# A tour through the programming choices: semantics and applications

Pedro Ribeiro[1][0000−0003−4319−4872], Kangfeng Ye[1][0000−0003−2460−7926], Frank Zeyda[2][0009−0009−4251−4740], and Alvaro Miyazawa[1][0000−0003−2233−9091]

[1] Department of Computer Science, University of York, York, YO10 5GH, UK
{pedro.ribeiro,kangfeng.ye,alvaro.miyazawa}@york.ac.uk
[2] Independent Researcher, Zapopan, Mexico
https://www.linkedin.com/in/frank-zeyda/
frank.zeyda@gmail.com

**Abstract.** The discipline of programming, as Edsger Dijkstra would call it, has provided a formal basis for the study of programs where choices are not necessarily deterministic. That is, despite reservations from pioneers, like Tony Hoare, about the inclusion of nondeterministic behaviour in a program. For over 65 years, nondeterminism has played an important role in modelling and reasoning about programs. Moreover, there are at least two major flavours: *angelic* and *demonic*. Further programming choices, such as preferential and probabilistic, have been proposed to capture other important phenomena, for example, in the context of cyber-physical systems, distributed systems, and cryptography. In this chapter, we provide a critical account of these programming choices and their semantics, and discuss their applications. Our account focuses mainly on denotational semantics, and in particular relational models, as is fitting in honour of Jim Woodcock. We discuss approaches based on, or inspired by, weakest preconditions semantics and alphabetised relations, as found in the Unifying Theories of Programming (UTP). They are at the core of the semantics of notations such as Z and *Circus*, and extensions to deal with time and probabilities. Some of the ideas discussed here have been developed in close collaboration with Jim.

**Keywords:** programming · semantics · choice · relations · refinement · UTP

## 1 Dedication

The authors met Jim Woodcock at York in the early 2000s. Pedro first heard about the UTP as an MEng student when he had the privilege to attend Jim's lectures on the topic in the Summer of 2010. Kangfeng's academic career began from when he came to study for his PhD at York in 2012 under Jim's supervision. While his industry background was in embedded systems, Jim taught him a lot about formal methods. He then worked closely with Jim on probabilistic semantics and verification in various RoboStar projects. Alvaro first heard of Jim's work during his undergraduate degree when he read Jim's excellent book

"Using Z: Specification, Refinement, and Proof", which boosted his interest in formal methods. Since then, he had the pleasure of attending several talks by Jim. Frank first met Jim at ZB 2002 and also later at the first symposium on Unifying Theories of Programming that he helped to organise in 2006. He was so taken by Jim's enthusiasm and didactic talent to explain difficult topics in intuitive and accessible ways that he did not think twice when offered the opportunity to work in Jim's research group at York after completing his PhD.

The authors have greatly benefited from Jim's inspiration and mentorship. Thank you, Jim!

## 2 Introduction

The mathematical rigour advocated by programming pioneers, like Dijkstra and Hoare, has greatly influenced how modern software engineering is approached. In [175] Woodcock et al. provide one of the most comprehensive reviews on the application of formal methods in industry. In recent years, household names such as Amazon, Facebook, and Microsoft, have quietly propelled formal methods to the forefront of their software engineering processes. In the domain of robotics and autonomous systems, however, their wider adoption is more challenging [57]. The mix of concerns, including discrete, continuous, timed and probabilistic phenomena, requires a reconciliation of formalisms for holistic system reasoning.

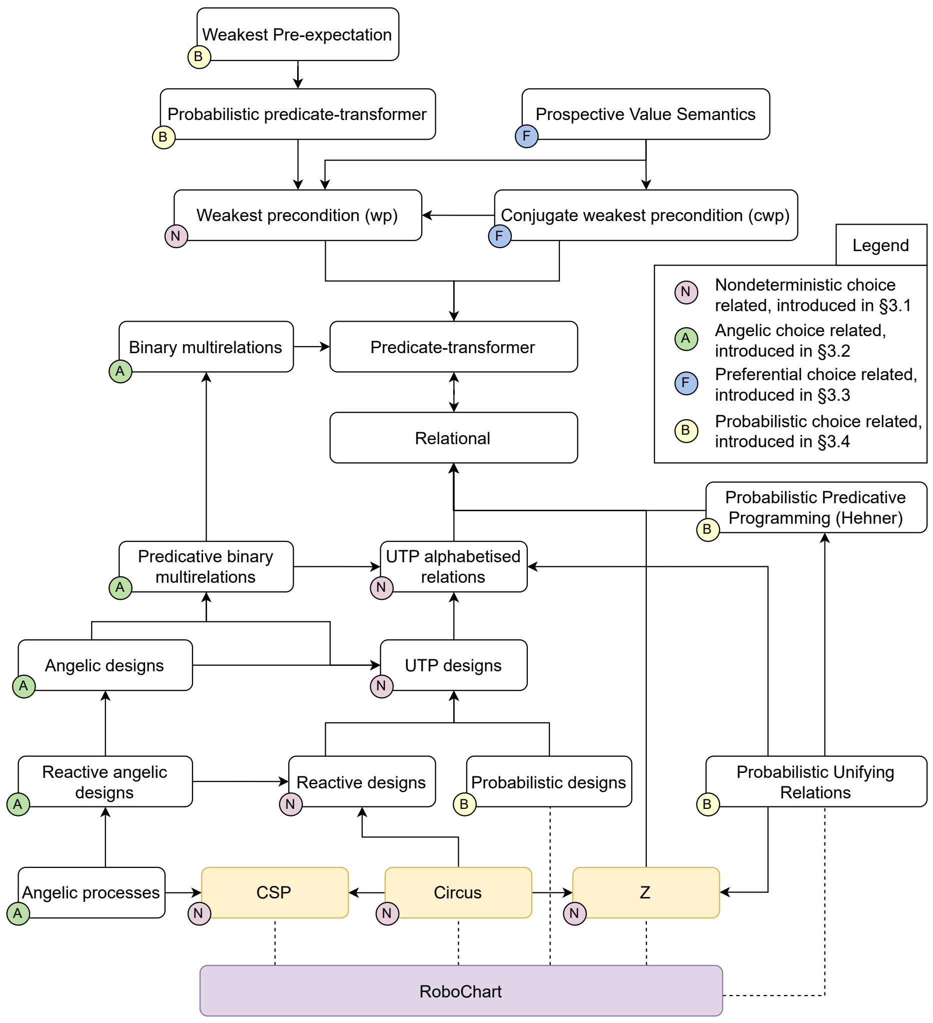
The unification of formalisms and their semantics has been at the forefront of much of Woodcock's research agenda [170], leveraging the pioneering work of Hoare and He [79] on the alphabetised relational calculus. In the UTP, programs, or relations, are specified via predicates where refinement gives rise to a lattice. It is convenient to model recursion, via fixed points, and *demonic* nondeterminism via the greatest lower bound. However, the latter is but one kind of nondeterministic choice, with other interpretations possible to model backtracking algorithms [45], namely as found in monotonic predicate transformers [8].

Similarly, probabilistic programs, widely used to model randomisation algorithms using probabilistic choice to simplify algorithms and improve performance, can be captured in both relational models [64,66,92] and predicate transformers [103,115]. Whether or not nondeterministic choice is supported and how it interacts with probabilistic choice is the most significant difference between different probabilistic models. A systematic account of predicate transformers is given by McIver and Morgan [102] for sequential and probabilistic programs. Here, in contrast, we are also interested in the study of choices for process calculi.

A somewhat lesser known and semantically more elusive notion of choice is that of *preference*. We think of the preferential choice $S \gg T$ as a kin of nondeterminism $S \sqcap T$ that prefers its first program $S$, provided that such turns out to be feasible[3]. While there exist several semantic investigations into preferential and prioritised choice, it can be shown that our notion of preference with its desired algebraic properties cannot be formally described using standard

[3] Feasible has a mathematical meaning here that we will elaborate on in Section 3.3.

predicate transformers. In [187] we show how a theory of expression transforms can capture the meaning of preference in terms of the *prospective value* of a given computation. This approach opens the doors for a whole new program model outside wp: that of preferential computations.



Weakest Pre-expectation
Probabilistic predicate-transformer
Prospective Value Semantics
Weakest precondition (wp)
Conjugate weakest precondition (cwp)
Legend
Nondeterministic choice related, introduced in §3.1
Angelic choice related, introduced in §3.2
Preferential choice related, introduced in §3.3
Probabilistic choice related, introduced in §3.4
Binary multirelations
Predicate-transformer
Relational
Probabilistic Predicative Programming (Hehner)
Predicative binary multirelations
UTP alphabetised relations
Angelic designs
UTP designs
Reactive angelic designs
Reactive designs
Probabilistic designs
Probabilistic Unifying Relations
Angelic processes
CSP
Circus
Z
RoboChart
N
A
F
B

**Fig. 1.** Hierarchical structure of theories and formal notations, where arrows loosely indicate theory dependencies and influences. RoboChart, a notation for design of robotics software discussed in this chapter, has process-algebraic and probabilistic semantics.

In Figure 1 we provide an overview of all the semantic models we consider in our discussion, and how they loosely relate to, or have influenced, each other. The top-half concerns predicate transformers, whereas the bottom-half concerns (multi)relational models. At the bottom, we highlight in purple RoboChart [110], a domain-specific notation for formal modelling of robotic controllers, considered as an application in Section 4.2, and in light-yellow the formal notations that underpin RoboChart. Solid arrows denote dependencies between theories, while dotted lines indicate the theories on which RoboChart directly depends.

This chapter's aim is to study and contrast different key notions of programming choices: *demonic*, *angelic*, *external*, *preferential* and *probabilistic*, and their fundamental properties. This includes discussions of how these choices may be characterised in different semantic models. We also sketch how they can be used to model different phenomena relevant to cyber-physical systems. In particular, we propose a novel process algebraic model for a co-simulation master algorithm, discuss the role *angelic* choice could play in the design of robotic controllers, and consider the modelling of random walkers using *probabilistic* choice.

In the spirit of some variants of modern literature, readers are not required to read the paper from start to finish but skip and jump to sections that mostly spark their interest, guided by Figure 1. The key notions of choice, namely *nondeterministic*, *angelic*, *preferential*, and *probabilistic* have each a dedicated subsection in Section 3, and we strove to make these sections self-contained.

The chapter is structured as follows. In Section 3, we provide an account of the examined programming choices, namely by discussing semantic models suitable for characterising choices in sequential programs and process calculi. Then, in Section 4, we sketch their novel applications. Finally, we conclude in Section 5.

## 3 Programming choices

In this section, we address nondeterministic, angelic, preferential and probabilistic choices and discuss their semantics in the context of a variety of semantic frameworks, including UTP and the wp-calculus. While Figure 1 gave us the big picture of the relationships between programming choices and semantic models, here we shall delve deeper into the intuitive understanding of these choices, and if and how they can be expressed in various semantic models and frameworks.

### 3.1 Nondeterministic choice

The ability to write programs that can lead to different executions when started from the same state did not feature in the early theories of computation. Pioneers [36, 87], including Turing [160], focused their attention on deterministic machines [3, 4], perhaps with good reservations as Dijkstra later quoted Hoare's observation that:

> "A system which permits user programs to become non-deterministic presents dreadful problems to the maintenance engineer: it is not a 'facility' to be lightly granted." [39, p.12]

Nondeterminism, however, is at the core of computer science, crucial in accounting for uncontrollable phenomena, and particularly elegant for capturing many computational paradigms. It is perhaps not so surprising then, that Rabin and Scott [134] considered nondeterministic automata. In that setting, nondeterminism does not confer extra expressiveness as the power set construction shows, but it can be advantageous in the number of states required. Similarly, context-free grammars allow for nondeterminism in rule application [30]. Importantly, in those models, language acceptance merely requires reaching an accepting state.

**Imperative** The earliest use of nondeterminism in a program is due to McCarthy [101], who proposed an ambiguity operator $amb(x, y)$ that nondeterministically returns the values of defined expressions $x$ or $y$. This would be used to define "computably ambiguous functions" and influence dialects [1,185] of Lisp.

In a different vain, Floyd [45] proposed a nondeterministic assignment that allows for an elegant characterisation of computations that would otherwise require explicit backtracking, such as the eight-queens problem. Novel contributions included the annotation of failure and success points in a program.

Dijkstra would eventually embrace nondeterminism in his language of guarded commands [40] (GCL) after overcoming a "considerable mental resistance" [39, p.12]. The semantics of commands is given using the *wp* calculus, where $wp(S, R)$ defines for a post-condition $R$ and command $S$, the weakest pre-condition that must be satisfied in order for execution of $S$ to terminate and establish $R$.

Importantly, Dijkstra would observe the "Law of the Excluded Miracle", by requiring that $wp(S, F) = F$, and consider conjunctive, rather than arbitrary, predicate transformers. Thus, in that context it is impossible to entertain a program that achieves either outcome as long as one is individually achievable. Consider the program $P = x := 1 \sqcap x := 2$ that has a choice ($\sqcap$) between assigning ($:=$) the value 1 or 2 to a variable $x$, then we have that $wp(P, \{x = 1 \vee x = 2\})$ is achievable from any state, yet both $wp(P, \{x = 1\})$ and $wp(P, \{x = 2\})$ have *infeasible* preconditions. In that setting, choice is *demonic* following Dijkstra's allegory of a "daemon [that] decides quite arbitrarily" [38, p.7].

The notion of *demonic* nondeterminism is in stark contrast with Floyd's choice points, that, by analogy, take the *angelic* view. While the treatment of both notions within a single framework is explored further in the next section, they played important roles in the refinement calculus of Back [8], Morgan [112] and Morris [117]. In particular, the adoption of the 'Miracle' would allow the definition of complete lattices. Woodcock would not only write a tutorial on this topic [167], but also discuss how to apply refinement in Z [166]. Such ideas would later be applied in the formalisation of the Mondex smart card, which achieved ITSEC Level E6 [31, 176], but not before Woodcock found that Z proof rules were incomplete as elegantly detailed in [152]. This led to the development of backward refinement rules. The Mondex would be one of the first projects to be tackled in the Grand Challenge in Verified Software [84] initiated by Hoare. Freitas and Woodcock would later mechanise their proofs using Z/EVES [52].

**Process calculi** When Hoare proposed Communicating Sequential Processes (CSP), he adopted Dijkstra's GCL as the "sole means of introducing and controlling nondeterminism" [77]. This seminal work spurred an interest in the study of the semantics of CSP, some of which we cover in further detail. Importantly, in contrast with CCS [108] and ACP [13], CSP has a theory for refinement.

One of the earliest denotational models for CSP was proposed by Francez et al. [51] based on *history trees*, where two types of determinism are explicitly distinguished: local, where a process can decide which value to communicate, and global, that depends on the interaction with other processes. This distinction would later lead to the consideration of two distinct choice operators over processes: internal choice ($\sqcap$), that decides irrespective of interaction with the environment; and external choice ($\Box$) that allows a process' environment to choose. An example from [51] is recast using these operators and a typed channel $e$:

*Example 3.1.* $P = (a \rightarrow e?x \rightarrow \mathit{Skip} \;\Box\; a \rightarrow e!0 \rightarrow \mathit{Skip}) \;[\![\, \{e\} \,]\!]\; e!1 \rightarrow \mathit{Skip}$
$Q = (e?x \rightarrow \mathit{Skip} \;\Box\; e!0 \rightarrow \mathit{Skip}) \;[\![\, \{e\} \,]\!]\; e!1 \rightarrow \mathit{Skip}$

Here, process $P$ is defined by a parallel composition ($[\![ \ldots ]\!]$) of two processes synchronising on channel $e$, but where event $a$ can happen independently. The left hand-side process has a choice ($\Box$) between a prefixing ($\rightarrow$) on event $a$, followed by an input ($e?x$) on $e$ and terminating ($\mathit{Skip}$), or, similarly, after $a$ communicating the value 0 via $e$, and then also terminating. The right hand-side process, however, can only agree on the value 1 being communicated ($e!1$) via $e$. We observe that in this case the external choice is actually nondeterministic, as the algebraic laws can show [143], and so $P$ can enter in deadlock. In contrast, process $Q$ does not deadlock, given that the choice can always be resolved.

