1 \cdot \chi_2 = \chi_2 \cdot \chi_1$$

p:D:Prod:Cmm

## Types of distributions

A *weighting distribution*  $\pi$  is a distribution such that  $\text{img}(\pi) \subseteq [0..1]$ .  
We use  $\mathcal{D}_w$  to note the subset of  $\mathcal{D}$  of all weighting distributions.

d:D:WD

Given a weighting distribution  $\pi$ , we define its complementary weighting distribution  $\bar{\pi}$  as:

d:D:WD:Cmp

$$\bar{\pi} \triangleq \iota_{\pi} - \pi$$

*Restriction* :  $\pi_1 \langle \pi_2 \rangle \in \mathcal{D}_w$

p:D:WD:Rst

See proof B.7

A *probability distribution*  $\delta$  is a weighting distribution such that  $\|\delta\| \leq 1$ :

d:D:PD

$$\delta \triangleq \{ \sigma_1 \mapsto p_1, \sigma_2 \mapsto p_2, \dots, \sigma_n \mapsto p_n, \dots \}$$

We use  $\mathcal{D}_p$  to note the subset of  $\mathcal{D}_w$  of all probability distributions.

In this case we will refer to the weight  $p_i$  as to the *probability*<sup>17</sup> of the state  $\sigma_i$ ; likewise we will talk of the probability of an abstract state<sup>18</sup>, rather than of its weight.

*Restriction* :  $\delta \langle \pi \rangle \in \mathcal{D}_p$

p:D:PD:Rst

See proof B.8

In the remainder of this document we will be talking mostly of probability distributions, so we will usually be referring to them simply as “distributions”.

Whenever we want to use this term in the more general meaning we have used so far, we will rather use “general distributions”.

## 6 Assignments

An assignment performed in a state  $\sigma$  is an operation  $v_i := e_i$ , that updates the value contained in  $v_i$  with  $\sigma(e_i)$ .

In *UTP* we usually use a dash to mark a variable, in order to refer to the new value  $v'_i$  it contains: the same convention is adopted here. Moreover we will use dashes in a similar way to denote *after-states* ( $\sigma'$ ) and *after-distributions* ( $\delta'$ ).

We use the following notation for  $n$  simultaneous assignments of the expressions  $e_1, e_2, \dots, e_n$  to the variables  $v_1, v_2, \dots, v_n \in \mathcal{V}$ :

d:S:SA

$$\underline{v} := \underline{e} \triangleq v_1, v_2, \dots, v_n := e_1, e_2, \dots, e_n$$

where

$$\underline{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \in \mathcal{V} \quad \text{and} \quad \underline{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix} \in \mathcal{E}$$

Moreover, we also use the following shorthand notation:

d:E:Ev:VSH

$$\sigma(\underline{v}) \triangleq \begin{pmatrix} \sigma(v_1) \\ \sigma(v_2) \\ \vdots \\ \sigma(v_n) \end{pmatrix}$$

By using this notation we can define a vectorial extension for the map operator:

d:S:VMap

$$\underline{v} \mapsto \underline{w} \triangleq \{ v_i \mapsto w_i \mid 1 \leq i \leq n \}$$

<sup>17</sup> $\delta(\sigma)$  is a function of  $\sigma$  and is what is usually referred to as the *probability mass function*: it represents the way the probability is distributed depending on  $\sigma$ .

<sup>18</sup>If we see a state as an *outcome*, we can see an abstract state as an *event* (i.e. a set of outcomes).

We can use this to give a compact definition of a state:

$$\sigma = \underline{v} \mapsto \underline{w}$$

where

$$\underline{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \in \mathcal{W}$$

We use the following notation for simultaneous substitutions<sup>19</sup>  $\{f_1/g_1\}\{f_2/g_2\}\dots\{f_n/g_n\}$ :

$$\{f/g\} \triangleq \{f_1/g_1\}\{f_2/g_2\}\dots\{f_n/g_n\}$$

where

$$\underline{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \quad \text{and} \quad \underline{g} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}$$

When the substitution  $\{f/g\}$  is applied to a vector of expressions  $\underline{e}$ , the meaning is the following:

$$\underline{e}\{f/g\} \triangleq \begin{pmatrix} e_1\{f/g\} \\ e_2\{f/g\} \\ \vdots \\ e_n\{f/g\} \end{pmatrix}$$

The composition of two expression vectors  $\underline{f}$  and  $\underline{e}$  is defined as a particular substitution that involves the variable vector  $\underline{v}$ :

$$\underline{f} \circ \underline{e} \triangleq \underline{f}\{e/v\} = \begin{pmatrix} f_1\{e/v\} \\ f_2\{e/v\} \\ \vdots \\ f_n\{e/v\} \end{pmatrix}$$

We can read the notation  $\underline{f} \circ \underline{e}$  as  $\underline{f}$  after  $\underline{e}$ .

Concerning the evaluation of this vector we have

$$\sigma(\underline{f} \circ \underline{e}) = \sigma(\underline{f}\{e/v\}) = \sigma(\underline{f}\{\sigma(e)/v\}) = \underline{f}\{\sigma(e)/v\}$$

This is equivalent to evaluating  $\underline{f}$  in a state  $\zeta$  such that  $\zeta(\underline{v}) = \sigma(\underline{e})$ .

Now it should be clear why we intentionally use a symbol like  $\circ$  and the word “after”, which both remind of functional composition: if for every expression and variable vectors  $\underline{e}$  and  $\underline{v}$  we define an associated function  $\underline{e}_v : \mathcal{W}' \rightarrow \mathcal{W}'$  as:

$$\underline{e}_v(\underline{w}) = \text{eval}_{\underline{v} \mapsto \underline{w}}(\underline{e})$$

then for any state  $\sigma = \underline{v} \mapsto \underline{w}$ , we have that  $\sigma(\underline{f} \circ \underline{e}) = \underline{f}_v(\underline{e}_v(\underline{w}))$ :

$$\begin{cases} \sigma(\underline{f} \circ \underline{e}) = \underline{f}_v(\underline{w}^*) \\ \underline{w}^* = \underline{e}_v(\underline{w}) \end{cases}$$

When composing the same expression for  $k \geq 1$  times, we use the following notation:

$$\underline{e}^k \triangleq \underbrace{\underline{e} \circ \underline{e} \circ \dots \circ \underline{e}}_{k \text{ times}}$$

We define that for  $k = 0$  this notation has the following meaning:

$$\underline{e}^0 \triangleq \underline{v}$$

<sup>19</sup>For this to make sense, it must be the case that  $\forall i \neq j \bullet g_i \neq g_j$ .

p:S:Alt

d:E:Sub

d:E:Sub2

d:E:Comp

p:E:Ev:Comp

d:E:Comp:Iter

## 6.1 The inverse-image set

Let us now define the *inverse-image set* for a generic assignment  $\underline{v} := \underline{e}$ , after which the new mapping of the variable vector is a state  $\sigma'$ :

$$Inv(\underline{v} := \underline{e}, \sigma') \triangleq \{ \sigma \mid \sigma'(\underline{v}) = \sigma(\underline{e}) \wedge \sigma \in \mathcal{S}_{\text{alph}(\sigma')} \}$$

We can generalize this to an abstract state  $\alpha'$ :

$$Inv(\underline{v} := \underline{e}, \alpha') \triangleq \bigcup_{\sigma' \in \alpha'} Inv(\underline{v} := \underline{e}, \sigma')$$

The abstract state  $\alpha$  is the set of all the possible states before the assignment that are compatible with the result of the new mapping being in the abstract state  $\alpha'$ .

Due to the fact that the evaluation of an expression is an injective function we have that:

$$Inv(\underline{v} := \underline{e}, \sigma_1) \cap Inv(\underline{v} := \underline{e}, \sigma_2) = \emptyset \Leftrightarrow \sigma_1 \neq \sigma_2$$

Thanks to this property, if the evaluation of an expression  $\underline{e}$  is defined on all of the states belonging to an abstract state  $\alpha$ , we have that it is possible to partition  $\alpha$  through  $\underline{e}$ .

In fact if we have a relation  $\mathcal{R}_e$  defined as:

$$\sigma_1 \mathcal{R}_e \sigma_2 \Leftrightarrow \sigma_1(\underline{e}) = \sigma_2(\underline{e})$$

this is an equivalence relation among states belonging to an abstract state  $\alpha$ , that is partitioned into equivalence classes corresponding to inverse-image sets  $\alpha'$ :

$$\alpha = \bigcup_{\sigma' \in \alpha'} Inv(\underline{v} := \underline{e}, \sigma')$$

where each class is represented by a state  $\sigma$  such that  $\sigma(\underline{e}) = \sigma'(\underline{v})$ .

*Nested inverse-image set* :  $Inv(\underline{v} := \underline{e}, Inv(\underline{v} := \underline{f}, \{\sigma\})) = Inv(\underline{v} := \underline{f}\{\underline{e}/\underline{v}\}, \{\sigma\})$

Examples relative to this subsection can be found in §A.1.

## 6.2 The remap operator

The *remap* operator is defined as follows:

$$\delta\{\underline{e}/\underline{v}\} \triangleq \left\{ \sigma' \mapsto \left\| \delta\left(Inv(\underline{v} := \underline{e}, \{\sigma'\})\right) \right\| \mid \text{alph}(\sigma') \in \text{alph}(\delta) \right\}$$

Which is:

$$(\delta\{\underline{e}/\underline{v}\})(\sigma') \triangleq \left\{ \sum \delta(\sigma) \mid \sigma' = \sigma \dagger \{\underline{v} \mapsto \text{eval}_\sigma(\underline{e})\} \right\}$$

From the definition we can see that after applying the remap operator the alphabet of the resulting distribution is the same as the alphabet of the original distribution:

$$\text{alph}(\delta\{\underline{e}/\underline{v}\}) = \text{alph}(\delta)$$

We overload this notation to account for assignment to a single variable:

$$\delta\{e_i/v_i\} \triangleq \delta\{(e_i)/(v_i)\}$$

We define a compact notation for multiple application of the same operation:

$$\delta\{\underline{e}/\underline{v}\}^k \triangleq \underbrace{\delta\{\underline{e}/\underline{v}\}\{\underline{e}/\underline{v}\} \dots \{\underline{e}/\underline{v}\}}_{k \text{ times}}$$

d:S:Inv

d:A:Inv

p:A:Inv:Dsj

p:A:Inv:EqR

p:S:Inv:Nest  
See proof B.9

d:D:Rmp

p:D:Rmp:Alt

p:D:Rmp:Alph

d:D:Rmp:Sng

d:D:Rmp:Iter

A sequence of assignments causes the distribution to evolve and we want to keep track of changes. In fact we have that after an assignment  $\underline{v} := \underline{e}$  when the distribution is  $\delta$ , we have an evolution towards a new  $\delta'$ .

We can write this as:

$$\delta' = \delta \{ \underline{e} / \underline{v} \}$$

Examples relative to this subsection can be found in §A.1.