In [78] Hoare considered a denotational model based on *traces*, that is, the sequences of events a process can perform, restricted to deterministic processes. Traces, however, are inadequate for distinguishing internal and external choice and for preserving liveness, for example, by having deadlock as the maximally refined process. Addressing this, a *failures* semantics would later be considered by Brookes et al. [16], where in addition to traces, the events refused at the end of every trace are also recorded. Related, De Nicola and Hennesy [123] would identify that failures refinement coincides with *must*-testing in CCS.

Refinement in the *failures* semantics, defined simply by (reverse) subset inclusion, similarly to *traces*, gives rise to a partial order where deadlock is no longer the maximally refined process. Instead, however, a process that can only perform a sequence of internal events is maximally refined. Such behaviour can easily be introduced in CSP via hiding ($\setminus$) the event of a guarded recursion. For example, if we let $R = a \rightarrow R$ and then consider $S = R \setminus \{a\}$, then $S$ would have no stable *failures*, and hence would be maximal in the refinement order. To overcome this, Roscoe et al. [143] would propose the *failures-divergences* semantic model, where $S$ is considered to be divergent, that is, the bottom in the refinement order, given that it cannot provide any guarantees on its behaviour.

**UTP** Total correctness, as found in *wp*, would be recast as a theory of designs [171] in the Unifying Theories of Programming (UTP) of Hoare and He [79], a framework of alphabetised relations, in the style of Hehner's predicative programming approach [65], proposed as a sound basis for studying several programming paradigms. Although only published in book form in 1998, the UTP would play a central role in Woodcock's contributions [174] in the following decades.

In the UTP, theories are characterised by three components: an alphabet, that defines the variables considered in a relation; a signature; and a set of healthiness conditions, usually defined by monotonic and idempotent functions over predicates whose fixed points define the valid predicates of a theory. The refinement order is defined by the universally quantified reverse implication of predicates, and gives rise to a complete lattice with *false* as the top element, allowing no observations, and *true* as the bottom, allowing any observation. Importantly, *demonic* nondeterminism is captured via the greatest lower bound.

*Designs* As observed by Hoare and He, relations are unsuitable, on their own, to define a model of total correctness, and so the theory of designs employs an auxiliary variable $ok'$ that records whether a program has terminated, and similarly an undashed version, $ok$, records whether the current program has started. The simplest design is therefore $ok \Rightarrow ok'$, equivalently stated using Hoare and He's turnstile notation as $true \vdash true$, that when started terminates successfully: its pre- and postcondition are both *true*. The above form rules out explicit non-termination $(\neg\, ok')$ or the ability to state anything about program variables before a program starts $(\neg\, ok)$. This is captured by two healthiness conditions **H1** and **H2** [79,172]. Another two optional conditions are considered: **H3**, that rules out designs that can make final observations without terminating; and **H4**, that rules out miraculous designs. **H3**-healthy designs correspond to (conjunctive) predicate transformers [25], while miracles allow the specification of *assumptions* and play an important role [169] in theories that we discuss next.

*Reactive designs* Amongst the process calculi studied in the UTP, Hoare and He would propose a recast of CSP [23, 79] via a theory of reactive processes that considers auxiliary variables $tr$, that records the trace of events, and $wait$, that records whether a process is waiting for an interaction with its environment. The key result is that the semantics of CSP processes can be defined as the image of designs through the application of the healthiness condition **R** for reactive processes, that is, using pre- and postconditions. For example, $\mathbf{R}(true \vdash wait' \wedge tr' = tr)$ deadlocks by stating that the process waits and keeps the trace unchanged. Importantly, reactive designs don't satisfy **H3**, as there can be intermediate interactions irrespective of termination in the reactive paradigm. Canham and Woodcock [20] would refer to a postcondition where $wait'$ is *true* as the *peri*condition, emphasising the intermediate nature. Foster et al. [47] would provide a systematic account of all three pre-/peri-/post-conditions of reactive contracts in their Isabelle/UTP [46, 50] mechanisation.

*Circus* The combination of state-based formalisms and process calculi has received much attention [18,44,144,151]. *Circus* is a combination of Z and CSP, incorporating Dijkstra's guarded commands and Morgan's specification statement, with a UTP semantics [127] given via reactive designs. It targets refinement [22] and has been exploited in the context of industrial and safety-critical applications [126, 168]. Woodcock and others would consider extensions of *Circus* to cater for other phenomena, such as time, by exploiting its UTP semantics.

One of the earliest works on *Circus Time* is that of Adnan and He [149, 150]. The semantics is given via reactive designs using a model of *failures* within a sequence of discrete time units. A similar structure would be exploited for slotted-*Circus* by Butterfield et al. [19], that can be captured via a generalisation [139] of Foster et al.'s trace algebra [48]. Importantly, reactive miracles would allow Wei et al. [164] to capture deadlines, by pruning periconditions once a deadline has been reached, thus making interactions urgent. Refinements that meet their deadlines more tightly than specified are non-urgent, in the same way that a design is non-miraculous in an environment that meets its assumptions.

## 3.2 Angelic choice

Amongst the pioneers [30, 146], Floyd [45] is perhaps first in employing nondeterminism as a specification mechanism abstracting away from implementation details of backtracking algorithms. This view would be used by Cook [32] to characterise the class of NP problems. The use of both *angelic* and *demonic* nondeterminism (alternation) would be considered for Turing machines in [27].

**Imperative** The earliest use of the term *angelic* seems to be due to Broy and Wirsing [17], who studied four types of nondeterminism via algebraic data types. However, they refer to *unbounded nondeterminism* as *angelic*, and *demonic* as backtracking [90], in contrast to recent works. *Angelic* choice has found use in: parsing [70], modelling of game-like scenarios [8], proof tactics [99, 125], constraint [82] and logic programming [91], and in conformance relations [21].

Hesselink [71] would propose a subsumption of the GCL to cope with the semantics of recursion. As Dijkstra points out [41], Hesselink would further study its operational semantics [74] and consider an extension with angelic choice [72]. Dijkstra's work [41] on reasoning about UNITY [28] via arbitrary monotonic predicate transformers also considers both *demonic* and *angelic* choices.

*Refinement calculus* Back and von Wright [8] extensively studied sublattices, where choice can be *angelic* or *demonic*. The lattice of monotonic predicate transformers is the most important, where *angelic* choice and *demonic* choice are duals in the lattice. Related, *angelic* choice has also been used by Gardiner and Morgan [54] to formalise logical variables, which is central to their calculational data-refinement approach, for example, by allowing the postcondition of a specification statement to refer to the initial value of program variables. Ward and Hayes [163] would emphasise in their work, that the angelic choice of the refinement calculus can "look ahead" and avoid aborting, if at all possible.

*Multirelations* In theories of total correctness, such as in Z and VDM [85], it is natural to model computations via relations. However, as observed by Rewitzky and Brink [138], and later Back and von Wright [8], relations can only capture one type of nondeterminism, either *angelic* or *demonic*, but not both. Instead, binary multirelations [100, 137], that is, relations between a state and a set of states, can be used to model both: the set can be understood to model *angelic* or *demonic* choices, while the relation models the dual, respectively. As observed by Hesselink [73] multirelations are equivalent to one of the constructions of the Free Distributive Completion (FDC) of the state space, originally studied by Morris [118]. More recently, Furusawa et al. [53] have extensively studied multirelations to provide an algebraic account of concurrent dynamic logic [129].

**Process calculi** As mentioned, in the UTP, *demonic* nondeterminism is captured by the greatest lower bound of the lattice. Because it is a theory of relations, it cannot capture both forms of nondeterminism, as established by Cavalcanti et al. [25] who show that, in general, UTP relations are isomorphic to conjunctive predicate transformers. They also consider a predicative encoding of upward-closed binary multirelations, as non-homogeneous alphabetised relations, that is shown isomorphic to arbitrary monotonic predicate transformers. Importantly, it is a theory of designs that satisfies **H3**, and so it is inadequate to give a treatment of *angelic* choice for reactive designs and process calculi.

Ribeiro and Cavalcanti [142] would study this problem further. First, they generalised the predicative encoding of binary multirelations to consider non-**H3** *angelic* designs, and showed that theory to be isomorphic to an extended notion of upward-closed binary multirelations where the target type includes the option of non-termination [141]. Secondly, they used that encoding to study the operators of CSP, recast in a theory of reactive angelic designs [140], and their interplay with *angelic* choice. In that theory, angelic choice can avoid immediate divergence, by satisfying the following law: $(\mathit{Stop} \sqcup \mathit{Skip})\,;\, \mathit{Chaos} = \mathit{Stop}$ where $\sqcup$ is *angelic* choice, defined as the least upper bound of the lattice, and $;$ is sequential composition. That is, the angel can choose to deadlock ($\mathit{Stop}$), rather than diverge ($\mathit{Chaos}$). However, interactions cannot be eliminated in the same way, even if they could to lead to a divergence, as can be seen from the example:

*Example 3.2.* $(a \rightarrow \mathit{Skip}) \sqcup (b \rightarrow \mathit{Chaos}) = (a \rightarrow \mathit{Skip}) \sqcup (b \rightarrow \mathit{Choice})$

where $a$ and $b$ are different events. Here, the divergence is avoided by behaving as the most nondeterministic process that does not diverge ($\mathit{Choice}$).

As it turns out, the standard healthiness conditions of reactive processes insist on extension of traces (i.e. no backtracking), and thus disallow any pruning of interactions that could lead to divergence. Instead, [142] considers a theory of angelic processes that does not insist on trace extension. In that context, the following law holds: $a \rightarrow \mathit{Chaos} \sqcup b \rightarrow \mathit{Skip} = b \rightarrow \mathit{Skip}$. That is, the prefixing on $a$ can be eliminated, given that it would lead to a divergence. In that theory, imposition of the standard healthiness conditions, can be used to scope the level of 'backtracking' similarly to a cut operator.

Related, Tyrrell et al. [161] have proposed an axiomatization of CSP, where external choice is referred to as angelic, but where deadlock is indistinguishable from divergence. Roscoe [143] has considered an alternative to external choice via operational combinator semantics of CSP, where, however, the possibility of divergence has no effect on 'backtracking'.

## 3.3 Preferential choice

In comparison to other forms of choice discussed in this chapter, preferential choice is a slightly odd companion: it is asymmetric, non-monotonic, and cannot be expressed in the theory of predicate transformers. Intuitively, a preferential choice $S \gg T$ behaves like $S$ as long as S and its continuation is feasible, meaning non-miraculous. It only behaves like $T$ if selection of $S$ results in a miracle further down the line. To illustrate this, we consider the computation:

$$(x := 1 \gg x := 2) \; ; \; U$$

Assuming we have $\mathbf{fis}(U) \equiv \mathsf{true}$[4], meaning that $U$ is feasible from all initial states[5], $S$ behaves similar to $x := 1$ and thus in a deterministic fashion.

However, let us assume $U$ becomes infeasible in a state where $x = 1$. The guarded computation $x \neq 1 \longrightarrow \mathbf{skip}$ precisely exhibits such behaviour. Then, we have that

$$(x := 1 \gg x := 2) \; ; \; x \neq 1 \longrightarrow \mathbf{skip}$$

becomes equivalent to $x := 2$. This behaviour of choosing the second alternative when the first one leads to infeasibility is reminiscent of (standard) nondeterministic choice $S \sqcap T$ and holds there too. We are likewise entitled to refine the second operand $T$ of a preferential choice to obtain a refinement of the entire construct, hence preference shares some semantic properties and traits with nondeterminism. However, things get hairy when we are trying to refine the first operand, which is where monotonicity unfortunately breaks down.

To illustrate this, let us consider the computation:

$$(x := 1 \gg x := 2) \; ; \; (x = 1 \mathrel{\rule[-0.2ex]{0.3em}{1.8ex}} \mathbf{skip})$$

where $x = 1 \mathrel{\rule[-0.2ex]{0.3em}{1.8ex}} \mathbf{skip}$ is a preconditioned program (Floyd assertion) that behaves like **skip** when $x = 1$, and otherwise **abort**s (does not terminate). We first observe that since the continuation $x = 1 \mathrel{\rule[-0.2ex]{0.3em}{1.8ex}} \mathbf{skip}$ is everywhere feasible, preference always chooses its first program $x := 1$ and the entire computation boils down to $x := 1$, provided that no infeasibility is encountered later on. (For the sake of simplicity, and without limiting generality, we assume the latter is the case.) Note that this is different from nondeterminism, since we can prove that

$$(x := 1 \sqcap x := 2) \; ; \; (x = 1 \mathrel{\rule[-0.2ex]{0.3em}{1.8ex}} \mathbf{skip})$$

is indeed equivalent to **abort**.

Let us now refine $x := 1$ in $x := 1 \gg x := 2$ by **magic**— the computation that is everywhere infeasible and thus the top of the refinement lattice of programs. As we already observed, the preference operator, just like nondeterministic choice,

[4] Strictly, it is enough to show that $x = 1 \Rightarrow \mathbf{fis}(U)$ universally holds.

[5] In the UTP theory of designs, this is equivalent to saying that healthiness condition **H4** holds for $S$.

cannot choose a miracle; hence, we have that **magic** $\gg x := 2$ must be equivalent to $x := 2$. However, the composition

$$x := 2 \;;\; (x = 1 \mathbin{\rule{2pt}{1.2em}} \textbf{skip})$$

can be shown to be **abort** (the bottom of the lattice). And **abort** is clearly not a refinement of $x := 1$ or, indeed, any other computation apart from itself. This exemplifies that preference is not monotonic in its first operand, given our standard intuitive notion of refinement.

It is moreover easy to see that $x := 1 \gg x := 2$ cannot be the same as $x := 2 \gg x := 1$, hence preference, unlike nondeterminism, is not commutative. A question that remains is whether we can express its semantics at all in the theory of wp predicate transformers, similar to other constructs of the guarded command language of Dijkstra [40]. Sadly, the answer is no, and to illustrate this let us recognise that both $x := 1$ and $x := 1 \gg x := 2$ ought be able to establish exactly the same postconditions $Q$, meaning that $\mathrm{wp}(x := 1, Q)$ and $\mathrm{wp}(x := 1 \gg x := 2, Q)$ are expected to be logically equivalent; but they are nevertheless algebraically distinct since they can be distinguished by a context $U \mathrel{\widehat{=}} x \neq 1 \longrightarrow \textbf{skip}$. That is $x := 1 \;;\; U$ is **magic** whereas $x := 1 \gg x := 2 \;;\; U$ is $x := 2$. The closest we get to preferential choice in a wp semantics is Nelson's biased choices, described in [122].

$$S \boxplus T \;=_{\mathbf{df}}\; S \sqcap \neg\, \textbf{fis}(S) \longrightarrow T$$

Unfortunately, this definition does not precisely capture our intuitive semantics of preference, since it is not sensitive to the continuation. For example, $x := 1 \boxplus x := 2$ evaluates to $x := 1 \sqcap \textbf{fis}(x := 1) \longrightarrow x := 2$ which simplifies to just $x := 1 \sqcap x := 2$, since assignment is universally feasible. In other words,

$$x := 1 \boxplus x := 2 \;;\; (x = 1 \mathbin{\rule{2pt}{1.2em}} \textbf{skip})$$

yields **abort** and not **skip**, as we would expect for preference. This makes biased choice less useful as a specification and implementation construct to capture preference. What we expect (see [43] for a discussion) is that preferential choice is sandwiched between nondeterministic choice and biased choice, pertaining to the following refinement relationship in a suitable semantics.

$$S \sqcap T \;\sqsubseteq\; S \gg T \;\sqsubseteq\; S \boxplus T$$

The endpoints of this refinement, we can express in wp semantics, but the middle point ($S \gg T$) not so. This raises the questions in what semantic framework the $S \gg T$ construct may be expressible. We answered this question in [43]: instead of using a semantics that is based on predicate transformers $\mathrm{wp}(S, Q)$, we us a semantics based on expression transformers $S \diamond E$.

Intuitively, expression transformers capture the value of an expression $E$ after execution of some computation $S$. Due to the fact that $S$ may be non-deterministic, miraculous or abortive, we require a more powerful language of expressions that, in effect, associates expressions with sets of possible outcomes.

Those sets may be empty (capturing the result of executing **magic**) or yield a special distinct value $\perp$ that denotes the outcome of the computation **abort**.

A bespoke mathematical structure to describe $S \diamond E$ turns out to be Hehner's notion of a *bunch* and the underlying bunch theory [68,69]. Bunch theory defines many properties that we expect, by default, to hold in a theory of expression transforms, and has already been used by Morris and Bunkenburg as a vehicle to develop their theory of term transformers [120]. We extended bunch theory in [153,187] to account for the improper bunch $\perp$, and thereby managed to give a model to the GCL in a semantics based on prospective values (pv semantics) that is fully isomorphic to conjunction computations, albeit not appropriate to capture angelic nondeterminism[6].

Not discussing our prospective-value semantics of the GCL in full detail, here is the definitional axiom for the preferential choice operator:

$$(S \gg T) \diamond E \;=_{\mathbf{df}}\; (S \diamond E)\;,\,(S \diamond E) = \mathbf{null} \rightarrow T \diamond E$$

To dissect the above notation, the comma corresponds to bunch union, and the short right arrow is a guarded form of bunch that evaluates to the empty bunch **null** when the guard is false. The predicate $S \diamond E = \mathbf{null}$ is a way to compute continuation-aware feasibility of $S$. We note that the continuation is captured by the expression $E$; we can see this more clearly by considering the expression transformer law for sequence[7]:

$$(S\;;\,T) \diamond E \;=_{\mathbf{df}}\; S \diamond (T \diamond E) \;=\; S \diamond F \quad \text{where} \quad F \mathrel{\widehat{=}} T \diamond E$$

where we can think of $T$ as the continuation of $S$ whose effect is captured by the expression $F$. So expression transformers together with improper[8] bunch theory provide a suitable abstraction and mathematical framework to formally capture preference, examine its algebraic properties, and prove various kinds of useful laws. (While we do not have space to indulge into this discussion here, details can be found in various papers [43,153,155].)

Bunch theory has sometimes been criticised on the grounds that its consistency is not apparent and thus cannot be taken for granted. Morris and Bunkenburg tried to alleviate this concern already in their publication [119] by providing a set-theoretic model. A more recent article by Stoddart et al. [153] addresses this problem too. Lastly, one might interject that bunch theory is as fundamental as set theory, and therefore may not require a model—just as we usually accept the axioms of ZFC set theory without further ado.

[6] Doing so may not be beyond the possibilities of a 'bunchy' semantics, but may require us to distinguish between different kinds of bunches: those that capture sets of demonic outcomes, and those that capture sets of angelic outcomes.

[7] In earlier publications, reviewers sometimes question whether the right hand of the axiom ought not be $T \diamond (S \diamond E)$. We reassure the reader that the law is indeed correct in the form given here, and refer to [186] for additional intuitive justifications.

[8] By this, we refer to Hehner's bunch theory, augmented with an improper (bottom) bunch $\perp$ for every type.

We share Jim Woodcock's curiosity and interest in examining the formalisation of theories of programming in various semantic frameworks and domains. While bunch theory gives us an elegant and concise formal model, it does make sense to explore preference in other semantic frameworks as well; and we shall comment on this in the subsequent paragraphs.

*Preference in wp* We already explained that a model of preference in conventional wp semantics is beset with difficulties and most likely unrealizable. However, there is a way to express preference and, more generally, the semantics of what we call preferential computations in a wp theory that uses three-valued logic. More specifically, Gödel logic [6] enables us to distinguish several degrees of truth and is a precursor of fuzzy logic. How can this be useful here?

Let us take inspiration again from the pv expression transformer semantics. Whether a computation $S$ and its continuation is feasible can be dynamically evaluated within the bunch logic via $(S \diamond E) \neq \mathbf{null}$; namely, the bunch of values that $E$ may take after executing $S$ is non-empty. In other words, there is at least one behaviour that gives us a prospective value for $E$. This enables us to effectively check whether the execution of $S$ succeeded or otherwise has failed due to infeasibility. It is a crucial bit of additional information that considers the continuation being captured implicitly in $E$.

The logical expression $\mathrm{wp}(S, Q)$ does not offer such a test: if, for some initial state, it yields true we have established $Q$ but we don't know how: both $\mathrm{wp}(x := 1, x = 1)$ and $\mathrm{wp}(\mathbf{magic}, x = 1)$ evaluate to true where the first one establishes $x = 1$ through a proper terminating behaviour, whereas the second one establishes $x = 1$ vacuously by a miracle. This distinction is lost in the term true; however, we can reintroduce it in a three-valued logic: true now means establishing the postcondition via non-miraculous behaviour, and supertrue means establishing it miraculously. We this have three truth values: false, true and supertrue. They are order in the way that we have enumerated them. We recall that in Gödel logic, such ordering determines the result of logical operators.

At this point, we ask the reader for a leap of faith. While it is out of scope for this publication to go into the details of such a wp theory built on top of a suitable three-valued logic, we note that we have developed a complete mechanisation[9] in Isabelle/HOL already to show that (a) such a theory can be defined, (b) that it supports the definition of all GCL operators including angelic choice, and (c) that it preserves all conventional laws of refinement if restricting ourselves to the core GCL operators. We call this theory a wpe semantics of guarded commands.

We moreover discovered that there is a neat Galois connection between monotonic wp and preferential wpe computations by approximating preference with nondeterminism, and that the resulting laws of this Galois connection can be used to recover a restricted form of compositional refinement in the absence of monotonicity of program operators with respect to refinement *in general*. The lose of monotonicity is a lamentable downside of our theory of preference, but we

[9] Unfortunately, we have not published this material yet but envisage to do so in the near future, via a separate publication and/or technical report.

could show that if preference is only introduced in the final step of refinement, we are entitled to replace any nondeterministic choice $S \sqcap T$ by a preferential choice $S \gg T$, irrespective of the absence of general monotonicity laws.

For computations that do *not* involve preference, our mechanised theory shows that refinement has precisely the same meaning as in the standard wp semantics of monotonic computations. Last but not least, we note that the work in [43] goes further by introducing a notion of feasibility-preserving refinement $S \sqsubseteq^* T$ which strengthen conventional refinement $S \sqsubseteq T$ with the caveat $\mathbf{fis}(S) \Rightarrow \mathbf{fis}(T)$ that provides a more useful basis for using (partial) miracles in implementations for backtracking algorithm implementations. Again, this causes monotonicity to break down: here, for sequential composition in the first operand. Nonetheless, the abovementioned replacement of $S \sqcap T$ by $S \gg T$ remains valid, even in this stronger refinement regime.

*Preference in UTP* Modelling preference in UTP is more challenging. While nondeterminism is naturally represented in UTP by predicates over before and after states, such as $x' = x - 1 \vee x' = x + 1$, preference (just like probability) gives rise to another structural dimension within the semantic space, with entanglement and closure properties between nondeterministic and preferential behaviours that have to be understood and made precise via healthiness conditions. A sensible strategy is to index computations with a natural number, i.e., by adding an auxiliary variable $idx$ in order to impose an explicit order on behaviours. E.g., $x := 1 \gg x := 2$ in the UTP theory of relations could be modelled as $idx = 1 \wedge x' = 1 \vee idx = 2 \wedge x' = 2$. More accurately, the $idx$ variable would need to be suitably typed in order to tabulate all choices made "along the way", and we would also need a variable $idx'$ to propagate the (updated) value of $idx$ through sequential compositions, similar to the $tr$ and $tr'$ variables in reactive computations. A notable open question is whether this will turn out sufficient for a compositional definition of sequential composition, and what the underlying healthiness conditions ought be. To account for unbounded nondeterminism[10], we would most likely have to move to some larger index type.

A denotational semantics may express preferential computations as sequences of functions (for deterministic programs) or relations (for nondeterminstic programs). Again, we have not developed such a model yet or identified any validity or closure constraints on the underlying mathematical structure. A UTP theory may likely enable us to easily derive a clean denotational model, since UTP typically encodes mathematical structures in a predicative manner, so the gap is inherently small between these approaches.

It is again not the ambition of this chapter to develop such a UTP theory, and the reason we mention it is that we believe it could be just the thing that Jim Woodcock may be excited about, with all the wonderful work he has done in producing novel theories of computation in UTP and pushing the UTP approach to and beyond its perceived limits. We flag this as a future work.

[10] Unbounded nondeterminism is a misnomer: often, we mean infinite albeit *bounded* nondeterminism, i.e., by some transfinite ordinal.

*Application of preference* So far, we confined our discussion mostly the semantic aspects of preference and the ability to mathematically describe it. A fair question is “What can it actually be used for?”. And “Is it worth the trouble?”. We believe there are a number of applications that still need to be explored, but one of them is to verify the implementation of heuristics in guiding backtracking search. While demonic nondeterminism is often used to express implementor’s choice, it is also well understood that nondeterminism exhibits backtracking[11] when combined with guarded computations. In this context, we think of nondeterministic choice as providing the possible moves (or steps) of an algorithmic search problem. An infeasible guard thus triggers backtracking from a dead end during search exploration.

The simple semantics of nondeterminism and guarded commands in wp promises a proof-economic technique to verify computations that make use of guards and choice in the way we hinted above, e.g. to solve puzzles like the Eight Queens or Knight’s Tour problem, or similar. The downside is that we have no control of the order in which choices are attempted. And implementing choice as preference is a refinement that resides outside the scope of formal reasoning. Preferential choice, as a semantic operator, lifts it into the realm of formal verification and thereby opens up the possibility of verifying algorithms that use preference as an implementation operator (which most algorithms do).

We concede that this deployment of choice and guards is not the only way to specify search procedures in formal languages: a dual method is to use angelic rather than demonic choice, and use **abort** as a means to trigger backtracking. This approach comes with its own issues: whether a computation fails to terminate and thus behaves like **abort** may be difficult to detect in practice[12], and, semantically, we may need to do so in a sufficiently expressive target language that supports iteration and recursion. Lastly, we may not require demonic or angelic choice either, and use a standard iterative or recursive sequential implementation instead to explore the search space via a stack.

Another application of preference relates to the example of a co-simulation master algorithm (MA) discussed later in Section 4.1. Though this is not further developed in this chapter, we believe that preference can be used to model a trial-and-error loop of the MA to steps a set of FMUs with step sizes that become progressively smaller, until all FMUs succeed to concurrently perform the simulation step. In order to reject a (too large) step size, an FMU can use a programming guard $\mathit{FMU}_i \mathrel{\widehat{=}} \mathit{stepsize} \leq \mathit{maxstep}_i \longrightarrow \mathit{FMU}_{beh}$. And the exploration of step sizes at the MA modelling level could be realised by a preferential choice as follows:

$$(\mathit{stepsize} := 4 \gg \mathit{stepsize} := 2 \gg \mathit{stepsize} := 1)\ ;(\mathit{FMU}_1 \mid\mid \mathit{FMU}_2 \mid\mid \cdots)$$

[11] More accurately, reversibility as all state is implicitly restored during backtracking.
[12] Note that **abort** is semantically equivalant to a non-terminating computation such as **while** *true* **do skip**.

where $FMU_i$ correspond to a model of the computation carried out by the Functional Mockup Units and the above parallel composition is a suitable version of parallel-by-merge, assuming that FMUs only modify their local disjoint states.

*Summary of preference* We conclude our mini exposition of preferential choice by observing that, despite its elusiveness to a simple semantic treatment, it turns out to be an operator that we believe has significant use in both specifications and implementations. We are hoping to publish further work in the near future that presents a wp(e) semantic model for preferential computations that has been fully formalised in Isabelle/HOL, with key laws and the aforementioned Galois connection being defined and verified. Development of a UTP model is a more long-term goal and we hope we sparked Jim's interest in this as well.

## 3.4 Probabilistic choice

Probabilistic algorithms [121] are algorithms that can make random choice at some points. They have been widely applied in many areas such as distributed systems (e.g. Rabin's mutual exclusion with bounded waiting [133], Aspnes et al.'s fast randomized consensus [5]), security and cryptography (public key protocols [42]), sorting (e.g. Hoare's randomized QuickSort [75]), number theory (e.g. Rabin's probabilistic primality testing algorithm [132]), data structuring (e.g. randomized hashing scheme [121]), and robotics (e.g. localisation, mapping, and planning [158]) to simplify algorithms and improve performance. These algorithms are classified as either *Las Vegas* (they always give correct results but the running time varies) or *Monte Carlo* (there is a small probability that they may give wrong results, but the running time is fixed).

To program probabilistic algorithms, conventional non-probabilistic programs are extended with a capability to model randomness, usually using a discrete probabilistic choice construct ($P \oplus_r Q$) [67, 104] or a random number generator (*rand*) [34, 66, 92] sampling from the uniform distribution over a set (either finite or infinite for discrete or continuous distributions). To capture high-level specifications for probabilistic programs or analyse them in the abstract level, nondeterministic choice is also introduced in probabilistic programming, such as *pGCL* [104,114], in which both nondeterministic choice ($P \sqcap Q$) and probabilistic choice are present.

The semantics for conventional programs such as weakest precondition [40], Hoare logic [76], and predicative programming [65] are boolean functions over state space. They can reason about these programs qualitatively, but not quantitatively which is natural in probabilistic programs. These semantics have been extended to deal with probabilistic programs. We mainly discuss the relational semantics and the predicate-transformer semantics for imperative sequential probabilistic programs, and briefly review probabilistic process algebras.

**Imperative - relational semantics** In Kozen's seminal papers [92,93], nondeterministic choice in conventional programs is replaced with probabilistic choice and boolean functions are generalised to real-valued functions over state spaces to give the semantics for probabilistic programs as partial measurable functions on a measurable space and continuous linear operators on a Banach space.

The pGCL has relational and predicate-transformer semantics by He, Morgan, and McIver [64]. The relational semantics extends the theory of designs for standard non-probabilistic programs in UTP to *probabilistic designs*. In the relational model, any probabilistic choice refines nondeterministic choice. Actually, the semantics of nondeterministic choice of two programs $P$ and $Q$ is the nondeterministic choice of the probabilistic choices of the embeddings of $P$ and $Q$ in all possible ways (all possible weights from 0 to 1 inclusive). The relational model embeds standard UTP designs in probabilistic designs through the weakest prespecification [80], a generalisation of wp from conditions (specified in only initial observation variables $v$) to relations (in both initial and final observation variables $v$ and $v'$). In probabilistic designs, distributions are captured in a probability distribution function *prob* of type $S \to [0, 1]$ over the observation state space $S$ and $\sum_{s \in S} prob(s) = 1$. Different from Kozen's semantics and the Weakest Pre-Expectation (WPE) [104,115] where probabilistic programs are real-valued functions over state space, the probabilistic programs in the relational model are probabilistic (non-homogeneous) designs relating initial observation state space $S$ and final probabilistic state space on *prob*.