## Properties

From the definitions of sum and multiplication, we have that the remap operator is a linear one:

$$(x \cdot \delta \{ \underline{e} / \underline{v} \} + y \cdot \delta \{ \underline{f} / \underline{v} \} ) \{ \underline{g} / \underline{v} \} = x \cdot \delta \{ \underline{e} / \underline{v} \} \{ \underline{g} / \underline{v} \} + y \cdot \delta \{ \underline{f} / \underline{v} \} \{ \underline{g} / \underline{v} \}$$

Here are some other properties:

*Composition (I)* :  $\delta \{ \underline{e} / \underline{v} \} \{ \underline{f} / \underline{v} \} = \delta \{ \underline{f} \{ \underline{e} / \underline{v} \} / \underline{v} \}$

*Composition (II)* :  $\delta \{ \underline{e} / \underline{v} \} \{ \underline{f} / \underline{v} \} = \delta \{ \underline{f} \circ \underline{e} / \underline{v} \}$

*Composition (III)* :  $\delta \{ \underline{e} / v_i \} \{ \underline{f} / v_j \} = \delta \{ (e, f \{ \underline{e} / v_i \}) / (v_i, v_j) \}$

*Composition (IV)* :  $\delta \{ \underline{e} / v_i \} \{ \underline{f} / v_i \} = \delta \{ \underline{f} \{ \underline{e} / v_i \} / v_i \}$

*Iteration* :  $\delta \{ \underline{e} / \underline{v} \}^k = \delta \{ \underline{e}^k / \underline{v} \}$

*Commutativity (I)* :  $\delta \{ \underline{e} / v_i \} \{ \underline{f} / v_j \} = \delta \{ \underline{f} \{ \underline{e} / v_i \} / v_j \} \{ \underline{e} / v_i \}$  iff  $v_j \notin fv(\underline{e})$

*Commutativity (II)* :  $\delta \{ \underline{e} / v_i \} \{ \underline{f} / v_j \} = \delta \{ \underline{f} / v_j \} \{ \underline{e} / v_i \}$  iff  $v_i \notin fv(\underline{f}) \wedge v_j \notin fv(\underline{e})$

*Expression substitution* :  $\delta \{ \underline{f} = \underline{g} \} \{ \underline{e} / \underline{v} \} = \delta \{ \underline{f} = \underline{g} \} \{ \underline{e} \{ \underline{f} / \underline{g} \} / \underline{v} \}$

*Contradiction* :  $\forall \sigma \in \text{dom}(\delta) \bullet \sigma(c \{ \underline{e} / \underline{v} \}) = \text{false} \wedge \delta \neq \epsilon \Leftrightarrow \delta \{ \underline{e} / \underline{v} \} \langle c \rangle = \epsilon$

*Assertion* :  $\forall \sigma \in \text{dom}(\delta) \bullet \sigma(c \{ \underline{e} / \underline{v} \}) = \text{true} \Leftrightarrow \delta \{ \underline{e} / \underline{v} \} \langle c \rangle = \delta \{ \underline{e} / \underline{v} \}$

*Remapping a condition* :  $\delta \{ \underline{e} / \underline{v} \} \langle c \rangle = \delta \langle c \{ \underline{e} / \underline{v} \} \rangle \{ \underline{e} / \underline{v} \}$

*Weight of a distribution after remapping* :  $\| \delta \{ \underline{e} / \underline{v} \} \| = \| \delta \|$  iff  $\sigma(\underline{e})$  is defined in  $\text{dom}(\delta)$

## 7 Conclusion and future work

We have provided an encoding of the semantics of pGCL in UTP, as a homogeneous relation on the alphabet  $\{\delta, \delta'\}$ , where the before and after variables are distributions over program states. The key is that our semantics models probabilistic programs as predicate transformers, so allowing us to claim that “probabilistic programs are predicates too”!

Such programs may feature both demonic and probabilistic choice: this is non-trivial and at the time of this writing there is still no satisfactory UTP theory that embeds such a feature — *unifying probabilism with other programming constructs in the style of Unifying Theories of Programming* is a *so-far-unachieved goal*, according to the aforementioned talk by Chen and Sanders [CS09] at FM09.

We hope that this is a step towards having a formalism that can deal with this issue properly.

We have shown that we can deal with variables by name, despite their being entangled in the semantic domain, and that the laws of pGCL are provable from our semantics.

d:D:Dash

p:D:Rmp:Lin  
See proof B.10

p:D:Rmp:Comp1  
See proof B.11

p:D:Rmp:Comp2  
See proof B.12

p:D:Rmp:Comp3  
See proof B.13

p:D:Rmp:Comp4  
See proof B.14

p:D:Rmp:Iter  
See proof B.15

p:D:Rmp:Cmm1  
See proof B.16

p:D:Rmp:Cmm2  
See proof B.17

p:D:Rst:ES  
See proof B.18

p:D:Rmp:Rst1  
See proof B.19

p:D:Rmp:Rst2  
See proof B.20

p:D:Rst:Rmp  
See proof B.21

p:D:Rmp:Wt  
See proof B.22



In addition we have formulated our semantics in such a way as to be able to view all choices as instances of a generic choice construct, and even to be able to allow disjunction back in as a form of choice. We have also shown that refinement meshes in effectively, not just with demonic choice but in fact with all generic variants.

There are some steps forward that need to be taken. The first one is exploring the precise linkages between our semantic model and the two models that feature in [HSM97; MM04]. This will also lead to a formalization of the healthiness conditions, which characterise the predicates in our framework: we need to show that we inherit all of the healthiness conditions, modulo an appropriate generalization, which are valid for pGCL.

The second is to explore the role of auxiliary variables such as  $o\kappa$  and  $o\kappa'$  that capture a behaviour such as termination, and in particular whether they are necessary (non-termination leads to probability sub-distributions, similar to what happens in pGCL, so could we manage without?), and how to introduce auxiliary variables such as *wait* and *wait'*, to move towards the encoding of reactive systems in this framework.

This is important, as the long term focus of this work is on a probabilistic variant of *Circus*, which requires semantic models for probabilistic process algebras like *pCSP* [Mor<sup>+</sup>96; Den<sup>+</sup>08] or *PTSC* [NS09]. These will then have to be integrated with our pGCL semantics in much the same way that the theory of Reactive Designs in UTP is the basis for the semantics of *Circus*-like languages.

## Acknowledgements

We wish to thank (some of) the anonymous referees, who have reviewed a conference paper derived from this work, for their insightful comments and suggestions, as well as people from the Foundations and Methods Group, TCD for many fruitful discussions.

## A Examples

### A.1 Definitions

This section is meant to give some examples aimed at a better explanation of the basic definitions given in §5 and §6.

#### States and distributions

- States:

$$\begin{array}{lll}
 \sigma_1 = \{(v_1 \mapsto 4)\} & \sigma_2 = \{(v_1 \mapsto 5, v_2 \mapsto \text{false})\} & \sigma_3 = \{(v_1 \mapsto 6, v_3 \mapsto \text{true})\} \\
 \sigma_4 = \{(v_1 \mapsto 5, v_3 \mapsto \text{true})\} & \sigma_5 = \{(v_1 \mapsto -1, v_3 \mapsto \text{true})\} & \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5 \in \mathcal{S} \\
 \text{alph } \sigma_1 = \{v_1\} & \text{alph } \sigma_2 = \{v_1, v_2\} & \text{alph } \sigma_3 = \{v_1, v_3\} \\
 \text{alph } \sigma_4 = \{v_1, v_3\} & \text{alph } \sigma_5 = \{v_1, v_3\} &
 \end{array}$$

- States satisfying a condition:

- All of these states satisfy the condition  $v_1 > -10$ ;
- all states with the exception of  $\sigma_5$  satisfy the condition  $v_1 > 0$ ;
- $\sigma_3, \sigma_4, \sigma_5$  satisfy the condition  $v_3 = \text{true}$ , but  $\sigma_1, \sigma_2$  do not.

- Abstract states:

$$\begin{array}{lll}
 \alpha_1 = \{\sigma_1, \sigma_2\} & \alpha_2 = \{\sigma_3, \sigma_4, \sigma_5\} & \alpha_1, \alpha_2 \in \mathcal{P}\mathcal{S} \\
 \text{alph}(\alpha_1) = \{\{v_1\}, \{v_1, v_2\}\} & \text{alph}(\alpha_2) = \{\{v_1, v_3\}\} &
 \end{array}$$

- Abstract states satisfying a condition:

- Both  $\alpha_1$  and  $\alpha_2$  satisfy the condition  $v_1 > -10$ ;
  - $\alpha_1$  satisfies the condition  $v_1 > 0$ , but  $\alpha_2$  does not;
  - $\alpha_2$  satisfies the condition  $v_3 = \text{true}$ , but  $\alpha_1$  does not.
- Largest abstract state with a fixed alphabet:

$$S_{\{v_1, v_2, v_3\}} = \{(v_1 \mapsto a, v_2 \mapsto b, v_3 \mapsto c) \mid a \in \mathbb{Z}, b, c \in \{\text{true}, \text{false}\}\}$$

- Restriction of abstract states:

$$\alpha_1 \langle v_1 > 0 \rangle = \{(v_1 \mapsto 4), (v_1 \mapsto 5, v_2 \mapsto \text{false})\} = \alpha_1$$

$$\alpha_2 \langle v_1 > 0 \rangle = \{(v_1 \mapsto 6, v_3 \mapsto \text{true}), (v_1 \mapsto 5, v_3 \mapsto \text{true})\} = \alpha_2 \setminus \{(v_1 \mapsto -1, v_3 \mapsto \text{true})\}$$

$$\alpha_1 \langle v_3 = \text{true} \rangle = \emptyset$$

- Distributions:

$$\chi_1 = \{\sigma_1 \mapsto 7, \sigma_2 \mapsto 4, \sigma_3 \mapsto -2, \sigma_4 \mapsto \sqrt{2}, \sigma_5 \mapsto 7/18\}$$

$$\chi_2 = \{\sigma_2 \mapsto 1, \sigma_3 \mapsto e^2, \sigma_4 \mapsto 8\}$$

$$\text{alph}(\chi_1) = \{\{v_1\}, \{v_1, v_2\}, \{v_1, v_3\}\} \quad \text{alph}(\chi_2) = \{\{v_1, v_2\}, \{v_1, v_3\}\}$$

- Restriction of distributions:

$$\chi_1 \langle v_3 = \text{true} \rangle = \{\sigma_3 \mapsto -2, \sigma_4 \mapsto \sqrt{2}, \sigma_5 \mapsto 7/18\}$$

$$\chi_2 \langle v_3 = \text{true} \rangle = \{\sigma_3 \mapsto e^2, \sigma_4 \mapsto 8\}$$

$$\chi_1 \langle \chi_2 \rangle = \chi_2 \langle \chi_1 \rangle = \{\sigma_2 \mapsto 4 \cdot 1, \sigma_3 \mapsto -2 \cdot e^2, \sigma_4 \mapsto \sqrt{2} \cdot 8\}$$

- Weight of conditions, states and abstract states:

$$\|\chi_1 \langle v_1 > 0 \rangle\| = \chi_1(\sigma_1) + \chi_1(\sigma_2) + \chi_1(\sigma_3) + \chi_1(\sigma_4) = 9 + \sqrt{2}$$

$$\|\chi_1 \langle v_3 = \text{true} \rangle\| = \chi_1(\sigma_3) + \chi_1(\sigma_4) + \chi_1(\sigma_5) = -29/18 + \sqrt{2}$$

$$\|\chi_1 \langle v_1 > -10 \rangle\| = \chi_1(\sigma_1) + \chi_1(\sigma_2) + \chi_1(\sigma_3) + \chi_1(\sigma_4) + \chi_1(\sigma_5) = 169/18 + \sqrt{2} \\ = \|\chi_1\|$$

$$\|\chi_1 \langle \sigma_1 \rangle\| = \chi_1(\sigma_1) = 7$$

$$\|\chi_1 \langle \alpha_1 \rangle\| = \chi_1(\sigma_1) + \chi_1(\sigma_2) = 7 + 4 = 11$$

$$\|\chi_1 \langle \alpha_2 \rangle\| = \chi_1(\sigma_3) + \chi_1(\sigma_4) + \chi_1(\sigma_5) = -2 + \sqrt{2} + 7/18 = -29/18 + \sqrt{2}$$

$$\|\chi_2 \langle \alpha_1 \rangle\| = \chi_2(\sigma_2) = 1$$

$$\|\chi_2 \langle \alpha_2 \rangle\| = \chi_2(\sigma_3) + \chi_2(\sigma_4) = e^2 + 8$$

- Operations on distributions:

$$\chi_1 + \chi_2 = \{\sigma_1 \mapsto 7, \sigma_2 \mapsto 5, \sigma_3 \mapsto -2 + e^2, \sigma_4 \mapsto 8 + \sqrt{2}, \sigma_5 \mapsto 7/18\}$$

$$\sqrt{7} \cdot \chi_2 = \{\sigma_2 \mapsto \sqrt{7}, \sigma_3 \mapsto e^2 \sqrt{7}, \sigma_4 \mapsto 8\sqrt{7}\}$$

$$\chi_1 \cdot \chi_2 = \chi_1 \langle \chi_2 \rangle = \{\sigma_2 \mapsto 4 \cdot 1, \sigma_3 \mapsto -2 \cdot e^2, \sigma_4 \mapsto \sqrt{2} \cdot 8\}$$

### The inverse-image Set

$$\text{Inv}(v_1 := (v_1^2), \sigma_1) = \{(v_1 \mapsto 2), (v_1 \mapsto -2)\}$$

$$\text{Inv}(v_1 := 4, \sigma_1) = \{(v_1 \mapsto i) \mid i \in \mathbb{Z}\}$$

$$\text{Inv}(v_2 := (v_1 < 10), \sigma_2) = \emptyset$$

$$\text{Inv}(v_1 := (2 \cdot v_1), \alpha_2) = \{(v_1 \mapsto 2), (v_1 \mapsto 2.5, v_2 \mapsto \text{false})\}$$

## The remap operator

$$\begin{aligned}
\delta_1 &= \{\sigma_1 \mapsto 0.55, \sigma_2 \mapsto 0.45\} \\
\delta_1 \{\{2 \cdot v_1 / v_1\}\} &= \{ \{(v_1 \mapsto 8)\} \mapsto 0.55, \{(v_1 \mapsto 10, v_2 \mapsto \text{false})\} \mapsto 0.45 \} \\
\delta_1 \{\{2/v_1\}\} &= \{ \{(v_1 \mapsto 2)\} \mapsto 0.55, \{(v_1 \mapsto 2, v_2 \mapsto \text{false})\} \mapsto 0.45 \} \\
\delta_1 \{\{\neg v_2 / v_2\}\} &= \{ \{(v_1 \mapsto 5, v_2 \mapsto \text{true})\} \mapsto 0.45 \} \\
\delta_1 \{\{(v_1+1, \text{true}) / (v_1, v_2)\}\} &= \{ \{(v_1 \mapsto 6, v_2 \mapsto \text{true})\} \mapsto 0.45 \}
\end{aligned}$$

## A.2 Interaction of probabilistic and non-deterministic choice

Let us take these two simple programs:

$$\begin{aligned}
A &\hat{=} x := 0 \sqcap x := 1 ; y := 0 \frac{1}{2} \oplus y := 1 \\
B &\hat{=} x := 0 \frac{1}{2} \oplus x := 1 ; y := 0 \sqcap y := 1
\end{aligned}$$

We evaluate what is the probability that after each program has run we have  $x = 1$ , as well as the probability of having  $x = y$ .

We start by examining A:

$$\begin{aligned}
&x := 0 \sqcap x := 1 ; y := 0 \frac{1}{2} \oplus y := 1 \\
\equiv &\quad \text{Translation: A} \\
&\exists \pi \bullet \delta' = \delta \langle \pi \rangle \{\{0/x\}\} + \delta \langle \bar{\pi} \rangle \{\{1/x\}\} ; \delta' = 1/2 \cdot \delta \{\{0/y\}\} + 1/2 \cdot \delta \{\{1/y\}\} \\
\equiv &\quad [\text{d:P:Seq}] \\
&\exists \pi, \delta_m \bullet \delta_m = \delta \langle \pi \rangle \{\{0/x\}\} + \delta \langle \bar{\pi} \rangle \{\{1/x\}\} \wedge \delta' = 1/2 \cdot \delta_m \{\{0/y\}\} + 1/2 \cdot \delta_m \{\{1/y\}\} \\
\equiv &\quad \text{One-point rule} \\
&\exists \pi \bullet \delta' = 1/2 \cdot (\delta \langle \pi \rangle \{\{0/x\}\} + \delta \langle \bar{\pi} \rangle \{\{1/x\}\}) \{\{0/y\}\} + 1/2 \cdot (\delta \langle \pi \rangle \{\{0/x\}\} + \delta \langle \bar{\pi} \rangle \{\{1/x\}\}) \{\{1/y\}\} \\
\equiv &\quad [\text{p:D:Rmp:Lin}] \\
&\exists \pi \bullet \delta' = 1/2 \cdot \delta \langle \pi \rangle \{\{0/x\}\} \{\{0/y\}\} + 1/2 \cdot \delta \langle \bar{\pi} \rangle \{\{1/x\}\} \{\{0/y\}\} + 1/2 \cdot \delta \langle \pi \rangle \{\{0/x\}\} \{\{1/y\}\} + 1/2 \cdot \delta \langle \bar{\pi} \rangle \{\{1/x\}\} \{\{1/y\}\}
\end{aligned}$$

The final distribution  $\delta'(\pi)$  is parametric in the weighting distribution  $\pi$ : let us try to use this to perform a worst-case analysis.

We can show that  $\forall \pi \bullet \|\delta'(\pi)\langle x = y \rangle\| = 1/2$ , which implies that We can show that  $\forall \pi \bullet \|\delta'(\pi)\langle x = y \rangle\| = 1/2$ , which implies that  $\forall \delta \bullet \min(\|A(\delta)\langle x = y \rangle\|) = 1/2$ :

$$\begin{aligned}
\|\delta'(\pi)\langle x = y \rangle\| &= \left\| \left( 1/2 \cdot \delta \langle \pi \rangle \{\{0/x\}\} \{\{0/y\}\} + 1/2 \cdot \delta \langle \bar{\pi} \rangle \{\{1/x\}\} \{\{1/y\}\} \right) \right\| \\
&= \left\| \left( 1/2 \cdot \delta \langle \pi \rangle + 1/2 \cdot \delta \langle \bar{\pi} \rangle \right) \right\| = \|1/2 \cdot \delta\| = 1/2
\end{aligned}$$

But if we choose  $\pi = \iota_{\delta'}$ , we have  $\delta'(\iota_{\delta'}) = 1/2 \cdot \delta \{\{0/x\}\} \{\{0/y\}\} + 1/2 \cdot \delta \{\{0/x\}\} \{\{1/y\}\}$  and therefore  $\|\delta'(\iota_{\delta'})\langle x = 1 \rangle\| = 0$  — so the minimum of the weight of program A is 0.

Likewise, we examine B:

$$\begin{aligned}
&x := 0 \frac{1}{2} \oplus x := 1 ; y := 0 \sqcap y := 1 \\
\equiv &\quad \text{Translation: B} \\
&\delta' = 1/2 \cdot \delta \{\{0/x\}\} + 1/2 \cdot \delta \{\{1/x\}\} ; \exists \pi \bullet \delta' = \delta \langle \pi \rangle \{\{0/y\}\} + \delta \langle \bar{\pi} \rangle \{\{1/y\}\} \\
\equiv &\quad [\text{d:P:Seq}] \\
&\exists \pi, \delta_m \bullet \delta_m = 1/2 \cdot \delta \{\{0/x\}\} + 1/2 \cdot \delta \{\{1/x\}\} \wedge \delta' = \delta_m \langle \pi \rangle \{\{0/y\}\} + \delta_m \langle \bar{\pi} \rangle \{\{1/y\}\} \\
\equiv &\quad \text{One-point rule} \\
&\exists \pi \bullet \delta' = \left( 1/2 \cdot \delta \{\{0/x\}\} + 1/2 \cdot \delta \{\{1/x\}\} \right) \langle \pi \rangle \{\{0/y\}\} + \left( 1/2 \cdot \delta \{\{0/x\}\} + 1/2 \cdot \delta \{\{1/x\}\} \right) \langle \bar{\pi} \rangle \{\{1/y\}\} \\
\equiv &\quad [\text{p:D:Rmp:Lin}] \\
&\exists \pi \bullet \delta' = 1/2 \cdot \delta \{\{0/x\}\} \langle \pi \rangle \{\{0/y\}\} + 1/2 \cdot \delta \{\{1/x\}\} \langle \pi \rangle \{\{0/y\}\} + 1/2 \cdot \delta \{\{0/x\}\} \langle \bar{\pi} \rangle \{\{1/y\}\} + 1/2 \cdot \delta \{\{1/x\}\} \langle \bar{\pi} \rangle \{\{1/y\}\}
\end{aligned}$$

The final distribution  $\delta'(\pi)$  is parametric in a weighting distribution  $\pi$  and very similar to the resulting distribution after  $\Lambda$ , but with one crucial difference:  $\langle \pi \rangle$  is put *after* the first occurrence of the remap operator.

We can show that  $\|\delta'(\pi)\langle x = 1 \rangle\| = 1/2$ .

But if we choose  $\pi = \iota_{\delta'}\langle x = 1 \rangle$ , we have  $\delta'(\iota_{\delta'}\langle x = 1 \rangle) = 1/2 \cdot \delta\{\{0/x\}\{1/y\}\} + 1/2 \cdot \delta\{\{1/x\}\{0/y\}\}$  and therefore  $\|\delta'(\iota_{\delta'}\langle x = 1 \rangle)\langle x = y \rangle\| = 0$

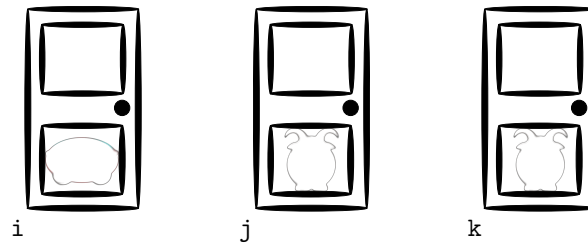
We have translated the programs and worked them out to express a predicate that links before-distributions with after-distributions: with this we can easily compute the minimum guaranteed probability that a condition will hold after the run of the program.

This is the same result we can achieve with *pGCL*, but:

- the notation is more general (and not as heavy as it would be with *pGCL*);
- we are not forced to stick with the minimum probability (“hard-linked” in the *pGCL* definition of demonic choice), but we have a set containing the probabilities of every branch of the execution tree;
- it is straight forward to refine the demonic choice with any other kind of choice — we simply have to constrain the existentially quantified  $\pi$ .

### A.3 The Monty Hall program



In the Monty Hall game a player is challenged to guess behind which of the three doors in front of him hides a car.

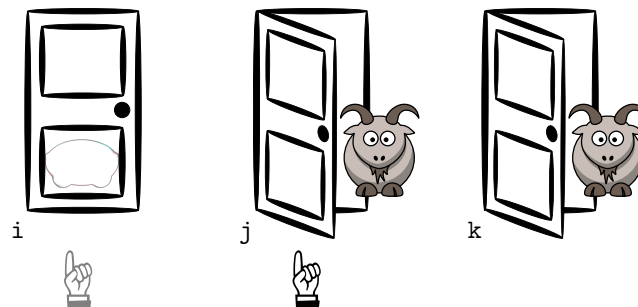


After having chosen a door among the three possible options, Monty Hall will open one of the remaining two doors. Monty Hall knows where the car is, so he is going to open one of the other two.

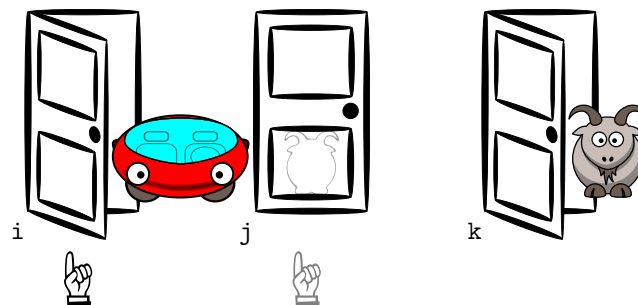
The player is given the chance to change his guess at this point.

It is known from the literature<sup>20</sup> that the player will maximize the probability of finding the car if now he changes the door he has chosen (the probability will be  $2/3$ ).

In fact the player can lose only if his first choice — indicated with  — was the  $i$ -th door, which is hiding the car (and this happens with probability  $1/3$ ) so after Monty Hall has opened the  $k$ -th door, that is one of the two hiding a goat, the switching strategy leads the player's final choice — indicated with  — to be the  $j$ -th door, which is hiding a goat:



Nevertheless this is a winning strategy with probability  $2/3$ , as the chances of winning equal the chances of choosing a door hiding a goat, when all doors are closed. In fact choosing the  $j$ -th door forces Monty Hall to open the  $k$ -th door, and switching makes the player choose the  $i$ -th door:



A short program, which uses the program constructs defined in §3.1, to implement the game is the following:

<sup>20</sup>Also back in 1935, it was known as Bertrand's box paradox (1889). This problem is often used as an example: among the papers cited as references, we can find it in McIver and Morgan [MM04] as well as in the more recent Chen and Sanders [CS09].

$$P \triangleq \text{setup}; \text{player}; \text{host}; \text{guess}$$

Let us use three variables  $a$ ,  $b$  and  $c$  with the following meaning:

$$\begin{aligned} a &\triangleq \text{the position of the car} \\ b &\triangleq \text{the player's guess} \\ c &\triangleq \text{Monty Hall's hint} \end{aligned}$$

we can then define each instruction as follows:

$$\text{setup} \triangleq a := 1 \sqcap (a := 2 \sqcap a := 3) \quad [1]$$

$$\text{player} \triangleq b := 1 \frac{1}{3} \oplus (b := 2 \frac{1}{2} \oplus b := 3) \quad [2]$$

$$\text{host} \triangleq c := \mathcal{S}(a, b) \triangleleft (a \neq b) \triangleright (c := \mathcal{H}_m(a) \sqcap c := \mathcal{H}_M(a)) \quad [3]$$

$$\text{guess} \triangleq b := \mathcal{S}(b, c) \quad [4]$$

Here is the definition of the functions mentioned in the program:

$$\begin{aligned} \mathcal{S}(x, y) &\triangleq \min(\{1, 2, 3\} \setminus \{x, y\}) \\ \mathcal{H}_m(x) &\triangleq \min(\{1, 2, 3\} \setminus \{x\}) \\ \mathcal{H}_M(x) &\triangleq \max(\{1, 2, 3\} \setminus \{x\}) \end{aligned}$$

Let  $\underline{v} = (a, b, c)$  and  $\text{type}(a) = \text{type}(b) = \text{type}(c) = \{1, 2, 3\}$ : the state space is

$$S = \{\sigma \mid \sigma = \underline{v} \mapsto \underline{w} \wedge \underline{w} \in \text{type}(a) \times \text{type}(b) \times \text{type}(c)\}$$

The initial distribution is a parameter of the problem: we assume its weight is 1, but make no further assumptions on the individual weight of each state.