Woodcock [173] became interested in this relational semantics when he considered the probabilistic semantics for RoboChart [110] because of its UTP semantics and algebraic properties which are suitable for reasoning and automation in Isabelle/UTP. He formalised the semantics of the relational model and the proof that the semantic embedding is a homomorphism on the structure of standard programs, probabilistic choice, nondeterministic choice, and sequential composition. He also found interesting details like a law requiring a side condition [173] of the H3 healthiness condition [79, 171] and the lifting law for sequential composition requiring finite state space [180]. The formalisation resulted in the mechanisation of probabilistic designs in Isabelle/UTP [46] by Ye, Woodcock, and Foster [180].

A uniform distribution algorithm modelled in RoboChart using a binary probabilistic choice is proved in [180]. Ye, Woodcock, and Foster [180] found that the most difficult part in proving probabilistic programs using probabilistic designs is the supplement of a witness $Q$ for the existential quantification in [180, Definition 8.2] used in sequential composition of two probabilistic designs in [180, Theorem 8.4]. This is due to a fact that probability distributions are encoded in the variable *prob* which is inside relations. In this view, *probabilistic programs are relational in terms of probability distributions over state space*, and so relations or predicates deal with distributions. To address this challenge, Woodcock and Ye started to look at other approaches with different views of probabilistic programs. They should also be suitable for the mechanisation in Isabelle/UTP.

Hehner's Probabilistic Predicative Programming (PPP) [66,67] takes another view. PPP generalises boolean functions for predicative programming to real-valued functions. In PPP, *conditional* and *joint* probability are modelled through sequential and parallel composition. One unique feature of the language is its capability to model both epistemic uncertainty (due to the lack of knowledge of information ) using the subjective Bayesian approach and aleatoric uncertainty (due to the natural randomness of physical processes). In PPP, *programs are probabilistic (real-valued functions) and relations are over state space*, and so relations only deal with usual state space, not distributions over state space like the relational semantics. This interested Woodcock because the relations over usual state space can be formalised using UTP's alphabetised relations, and so PPP can be mechanised in Isabelle/UTP.

There are, however, several obstacles to the broader adoption of Hehner's work, such as informal syntax and semantics, and simple semantics for probabilistic loops but that is difficult to mechanise. For these reasons, Woodcock introduced an Iverson bracket notation $[\![P]\!]$ [183] (of type $S \to \mathbb{R}$), which maps a predicate $P$ (of type $S \to \mathbb{B}$) into real numbers. This also separates relations from arithmetic. Woodcock replaced PPP's relations with UTP's alphabetised relations, which allows UTP theories on relations and their mechanisation in Isabelle/UTP reused.

Ye, Woodcock, and Foster [183] formalised the syntax and semantics of this new probabilistic framework, called *Probabilistic Unifying Relations* (ProbURel), and introduced the constructive semantics for probabilistic loops using Kleene's fixed-point theorem and the unique fixed-point theorem to simplify the reasoning about probabilistic loops. ProbURel is mechanised in Isabelle/UTP, leveraging the recent mechanisation of the Z mathematical toolkit[13] in Isabelle/UTP by Foster and Woodcock.

PPP does not include nondeterminism. In [66], Hehner discussed informally an approach to reason about nondeterministic choice. Nondeterministic choice is equivalent to an existentially quantified deterministic choice.

$$P \sqcap Q = \exists\, b : \mathbb{B} \bullet \textbf{if } b \textbf{ then } P \textbf{ else } Q$$

Each nondeterministic choice is replaced by such a quantified conditional choice with a new fresh quantifier variable $b$. The strategy is to move all quantifiers outwards. Based on this definition, nondeterministic choice after a probabilistic choice, such as $P \oplus_r Q;\; P \sqcap Q$, is oblivious (that is, making a choice without looking at the current (or past) state).

Similarly to PPP, ProbURel does not support nondeterministic choice and continuous distributions. Woodcock and Ye are interested in extending ProbURel in these aspects.

**Imperative - predicate-transformer semantics** He, Morgan, and McIver's predicate-transformer semantics [64] for pGCL uses the same weakest prespecification [80] technique to inject standard predicates in Dijkstra's predicate-transformer semantics (the weakest pre-condition) [40] into probabilistic predicates. Based on this injection, probabilistic predicate transformer is an embedding of standard predicate transformer. However, for the constructs that are not

[13] https://github.com/isabelle-utp/Z_Toolkit.

standard, such as probabilistic choice, an additional healthiness condition *continuity* is imposed on the probabilistic transformer. Monotonicity and "scaling" can be derived from continuity. They showed the embedding preserves program structure. Interestingly, the embedding of demonic non-deterministic choice (conjunction in standard predicate transformer) becomes *minimum* for probabilistic transformer. The key healthiness condition for this transformer semantics is sublinearity [105].

McIver and Morgan gave WPE or expectation transformer semantics [104, 115] to pGCL. In WPE, the real-valued functions or expressions over state space are called expectations (indeed random variables). So a probabilistic program maps a post-expectation (postE) to a pre-expectation (preE). Indeed, its WPE semantics defines the greatest preE (gp) for a postE: $preE = gp(P, [postE])$ where the square bracket [_] converts a boolean-valued predicate to an arithmetic value, especially $[true] = 1$ and $[false] = 0$. The semantics for probabilistic choice are the standard weighted average of pre-expectations of its all alternatives. The semantics for nondeterministic choice, however, are the *smaller* pre-expectation (that is, the demonic behaviour against the greatest pre-expectation) in the pre-expectations of its two alternatives. We show below an example of gp for a probabilistic choice.

$$\begin{aligned}
&gp(x := x + 1 \oplus_{(2/3)} x := x - 1, [x \geq 0]) \\
=&(1/3) * [x + 1 \geq 0] + (2/3) * [x - 1 \geq 0] \\
=&(1/3) * [x \geq -1] + (2/3) * [x \geq 1] \\
=&(1/3) * [x = -1 \vee x = 0] + [x \geq 1]
\end{aligned}$$

It means that in order for the program to establish $x \geq 0$, the probability of $x$ being $-1$ or $0$ (or $x \geq 1$, or $x < -1$) in its initial state is at least $1/3$ (or 1, or 0). McIver and Morgan [106] also developed refinement for probabilistic programs (pGCL) to ensure correctness by construction.

McIver and Morgan [102] extended expectation transformer from finite state spaces to infinite state spaces, and constructed deterministic, demonic, demonic and angelic probabilistic transformers sequentially (characterised by transformer properties linearity, sublinearity, semi-sublinearity respectively), as Back and von Wright [7] did for non-probabilistic transformers (characterised by transformer properties disjunctivity and conjunctivity, conjunctivity, monotonicity respectively). [102, Fig. 9] illustrates the transformer structure.

WPE attracted the interest of many. Kaminski [88] developed an advanced weakest precondition calculus. McIver et al. [107] presented additional proof rules for loops and loop termination. Further three advancements: expected runtimes, conditioning [128], and mixed-sign expectations were developed and presented in Kaminski's PhD thesis. Schröer et al. [145] developed a deductive verification infrastructure for discrete probabilistic programs to use SMT solvers to verify programs automatically. Batz et al. [11] proposed an approach to use inductive synthesis to find loop invariants for probabilistic programs from templates. Batz et al. [10] used deductive verification technique [145] to synthesise nondeterministic probabilistic programs to find memoryless and deterministic strategies.

Stoddart and Zeyda et al. [153, 154] proposed a guarded command language that is able to model nondeterministic choice, preferential choice, and probabilistic choice. The semantics of the language is based on wp and conjugate weakest precondition (cwp), and interpreted in bunch theory [69], instead of set theory. They use $S \diamond E$ to denote the prospective values of an expression $E$ after the execution of a program $S$. A basic law is developed to link prospective values to cwp. The PV semantics is shown to represent backtracking and the implementation of backtracking via reversible computing [186]. Their semantics for probabilistic choice is a weighted choice of a bunch of three terms, which takes into account the interaction of probabilistic with nondeterministic choice, and the feasibility/continuation and abortion behaviour of probabilistic choice.

In addition to Kozen's, relational, and predicate transformer semantics, many other semantics for imperative probabilistic programming exist. Dahlqvist et al.'s simple imperative probabilistic language [34] uses two constructs, $coin()$ and $rand()$, to introduce discrete and continuous uniform distributions, and both operational and denotational semantics are presented. Hoare logic has also been used to reason about probabilistic programs [9, 26, 37, 135, 136].

**Probabilistic process algebras** Woodcock's interest in probabilistic programming is not restricted to sequential imperative programs. Indeed, he is interested in probabilistic programming based on CSP and *Circus*, and UTP in general, and how it can be used to give semantics to RoboChart. We review different probabilistic models in process algebras, particularly in (1) how to model probability, (2) how to interact with their environment, and (3) how probabilistic choice interacts with nondeterministic choice. Then we discuss the approach that RoboChart uses for probabilistic modelling.

Hansson's *alternating model* [62] distinguishes between nondeterministic and probabilistic choice. Either a nondeterministic choice or a probabilistic choice can be made in each state of this model, and the order of availability of these choices is strictly alternating between a nondeterministic and a probabilistic choice.

In Segala and Lynch's *probabilistic automata* [147], transitions are labelled with probability values. The transition relation is a *steps* function which maps source states to probability distributions over (action, target state) pairs. Actions can be external, modelling interactions with the environment through events, or internal, modelling computation steps through internal events $\tau$. In probabilistic automata, a distribution from a source state may contain more than one action. If all distributions in a probabilistic automaton contain only one action, this automaton is called *simple*. Otherwise, it is called *general*. If from each source state, there is at most one step (or one distribution) enabled, this automaton is called *fully probabilistic*.

Van Glabbeek et al. [162]'s *reactive*, *generative*, and *stratified* models are the different ways to interpret probabilities [98] in processes. Multiple actions can be offered by a process to its environment, but the environment is allowed to choose only one in the reactive model or some in the generative and stratified models. After actions are chosen by the environment, the process makes an internal transition based on the current state and (1) the probability distribution associated with the action in the reactive model, or (2) the globally (or locally) redistributed probability distribution for chosen actions in the generative (or the stratified) model. The reactive model corresponds to the simple probabilistic automaton model without internal nondeterministic choice involving the same actions, in that both allow external nondeterministic choice between different actions (that is, multiple transitions with different external actions from the same state). Interestingly, zero probabilities are permitted in the stratified model to model process priorities.

There are many probability extensions based on: CSP [55, 58, 94, 97, 113, 116, 124, 148, 157], CCS [56, 61, 162, 184], and ACP [2], probabilistic transition systems [15, 86, 96], and automata [63, 177]. We particularly review CSP-based extensions because CSP supports refinement and has its semantics in UTP.

Seidel [148] proposed two semantics models for probabilistic CSP: the independent model with external and internal choice and the conditional model without internal choice and hiding.

Lowe [97] presented two languages as refinements of Timed CSP [35, 143]: a fully deterministic model (DTCSP) with a notion of priority (the prioritized model) and a probabilistic model (PDTCSP) extended from DTCSP. The two models feature biased external choice, parallel composition, and interleaving. They have no internal nondeterministic choice though PDTCSP has a (internal) probabilistic choice.

Gómez et al.'s probabilistic variant [58] of CSP has no internal nondeterministic choice but has two versions of probabilistic choice: one generative and one reactive to replace internal and external choice in CSP. Similarly, Núñez et al.'s PPA [124] has similar two versions of probabilistic choice.

Morgan and McIver et al.'s PCSP [113,116] uses Jones's general construction to extend CSP with probabilistic choice. In PCSP, probabilistic choice distributes through all other operators, including external choice and internal choice. But internal choice is not idempotent in PCSP. Morgan [113] linked probabilistic action systems (written in pGCL with expectation transformer semantics) and probabilistic CSP, gave traces, failures, and divergences from the probabilistic CSP to the probabilistic action systems.

Mislove's [109] proposed an approach to consider the family of probability convex sets in three possible power domains: lower, upper, and convex to provide a probabilistic extension of CSP (Morgan's PCSP) with both nondeterminism choice and probabilistic choice, importantly, all the laws for them are valid, including the idempotent law for nondeterministic choice.

Georgievska and Andova [55] presented a probabilistic extension of CSP which preserves the distributivity laws and the idempotent law for internal nondeterministic choice, via restricted schedulers. This extension supports internal nondeterministic choice, external choice, probabilistic choice, and parallel composition with hiding.

Sun et al. [157] proposed PCSP#, a probability extension of CSP# which combines high-level modelling operators with low-level procedural codes. The semantics of PCSP# is Markov Decision Processes (MDPs) [130]. The verification of PCSP# programs is supported by the Process Analysis Toolkit (PAT) [156].

**Semantics for RoboChart** Because RoboChart's standard semantics is based on CSP and tock-CSP [12], Woodcock showed interest in probabilistic extensions of CSP, discussed previously, to give RoboChart a probabilistic semantics. However, the main issue is the lack of tool support. For this reason, inspired by Jansen et al.'s P-statecharts [83], Woodcock and Ye et al. [179] gave RoboChart a semantics based on the PRISM language [95]. RoboChart adopts the same alternation as [62] between nondeterministic and probabilistic choice, but a probabilistic choice is made only within a transition originating from a nondeterministic state. The PRISM semantics for RoboChart, however, is subject to the state space explosion problem, as discussed in [182] by Ye and Woodcock. Because of these challenges, Woodcock is also interested in the UTP semantics for probabilistic process algebras and its mechanisation in Isabelle/UTP to verify them using theorem proving. However, there are some important questions left to be answered, such as "Out of the existing and possible extensions of CSP to cover probability, which one should be adopted?", "should probability be encoded in traces, or should traces be probabilistic?", etc.

In [181], Ye et al. gave RoboChart operational semantics based on the mechanised CSP and *Circus* [49] in Isabelle/UTP using interaction trees (ITrees) [178], and used priority-based hiding and renaming operators to resolve nondeterminism. Ye, Woodcock, and Foster are interested in extending CSP and *Circus* with probabilistic choice to have sound animation for probabilistic RoboChart models.

## 4 Applications

In this section, we explore the application of the various choice operators to co-simulation, the semantics of RoboChart, and the modelling of random walks.

### 4.1 Co-simulation for cyber-physical systems

Simulation techniques are widely adopted in the development of cyber-physical systems (CPS). Typically, their components may be developed using different techniques. To simulate a complete system, co-simulation can be used where components are encapsulated as simulation units (SUs) and a master algorithm orchestrates the simulation by exchanging values between SUs at sample times.