Let us now go through the first instruction:

$$\begin{aligned} a := i &= \delta' = \delta \{i/a\} \\ \text{setup} &= \exists \pi_1, \pi_2 \bullet \delta' = \delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle 1 - \pi_1 - \pi_2 \rangle \{3/a\} \end{aligned}$$

After the second instruction we have:

$$\begin{aligned} b := i &= \delta'_i = \delta \{i/b\} \\ \text{player} &= \delta' = 1/3 \cdot \delta \{1/b\} + 1/3 \cdot \delta \{2/b\} + 1/3 \cdot \delta \{3/b\} \end{aligned}$$

We have an if-statement in the third instruction, so we have:

$$\begin{aligned} c := \mathcal{S}(a, b) &= \delta' = \delta \{ \mathcal{S}(a, b) / c \} \\ c := \mathcal{H}_m(a) &= \delta' = \delta \{ \mathcal{H}_m(a) / c \} \\ c := \mathcal{H}_M(a) &= \delta' = \delta \{ \mathcal{H}_M(a) / c \} \\ c := \mathcal{H}_m(a) \sqcap c := \mathcal{H}_M(a) &= \exists \pi_{\mathcal{H}} \bullet \delta' = \delta \langle \pi_{\mathcal{H}} \rangle \{ \mathcal{H}_m(a) / c \} + \delta \langle 1 - \pi_{\mathcal{H}} \rangle \{ \mathcal{H}_M(a) / c \} \\ \text{host} &= \exists \pi_{\mathcal{H}} \bullet \delta' = \delta \langle a \neq b \rangle \{ \mathcal{S}(a, b) / c \} + \\ &\quad + \delta \langle a = b \rangle \langle \pi_{\mathcal{H}} \rangle \{ \mathcal{H}_m(a) / c \} + \delta \langle a = b \rangle \langle 1 - \pi_{\mathcal{H}} \rangle \{ \mathcal{H}_M(a) / c \} \end{aligned}$$

Finally the fourth instruction gives

$$b := \mathcal{S}(b, c) = \delta' = \delta \{ \mathcal{S}(b, c) / b \}$$

Let us now compose sequentially these constructs:

$$\begin{aligned}
& \text{setup}; \text{player}; \text{host}; \text{guess} \\
\equiv & \quad \text{Translation: } \text{setup}; \text{player} \text{ — with the position } \bar{\pi}_{12} = \iota - \pi_1 - \pi_2 \\
& \exists \pi_1, \pi_2 \bullet \delta' = \delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\} ; \\
& \quad ; \delta' = 1/3 \cdot \delta \{1/b\} + 1/3 \cdot \delta \{2/b\} + 1/3 \cdot \delta \{3/b\} ; \text{host}; \text{guess} \\
\equiv & \quad [\text{d:P:Seq}] \\
& \exists \pi_1, \pi_2, \delta_m \bullet \delta_m = \delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\} \wedge \\
& \quad \wedge \delta' = 1/3 \cdot \delta_m \{1/b\} + 1/3 \cdot \delta_m \{2/b\} + 1/3 \cdot \delta_m \{3/b\} ; \text{host}; \text{guess} \\
\equiv & \quad \text{One-point rule} \\
& \exists \pi_1, \pi_2 \bullet \delta' = 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} ; \text{host}; \text{guess} \\
\equiv & \quad \text{Translation: } \text{host} \\
& \exists \pi_1, \pi_2 \bullet \delta' = 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} ; \\
& \quad ; \exists \pi_{\mathcal{G}_C} \bullet \delta' = \delta \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(\mathbf{a}, \mathbf{b})/c\} + \\
& \quad + \delta \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} + \delta \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} ; \text{guess} \\
\equiv & \quad [\text{d:P:Seq}] \\
& \exists \pi_1, \pi_2, \pi_{\mathcal{G}_C}, \delta_m \bullet \delta_m = 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} \wedge \\
& \quad \wedge \delta' = \delta_m \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(\mathbf{a}, \mathbf{b})/c\} + \\
& \quad + \delta_m \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} + \delta_m \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} ; \text{guess} \\
\equiv & \quad \text{One-point rule} \\
& \exists \pi_1, \pi_2, \pi_{\mathcal{G}_C} \bullet \delta' = \left( 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \right. \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} \left. \right) \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(\mathbf{a}, \mathbf{b})/c\} + \\
& \quad + \left( 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \right. \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} + \\
& \quad + \left( 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \right. \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
& \quad + 1/3 \cdot (\delta \langle \pi_1 \rangle \{1/a\} + \delta \langle \pi_2 \rangle \{2/a\} + \delta \langle \bar{\pi}_{12} \rangle \{3/a\}) \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{G}_C} \rangle \{J_{\mathcal{C}_M}(\mathbf{a})/c\} ; \text{guess}
\end{aligned}$$





$$\begin{aligned}
&\equiv \quad [d:P:Seq] \\
&\exists \pi_1, \pi_2, \pi_{\mathcal{J}C}, \delta_m \bullet \delta_m = \left( \frac{1}{3} \cdot (\delta\langle \pi_2 \rangle \{2/a\} + \delta\langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \right. \\
&\quad + \frac{1}{3} \cdot (\delta\langle \pi_1 \rangle \{1/a\} + \delta\langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
&\quad + \frac{1}{3} \cdot (\delta\langle \pi_1 \rangle \{1/a\} + \delta\langle \pi_2 \rangle \{2/a\}) \{3/b\} \left. \right) \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{J}C} \rangle \{J_m^C(a)/c\} + \\
&\quad + \left( \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{1/b\} + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{2/b\} + \right. \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{J}C} \rangle \{J_M^C(a)/c\} \wedge \\
&\quad \wedge \delta' = \delta_m \{S(b,c)/b\} \\
&\equiv \quad \text{One-point rule} \\
&\exists \pi_1, \pi_2, \pi_{\mathcal{J}C} \bullet \delta' = \left( \frac{1}{3} \cdot (\delta\langle \pi_2 \rangle \{2/a\} + \delta\langle \bar{\pi}_{12} \rangle \{3/a\}) \{1/b\} + \right. \\
&\quad + \frac{1}{3} \cdot (\delta\langle \pi_1 \rangle \{1/a\} + \delta\langle \bar{\pi}_{12} \rangle \{3/a\}) \{2/b\} + \\
&\quad + \frac{1}{3} \cdot (\delta\langle \pi_1 \rangle \{1/a\} + \delta\langle \pi_2 \rangle \{2/a\}) \{3/b\} \left. \right) \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \left( \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{1/b\} + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{2/b\} + \right. \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{J}C} \rangle \{J_m^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \left( \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{1/b\} + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{2/b\} + \right. \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \left. \right) \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{J}C} \rangle \{J_M^C(a)/c\} \{S(b,c)/b\} \\
&\equiv \quad [p:D:Rmp:Lin] \\
&\exists \pi_1, \pi_2, \pi_{\mathcal{J}C} \bullet \delta' = \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{1/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{1/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{2/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{2/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{3/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{3/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{1/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{J}C} \rangle \{J_m^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{2/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{J}C} \rangle \{J_m^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi_{\mathcal{J}C} \rangle \{J_m^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_1 \rangle \{1/a\} \{1/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{J}C} \rangle \{J_M^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \pi_2 \rangle \{2/a\} \{2/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{J}C} \rangle \{J_M^C(a)/c\} \{S(b,c)/b\} + \\
&\quad + \frac{1}{3} \cdot \delta\langle \bar{\pi}_{12} \rangle \{3/a\} \{3/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \bar{\pi}_{\mathcal{J}C} \rangle \{J_M^C(a)/c\} \{S(b,c)/b\}
\end{aligned}$$

Now that we have a statement describing the final distribution that results after the execution of the program, we can recognize two kind of terms:

- $\delta\{i/a\} \{i/b\} \langle \mathbf{a} \neq \mathbf{b} \rangle \{S(a,b)/c\} \{S(b,c)/b\}$
- $\delta\{i/a\} \{i/b\} \langle \mathbf{a} = \mathbf{b} \rangle \langle \pi \rangle \{J^C(a)/c\} \{S(b,c)/b\}$

where  $i \neq j$ .

We can see that the ones of the first kind account for cases when the player wins, while those of the second kind account for the cases when the player loses — let us see this by working out these terms, under the winning condition, *i.e.*  $a = b$ .

For terms of the first kind we have:

$$\begin{aligned}
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(a,b)/c\} \{S(b,c)/b\} \{a = b\} \\
= & \quad [p:D:Rmp:Comp1] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(a,b), S(b,c)\{S(a,b)/c\}/c, b\} \{a = b\} \\
= & \quad \text{Substitution: } c = S(a, b) \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(a,b), S(b, S(a,b))/c, b\} \{a = b\} \\
= & \quad [p:D:Sum:CS] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) = a\} \{S(a,b), S(b, S(a,b))/c, b\} \{a = b\} + \\
& \quad + \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) \neq a\} \{S(a,b), S(b, S(a,b))/c, b\} \{a = b\} \\
= & \quad [p:D:Rmp:Rst1] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) = a\} \{S(a,b), S(b, S(a,b))/c, b\} \{a = b\} + \epsilon \\
= & \quad [d:D:Sum] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) = a\} \{S(a,b), S(b, S(a,b))/c, b\} \{a = b\} \\
= & \quad [p:D:Rst:Rmp] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) = a\} \{S(a,b), a/c, b\} \{a = b\} \\
= & \quad [p:D:Rmp:Rst2] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(b, S(a, b)) = a\} \{S(a,b), a/c, b\} \\
= & \quad [p:D:Rst:ImC1] \\
& \delta \{i/a\} \{j/b\} \{a \neq b\} \{S(a,b), a/c, b\} \\
= & \quad [p:D:Rmp:Comp1] \\
& \delta \{i,j/a,b\} \{a \neq b\} \{S(a,b), a/c, b\} \\
= & \quad [p:D:Rmp:Rst2] \\
& \delta \{i,j/a,b\} \{S(a,b), a/c, b\}
\end{aligned}$$

As both remapping operations use expressions defined everywhere, thanks to [p:D:Rmp:Wt] we have that:

$$\|\delta \{i,j/a,b\} \{S(a,b), a/c, b\}\| = \|\delta\|$$

For terms of the second kind we have:

$$\begin{aligned}
& \delta \{i/a\} \{j/b\} \{a = b\} \langle \pi \rangle \{H(a)/c\} \{S(b,c)/b\} \{a = b\} \\
= & \quad [p:D:Rmp:Comp1] \\
& \delta \{i/a\} \{j/b\} \{a = b\} \langle \pi \rangle \{H(a), S(b,c)\{H(a)/c\}/c, b\} \{a = b\} \\
= & \quad \text{Substitution: } c = H(a) \\
& \delta \{i/a\} \{j/b\} \{a = b\} \langle \pi \rangle \{H(a), S(b, H(a))/c, b\} \{a = b\} \\
= & \quad [p:D:Rst:ES] \\
& \delta \{i/a\} \{j/b\} \{a = b\} \langle \pi \rangle \{H(a), S(a, H(a))/c, b\} \{a = b\} \\
= & \quad [p:D:Rst:Wt] \\
& \delta \{i/a\} \{j/b\} \{a = b\} \langle \pi \rangle \{H(a), S(a, H(a))/c, b\} \{a = b\} \\
= & \quad [p:D:Rmp:Rst1] \\
& \epsilon
\end{aligned}$$

Therefore we have:

$$\|\delta' \langle a = b \rangle\| = \|2 \cdot (1/3 \cdot \delta \langle \pi_1 \rangle + 1/3 \cdot \delta \langle \pi_2 \rangle + 1/3 \cdot \delta \langle \pi_3 \rangle)\| = 2/3 \cdot \|\delta\|$$

We have assumed that the weight of the initial distribution is 1, so the weight of all winning states is  $2/3$  — it is now clear why we did not need to make any other assumption, as this is all that

matters, as all the variables undergo at least an assignment during the run of the program.  $\frac{2}{3}$  is also the expected value for each of the initial states, so the pre-expectation assigning this weight to every state corresponds to the post-expectation of the predicate  $\iota(a = b)$ .

## B Proofs

B.1 Restriction of the state space  $\alpha\langle c \rangle = S\langle c \rangle \cap \alpha$

$p:A:Rst:Alt$

$$\begin{aligned}
 & \alpha\langle c \rangle \\
 = & [d:A:Rst] \text{ — §5.1} \\
 & \{\sigma \mid \sigma \in \alpha \wedge \sigma(c) = true\} \\
 = & [d:S] \text{ — §5.1} \\
 & \{\sigma \mid \sigma \in S \wedge \sigma \in \alpha \wedge \sigma(c) = true\} \\
 = & \text{Set theory} \\
 & \{\sigma \mid \sigma \in S \wedge \sigma(c) = true\} \cap \{\sigma \mid \sigma \in \alpha\} \\
 = & [d:A:Rst] \\
 & S\langle c \rangle \cap \alpha
 \end{aligned}$$

□

B.2 Restriction through equivalent condition  $(c_1 \Leftrightarrow c_2) \Rightarrow \chi\langle c_1 \rangle = \chi\langle c_2 \rangle$

$p:D:Rst:EqC$

$$\text{dom}(\chi)\langle c_1 \rangle = \text{dom}(\chi)\langle c_2 \rangle$$

□

B.3 Restriction through implied condition (I)  $(c_2 \Rightarrow c_1) \Leftrightarrow \chi\langle c_1 \rangle\langle c_2 \rangle = \chi\langle c_1 \rangle$

$p:D:Rst:ImC1$

$$\begin{aligned}
 & \chi\langle c_1 \rangle\langle c_2 \rangle \\
 = & [p:D:Rst:Cnj] \text{ — §5.2} \\
 & \chi\langle c_1 \wedge c_2 \rangle \\
 = & [p:D:Rst:EqC] \text{ — §B.2: } (c_2 \Rightarrow c_1) \wedge (c_1 \wedge c_2) \Leftrightarrow c_1 \\
 & \chi\langle c_1 \rangle
 \end{aligned}$$

□

B.4 Restriction through implied condition (II)  $(c_1 \Rightarrow \neg c_2) \Rightarrow \chi\langle c_1 \rangle\langle c_2 \rangle = \epsilon$

$p:D:Rst:ImC2$

$$\begin{aligned}
 & \chi\langle c_1 \rangle\langle c_2 \rangle \\
 = & [p:D:Rst:Cnj] \text{ — §5.2} \\
 & \chi\langle c_1 \wedge c_2 \rangle \\
 = & [p:D:Rst:EqC] \text{ — §B.2: } (c_1 \Rightarrow \neg c_2) \wedge (c_1 \wedge c_2) \Leftrightarrow false \\
 & \chi\langle false \rangle = \epsilon
 \end{aligned}$$

□

**B.5 Restriction through a restricted unitary distribution**  $\chi\langle c \rangle = \chi\{\iota_\chi\langle c \rangle\}$

$p:D:Rst:Alt$

$$\begin{aligned}
& \chi\{\iota_\chi\langle c \rangle\} \\
= & \quad [d:D:RstD] \text{ — §5.2} \\
& \left\{ \sigma \mapsto \chi(\sigma) \cdot \iota_\chi(\sigma) \mid \sigma \in \text{dom}(\chi) \cap \text{dom}(\iota_\chi\langle c \rangle) \right\} \\
= & \quad \text{Set theory: } \text{dom}(\iota_\chi\langle c \rangle) = \text{dom}(\chi\langle c \rangle) \subseteq \text{dom}(\chi) \\
& \left\{ \sigma \mapsto \chi(\sigma) \cdot \iota_\chi(\sigma) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \right\} \\
= & \quad [d:D:UD] \text{ — §5.2} \\
& \left\{ \sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \right\} \\
= & \quad [d:D:Rst] \text{ — §5.2} \\
& \chi\langle c \rangle
\end{aligned}$$

□

**B.6 Case Split**  $\chi = \chi\langle c \rangle + \chi\langle -c \rangle$

$p:D:Sum:CS$

$$\begin{aligned}
& \chi\langle c \rangle + \chi\langle -c \rangle \\
= & \quad [d:D:Sum] \text{ — §5.2} \\
& \left\{ \sigma \mapsto (\chi\langle c \rangle(\sigma) + \chi\langle -c \rangle(\sigma)) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \cup \text{dom}(\chi\langle -c \rangle) \right\} \\
= & \quad \text{Set theory} \\
& \left\{ \sigma \mapsto (\chi\langle c \rangle(\sigma) + 0) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \right\} \cup \left\{ \sigma \mapsto (0 + \chi\langle -c \rangle(\sigma)) \mid \sigma \in \text{dom}(\chi\langle -c \rangle) \right\} \\
= & \quad [d:D:Rst] \text{ — §5.2} \\
& \left\{ \sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \right\} \cup \left\{ \sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi\langle -c \rangle) \right\} \\
= & \quad \text{Set theory} \\
& \left\{ \sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi\langle c \rangle) \cup \text{dom}(\chi\langle -c \rangle) \right\} \\
= & \quad \text{Set theory} \\
& \left\{ \sigma \mapsto \chi(\sigma) \mid \sigma \in \text{dom}(\chi) \right\} \\
= & \quad [d:D] \text{ — §5.2} \\
& \chi
\end{aligned}$$

□

**B.7 Restriction**  $\pi_1\langle \pi_2 \rangle \in \mathcal{D}_w$

$p:D:WD:Rst$

$$\pi_1\langle \pi_2 \rangle(\sigma) = \pi_1(\sigma) \cdot \pi_2(\sigma) \leq \pi_1(\sigma)$$

□

**B.8 Restriction**  $\delta\langle \pi \rangle \in \mathcal{D}_p$

$p:D:PD:Rst$

$$\delta\langle \pi \rangle(\sigma) = \delta(\sigma) \cdot \pi(\sigma) \leq \delta(\sigma)$$

□

B.9 Nested inverse-image set  $Inv(\underline{v} := \underline{e}, Inv(\underline{v} := \underline{f}, \{\sigma\})) = Inv(\underline{v} := \underline{f}\{\underline{e}/\underline{v}\}, \{\sigma\})$

$p:S:Inv:Nest$

$$\begin{aligned}
& Inv(\underline{v} := \underline{e}, Inv(\underline{v} := \underline{f}, \{\sigma'\})) \\
= & [d:S:Inv] \text{ — §6.1} \\
& Inv(\underline{v} := \underline{e}, \{\sigma \mid \sigma'(\underline{v}) = \sigma(\underline{f}) \wedge \sigma \in \mathcal{S}_{\text{alph}(\sigma')}\}) \\
= & [d:S:Inv] \\
& \bigcup_{\zeta' \in \{\sigma \mid \sigma'(\underline{v}) = \sigma(\underline{f}) \wedge \sigma \in \mathcal{S}_{\text{alph}(\sigma')}\}} \{\zeta \mid \zeta'(\underline{v}) = \zeta(\underline{e}) \wedge \zeta \in \mathcal{S}_{\text{alph}(\zeta')}\} \\
= & \text{Property of distributed union} \\
& \{\zeta \mid \sigma(\underline{v}) = \zeta(\underline{e}) \wedge \sigma'(\underline{v}) = \sigma(\underline{f}) \wedge \sigma, \zeta \in \mathcal{S}_{\text{alph}(\sigma')}\} \\
= & [p:E:Ev:Comp] \text{ — §6} \\
& \{\zeta \mid \sigma'(\underline{v}) = \zeta(\underline{f} \circ \underline{e}) \wedge \zeta \in \mathcal{S}_{\text{alph}(\sigma')}\} \\
= & [d:E:Comp] \text{ — §6} \\
& \{\zeta \mid \sigma'(\underline{v}) = \zeta(\underline{f}\{\underline{e}/\underline{v}\}) \wedge \zeta \in \mathcal{S}_{\text{alph}(\sigma')}\} \\
= & [d:S:Inv] \\
& Inv(\underline{v} := \underline{f}\{\underline{e}/\underline{v}\}, \{\sigma'\})
\end{aligned}$$

□

B.10 Linearity of the remap operator  $(x \cdot \delta\{\underline{e}/\underline{v}\} + y \cdot \delta\{\underline{f}/\underline{v}\})\{\underline{g}/\underline{v}\} = x \cdot \delta\{\underline{e}/\underline{v}\}\{\underline{g}/\underline{v}\} + y \cdot \delta\{\underline{f}/\underline{v}\}\{\underline{g}/\underline{v}\}$

$p:D:Rmp:Lin$

$$\begin{aligned}
& (x \cdot \delta\{\underline{e}/\underline{v}\} + y \cdot \delta\{\underline{f}/\underline{v}\})\{\underline{g}/\underline{v}\}(\sigma) \\
= & [d:D:Rmp] \text{ — §6.2} \\
& \|(x \cdot \delta\{\underline{e}/\underline{v}\} + y \cdot \delta\{\underline{f}/\underline{v}\})\{Inv(\underline{v} := \underline{g}, \{\sigma\})\}\| \\
= & [p:D:Sum:Wt] \text{ — §5.2} \\
& \|x \cdot \delta\{\underline{e}/\underline{v}\}\{Inv(\underline{v} := \underline{g}, \{\sigma\})\} + y \cdot \delta\{\underline{f}/\underline{v}\}\{Inv(\underline{v} := \underline{g}, \{\sigma\})\}\| \\
= & [d:D:Rmp] \\
& x \cdot \delta\{\underline{e}/\underline{v}\}\{\underline{g}/\underline{v}\}(\sigma) + y \cdot \delta\{\underline{f}/\underline{v}\}\{\underline{g}/\underline{v}\}(\sigma) \\
= & [d:D:Sum] \\
& (x \cdot \delta\{\underline{e}/\underline{v}\}\{\underline{g}/\underline{v}\} + y \cdot \delta\{\underline{f}/\underline{v}\}\{\underline{g}/\underline{v}\})(\sigma)
\end{aligned}$$

□

**B.11 Composition (I)**  $\delta\{\underline{e}/\underline{v}\}\{\underline{f}/\underline{v}\} = \delta\{\underline{f}\{\underline{e}/\underline{v}\}/\underline{v}\}$

$p:D:Rmp:Comp1$

$$\begin{aligned}
& \delta\{\underline{e}/\underline{v}\}\{\underline{f}/\underline{v}\}(\sigma) \\
= & [d:D:Rmp] \text{ — §6.2} \\
& \|\delta\{\underline{e}/\underline{v}\}\{Inv(\underline{v} := \underline{f}, \{\sigma\})\}\| \\
= & [p:D:RstA:Wt] \text{ — §5.2} \\
& \sum_{\zeta \in Inv(\underline{v} := \underline{f}, \{\sigma\})} \delta\{\underline{e}/\underline{v}\}(\zeta) \\
= & [d:D:Rmp] \\
& \sum_{\zeta \in Inv(\underline{v} := \underline{f}, \{\sigma\})} \|\delta\{Inv(\underline{v} := \underline{e}, \{\zeta\})\}\| \\
= & [d:A:Inv]: \bigcup_{\zeta \in Inv(\underline{v} := \underline{f}, \{\sigma\})} Inv(\underline{v} := \underline{e}, \{\zeta\}) = Inv(\underline{v} := \underline{e}, Inv(\underline{v} := \underline{f}, \{\sigma\})) \\
& \|\delta\{Inv(\underline{v} := \underline{e}, Inv(\underline{v} := \underline{f}, \{\sigma\}))\}\| \\
= & [p:S:Inv:Nest] \text{ — §B.9} \\
& \|\delta\{Inv(\underline{v} := \underline{f}\{\underline{e}/\underline{v}\}, \{\sigma\})\}\| \\
= & [d:D:Rmp] \\
& \delta\{\underline{f}\{\underline{e}/\underline{v}\}/\underline{v}\}(\sigma)
\end{aligned}$$

□

**B.12 Composition (II)**  $\delta\{\underline{e}/\underline{v}\}\{\underline{f}/\underline{v}\} = \delta\{\underline{f} \circ \underline{e}/\underline{v}\}$

$p:D:Rmp:Comp2$

$$\begin{aligned}
& \delta\{\underline{e}/\underline{v}\}\{\underline{f}/\underline{v}\} \\
= & [p:D:Rmp:Comp1] \text{ — §B.11} \\
& \delta\{\underline{f}\{\underline{e}/\underline{v}\}/\underline{v}\} \\
= & [d:E:Comp] \text{ — §6} \\
& \delta\{\underline{f} \circ \underline{e}/\underline{v}\}
\end{aligned}$$

□

**B.13 Composition (III)**  $\delta\{\underline{e}/v_i\}\{\underline{f}/v_j\} = \delta\{(e, f\{e/v_i\})/(v_i, v_j)\}$

$p:D:Rmp:Comp3$

*Special case of B.11, where  $\underline{v} = \begin{pmatrix} v_i \\ v_j \end{pmatrix}$ ,  $\underline{e} = \begin{pmatrix} e \\ v_j \end{pmatrix}$  and  $\underline{f} = \begin{pmatrix} v_i \\ f \end{pmatrix}$ .*

□

**B.14 Composition (IV)**  $\delta\{\underline{e}/v_i\}\{\underline{f}/v_i\} = \delta\{f\{e/v_i\}/v_i\}$

$p:D:Rmp:Comp4$

*Special case of B.11, where  $\underline{v} = (v_i)$ ,  $\underline{e} = (e)$  and  $\underline{f} = (f)$ .*

□

**B.15 Iteration**  $\delta\{e/v\}^k = \delta\{e^k/v\}$

$p:D:Rmp:Iter$

By induction, the base case is trivial for  $k = \{0, 1\}$ .