Co-simulation has advantages, in that stakeholders can protect their intellectual property by conforming to an API, such as FMI [14]. However, the master algorithm needs to handle SUs that may reject a simulation step if the chosen time step is too large. Backtracking algorithms have been proposed [33] that can perform step revision. An example of using FMI is reproduced [33] in Figure 2,

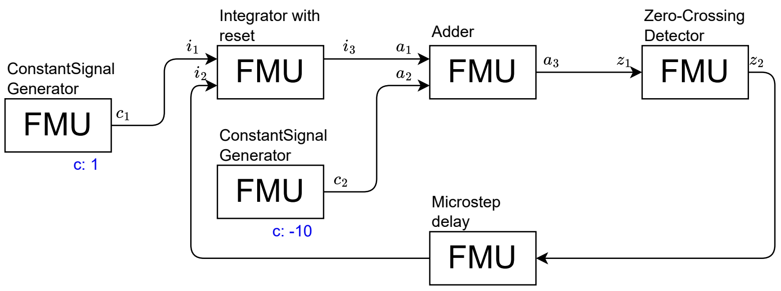


**Fig. 2.** Example of FMI setup with 6 connected FMUs.

consisting of 6 connected SUs, known as FMUs: constant signal generators, one integrator, an adder, a microstep delay, and a zero-crossing detector (ZCD).

The verification of co-simulation algorithms is a topic that Woodcock and his collaborators approached in [60], where they consider an UPPAAL encoding, however, the step revision procedure is modelled explicitly. The question is whether a formal specification can be constructed using abstract choice operators.

**Process algebraic model** In what follows, we sketch a process algebraic specification for execution of SUs in FMI with step revision using the *Circus* notation. It is loosely based on the architecture in [24] and follows the constraints of [60]. The communications between SUs are modelled using the following channels:

$$Value ::= absent \mid value\langle\!\langle \mathbb{R} \rangle\!\rangle \qquad \mathbf{channel}\ get : OUTPUT \times Value$$
$$\mathbf{channel}\ doStep : SU \times \mathbb{T} \times \mathbb{T} \qquad \mathbf{channel}\ set : INPUT \times Value$$

*get* to obtain the output of a SU, and *set* to provide an input, respectively, and *doStep* to step the simulation of a SU. *OUTPUT* and *INPUT* are types labeling the inputs and outputs of each SU, so that the connections between SUs can be modelled, and *Value* is the type of values admissible for communications, a real value or *absent*. *doStep* contains three values, the first drawn from *SU*, a set of SU identifiers, the second is the requested step size, and the third the actual step size accepted, over a time domain $\mathbb{T}$. We observe that according to the FMI2 standard it is possible that a SU may not report an accepted step size at all, and just yield an error. In this model, we omit this detail without loss of generality.

*Connections* Input and output labels are associated with SUs via total functions while the connections between them are defined by a total surjection $L$ from *INPUT* to *OUTPUT*. From $L$, we define a process *Exchange* that captures the valid interactions for FMUs, via synchronisation on *get*, *set* and *doStep*.

$$\mathbf{process}\ Exchange \mathrel{\widehat{=}} \mathop{|||}\ x : L \bullet SetC(first(x), second(x))$$

It is defined by an iterated interleaving over pairs $x$, drawn from $L$, of processes *SetC*, omitted here and parameterised by input and output labels, that define when an output is available given its dependency on an input, and whether a SU has been stepped forward in time, following the criteria defined in [60].

Then, given a process algebraic model of all SUs, captured by a process of the same name and whose definition we omit here, we can capture all possible interactions, by taking into account their connections, via a parallel composition with *Exchange* synchronising on *doStep*, *get* and *set*, defined as follows.

$$\mathbf{process}\ SUExchange \mathrel{\widehat=} SUs \llbracket \{\!| doStep, get, set |\!\} \rrbracket\ Exchange$$

This process captures all possible simulations, including those that may not step through by the requested step size. A valid simulation is therefore defined by

$$\mathbf{process}\ \ Simulation \mathrel{\widehat=} SUExchange \llbracket \{\!| doStep |\!\} \rrbracket\ ChooseStep$$

where *ChooseStep* captures the valid steps, that is, where *doStep* is observed with requested and accepted simulation step sizes that are the same. We consider an approach where angelic choice is used to choose a step size, before a prefixing on $doStep.u.t_r.t_a$, for every SU $u$. This is sketched in the following *Circus* process.

$$\begin{array}{l}
\mathbf{process}\ \ AngelicStep \mathrel{\widehat=} \mathbf{begin\ state}\ \ S == [h : \mathbb{T}_1] \\
\quad DoStep \mathrel{\widehat=} su : SU \bullet doStep!su!h?t \to \{h = t\} \\
\quad Step \mathrel{\widehat=} \mathop{|||}\ su : SU \bullet DoStep(su) \\
\quad Main \mathrel{\widehat=} h : \mathbf{guess}(\mathbb{T}_1)\ ;\ Step\ ;\ Main \\
\bullet\ Main \\
\mathbf{end}
\end{array}$$

It has a state variable $h$ and its behaviour is defined by the *Main* action, that angelically **guess**es a step size $h$ from a finite domain $\mathbb{T}_1$, and then behaves as *Step* followed by the recursion. *Step* is defined by the iterated interleaving over $su$, drawn from the set SUs, of actions $DoStep(su)$. Each action has a prefixing on $doStep!su!h?t$, that uses the chosen step $h$ and is prepared to synchronise on any accepted size $t$, and afterwards there is an assertion ($\{h = t\}$). If it fails, then the action aborts, including up to the point in *Main* where $h$ was chosen. While this captures all valid step sizes, it does not require $h$ to be maximal.

*Maximum angelic step* To address the shortcomings of the previous solution, we consider a modified version of the *AngelicStep* using both angelic and demonic choice. We assume that a SU that accepts a step size $h$ may also accept a step size $h'$, such that $h' \leq h$. Intuitively, to determine the maximum admitted step size, we need “to experiment” with stepping a SU with values above and below an angelically chosen maximum step size, such that step sizes less than or equal to the maximum suceed, but values above do not. Crucially, failed experiments should be pruned from the model. We frame this in a game-like way as follows.

$$
\begin{array}{l}
\textbf{process}\ \ \mathit{MaxAngelicStep} \mathrel{\widehat=} \textbf{begin state}\ \ S == [h : \mathbb{T}_1;\ \mathit{real} : \mathbb{B}] \\
\quad \mathit{DoStep} \mathrel{\widehat=} su : SU \bullet ([\mathit{real} = \textbf{True}]\ ;\ \mathit{Stop}) \\
\qquad\qquad\qquad\qquad\qquad \Box\ \mathit{doStep}!su!h?t \rightarrow \{\mathit{real} = \textbf{True} \Leftrightarrow h = t\} \\
\quad \mathit{Step} \mathrel{\widehat=} \big|\big|\big|\ su : SU \bullet \mathit{DoStep}(su) \\
\quad \mathit{Main} \mathrel{\widehat=} h : \textbf{guess}(\mathbb{T}_1); \\
\qquad\qquad h, \mathit{real} : [\mathit{true}, (h' \leq h \wedge \mathit{real}' = \textbf{True}) \vee (h' > h \wedge \mathit{real}' = \textbf{False})]; \\
\qquad\qquad \mathit{Step}\ ;\ [\mathit{real} = \textbf{True}]\ ;\ \mathit{Main} \\
\bullet\ \mathit{Main} \\
\textbf{end}
\end{array}
$$

Differently from before, *MaxAngelicStep* has an additional boolean variable *real*, that is used to differentiate between an actual stepping, so it is **True**, and otherwise when an "experiment" with a larger step size is conducted it is **False**.

As before, the *Main* action begins by **guess**ing the step size $h$. Afterwards, we have a specification statement, whereby the value of $h$ is demonically changed, such that, for lower values the variable *real* is **True**, and for higher values it is **False**. This is followed by a composition with a revised *Step* action and an assumption ($[\mathit{real} = \textbf{True}]$), followed by the recursion. In *DoStep* there is now an external choice between two processes: the first has an assumption that *real* is **True**, in which case it behaves as *Stop*, and otherwise is miraculous; the second has a prefix on *doStep* followed by an assertion that requires *real* to be **True** if, and only if, the step succeeds with value $h$. We consider all possible cases:

1. $\mathit{real} = \textbf{True} \wedge h \neq t$: the assertion fails as before, and so the process *Step* aborts given that $h$ is not an admissible step size;
2. $\mathit{real} = \textbf{True} \wedge h = t$: the assertion succeeds and so do both assumptions, so $h$ is an admissible step size;
3. $\mathit{real} = \textbf{False} \wedge h \neq t$: the assertion succeeds, but both assumptions fail. Therefore the external choice becomes equal to $\top \Box \mathit{doStep} \rightarrow ...$, where $\top$ is miracle, which prunes waiting behaviours from the process, that is, the pericondition. When this behaviour is propagated to *Main*, as *Step*, its composition with the subsequent assumption $[\mathit{real} = \textbf{True}]$ ensures that overall that behaviour is pruned, given that a failed assumption is miraculous.
4. $\mathit{real} = \textbf{False} \wedge h = t$: both the assertion and the assumptions fail. This corresponds to an experiment for which a larger step size is admissible, but because of the failed assertions and assumptions the behaviour is also pruned.

The above is but a sketch of a specification for a master algorithm with step revision. Further work is required to establish the laws that would allow introducing the specific mechanisms for step revision in a refinement.

## 4.2 RoboChart

RoboChart [110] is a diagrammatic modelling language designed to support the specification of robotic software in terms of reactive state machines. These state machines can be combined in parallel to form controllers, which can be composed in parallel to describe the overall software. State machines are formed of transitions, states, and junctions. Transitions represent changes in a system's configuration; they connect two nodes (states and junctions), can be triggered by events and guarded by conditions, and can cause actions to be executed.

States model stable decision points, that is, configurations of the software in which the machine is allowed, potentially indefinitely, to wait for a particular input. Junctions, on the other hand, are not stable; they allow for decisions to be made but do not allow the system to wait indefinitely.

The semantics of RoboChart is defined in terms of *Circus*/CSP processes and supports formal analysis via model-checking and theorem proving. Several well-formedness conditions are necessary to adequately specify the semantics of RoboChart models. Of particular interest is the requirement that the guards of transitions leaving the same junction form a cover (the disjunction of the guards is true). This requirement guarantees that if a junction is reached, there is always one transition whose guard is true, and the system does not get stuck.

There are a number of reasons why this well-formedness condition might be inconvenient. For example, the particular path taken to a junction could guarantee that one of the outgoing transitions is always enabled. Alternatively, a modeller may wish to have simpler models (fewer transitions) and rely on an interpretation in which a transition from a state is only taken if there is a path to another state in which all the transitions are enabled.

Angelic choice discussed in Section 3.2 is a suitable alternative to specify the semantics of RoboChart models, in which the well-formedness condition above is relaxed and where the previously discussed interpretation is adopted. The standard semantics of transitions leaving a state can be specified as the external choice of the events that trigger the transitions. In order to adapt this semantics, we must first replace the external choice (in the selection of transitions) with angelic choice and adopt the theory described in [142]. Finally, we must also encode the omitted guards as Chaos. This guarantees that if a junction is reached in which none of the guards of the outgoing transitions is true, the process would backtrack, and an alternative path would be taken.

While such an approach has the potential for more succinct diagrams, it presents practical and theoretical challenges for verification due to the lack of tool support around angelic choice and the extra complexity in analysing the possible paths. The standard semantics balances specification power and automation, with a well-formedness condition that, while not amenable to static analysis, can be checked locally using automated theorem provers.

### 4.3 Random walks

Grimmett and Welsh [59] defined various random walks (RW). A RW is *simple* if at each time step it can move only to its next (or neighbouring) positions randomly in one of the lattice directions. A *symmetric* RW has the equal probability for each direction. Otherwise, it is *asymmetric*.

According to Pólya's recurrence theorem [131], a symmetric random walk is recurrent (the probability of revisiting its starting point is 1) only if it is one- or two-dimensional. This simple random walk does not terminate. For this reason, researchers [29,81,89,103,107] in probabilistic programming and verification are more interested in a variant of the one-dimensional simple random walk (SRW). It is also the Gambler's Ruin Problem with an absorbing barrier at 0. We can model it as a probabilistic program below.

$$x := m;\ \text{while}(x > 0)\{x := x - 1 \oplus_p x := x + 1\} \tag{PSRW}$$

where $x$ is an integer variable, $m$ denotes the starting position and $m > 0$, and $p$ is the probability of decreasing $x$ by 1.

Grimmett and Welsh [59] proved the termination probability for SRW is $(p/q)^m$ if $q > p$ where $q = 1 - p$, and 1 otherwise. This program is proved to have its termination probability equal to 1 (that is, almost-sure termination (AST)) if $p$ is equal to 1/2 (so symmetric, as SSRW), according to [29,81,103,107], using the invariant and variant reasoning technique for probabilistic loops. McIver et al. [107] and Chatterjee et al. [29]'s variant rules for loops are based on *supermartingale* [165], a sequence of random variables (RVs) for which the expected value of the current random variable is larger than or equal to that of the subsequent random variable. Moosbrugger et al. [111] developed the tool Amber to automatically prove the one-dimensional SSRW is AST. Furthermore, McIver et al. [107] proved the two-dimensional SSRW terminates almost-surely. Kaminski [89] proposed a wp-style reasoning for bounded expected runtimes and proved the expected runtime of SSRW is infinite.

These studies show the verification of AST for probabilistic loop programs. They, however, have not inferred the exact value of probabilities (either distribution or sub-distribution depending on $p$ and $m$) and expected runtimes in terms of steps or iterations, or have not automated the inference, which are important for the evaluation of the correctness and performance of probabilistic algorithms (both Las Vegas and Monte Carlo). ProbURel [183] uses a unique (Kleene's) fixed-point theorem to give the semantics to probabilistic loops, which makes it able to infer the exact value of probabilities and expected runtimes. The most difficult part using this theorem is to construct a fixed point *fp*, or the invariant for the loop. As shown in [183], the exact value of probabilities and expected runtimes of two loop examples (flip a coin till a heads and throw a pair of dices till they have the same outcome) are proved and mechanised in Isabelle/UTP.

Here, we sketch a strategy for the exact inference (to explicitly represent the probability distribution or sub-distribution of PSRW in terms of iterations, parametric in $m$ and $p$) of this SRW using mathematics and then use probabilistic programming to give the semantics by constructing a fixed point or invariant for the loop in PSRW. Mathematical calculation is not used in programming but is used for comparison.

**Mathematics** We define $X_i$ (where $i \in \mathbb{N}$) for the $i$th toss of the coin or the $i$th step of the moves, and

$$\mathbb{P}\{X_i = 1\} = q \qquad \mathbb{P}\{X_i = -1\} = p$$

where both $p$ and $q$ are non-negative real numbers and $p + q = 1$.

We use $S_n^m$ to denote the current position of the walker (when starting at position $m$) after $n$ time steps, so $S_n^m = m + \sum_{i=0}^n X_i$. Then the minimum number of time steps the walker reaches 0 from $m$ is defined below.

$$\sigma_0(m) = min(\{n : \mathbb{N} \mid S_n^m = 0\})$$

So $\sigma_0(1)$ is the minimum number of time steps the walker reaches 0 from 1. We omit the subscript 0 here for simplicity. What we are interested in is the distribution in terms of $\sigma(m)$ or the number of iterations in the PSRW program.