Inductive hypothesis:  $\delta\{e/v\}^n = \delta\{e^n/v\}$

$$\begin{aligned}
 & \delta\{e/v\}^{n+1} \\
 = & \quad [d:D:Rmp:Iter] \text{ — §6.2} \\
 & \delta\{e/v\}^n \{e/v\} \\
 = & \quad \text{Inductive hypothesis} \\
 & \delta\{e^n/v\} \{e/v\} \\
 = & \quad [p:D:Rmp:Comp1] \text{ — §B.11} \\
 & \delta\{e\{e^n/v\}/v\} \\
 = & \quad [d:E:Comp] \text{ — §6} \\
 & \delta\{e \circ e^n/v\} \\
 = & \quad [d:E:Comp:Iter] \text{ — §6} \\
 & \delta\{e^{n+1}/v\}
 \end{aligned}$$

□

**B.16 Commutativity (I)**  $\delta\{e/v_i\} \{f/v_j\} = \delta\{f\{e/v_i\}/v_j\} \{e/v_i\}$  iff  $v_j \notin fv(e)$

$p:D:Rmp:Cmm1$

$$\begin{aligned}
 & \delta\{e/v_i\} \{f/v_j\} \\
 = & \quad [p:D:Rmp:Comp3] \text{ — §B.13} \\
 & \delta\{(e, f\{e/v_i\})/(v_i, v_j)\} \\
 = & \quad \text{Substitution: } v_j \notin fv(e) \Rightarrow e\{x/v_j\} = e \\
 & \delta\{(e\{f\{e/v_i\}/v_j\}, f\{e/v_i\})/(v_i, v_j)\} \\
 = & \quad \text{Substitution: } x = y\{x/y\} \\
 & \delta\{(e\{f\{e/v_i\}/v_j\}, v_j\{f\{e/v_i\}/v_j\})/(v_i, v_j)\} \\
 = & \quad [p:D:Rmp:Comp3] \\
 & \delta\{f\{e/v_i\}/v_j\} \{e/v_i\}
 \end{aligned}$$

□

**B.17 Commutativity (II)**  $\delta\{e/v_i\} \{f/v_j\} = \delta\{f/v_j\} \{e/v_i\}$  iff  $v_i \notin fv(f) \wedge v_j \notin fv(e)$

$p:D:Rmp:Cmm2$

$$\begin{aligned}
 & \delta\{e/v_i\} \{f/v_j\} \\
 = & \quad [p:D:Rmp:Cmm1] \text{ — §B.16} \\
 & \delta\{f\{e/v_i\}/v_j\} \{e/v_i\} \\
 = & \quad \text{Substitution: } v_i \notin fv(f) \Rightarrow f\{e/v_i\} = f \\
 & \delta\{f/v_j\} \{e/v_i\}
 \end{aligned}$$

□

**B.18 Expression substitution**  $\delta\{f = g\} \{e/v\} = \delta\{f = g\} \{e\{f/g\}/v\}$

$p:D:Rst:ES$

$$\begin{aligned}
 & \delta\{f = g\} \{e\{f/g\}/v\} \\
 = & \quad [d:E:Ev] \text{ — §5.1} \\
 & \delta\{f = g\} \{e/v\}
 \end{aligned}$$

□

**B.19 Contradiction**  $\forall \sigma \in \text{dom}(\delta) \bullet \sigma(\mathbf{c}\{\underline{e}/\underline{v}\}) = \text{false} \wedge \delta \neq \epsilon \Leftrightarrow \delta\{\underline{e}/\underline{v}\}(\mathbf{c}) = \epsilon$

$p:D:Rmp:Rst1$

$$\begin{aligned} & \forall \sigma \in \text{dom}(\delta) \bullet \sigma(\mathbf{c}\{\underline{e}/\underline{v}\}) = \text{false} \wedge \delta \neq \epsilon \\ \equiv & [d:D:Rmp] \text{ --- } \S 6.2 \\ & \forall \sigma' \in \text{dom}(\delta\{\underline{e}/\underline{v}\}) \bullet \sigma'(\mathbf{c}) = \text{false} \wedge \delta \neq \epsilon \\ \equiv & [d:D:Rst] \text{ --- } \S 5.2 \\ & \delta\{\underline{e}/\underline{v}\}(\mathbf{c}) = \epsilon \end{aligned}$$

□

**B.20 Assertion**  $\forall \sigma \in \text{dom}(\delta) \bullet \sigma(\mathbf{c}\{\underline{e}/\underline{v}\}) = \text{true} \Leftrightarrow \delta\{\underline{e}/\underline{v}\}(\mathbf{c}) = \delta\{\underline{e}/\underline{v}\}$

$p:D:Rmp:Rst2$

$$\begin{aligned} & \forall \sigma \in \text{dom}(\delta) \bullet \sigma(\mathbf{c}\{\underline{e}/\underline{v}\}) = \text{true} \\ \equiv & [d:D:Rmp] \text{ --- } \S 6.2 \\ & \forall \sigma' \in \text{dom}(\delta\{\underline{e}/\underline{v}\}) \bullet \sigma'(\mathbf{c}) = \text{true} \\ \equiv & [d:D:Rst] \text{ --- } \S 5.2 \\ & \delta\{\underline{e}/\underline{v}\}(\mathbf{c}) = \delta\{\underline{e}/\underline{v}\} \end{aligned}$$

□

**B.21 Remapping a condition**  $\delta\{\underline{e}/\underline{v}\}(\mathbf{c}) = \delta(\mathbf{c}\{\underline{e}/\underline{v}\})\{\underline{e}/\underline{v}\}$

$p:D:Rst:Rmp$

$$\begin{aligned} & \delta\{\underline{e}/\underline{v}\}(\mathbf{c}) \\ = & [p:D:Sum:CS] \text{ --- } \S B.6 \\ & \delta(\mathbf{c}\{\underline{e}/\underline{v}\})\{\underline{e}/\underline{v}\}(\mathbf{c}) + \delta(\neg\mathbf{c}\{\underline{e}/\underline{v}\})\{\underline{e}/\underline{v}\}(\mathbf{c}) \\ = & [p:D:Rmp:Rst1] \text{ --- } \S B.19 \\ & \delta(\mathbf{c}\{\underline{e}/\underline{v}\})\{\underline{e}/\underline{v}\}(\mathbf{c}) + \epsilon \\ = & [p:D:Rmp:Rst2] \text{ --- } \S B.20 \\ & \delta(\mathbf{c}\{\underline{e}/\underline{v}\})\{\underline{e}/\underline{v}\} \end{aligned}$$

□

**B.22 Weight of a distribution after remapping**  $\|\delta\{\underline{e}/\underline{v}\}\| = \|\delta\|$  iff  $\sigma(\underline{e})$  is defined in  $\text{dom}(\delta)$

$p:D:Rmp:Wt$

$$\begin{aligned} & \|\delta\{\underline{e}/\underline{v}\}\| \\ = & [d:D:Wt] \text{ --- } \S 5.2 \\ & \sum_{\sigma' \in \text{dom } \delta\{\underline{e}/\underline{v}\}} \delta\{\underline{e}/\underline{v}\}(\sigma') \\ = & [d:D:Rmp] \text{ --- } \S 6.2 \\ & \sum_{\sigma' \in \text{dom } \delta\{\underline{e}/\underline{v}\}} \|\delta(\text{Inv}(\underline{v} := \underline{e}, \{\sigma'\}))\| \\ = & [p:D:RstA:Wt] \text{ --- } \S 5.2 \\ & \sum_{\sigma' \in \text{dom } \delta\{\underline{e}/\underline{v}\}} \left( \sum_{\sigma \in \text{Inv}(\underline{v} := \underline{e}, \{\sigma'\})} \delta(\sigma) \right) \\ = & [p:A:Inv:EqR] \text{ --- } \S 6.1: \bigcup_{\sigma' \in \text{dom } \delta\{\underline{e}/\underline{v}\}} \text{Inv}(\underline{v} := \underline{e}, \{\sigma'\}) = \text{dom } \delta \text{ iff } \sigma(\underline{e}) \text{ is defined in } \text{dom}(\delta) \\ & \sum_{\sigma \in \text{dom } \delta} \delta(\sigma) \\ = & [d:D:Wt] \\ & \|\delta\| \end{aligned}$$

□



### B.23 Pseudo-associativity of probabilistic choice

$p:P:Ch:Prb:Assoc$

$$\begin{aligned}
& A \text{ }_p\oplus (B \text{ }_q\oplus C) \equiv (A \text{ }_r\oplus B) \text{ }_s\oplus C \wedge p = rs \wedge (1-s) = (1-p)(1-q) \\
& \equiv A \text{ }_p\oplus (B \text{ }_q\oplus C) \\
& \equiv [d:P:Ch:Prb] \text{ --- } \S 3.1 \\
& \equiv \exists \delta_A, \delta_{BC} \bullet A(p \cdot \delta, \delta_A) \wedge (B \text{ }_q\oplus C)((1-p) \cdot \delta, \delta_{BC}) \wedge \delta' = \delta_A + \delta_{BC} \\
& \equiv [d:P:Ch:Prb] \wedge \delta_{BC} = \delta_B + \delta_C \text{ (One-point rule)} \\
& \equiv \exists \delta_A, \delta_B, \delta_C \bullet A(p \cdot \delta, \delta_A) \wedge B(q(1-p) \cdot \delta, \delta_B) \wedge C((1-q)(1-p) \cdot \delta, \delta_C) \wedge \delta' = \delta_A + \delta_B + \delta_C \\
& \equiv (1-p)(1-q) = (1-s) \wedge p = rs \Rightarrow q(1-p) = (1-r)s \\
& \equiv \exists \delta_A, \delta_B, \delta_C \bullet A(rs \cdot \delta, \delta_A) \wedge B((1-r)s \cdot \delta, \delta_B) \wedge C((1-s) \cdot \delta, \delta_C) \wedge \delta' = \delta_A + \delta_B + \delta_C \\
& \equiv [d:P:Ch:Prb] \wedge \delta_{AB} = \delta_A + \delta_B \text{ (One-point rule)} \\
& \equiv \exists \delta_{AB}, \delta_C \bullet (A \text{ }_r\oplus B)(s \cdot \delta, \delta_{AB}) \wedge C((1-s) \cdot \delta, \delta_C) \wedge \delta' = \delta_{AB} + \delta_C \\
& \equiv [d:P:Ch:Prb] \\
& \equiv (A \text{ }_r\oplus B) \text{ }_s\oplus C
\end{aligned}$$

□

### B.24 Idempotency of choice operators $\forall X \bullet choice(A, A, X) \equiv A$

$p:P:Ch:Idem$

$$\begin{aligned}
& choice(A, A, X) \\
& \equiv [d:P:Ch] \text{ --- } \S 3.3 \\
& \equiv \exists \pi, \delta_A, \delta_{\bar{A}} \bullet \pi \in X \wedge A(\delta\langle\pi\rangle, \delta_A) \wedge A(\delta\langle\bar{\pi}\rangle, \delta_{\bar{A}}) \wedge \delta' = \delta_A + \delta_{\bar{A}} \\
& \equiv [d:P:Structure] \text{ --- } \S 3.1 \\
& \equiv \exists \pi, \delta_A, \delta_{\bar{A}}, QuantOf(A) \bullet \pi \in X \wedge \delta_A = BodyOf(A) \circ \delta\langle\pi\rangle \wedge \delta_{\bar{A}} = BodyOf(A) \circ \delta\langle\bar{\pi}\rangle \wedge \delta' = \delta_A + \delta_{\bar{A}} \\
& \equiv One-point rule \\
& \equiv \exists \pi, QuantOf(A) \bullet \pi \in X \wedge \delta' = BodyOf(A) \circ \delta\langle\pi\rangle + BodyOf(A) \circ \delta\langle\bar{\pi}\rangle \\
& \equiv [p:D:Sum:CS] \text{ --- } \S B.6 \\
& \equiv \exists \pi, QuantOf(A) \bullet \pi \in X \wedge \delta' = BodyOf(A) \circ \delta \\
& \equiv [d:P] \text{ --- } \S 3 \\
& A
\end{aligned}$$

□

### B.25 Discarding right-hand option $choice(A, B, \{\iota\}) \equiv A$

$p:P:Ch:Dscrd$

$$\begin{aligned}
& choice(A, B, \{\iota\}) \\
& \equiv [d:P:Ch] \text{ --- } \S 3.3 \\
& \equiv \exists \pi, \delta_A, \delta_B \bullet \pi \in \{\iota\} \wedge A(\delta\langle\pi\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}\rangle, \delta_B) \wedge \delta' = \delta_A + \delta_B \\
& \equiv One-point rule \\
& \equiv \exists \delta_A, \delta_B \wedge A(\delta\langle\iota\rangle, \delta_A) \wedge B(\delta\langle\epsilon\rangle, \delta_B) \wedge \delta' = \delta_A + \delta_B \\
& \equiv [d:D:Rst] \text{ --- } \S 5.2 \\
& \equiv \exists \delta_A, \delta_B \wedge A(\delta, \delta_A) \wedge B(\epsilon, \delta_B) \wedge \delta' = \delta_A + \delta_B \\
& \equiv One-point rule \\
& A
\end{aligned}$$

□

## B.26 Distributivity of choice operators

p:P:Ch:Dst

$$\begin{aligned}
& \text{choice}(A, (\text{choice}(B, C, X_2)), X_1) \equiv \text{choice}(\left(\text{choice}(A, B, X_1)\right), \left(\text{choice}(A, C, X_1)\right), X_2) \\
& \text{choice}(A, (\text{choice}(B, C, X_2)), X_1) \\
\equiv & \quad [d:P:Ch] \text{ — §3.3} \\
& \exists \pi_1, \delta_A, \delta_{BC} \bullet \pi_i \in X_i \wedge A(\delta\langle\pi_1\rangle, \delta_A) \wedge (\text{choice}(B, C, X_2))(\delta\langle\bar{\pi}_1\rangle, \delta_{BC}) \wedge \delta' = \delta_A + \delta_{BC} \\
\equiv & \quad [d:P:Ch] \wedge \delta_{BC} = \delta_B + \delta_C \text{ (One-point rule)} \\
& \exists \pi_1, \pi_2, \delta_A, \delta_B, \delta_C \bullet \pi_i \in X_i \wedge A(\delta\langle\pi_1\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}_1\rangle\langle\pi_2\rangle, \delta_B) \wedge C(\delta\langle\bar{\pi}_1\rangle\langle\bar{\pi}_2\rangle, \delta_C) \\
& \quad \wedge \delta' = \delta_A + \delta_B + \delta_C \\
\equiv & \quad [p:D:Sum:CS] \text{ — §B.6} \\
& \exists \pi_1, \pi_2, \delta_A, \delta_B, \delta_C \bullet \pi_i \in X_i \wedge A(\delta\langle\pi_1\rangle\langle\pi_2\rangle + \delta\langle\pi_1\rangle\langle\bar{\pi}_2\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}_1\rangle\langle\pi_2\rangle, \delta_B) \wedge \\
& \quad \wedge C(\delta\langle\bar{\pi}_1\rangle\langle\bar{\pi}_2\rangle, \delta_C) \wedge \delta' = \delta_A + \delta_B + \delta_C \\
\equiv & \quad \text{Linearity} \\
& \exists \pi_1, \pi_2, \delta_A, \delta_{\bar{A}}, \delta_B, \delta_C \bullet \pi_i \in X_i \wedge A(\delta\langle\pi_1\rangle\langle\pi_2\rangle, \delta_A) \wedge A(\delta\langle\pi_1\rangle\langle\bar{\pi}_2\rangle, \delta_{\bar{A}}) \wedge \\
& \quad \wedge B(\delta\langle\bar{\pi}_1\rangle\langle\pi_2\rangle, \delta_B) \wedge C(\delta\langle\bar{\pi}_1\rangle\langle\bar{\pi}_2\rangle, \delta_C) \wedge \delta' = \delta_A + \delta_{\bar{A}} + \delta_B + \delta_C \\
\equiv & \quad [d:P:Ch] \wedge \delta_{AB} = \delta_A + \delta_B \wedge \delta_{\bar{A}C} = \delta_{\bar{A}} + \delta_C \text{ (One-point rule)} \\
& \exists \pi_2, \delta_{AB}, \delta_{\bar{A}C} \bullet \pi_2 \in X_i \wedge (\text{choice}(A, B, X_1))(\delta\langle\pi_2\rangle, \delta_{AB}) \wedge (\text{choice}(\bar{A}, C, X_1))(\delta\langle\bar{\pi}_2\rangle, \delta_{\bar{A}C}) \\
& \quad \wedge \delta' = \delta_{AB} + \delta_{\bar{A}C} \\
\equiv & \quad [d:P:Ch] \\
& \text{choice}(\left(\text{choice}(A, B, X_1)\right), \left(\text{choice}(A, C, X_1)\right), X_2)
\end{aligned}$$

□

B.27 Sequential composition  $\text{choice}(A, B, X); C \equiv \text{choice}((A; C), (B; C), X)$ 

p:P:Ch:Seq

$$\begin{aligned}
& \text{choice}(A, B, X); C \\
\equiv & \quad [d:P:Seq] \text{ — §3.1} \\
& \exists \delta_m \bullet \text{choice}(A, B, X)(\delta, \delta_m) \wedge C(\delta_m, \delta') \\
\equiv & \quad [d:P:Ch] \text{ — §3.3} \\
& \exists \pi, \delta_A, \delta_B, \delta_m \bullet \pi \in X \wedge A(\delta\langle\pi\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}\rangle, \delta_B) \wedge \delta_m = \delta_A + \delta_B \wedge C(\delta_m, \delta') \\
\equiv & \quad \text{One-point rule} \\
& \exists \pi, \delta_A, \delta_B \bullet \pi \in X \wedge A(\delta\langle\pi\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}\rangle, \delta_B) \wedge C(\delta_A + \delta_B, \delta') \\
\equiv & \quad \text{Linearity} \\
& \exists \pi, \delta_A, \delta_B, \delta_C, \delta_{\bar{C}} \bullet \pi \in X \wedge A(\delta\langle\pi\rangle, \delta_A) \wedge B(\delta\langle\bar{\pi}\rangle, \delta_B) \wedge C(\delta_A, \delta_C) \wedge C(\delta_B, \delta_{\bar{C}}) \wedge \delta' = \delta_C + \delta_{\bar{C}} \\
\equiv & \quad [d:P:Seq] \\
& \exists \pi, \delta_C, \delta_{\bar{C}} \bullet \pi \in X \wedge (A; C)(\delta\langle\pi\rangle, \delta_C) \wedge (B; C)(\delta\langle\bar{\pi}\rangle, \delta_{\bar{C}}) \wedge \delta' = \delta_C + \delta_{\bar{C}} \\
\equiv & \quad [d:P:Ch] \\
& \text{choice}((A; C), (B; C), X)
\end{aligned}$$

□

**B.28 Choice flipping**  $\forall X \bullet \text{choice}(A, B, X) \equiv \text{choice}(B, A, \bar{X}) \wedge \bar{X} = \bigcup_{\pi \in X} \bar{\pi}$

*p:P:Ch:Flip*

$$\begin{aligned}
& \text{choice}(A, B, X) \\
\equiv & \quad [d:P:Ch] \text{ — §3.3} \\
& \exists \pi, \delta_A, \delta_B \bullet \pi \in X \wedge A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B \\
\equiv & \quad \bar{X} = \bigcup_{\pi \in X} \bar{\pi} \\
& \exists \bar{\pi}, \delta_A, \delta_B \bullet \bar{\pi} \in \bar{X} \wedge A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B \\
\equiv & \quad [d:P:Ch] \\
& \text{choice}(B, A, \bar{X})
\end{aligned}$$

□

**B.29 Monotonicity of generic choice**  $\forall \delta \bullet X_1 \subseteq X_2 \Rightarrow \text{choice}(A, B, X_1)(\delta) \subseteq \text{choice}(A, B, X_2)(\delta)$

*p:P:Ch:Mntn*

$$\begin{aligned}
& \text{choice}(A, B, X_2)(\delta) \\
= & \quad [d:P:Ch] \text{ — §3.3} \\
& (\exists \pi, \delta_A, \delta_B \bullet \pi \in X_2 \wedge A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B)(\delta) \\
= & \quad \text{Set theory} \wedge X_1 \subseteq X_2 \\
& (\exists \pi, \delta_A, \delta_B \bullet \pi \in X_1 \cup (X_2 \setminus X_1) \wedge A(\delta(\pi), \delta_A) \wedge B(\delta(\bar{\pi}), \delta_B) \wedge \delta' = \delta_A + \delta_B)(\delta) \\
= & \quad [d:P:Ch] \\
& \text{choice}(A, B, X_1)(\delta) \cup \text{choice}(A, B, X_2 \setminus X_1)(\delta)
\end{aligned}$$

□

**B.30 Refinement relation for choices involving  $X_2 \subseteq X_1$**   $X_2 \subseteq X_1 \Rightarrow \text{choice}(A, B, X_1) \sqsubseteq \text{choice}(A, B, X_2)$

*p:P:Rfn:Ch*

$$\begin{aligned}
& \text{choice}(A, B, X_1) \sqsubseteq \text{choice}(A, B, X_2) \\
\equiv & \quad [d:P:Rfn:Alt] \text{ — §4} \\
& \forall \delta \bullet \text{choice}(A, B, X_2)(\delta) \subseteq (\text{choice}(A, B, X_1)(\delta))^\Delta \\
\equiv & \quad [p:P:Ch:Mntn] \text{ — §B.29} \wedge \forall X \bullet X \subseteq (X)^\Delta \\
& \forall \delta \bullet \text{true}
\end{aligned}$$

□

**B.31 Refinement of the disjunction of two programs**  $A \vee B \sqsubseteq A \text{ }_p\oplus\text{ } B$

*p:P:Rfn:Dsj*

$$\begin{aligned}
& A \vee B \sqsubseteq A \text{ }_p\oplus\text{ } B \\
\equiv & \quad [d:P:Rfn:Alt] \text{ — §4} \\
& \forall \delta \bullet (A \text{ }_p\oplus\text{ } B)(\delta) \subseteq ((A \vee B)(\delta))^\Delta \\
\equiv & \quad \text{Set theory} \\
& \forall \delta, \delta' \bullet \delta' \in (A \text{ }_p\oplus\text{ } B)(\delta) \wedge \delta' \in ((A \vee B)(\delta))^\Delta \\
\equiv & \quad [d:P:Ch:Prb] \text{ — §3.1} \\
& \forall \delta, \delta', \delta'_A, \delta'_B \bullet \delta'_A \in A(\delta), \delta'_B \in B(\delta) \wedge \delta' = (p \cdot \delta'_A + (1-p) \cdot \delta'_B) \wedge \delta' \in ((A \vee B)(\delta))^\Delta \\
\equiv & \quad [d:P:RfnSet] \text{ — §4} \\
& \forall \delta \bullet \text{true}
\end{aligned}$$

□

### B.32 Refinement of the disjunction of two programs $A \vee B \sqsubseteq A \triangleleft c \triangleright B$

$p:P:Rfn:Dsj2$

$$\begin{aligned}
& A \vee B \sqsubseteq A \triangleleft c \triangleright B \\
\equiv & \quad [d:P:Rfn:Alt] \text{ — } \S 4 \\
& \forall \delta \bullet (A \triangleleft c \triangleright B)(\delta) \sqsubseteq ((A \vee B)(\delta))^\Delta \\
\equiv & \quad \text{Set theory} \\
& \forall \delta, \delta' \bullet \delta' \in (A \triangleleft c \triangleright B)(\delta) \wedge \delta' \in ((A \vee B)(\delta))^\Delta \\
\equiv & \quad [d:P:Ch:Cnd] \text{ — } \S 3.1 \\
& \forall \delta, \delta', \delta'_A, \delta'_B \bullet \delta'_A \in A(\delta \langle c \rangle), \delta'_B \in B(\delta \langle -c \rangle) \wedge \delta' = \delta'_A + \delta'_B \wedge \delta' \in ((A \vee B)(\delta))^\Delta \\
\equiv & \quad [d:P:RfnSet] \text{ — } \S 4 \\
& \forall \delta \bullet true
\end{aligned}$$