From $m$, to reach the position 0, the walker needs to reach $m-1$ first, then $m-2$ etc. We note that $X_i$ is independent and the distribution of the walker reaching $m-1$ from $m$ is exactly the same as reaching 0 from 1. We can treat $\sigma(m)$ as the summation of $m$ copies of $\sigma(1)$. Therefore, $\sigma(m) = \sum_{i=1}^m \sigma(1)$. So our question is to find out the distribution in terms of the discrete random variable $\sigma(1)$: $\mathbb{P}\{\sigma(1) = n\}$, denoted as $\varphi(n)$. The probability generation function [59] of $\sigma(1)$ is

$$G_{\sigma(1)}(z) = \mathbb{E}\left(z^{\sigma(1)}\right) = \sum_{n=0}^{\infty} \mathbb{P}\{\sigma(1) = n\} z^n = \sum_{n=0}^{\infty} \varphi(n) z^n \tag{G1}$$

Then according to the theorem [59] that the generation function for the sum of independent random variables is the product of those for each independent random variable,

$$G_{\sigma(m)}(z) = G_{\sum_{i=0}^m \sigma(1)}(z) = \left(G_{\sigma(1)}\right)^m = \left(\sum_{n=0}^{\infty} \varphi(n) z^n\right)^m \tag{Gm}$$

So our question now is to determine $\varphi(n)$. First, we can get $\mathbb{E}(z^{\sigma(1)})$ by solving an equation which is formed by conditioning the first move of the walker, as described in [159]. The result is shown below.

$$G_{\sigma(1)}(z) = \mathbb{E}(z^{\sigma(1)}) = \frac{1 - \sqrt{1 - 4pqz^2}}{2qz} \tag{G1r}$$

The sum of probabilities is just the generation function when $z = 1$, so

$$G_{\sigma(1)}(1) = \frac{1 - \sqrt{1 - 4pq}}{2q} = \begin{cases} p/q & \text{if } p < q \\ 1 & \text{if } p \geq q \end{cases}$$

SRW for $m = 1$ (that is, $\sigma(1)$) is AST only when $p \geq q$. The Taylor expansion of (G1r) gives the power series below.

$$\frac{1 - \sqrt{1 - 4pqz^2}}{2qz} = \sum_{n=1}^{\infty} \frac{(2n-3)!!2^{n-1}}{n!} p^n q^{n-1} z^{2n-1}$$

where !! denotes the double factorial. Particularly, $(-1)!! = 1$. We use $a(i)$ to denote the coefficient in the power series for $z^i$, and so

$$a(2n-1) = \frac{(2n-3)!!2^{n-1}}{n!} p^n q^{n-1}$$

We note that the corresponding $a(2*n)$ to $z^{2n}$ is 0. Let $i = 2n-1$, so $n = \frac{i+1}{2}$.

$$a(i) = \frac{(i-2)!!2^{\frac{i-1}{2}}}{(\frac{i+1}{2})!} p^{\frac{i+1}{2}} q^{\frac{i-1}{2}}$$

We can rewrite this power series as follows using the Iverson bracket notation defined in [183].

$$\sum_{n=1}^{\infty} a(2n-1)z^{2n-1} = \sum_{n=0}^{\infty} [\![ n\%2 = 1 ]\!] a(n) z^n \tag{G1pgf}$$

where $[\![ n\%2 = 1 ]\!]$ is 0 for all $n$ that is even. According to (G1), now $\varphi(n) = [\![ n\%2 = 1 ]\!] a(n)$. Now we expand the $m$th power in (Gm),

$$\left(\sum_{n=0}^{\infty} \varphi(n) z^n\right)^m = \sum_{n=0}^{\infty} \left( \overbrace{\underbrace{\sum_{i_1=0}^{n} \sum_{i_2=0}^{i_1} \dots \sum_{i_{m-1}=0}^{i_{m-2}}}_{m-1} \varphi(n-i_1)\varphi(i_1-i_2)\dots\varphi(i_{m-1})}^{\mu(m,n)} \right) z^n$$

Finally, $\mu(m,n)$ above gives the probability of SRW reaching 0 after exact $n$ steps, when starting from $m$.

**Probabilistic programming** To reason about expected runtimes, as shown in [183], an additional variable $t$ of type natural numbers is introduced in PSRW to count the iterations.

$$x := m;\ t := 0;\ \text{while}(x > 0)\{x := x-1 \oplus_p x := x+1;\ t := t+1\} \quad \text{(TPSRW)}$$

Unlike the other two verified loop examples (flip a coin and throw a pair of dices) in [183] whose every experiment or iteration is independent of the previous iterations, TPSRW's each iteration is related to all previous iterations because the value of $x$ is updated in each step. For this reason, its loop invariant or the fixed point does not share the same pattern: $p_f^{(n-1)} * p_s$ where $p_f$ (or $p_s$) is the probability of non-termination (or termination) for each experiment.

However, we observe that $\mu(m,n) = p*\mu(m-1, n-1) + q*\mu(m+1, n-1)$,[14] that is, the termination from $m$ in $n$ steps is only through two alternative ways: move to the left (with probability $p$) and then terminate from $m-1$ in $n-1$ steps, or move to the right (with probability $q$) and then terminate from $m+1$ in $n-1$ steps. This observation helps us to define the fixed point of the loop.

$$\begin{aligned} Ht \mathrel{\widehat{=}} & [\![ \neg\, x > 0 ]\!] * [\![ x' = x ]\!] * [\![ t' = t ]\!] + [\![ x > 0 ]\!] * [\![ x' = 0 ]\!] * \\ & \left( \begin{array}{l} [\![ t' - t \geq x ]\!] * [\![ ((t'-t) - x)\%2 = 0 ]\!] * \mu(x-1, t'-t-1) * p + \\ [\![ t' - t \geq x+2 ]\!] * [\![ ((t'-t) - (x+2))\%2 = 0 ]\!] * \mu(x+1, t'-t-1) * q \end{array} \right) \end{aligned}$$

[14] This equation equips us another way to calculate the distribution $\mu(m,n)$ of SRW through recursive functions.

which means if the initial value of $x$ is not larger than 0, the loop behaves like skip (unchanged variables). Otherwise, the loop terminates (that is, when the final value of $x$ is 0, $x' = 0$) through two possible initial steps: $x$ decreased by 1 with probability $p$ or increased by 1 with probability $q$, corresponding to two operands of the addition in the parenthesis. In the first case, $x$ is equal to 0 only after more than $x$ steps ($t' - t \geq x$) and the number ($t' - t$) of steps must be even (or odd) if $x$ is even (or odd), as encoded in $((t' - t) - x)\%2 = 0$. In the second case, $x$ is equal to 0 only after more than $x + 2$ steps ($t' - t \geq x + 2$) because the first step moves away from 0 and the number ($t' - t$) of steps must be even (or odd) if $x + 2$ is even (or odd), as encoded in $((t' - t) - (x + 2))\%2 = 0$.

We have proved that $Ht$ is a fixed point. And the proof is omitted here for brevity. The proof, however, has not been mechanised in Isabelle/UTP. This is part of the future work Ye and Woodcock are interested in.

As shown above, finding the distributions of probabilistic loops and proving their semantics is non-trivial, even for this simple program—PSRW. In ProbURel, Ye et al. [183] developed a constructive semantics for probabilistic loops using Kleene's fixed-point theorem. This can be used to approximate semantics for loops in practice based on iterations. We explain the algorithm as follows.

A loop characterisation function is defined below where $P$ is the loop body.

$$\mathcal{F}(b, P, X) \mathrel{\widehat=} \textbf{if } b \textbf{ then } \left(P \mathbin{;_p} X\right) \textbf{ else } \mathit{I\!I}_p$$

According to the theorems [183] about the least fixed point (lfp) and greatest fixed point by construction, lfp and gfp can be constructed using the iteration $\mathcal{F}^n(b, P, \bot)$ from the bottom and the iteration $\mathcal{F}^n(b, P, \top)$ from the top of the complete lattice. Additionally, $\mathcal{F}^n(b, P, \bot)$ is an increasing chain and $\mathcal{F}^n(b, P, \top)$ is an decreasing chain. The unique fixed point theorem shows that if, for all states, the difference $\mathcal{F}^n(b, P, \top) - \mathcal{F}^n(b, P, \bot)$ between the two iterations tends to 0 when $n$ approaches $\infty$, then the iterations coincide. Using the iterations to determine lfp and gfp, in general, are not decidable because the difference might never be 0 for any $n$ (though its limit might be 0). However, if we bound the difference as $\varepsilon > 0$, there always exists a $n$ such that, for all states, $\mathcal{F}^n(b, P, \top) - \mathcal{F}^n(b, P, \bot) < \varepsilon$. Then $\mathcal{F}^n(b, P, \top)$ is an approximation of lfp and $\mathcal{F}^n(b, P, \top)$ is an approximation of gfp though they bias. Then this bounded approximation is decidable. For PSRW, the algorithm is shown below.

(1) choose a small $\varepsilon$, such as $1e - 9$;
(2) compute a $n$ such that the differences for all states are less than $\varepsilon$;
(3) compute $\mathcal{F}^i(b, P, \bot)$ for $i$ from 0 to $n$; these are the termination probabilities in terms of the number of steps ($i$), that is, the probability distribution of PSRW up to $n$ steps.

We note that in the approximated distribution, the probabilities for 0 to $n$ steps are exact and only the probabilities after $n+1$ are cut off (that is, all zero).

This approach could address the difficulty of finding and proving the distribution in practice. We plan to also investigate this further in the future.

## 5 Conclusions

In this chapter, we have taken a *tour* through the programming choices available across state- and process-based paradigms. Nondeterminism plays a fundamental role and traces its roots to the early theories of computation, where it can be viewed as *demonic* or *angelic*. Semantically, the predicate transformer view is perhaps simpler than the (multi)relational one, yet it's pleasing that the choices are duals in the respective lattices induced by refinement. The applications sketched in Sections 4.1 and 4.2 highlight the need for further work in this area, not least the mechanisation of the theory [142] in Isabelle/UTP [46] and the exploration of refinement laws. Extensions of that theory to calculi covering time [21], perhaps via a relational account of *tock*-CSP [12], and other aspects are also future work. A related question is whether Foster's interaction trees [49] could be extended to give an operational account to both choices, thus paving the way for sound animation, an important bridge for engaging with practitioners.

Preferential choice forces us to step outside the realm of wp predicate transformers, while having an elegant and easy to define model in prospective-value (pv) expression transformers. The core logic of a theory of prospective values that is able to account for both nondeterminism and nontermination requires *improper bunch theory* (a self-flattening set theory with a $\perp$ value for every type) as an integral part, i.e., to model the outcome of computations. Preferential computations forfeit monotonicity, e.g., in the first operand of $S \gg T$, but a restricted form of compositional refinement can be recovered via a Galois connection between wp and pv computations. This enables us to safely replace $S \sqcap T$ in specification by $S \gg T$ in implementations, and thereby formally justify the introduction of heuristics when exploring search spaces.

As hinted in Section 3.3, we managed to define and mechanise a suitable model for preference and preferential computations in a wp calculus that builds on top of a three-value logic—an instance of Gödel-Dummett logic—and managed to verify the existence of the abovementioned Galois connection between plain monotonic computations in wp semantics and preferential computations. We expect this work to be published in the near future.

Probabilistic choice can be seen as a refinement of nondeterministic choice because it provides more specific information (probabilities) about how the choice can be made. The semantics of probabilistic choice with the presence of nondeterministic choice are very different in the relational model and the predicate transformer for imperative programs, and in process algebras. The review of probabilistic choice in Section 3.4 and its application presented in Section 4.3 reveal the need for further work (1) in the extension of ProbURel to support nondeterministic choice and concurrency, (2) in the practical tool support for approximating probabilistic loops using ProbURel in addition to theorem proving for exact verification, and (3) in the extension of ITrees-based CSP and *Circus* [49, 181] to support probabilistic choice for the sound animation of probabilistic programs with nondeterministic choice resolved by priorities.

In summary, the type of programming choice that is most suitable very much depends on our modelling needs. In some cases, several types of choices are even feasible to adopt: e.g., backtracking can be modelled by both demonic and angelic choice, using either **magic** or **abort** as a special program to trigger backtracking. The deeper connections between choices at a semantic level are a fascinating area of fundamental research, and the combination of several notions of choice in a single universal theory remains a key challenge for unification.

## Acronyms

| | |
|---|---|
| **ACP** | Algebra of Communicating Processes |
| **API** | Application Programming Interface |
| **AST** | Almost Sure Termination |
| **CCS** | Calculus of Communicating Systems |
| **CPS** | Cyber-Physical System |
| **CSP** | Communicating Sequential Processes |
| **cwp** | Conjugate Weakest Precondition |
| **FDC** | Free Distributive Completion |
| **FMI** | Functional Mock-up Interface |
| **FMU** | Functional Mock-up Unit |
| **GCL** | Guarded Command Language |
| **MA** | Master Algorithm |
| **MDP** | Markov Decision Process |
| **PPP** | Probabilistic Predicative Programming |
| **ProbURel** | Probabilistic Unifying Relations |
| **PV** | Prospective Value |
| **RW** | Random Walk |
| **(S)SRW** | (Symmetric) Simple Random Walk |
| **SU** | Simulation Unit |
| **UTP** | Unifying Theories of Programming |
| **wp** | Weakest Precondition |
| **wpe** | A wp theory but built on top of three-valued logic |
| **WPE** | Weakest Pre-Expectation |
| **ZFC** | Zermelo–Fraenkel set theory with axiom of choice |

## Acknowledgments

The work on FMI benefited from discussions with Claudio Gomes, Simon Hansen, Jaco van de Pol and Jim Woodcock. EPSRC projects RoboCalc (EP/M025756/1), RoboTest (EP/R025479/1), and CyPhyAssure (EP/S001190/1) funded the work discussed here related to RoboChart.

## References

1. Abelson, H., Sussman, G.J.: Structure and Interpretation of Computer Programs, Second Edition. MIT Press (1996)

2. Andova, S.: Probabilistic process algebra. Ph.D. thesis, Mathematics and Computer Science, Technische Universiteit Eindhoven (2002). 10.6100/IR561343
3. Apt, K.R., Olderog, E.: Nondeterminism and guarded commands. In: Apt, K.R., Hoare, T. (eds.) Edsger Wybe Dijkstra: His Life, Work, and Legacy, ACM Books, vol. 45, pp. 169–204. ACM / Morgan & Claypool (2022). 10.1145/3544585.3544595
4. Armoni, M., Ben-Ari, M.: The concept of nondeterminism: its development and implications for teaching. ACM SIGCSE Bull. **41**(2), 141–160 (2009). 10.1145/1595453.1595495
5. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. Journal of Algorithms **11**(3), 441–461 (Sep 1990). 10.1016/0196-6774(90)90021-6
6. Baaz, M., Preining, N., Zach, R.: First-order Gödel logics. Annals of Pure and Applied Logic **147**(1), 23–47 (Jun 2007). 10.1016/j.apal.2007.03.001
7. Back, R.J.R., von Wright, J.: Duality in specification languages: a lattice-theoretical approach. Acta Informatica **27**(7), 583–625 (Jul 1990). 10.1007/bf00259469
8. Back, R., von Wright, J.: Refinement Calculus - A Systematic Introduction. Graduate Texts in Computer Science, Springer (1998). 10.1007/978-1-4612-1674-2
9. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Formal certification of code-based cryptographic proofs. In: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. p. 90–101. POPL '09, Association for Computing Machinery, New York, NY, USA (2009). 10.1145/1480881.1480894
10. Batz, K., Biskup, T.J., Katoen, J.P., Winkler, T.: Programmatic Strategy Synthesis: Resolving Nondeterminism in Probabilistic Programs. Proceedings of the ACM on Programming Languages **8**(POPL), 2792–2820 (Jan 2024). 10.1145/3632935
11. Batz, K., Chen, M., Junges, S., Kaminski, B.L., Katoen, J.P., Matheja, C.: Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants, pp. 410–429. Springer Nature Switzerland (2023). 10.1007/978-3-031-30820-8_25
12. Baxter, J., Ribeiro, P., Cavalcanti, A.: Sound reasoning in tock-CSP. Acta Informatica (Apr 2021). 10.1007/s00236-020-00394-3
13. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theor. Comput. Sci. **37**, 77–121 (1985). 10.1016/0304-3975(85)90088-X
14. Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., et al.: Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: 9th International Modelica Conference. pp. 173–184. The Modelica Association (2012). 10.3384/ecp12076173
15. Bloom, B., Meyer, A.R.: A remark on bisimulation between probabilistic processes. In: Meyer, A.R., Taitslin, M.A. (eds.) Logic at Botik '89. pp. 26–40. Springer Berlin Heidelberg, Berlin, Heidelberg (1989). 10.1007/3-540-51237-3_4
16. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984). 10.1145/828.833
17. Broy, M., Wirsing, M.: On the algebraic specification of nondeterministic programming languages. In: Astesiano, E., Böhm, C. (eds.) CAAP '81, Trees in Algebra and Programming, 6th Colloquium, Genoa, Italy, March 5-7, 1981, Proceedings. Lecture Notes in Computer Science, vol. 112, pp. 162–179. Springer (1981). 10.1007/3-540-10828-9_61