□

## C Notation

### C.1 Logic

$\neg$  : logical negation  
 $\wedge$  : logical conjunction  
 $\vee$  : logical disjunction  
 $\Rightarrow$  : implication  
 $\Leftrightarrow$  : double implication  
*true* : logical true  
*false* : logical false

### C.2 Relations and functions

$\mapsto$  : maps to  
 $\dagger$  : override  
 $\rightarrow$  : total function  
 $\rightrightarrows$  : partial function  
 $\mathcal{R}$  : relation  
 dom : domain operator  
 codom : codomain operator  
 img : image operator

### C.3 Probability

p, q, r, s : probability  
 P, Q : stochastic variable  
 $f_P$  : probability density function  
 $F_P$  : cumulative density function

#### C.4 Variables, values and expressions

$:=$  : assignment  
 $v$  : variable  
 $w$  : value  
 $e, f, g$  : expression  
 $c, d, z$  : boolean expression  
 $\underline{v}$  : vector of variables  
 $\underline{w}$  : vector of values  
 $\underline{e}, \underline{f}, \underline{g}$  : vector of expressions  
 $\mathcal{V}$  : set of variables  
 $\mathcal{W}$  : set of values  
 $\mathcal{E}$  : set of expressions  
 $\text{eval}$  : expression evaluation operator  
 $\text{type}$  : variable type operator  
 $fv$  : free variable operator  
 $bv$  : bound variable operator

#### C.5 States and distributions

$\sigma, \zeta$  : state  
 $\alpha$  : abstract state  
 $\mathcal{S}$  : set of all states (state space)  
 $\mathcal{A}$  : alphabet  
 $\text{alph}$  : alphabet operator  
 $\chi, \xi$  : distribution  
 $\epsilon$  : empty distribution  
 $\iota$  : unitary distribution  
 $\pi$  : weighting distribution  
 $\bar{\pi}$  : complementary weighting distribution  
 $\delta$  : probability distribution  
 $\mathcal{X}, \mathcal{Y}$  : set of distributions  
 $\mathcal{D}$  : set of all distributions  
 $\mathcal{D}_w$  : set of all weighting distributions  
 $\mathcal{D}_p$  : set of all probability distributions  
 $\|\_ \|$  : weight  
 $\_ \langle \_ \rangle$  : restriction  
 $\text{Inv}(\_, \_)$  : inverse-image set  
 $\_ \{ / \_ \}$  : remap

---

## C.6 Programs

*skip* : skip  
*abort* : abort  
*miracle* : miracle  
\* : iteration  
 $\_ \triangleleft \_ \triangleright \_$  : conditional choice  
 $\sqcup$  : angelic choice  
 $\sqcap$  : demonic choice  
 $p^\oplus, p^\ominus(1-p)$  : probabilistic choice  
*choice*( $\_ , \_ , \_$ ) : choice  
 $\sqsubseteq$  : refinement  
 $(\_)^\Delta$  : refinement set

## References

- [But10] Andrew Butterfield, ed. *Unifying Theories of Programming, Second International Symposium, UTP 2008, Dublin, Ireland, September 8-10, 2008, Revised Selected Papers*. Vol. 5713. Lecture Notes in Computer Science. Springer, 2010.
- [CS09] Yifeng Chen and Jeff W. Sanders. “Unifying Probability with Nondeterminism”. In: *FM 2009, LNCS 5850*. 2009, pp. 467–482.
- [Den<sup>+</sup>08] Yuxin Deng et al. “Characterising Testing Preorders for Finite Probabilistic Processes”. In: *Logical Methods in Computer Science* 4.4 (2008).
- [Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [DK03] L. De Raedt and K. Kersting. “Probabilistic Logic Learning”. In: *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining* 5.1 (2003). Ed. by S. Džeroski and L. De Raedt, pp. 31–48.
- [DS06] Steve Dunne and Bill Stoddart, eds. *Unifying Theories of Programming, First International Symposium, UTP 2006, Walworth Castle, County Durham, UK, February 5-7, 2006, Revised Selected Papers*. Vol. 4010. Lecture Notes in Computer Science. Springer, 2006.
- [FHM90] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. “A Logic for Reasoning about Probabilities”. In: *Information and Computation* 87 (1990), pp. 78–128.
- [FWB08] L. Freitas, J. Woodcock, and A. Butterfield. “POSIX and the Verification Grand Challenge: A Roadmap”. In: *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on* (2008), pp. 153–162.
- [GB09] Paweł Gancarski and Andrew Butterfield. “The Denotational Semantics of *slotted-Circus*”. In: *FM2009: Formal Methods*. Ed. by Ana Cavalcanti and Dennis Dams. Vol. 5850. LNCS. Springer, 2009, pp. 451–466.
- [Hae<sup>+</sup>01] R. Haenni et al. “A Survey on Probabilistic Argumentation”. In: *ECSQARU’01, Toulouse. Workshop: Adventures in Argumentation*. 2001, pp. 19–25.
- [Hae<sup>+</sup>08] Rolf Haenni et al. “Possible Semantics for a Common Framework of Probabilistic Logics”. In: *Interval / Probabilistic Uncertainty and Non-Classical Logics*. 2008, pp. 268–279.
- [He10] Jifeng He. “A Probabilistic BPEL-Like Language”. In: *UTP*. Ed. by Shengchao Qin. Vol. 6445. Lecture Notes in Computer Science. Springer, 2010, pp. 74–100.
- [Heh06] Eric C. R. Hehner. “Retrospective and Prospective for Unifying Theories of Programming”. In: *UTP 2006, LNCS 4010*. 2006, pp. 1–17.
- [HJ98] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall International Series in Computer Science, 1998.
- [HMM05] Joe Hurd, Annabelle McIver, and Carroll Morgan. “Probabilistic guarded commands mechanized in HOL”. In: *Theor. Comput. Sci* 346.1 (2005), pp. 96–112.
- [Hoa85a] C. A. R. Hoare. “Programs are predicates”. In: *Proceedings of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*. Upper Saddle River, NJ, USA: Prentice-Hall, 1985, pp. 141–155.
- [Hoa85b] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [HP07] Joseph Y. Halpern and Riccardo Pucella. “Characterizing and reasoning about probabilistic and non-probabilistic expectation”. In: *J. ACM* 54.3 (2007), p. 15.
- [HS06] Jifeng He and Jeff W. Sanders. “Unifying Probability”. In: *UTP 2006, LNCS 4010*. Ed. by Steve Dunne and Bill Stoddart. Vol. 4010. Lecture Notes in Computer Science. Springer, 2006, pp. 173–199.
- [HSM97] Jifeng He, K. Seidel, and A. McIver. “Probabilistic models for the guarded command language”. In: *Science of Computer Programming* 28.2-3 (1997). Formal Specifications: Foundations, Methods, Tools and Applications, pp. 171–192.

- [JKB07] Dominik Jain, Bernhard Kirchlechner, and Michael Beetz. “Extending Markov Logic to Model Probability Distributions in Relational Domains”. In: *KI '07: Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*. Osnabrück, Germany: Springer-Verlag, 2007, pp. 129–143.
- [Jøs01] Audun Jøsang. “A logic for uncertain probabilities”. In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 9.3 (2001), pp. 279–311.
- [KJH08] R. Kohlas, J. Jonczy, and R. Haenni. “A Trust Evaluation Method Based on Logic and Probability Theory”. In: *IFIPTM'08, 2nd Joint iTrust and PST Conferences on Privacy Trust Management and Security*. Ed. by Y. Karabulut et al. Vol. II. Trust Management. Trondheim, Norway, 2008, pp. 17–32.
- [Koh03] J. Kohlas. “Probabilistic Argumentation Systems: A New Way to Combine Logic With Probability”. In: *Journal of Applied Logic* 1.3-4 (2003), pp. 225–253.
- [Koz81] Dexter Kozen. “Semantics of Probabilistic Programs”. In: *J. Comput. Syst. Sci.* 22.3 (1981), pp. 328–350.
- [Koz85] Dexter Kozen. “A Probabilistic PDL”. In: *J. Comput. Syst. Sci.* 30.2 (1985), pp. 162–178.
- [Kra<sup>+</sup>95] Paul Krause et al. “A Logic of Argumentation for Reasoning under Uncertainty.” In: *Computational Intelligence* 11 (1995), pp. 113–131.
- [McI06] Annabelle McIver. “Quantitative Refinement and Model Checking for the Analysis of Probabilistic Systems”. In: *UTP 2006, LNCS 4010*. 2006, pp. 131–146.
- [MCM06] Annabelle McIver, E. Cohen, and Carroll Morgan. “Using Probabilistic Kleene Algebra for Protocol Verification”. In: *RelMiCS*. 2006, pp. 296–310.
- [Mis00] Michael W. Mislove. “Nondeterminism and Probabilistic Choice: Obeying the Laws”. In: *CONCUR 2000, LNCS 1877*. 2000, pp. 350–364.
- [MM02] Annabelle McIver and Carroll Morgan. “Games, Probability and the Quantitative  $\mu$ -calculus  $qM\mu$ ”. In: *LPAR*. 2002, pp. 292–310.
- [MM04] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004.
- [MM05] Annabelle McIver and Carroll Morgan. “Abstraction and refinement in probabilistic systems”. In: *SIGMETRICS Performance Evaluation Review* 32.4 (2005), pp. 41–47.
- [MM97] Carroll Morgan and Annabelle McIver. *A Probabilistic Temporal Calculus Based on Expectations*. Tech. rep. PRG-TR-13-97. Oxford University Computing Laboratory, 1997.
- [MM98] A.K. McIver and Carroll Morgan. “Demonic, Angelic and Unbounded Probabilistic Choices in Sequential Programs”. In: *Acta Informatica* 37 (1998), p. 2001.
- [Mor04] Carroll Morgan. “Of Probabilistic Wp and SP-and Compositionality”. In: *Communicating Sequential Processes: The First 25 Years, Symposium on the Occasion of 25 Years of CSP, in LNCS 3525*. Springer, 2004, pp. 220–241.
- [Mor<sup>+</sup>95] Carrol Morgan et al. *Argument Duplication in Probabilistic CSP*. Tech. rep. PRG-TR-11-95. Oxford University, 1995.
- [Mor<sup>+</sup>96] Carroll Morgan et al. “Refinement-Oriented Probability for CSP”. In: *Formal Asp. Comput.* 8.6 (1996), pp. 617–647.
- [MV04] Pedro R. D’Argenio Miguel Vásquez Nicolás Wolovick. *Probabilistic Hoare-like Logics in Comparison*. Tech. rep. Universidad Nacional de Córdoba, 2004.
- [MW05] Annabelle McIver and Tjark Weber. “Towards Automated Proof Support for Probabilistic Distributed Systems”. In: *LPAR*. 2005, pp. 534–548.
- [NM10] Ukachukwu Ndukwu and Annabelle McIver. “An expectation transformer approach to predicate abstraction and data independence for probabilistic programs”. In: *CoRR* (2010).



- 
- [NS09] Ukachukwu Ndukwu and J. W. Sanders. “Reasoning about a Distributed Probabilistic System”. In: *Fifteenth Computing: The Australasian Theory Symposium (CATS 2009)*. Ed. by Rod Downey and Prabhu Manyem. Vol. 94. CRPIT. Wellington, New Zealand: ACS, 2009, pp. 35–42.
- [OCW09] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. “A UTP semantics for Circus”. In: *Formal Asp. Comput* 21.1-2 (2009), pp. 3–32.
- [Qin10] Shengchao Qin, ed. *Unifying Theories of Programming - Third International Symposium, UTP 2010, Shanghai, China, November 15-16, 2010. Proceedings*. Vol. 6445. Lecture Notes in Computer Science. Springer, 2010.
- [SH03] Adnan Sherif and Jifeng He. “Towards a Time Model for Circus”. In: *Lecture Notes in Computer Science* 2495 (2003), pp. 613–624.
- [SSL10] Jun Sun, Songzheng Song, and Yang Liu. “Model Checking Hierarchical Probabilistic Systems”. In: *ICFEM*. Ed. by Jin Song Dong and Huibiao Zhu. Vol. 6447. Lecture Notes in Computer Science. Springer, 2010, pp. 388–403.
- [SZ99] J. W. Sanders and P. Zuliani. “Quantum Programming”. In: *In Mathematics of Program Construction*. Springer-Verlag, 1999, pp. 80–99.
- [Wil02] Jon Williamson. “Handbook of the Logic of Argument and Inference: the Turn Toward the Practical”. In: ed. by H. J. Ohlbach D. Gabbay R. Johnson and J. Woods. Elsevier, 2002. Chap. Probability Logic, pp. 397–424.
- [Yin03] Mingsheng Ying. “Reasoning about probabilistic sequential programs in a probabilistic logic”. In: *Acta Inf.* 39.5 (2003), pp. 315–389.