18. Butler, M.J., Leuschel, M.: Combining CSP and B for specification and property verification. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3582, pp. 221–236. Springer (2005). 10.1007/11526841_16
19. Butterfield, A., Sherif, A., Woodcock, J.: Slotted-circus. In: Davies, J., Gibbons, J. (eds.) Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4591, pp. 75–97. Springer (2007). 10.1007/978-3-540-73210-5_5
20. Canham, S., Woodcock, J.: Three approaches to timed external choice in UTP. In: Naumann, D.A. (ed.) Unifying Theories of Programming - 5th International Symposium, UTP 2014, Singapore, May 13, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8963, pp. 1–20. Springer (2014). 10.1007/978-3-319-14806-9_1
21. Cavalcanti, A., Mota, A., Woodcock, J.: Simulink timed models for program verification. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. Lecture Notes in Computer Science, vol. 8051, pp. 82–99. Springer (2013). 10.1007/978-3-642-39698-4_6
22. Cavalcanti, A., Sampaio, A., Woodcock, J.: A refinement strategy for circus. Formal Aspects Comput. **15**(2-3), 146–181 (2003). 10.1007/S00165-003-0006-5
23. Cavalcanti, A., Woodcock, J.: A tutorial introduction to CSP in *Unifying Theories of Programming*. In: Cavalcanti, A., Sampaio, A., Woodcock, J. (eds.) Refinement Techniques in Software Engineering, First Pernambuco Summer School on Software Engineering, PSSE 2004, Recife, Brazil, November 23-December 5, 2004, Revised Lectures. Lecture Notes in Computer Science, vol. 3167, pp. 220–268. Springer (2004). 10.1007/11889229_6
24. Cavalcanti, A., Woodcock, J., Amálio, N.: Behavioural models for FMI co-simulations. In: Sampaio, A., Wang, F. (eds.) Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan, ROC, October 24-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9965, pp. 255–273 (2016). 10.1007/978-3-319-46750-4_15
25. Cavalcanti, A., Woodcock, J., Dunne, S.: Angelic nondeterminism in the unifying theories of programming. Formal Aspects Comput. **18**(3), 288–307 (2006). 10.1007/S00165-006-0001-8
26. Chadha, R., Cruz-Filipe, L., Mateus, P., Sernadas, A.: Reasoning about probabilistic sequential programs. Theor. Comput. Sci. **379**(1-2), 142–165 (2007). 10.1016/j.tcs.2007.02.040
27. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981). 10.1145/322234.322243
28. Chandy, K.M., Misra, J.: Parallel program design - a foundation. Addison-Wesley (1989)
29. Chatterjee, K., Fu, H., Novotný, P.: Termination Analysis of Probabilistic Programs with Martingales, p. 221–258. Cambridge University Press (2020). 10.1017/9781108770750
30. Chomsky, N.: Context-free grammars and pushdown storage. MIT Res. Lab. Electron. Quart. Prog. Report. **65**, 187–194 (1962)
31. Commission of the European Communities: Information technology security evaluation criteria (ITSEC): Preliminary harmonised criteria. (June 1991)

32. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA. pp. 151–158. ACM (1971). 10.1145/800157.805047
33. Cremona, F., Lohstroh, M., Broman, D., Natale, M.D., Lee, E.A., Tripakis, S.: Step revision in hybrid co-simulation with FMI. In: 2016 ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2016, Kanpur, India, November 18-20, 2016. pp. 173–183. IEEE (2016). 10.1109/MEMCOD.2016.7797762
34. Dahlqvist, F., Silva, A., Kozen, D.: Semantics of Probabilistic Programming: A Gentle Introduction. In: Barthe, G., Katoen, J.P., Silva, A. (eds.) Foundations of Probabilistic Programming, p. 1–42. Cambridge University Press (2020). 10.1017/9781108770750.002
35. Davies, J., Schneider, S.: A brief history of Timed CSP. Theoretical Computer Science **138**(2), 243–271 (Feb 1995). 10.1016/0304-3975(94)00169-j
36. Davis, M.D.: Computability and Unsolvability. McGraw-Hill Series in Information Processing and Computers, McGraw-Hill (1958)
37. den Hartog, J., De Vink, E.: Verifying Probabilistic Programs Using a Hoare like Logic. International journal of foundations of computer science **13**(3), 315–340 (Jun 2002). 10.1142/S012905410200114X, imported from DIES
38. Dijkstra, E.W.: Correctness concerns and, among other things, why they are resented (Nov 1974), `http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD450.PDF`, invited paper, to be presented at the International Conference on Reliable Software, Los Angeles, 21–23 April 1975; circulated privately
39. Dijkstra, E.W.: Guarded commands, non-determinacy and a calculus for the derivation of programs (Jun 1974), `http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD418.PDF`, see EWD:EWD472; circulated privately
40. Dijkstra, E.: A Discipline of Programming. Prentice-Hall Series in Automa, Prentice-Hall (1976)
41. Dijkstra, R.M.: DUALITY: A simple formalism for the analysis of UNITY. Formal Aspects Comput. **7**(4), 353–388 (1995). 10.1007/BF01211214
42. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). 10.1109/TIT.1983.1056650
43. Dunne, S., Ferreira, J.F., Mendes, A., Ritchie, C., Stoddart, B., Zeyda, F.: bGSL: An imperative language for specification and refinement of backtracking programs. Journal of Logical and Algebraic Methods in Programming **130**, 100811 (Jan 2023). 10.1016/j.jlamp.2022.100811
44. Fischer, C.: How to combine Z with a process algebra. In: Bowen, J.P., Fett, A., Hinchey, M.G. (eds.) ZUM '98: The Z Formal Specification Notation. pp. 5–23. Springer, Berlin, Heidelberg (1998). 10.1007/978-3-540-49676-2_2
45. Floyd, R.W.: Nondeterministic algorithms. J. ACM **14**(4), 636–644 (1967). 10.1145/321420.321422
46. Foster, S., Baxter, J., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying semantic foundations for automated verification tools in Isabelle/UTP. Science of Computer Programming **197**, 102510 (2020). 10.1016/j.scico.2020.102510
47. Foster, S., Cavalcanti, A., Canham, S., Woodcock, J., Zeyda, F.: Unifying theories of reactive design contracts. Theor. Comput. Sci. **802**, 105–140 (2020). 10.1016/J.TCS.2019.09.017
48. Foster, S., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying theories of time with generalised reactive processes. Inf. Process. Lett. **135**, 47–52 (2018). 10.1016/J.IPL.2018.02.017

49. Foster, S., Hur, C., Woodcock, J.: Formally verified simulations of state-rich processes using interaction trees in Isabelle/HOL. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference. LIPIcs, vol. 203, pp. 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). 10.4230/LIPICS.CONCUR.2021.20
50. Foster, S., Zeyda, F., Nemouchi, Y., Ribeiro, P., Wolff, B.: Isabelle/utp: Mechanised theory engineering for unifying theories of programming. Arch. Formal Proofs **2019** (2019), `https://www.isa-afp.org/entries/UTP.html`
51. Francez, N., Hoare, C.A.R., Lehmann, D.J., de Roever, W.P.: Semantics of nondeterminism, concurrency, and communication. J. Comput. Syst. Sci. **19**(3), 290–308 (1979). 10.1016/0022-0000(79)90006-0
52. Freitas, L., Woodcock, J.: Mechanising mondex with z/eves. Formal Aspects Comput. **20**(1), 117–139 (2008). 10.1007/S00165-007-0059-Y
53. Furusawa, H., Struth, G.: Taming multirelations. ACM Trans. Comput. Log. **17**(4), 28 (2016). 10.1145/2964907
54. Gardiner, P.H.B., Morgan, C.: Data refinement of predicate transformers. Theor. Comput. Sci. **87**(1), 143–162 (1991). 10.1016/0304-3975(91)90029-2
55. Georgievska, S., Andova, S.: Probabilistic CSP: Preserving the Laws via Restricted Schedulers. In: Schmitt, J.B. (ed.) Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance. pp. 136–150. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). 978-3-642-28540-0_10
56. Giacalone, A., Jou, C., Smolka, S.A.: Algebraic Reasoning for Probabilistic Concurrent Systems. In: Broy, M., Jones, C.B. (eds.) Programming concepts and methods: Proceedings of the IFIP Working Group 2.2, 2.3 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel, 2-5 April, 1990. pp. 443–458. North-Holland (1990)
57. Gleirscher, M., Foster, S., Woodcock, J.: New opportunities for integrated formal methods. ACM Comput. Surv. **52**(6), 117:1–117:36 (2020). 10.1145/3357231
58. Gómez, F.C., de Frutos Escrig, D., Ruiz, V.V.: A sound and complete proof system for probabilistic processes. In: Bertran, M., Rus, T. (eds.) Transformation-Based Reactive Systems Development. pp. 340–352. Springer Berlin Heidelberg, Berlin, Heidelberg (1997). 3-540-63010-4_23
59. Grimmett, G., Welsh, D.: Probability: An Introduction. Oxford University Press, Clarendon Press (1986)
60. Hansen, S.T., Gomes, C., Palmieri, M., Thule, C., van de Pol, J., Woodcock, J.: Verification of co-simulation algorithms subject to algebraic loops and adaptive steps. In: Lluch-Lafuente, A., Mavridou, A. (eds.) Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS 2021, Paris, France, August 24-26, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12863, pp. 3–20. Springer (2021). 10.1007/978-3-030-85248-1_1
61. Hansson, H., Jonsson, B.: A calculus for communicating systems with time and probabilities. In: [1990] Proceedings 11th Real-Time Systems Symposium. pp. 278–287 (1990). 10.1109/REAL.1990.128759
62. Hansson, H.: Time and Probabilities in Formal Design of Distributed Systems. PhD thesis, Department of Computer Systems, Uppsala University (1991)
63. Hartmanns, A., Hermanns, H.: In the quantitative automata zoo. Science of Computer Programming **112**, 3–23 (2015). 10.1016/j.scico.2015.08.009, fundamentals of Software Engineering (selected papers of FSEN 2013)
64. He, J., Morgan, C., McIver, A.: Deriving Probabilistic Semantics Via the 'Weakest Completion'. In: Davies, J., Schulte, W., Barnett, M. (eds.) Formal Methods and

Software Engineering. pp. 131–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). 978-3-540-30482-1_17
65. Hehner, E.C.R.: Predicative programming part i. Commun. ACM **27**(2), 134–143 (feb 1984). 10.1145/69610.357988
66. Hehner, E.C.R.: Probabilistic predicative programming. In: Kozen, D., Shankland, C. (eds.) Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3125, pp. 169–185. Springer (2004). 10.1007/978-3-540-27764-4_10
67. Hehner, E.C.R.: A probability perspective. Formal Aspects Comput. **23**(4), 391–419 (2011). 10.1007/s00165-010-0157-0
68. Hehner, E.C.R.: a Practical Theory of Programming (2024-1-14 edition). Springer (first edition) (Jan 2024), `https://www.cs.toronto.edu/~hehner/aPToP/aPToP.pdf`
69. Hehner, E.C.: Bunch theory: A simple set theory for computer science. Information Processing Letters **12**(1), 26–30 (Feb 1981). 10.1016/0020-0190(81)90071-5
70. Hesselink, W.H.: LR-parsing derived. Sci. Comput. Program. **19**(2), 171–196 (1992). 10.1016/0167-6423(92)90007-X
71. Hesselink, W.H.: Programs, Recursion and Unbounded Choice. Cambridge University Press (1992)
72. Hesselink, W.H.: Nondeterminacy and recursion via stacks and games. Theor. Comput. Sci. **124**(2), 273–295 (1994). 10.1016/0304-3975(92)00016-K
73. Hesselink, W.H.: Alternating states for dual nondeterminism in imperative programming. Theor. Comput. Sci. **411**(22-24), 2317–2330 (2010). 10.1016/J.TCS.2010.03.016
74. Hesselink, W.H., Reinds, R.: Temporal preconditions of recursive procedures. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Sematics: Foundations and Applications, REX Workshop, Beekbergen, The Netherlands, June 1-4, 1992, Proceedings. Lecture Notes in Computer Science, vol. 666, pp. 236–260. Springer (1992). 10.1007/3-540-56596-5_36
75. Hoare, C.A.R.: Algorithm 64: Quicksort. Commun. ACM **4**(7), 321 (jul 1961). 10.1145/366622.366644
76. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). 10.1145/363235.363259
77. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978). 10.1145/359576.359585
78. Hoare, C.A.R.: A model for communicating sequential processes. In: McKeag, R.M., Macnaghten, A.M. (eds.) On the Construction of Programs, pp. 229–254. Cambridge University Press (1980)
79. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice-Hall (1998)
80. Hoare, C., He, J.: The weakest prespecification. Information Processing Letters **24**(2), 127–132 (Jan 1987). 10.1016/0020-0190(87)90106-2
81. Hurd, J.: Formal verification of probabilistic algorithms. Tech. Rep. UCAM-CL-TR-566, University of Cambridge, Computer Laboratory (May 2003). 10.48456/tr-566
82. Jagadeesan, R., Shanbhogue, V., Saraswat, V.: Angelic non-determinism in concurrent constraint programming. Tech. rep., Technical report, Xerox Park (1991)
83. Jansen, D.N., Hermanns, H., Katoen, J.P.: A Probabilistic Extension of UML Statecharts. In: Formal Tec. in Real-Time and Fault-Tolerant Syst. LNCS, vol. 2469, pp. 355–374. Springer (2002). 3-540-45739-9_21

84. Jones, C.B., O'Hearn, P.W., Woodcock, J.: Verified software: A grand challenge. Computer **39**(4), 93–95 (2006). 10.1109/MC.2006.145
85. Jones, C.B.: Systematic software development using VDM. Prentice Hall International Series in Computer Science, Prentice Hall (1986)
86. Jonsson, B., Yi, W., Larsen, K.G.: Chapter 11 - probabilistic extensions of process algebras**this chapter is dedicated to the fond memory of Linda Christoff. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 685–710. Elsevier Science, Amsterdam (2001). 10.1016/B978-044482830-9/50029-1
87. Jr., H.R.: Theory of recursive functions and effective computability (Reprint from 1967). MIT Press (1987)
88. Kaminski, B.L.: Advanced weakest precondition calculi for probabilistic programs. Ph.D. thesis, RWTH Aachen University, Germany (2019), `http://publications.rwth-aachen.de/record/755408`
89. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. J. ACM **65**(5) (aug 2018). 10.1145/3208102
90. Kennaway, R., Hoare, C.A.R.: A theory of nondeterminism. In: de Bakker, J.W., van Leeuwen, J. (eds.) Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings. Lecture Notes in Computer Science, vol. 85, pp. 338–350. Springer (1980). 10.1007/3-540-10003-2_82
91. Kok, J.N.: On Logic Programming and the Refinement Calculus: Semantics Based Program Transformations. Technical Report RUU-CS-90-39, Utrecht University (December 1990)
92. Kozen, D.: Semantics of probabilistic programs. Journal of Computer and System Sciences **22**(3), 328–350 (1981). 10.1016/0022-0000(81)90036-2
93. Kozen, D.: A probabilistic pdl. Journal of Computer and System Sciences **30**(2), 162–178 (1985). 10.1016/0022-0000(85)90012-1
94. Kwiatkowska, M., Norman, G.: A fully abstract metric-space denotational semantics for reactive probabilistic processes. Electronic Notes in Theoretical Computer Science **13**, 182 (1998). 10.1016/S1571-0661(05)80222-1, comprox III, Third Workshop on Computation and Approximation
95. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011). 10.1007/978-3-642-22110-1_47
96. Larsen, K.G., Skou, A.: Bisimulation through Probabilistic Testing. Inf. Comput. **94**(1), 1–28 (1991). 10.1016/0890-5401(91)90030-6
97. Lowe, G.: Probabilistic and prioritized models of timed csp. Theoretical Computer Science **138**(2), 315–352 (1995). 10.1016/0304-3975(94)00171-E, meeting on the mathematical foundation of programing semantics
98. López, N., Núñez, M.: An Overview of Probabilistic Process Algebras and Their Equivalences, pp. 89–123. Springer Berlin Heidelberg (2004). 10.1007/978-3-540-24611-4_3
99. Martin, A.P., Gardiner, P.H.B., Woodcock, J.: A tactic calculus-abridged version. Formal Aspects Comput. **8**(4), 479–489 (1996). 10.1007/BF01213535
100. Martin, C.E., Curtis, S.A., Rewitzky, I.: Modelling nondeterminism. In: Kozen, D., Shankland, C. (eds.) Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceed-

ings. Lecture Notes in Computer Science, vol. 3125, pp. 228–251. Springer (2004). 10.1007/978-3-540-27764-4_13

101. McCarthy, J.: A basis for a mathematical theory of computation, preliminary report. In: Bauer, W.F. (ed.) Papers presented at the 1961 western joint IRE-AIEE-ACM computer conference, IRE-AIEE-ACM 1961 (Western), Los Angeles, California, USA, May 9-11, 1961. pp. 225–238. ACM (1961). 10.1145/1460690.1460715
102. McIver, A., Morgan, C.: Demonic, angelic and unbounded probabilistic choices in sequential programs. Acta Informatica **37**(4–5), 329–354 (Jan 2001). 10.1007/s002360000046
103. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science, Springer (2005). 10.1007/b138392
104. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems, chap. Introduction to pGCL: Its logic and its model, pp. 3–36. Springer New York, New York, NY (2005). 10.1007/0-387-27006-X_1
105. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems, chap. Introduction to pGCL, pp. 3–35. Monographs in Computer Science, Springer (2005). 10.1007/b138392
106. McIver, A., Morgan, C.: Correctness by construction for probabilistic programs. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12476, pp. 216–239. Springer (2020). 10.1007/978-3-030-61362-4_12
107. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.P.: A New Proof Rule for Almost-Sure Termination. Proc. ACM Program. Lang. **2**(POPL) (dec 2017). 10.1145/3158121
108. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). 10.1007/3-540-10235-3
109. Mislove, M.: Nondeterminism and Probabilistic Choice: Obeying the Laws, pp. 350–365. Springer Berlin Heidelberg (2000). 10.1007/3-540-44618-4_26
110. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: Robochart: modelling and verification of the functional behaviour of robotic applications. Softw. Syst. Model. **18**(5), 3097–3149 (2019). 10.1007/s10270-018-00710-z
111. Moosbrugger, M., Bartocci, E., Katoen, J., Kovács, L.: The Probabilistic Termination Tool Amber. In: Huisman, M., Pasareanu, C.S., Zhan, N. (eds.) Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13047, pp. 667–675. Springer (2021). 10.1007/978-3-030-90870-6_36
112. Morgan, C.: Programming from specifications. Prentice Hall International Series in computer science, Prentice Hall (1990)
113. Morgan, C.: Of probabilistic wp and csp—and compositionality. In: Abdallah, A.E., Jones, C.B., Sanders, J.W. (eds.) Communicating Sequential Processes. The First 25 Years: Symposium on the Occasion of 25 Years of CSP, London, UK, July 7-8, 2004. Revised Invited Papers, pp. 220–241. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). 10.1007/11423348_12
114. Morgan, C., McIver, A.: pGCL: formal reasoning for random algorithms. South African Computer Journal **22**, 14–27 (1999), `http://hdl.handle.net/10500/24296`
115. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems (TOPLAS) **18**(3), 325–353 (1996). 10.1145/229542.229547

116. Morgan, C., McIver, A., Seidel, K., Sanders, J.W.: Refinement-oriented probability for csp. Form. Asp. Comput. **8**(6), 617–647 (nov 1996). 10.1007/BF01213492
117. Morris, J.M.: A theoretical basis for stepwise refinement and the programming calculus. Sci. Comput. Program. **9**(3), 287–306 (1987). 10.1016/0167-6423(87)90011-6
118. Morris, J.M.: Augmenting types with unbounded demonic and angelic nondeterminacy. In: Kozen, D., Shankland, C. (eds.) Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3125, pp. 274–288. Springer (2004). 10.1007/978-3-540-27764-4_15
119. Morris, J.M., Bunkenburg, A.: A theory of bunches. Acta Informatica **37**(8), 541–561 (May 2001). 10.1007/PL00013316
120. Morris, J.M., Bunkenburg, A., Tyrrell, M.: Term Transformers: A New Approach to State. ACM Transactions on Programming Languages and Systems **31**(4) (May 2009). 10.1145/1516507.1516511
121. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
122. Nelson, G.: A Generalization of Dijkstra's Calculus. ACM Transactions on Programming Languages and Systems **11**(4), 517–561 (Oct 1989). 10.1145/69558.69559
123. Nicola, R.D., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. **34**, 83–133 (1984). 10.1016/0304-3975(84)90113-0
124. Núñez, M., de Frutos, D., Llana, L.: Acceptance trees for probabilistic processes. In: Lee, I., Smolka, S.A. (eds.) CONCUR '95: Concurrency Theory. pp. 249–263. Springer Berlin Heidelberg, Berlin, Heidelberg (1995). 10.1007/3-540-60218-6_18
125. Oliveira, M., Cavalcanti, A., Woodcock, J.: ArcAngel: a tactic language for refinement. Formal Aspects Comput. **15**(1), 28–47 (2003). 10.1007/S00165-003-0003-8
126. Oliveira, M., Cavalcanti, A., Woodcock, J.: Formal development of industrial-scale systems in *Circus*. Innov. Syst. Softw. Eng. **1**(2), 125–146 (2005). 10.1007/S11334-005-0014-0
127. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for *Circus*. Formal Aspects Comput. **21**(1-2), 3–32 (2009). 10.1007/S00165-007-0052-5
128. Olmedo, F., Gretz, F., Jansen, N., Kaminski, B.L., Katoen, J.P., Mciver, A.: Conditioning in probabilistic programming. ACM Transactions on Programming Languages and Systems **40**(1), 1–50 (Jan 2018). 10.1145/3156018
129. Peleg, D.: Concurrent dynamic logic. J. ACM **34**(2), 450–479 (1987). 10.1145/23005.23008
130. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., USA, 1st edn. (1994)
131. Pólya, G.: Über eine aufgabe der wahrscheinlichkeitsrechnung betreffend die irrfahrt im straßennetz. Mathematische Annalen **84**(1–2), 149–160 (Mar 1921). 10.1007/bf01458701
132. Rabin, M.O.: Probabilistic algorithm for testing primality. Journal of Number Theory **12**(1), 128–138 (Feb 1980). 10.1016/0022-314x(80)90084-0
133. Rabin, M.O.: N-process mutual exclusion with bounded waiting by 4 · log2 n-valued shared variable. Journal of Computer and System Sciences **25**(1), 66–75 (Aug 1982). 10.1016/0022-0000(82)90010-1
134. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). 10.1147/RD.32.0114
135. Ramshaw, L.H.: Formalizing the Analysis of Algorithms. Ph.D. thesis, Stanford University, Stanford, CA, USA (1979), aAI8001994

136. Rand, R., Zdancewic, S.: VPHL: A verified partial-correctness logic for probabilistic programs. In: Ghica, D.R. (ed.) The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015. Electronic Notes in Theoretical Computer Science, vol. 319, pp. 351–367. Elsevier (2015). 10.1016/j.entcs.2015.12.021
137. Rewitzky, I.: Binary multirelations. In: de Swart, H.C.M., Orlowska, E., Schmidt, G., Roubens, M. (eds.) Theory and Applications of Relational Structures as Knowledge Instruments, COST Action 274, TARSKI, Revised Papers, Lecture Notes in Computer Science, vol. 2929, pp. 256–271. Springer (2003). 10.1007/978-3-540-24615-2_12
138. Rewitzky, I., Brink, C.: Predicate transformers as power operations. Formal Aspects Comput. **7**(2), 169–182 (1995). 10.1007/BF01211604
139. Ribeiro, P.: A unary semigroup trace algebra. In: Fahrenberg, U., Jipsen, P., Winter, M. (eds.) Relational and Algebraic Methods in Computer Science - 18th International Conference, RAMiCS 2020, Palaiseau, France, April 8-11, 2020, Proceedings [postponed]. Lecture Notes in Computer Science, vol. 12062, pp. 270–285. Springer (2020). 10.1007/978-3-030-43520-2_17
140. Ribeiro, P., Cavalcanti, A.: Angelicism in the theory of reactive processes. In: Naumann, D.A. (ed.) Unifying Theories of Programming - 5th International Symposium, UTP 2014, Singapore, May 13, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8963, pp. 42–61. Springer (2014). 10.1007/978-3-319-14806-9_3
141. Ribeiro, P., Cavalcanti, A.: UTP designs for binary multirelations. In: Ciobanu, G., Méry, D. (eds.) Theoretical Aspects of Computing - ICTAC 2014 - 11th International Colloquium, Bucharest, Romania, September 17-19, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8687, pp. 388–405. Springer (2014). 10.1007/978-3-319-10882-7_23
142. Ribeiro, P., Cavalcanti, A.: Angelic processes for CSP via the UTP. Theor. Comput. Sci. **756**, 19–63 (2019). 10.1016/J.TCS.2018.10.008
143. Roscoe, A.W.: Understanding Concurrent Systems. Texts in Computer Science, Springer (2011)
144. Schneider, S.A., Treharne, H.: Communicating B machines. In: Bert, D., Bowen, J.P., Henson, M.C., Robinson, K. (eds.) ZB 2002: Formal Specification and Development in Z and B, 2nd International Conference of B and Z Users, Grenoble, France, January 23-25, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2272, pp. 416–435. Springer (2002). 10.1007/3-540-45648-1_22
145. Schröer, P., Batz, K., Kaminski, B.L., Katoen, J.P., Matheja, C.: A Deductive Verification Infrastructure for Probabilistic Programs. Proceedings of the ACM on Programming Languages **7**(OOPSLA2), 2052–2082 (Oct 2023). 10.1145/3622870
146. Schützenberger, M.P.: On context-free languages and push-down automata. Inf. Control. **6**(3), 246–264 (1963). 10.1016/S0019-9958(63)90306-1
147. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic J. of Computing **2**(2), 250–273 (jun 1995)
148. Seidel, K.: Probabilistic communicating processes. Theoretical Computer Science **152**(2), 219–249 (1995). 10.1016/0304-3975(94)00286-0
149. Sherif, A., He, J.: Towards a time model for circus. In: George, C., Miao, H. (eds.) Formal Methods and Software Engineering, 4th International Conference on Formal Engineering Methods, ICFEM 2002 Shanghai, China, October 21-25, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2495, pp. 613–624. Springer (2002). 10.1007/3-540-36103-0_62

150. Sherif, A., He, J., Cavalcanti, A., Sampaio, A.: A framework for specification and validation of real-time systems using *Circus* actions. In: Liu, Z., Araki, K. (eds.) Theoretical Aspects of Computing - ICTAC 2004, First International Colloquium, Guiyang, China, September 20-24, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3407, pp. 478–493. Springer (2004). 10.1007/978-3-540-31862-0_34
151. Smith, G., Derrick, J.: Specification, refinement and verification of concurrent systems-an integration of object-z and CSP. Formal Methods Syst. Des. **18**(3), 249–284 (2001). 10.1023/A:1011269103179
152. Stepney, S., Cooper, D., Woodcock, J.: More powerful Z data refinement: Pushing the state of the art in industrial refinement. In: Bowen, J.P., Fett, A., Hinchey, M.G. (eds.) ZUM '98: The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, September 24-26, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1493, pp. 284–307. Springer (1998). 10.1007/978-3-540-49676-2_20
153. Stoddart, B., Dunne, S., Mu, C., Zeyda, F.: Bunch theory: Axioms, logic, applications and model. Journal of Logical and Algebraic Methods in Programming **140**, 100977 (Aug 2024). 10.1016/j.jlamp.2024.100977
154. Stoddart, B., Zeyda, F.: A unification of probabilistic choice within a design-based model of reversible computation. Formal Aspects of Computing **25**(1), 107–131 (Jan 2013). 10.1007/s00165-007-0048-1
155. Stoddart, B., Zeyda, F., Dunne, S.: Preference and Non-deterministic Choice. In: Theoretical Aspects of Computing – ICTAC 2010. LNCS, vol. 6255, pp. 137–152. Springer (Sep 2010). 10.1007/11415787_12
156. Sun, J., Liu, Y., Dong, J.S., Pang, J.: Pat: Towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. pp. 709–714. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). 10.1007/978-3-642-02658-4_59
157. Sun, J., Song, S., Liu, Y.: Model checking hierarchical probabilistic systems. In: Dong, J.S., Zhu, H. (eds.) Formal Methods and Software Engineering. pp. 388–403. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). 10.1007/978-3-642-16901-4_26
158. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)
159. Tracy, C.A.: Lecture note in First Passage of a One-Dimensional Random Walker. `http://www.math.ucdavis.edu/~tracy/courses/math135A/UsefullCourseMaterial/firstPassage.pdf` (2020)
160. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc. **s2-42**(1), 230–265 (1937). 10.1112/PLMS/S2-42.1.230
161. Tyrrell, M., Morris, J.M., Butterfield, A., Hughes, A.: A lattice-theoretic model for an algebra of communicating sequential processes. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4281, pp. 123–137. Springer (2006). 10.1007/11921240_9
162. Vanglabbeek, R., Smolka, S., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Information and Computation **121**(1), 59–80 (1995). 10.1006/inco.1995.1123

163. Ward, N., Hayes, I.: Applications of angelic nondeterminism. In: Australian Software Engineering Conference 1991: Engineering Safe Software; Proceedings. pp. 391–404. Australian Computer Society, Sydney, N.S.W. (1991). 10.3316/informit.553249589811640
164. Wei, K., Woodcock, J., Burns, A.: A timed model of circus with the reactive design miracle. In: Fiadeiro, J.L., Gnesi, S., Maggiolo-Schettini, A. (eds.) 8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010. pp. 315–319. IEEE Computer Society (2010). 10.1109/SEFM.2010.40
165. Williams, D.: Probability with Martingales. Cambridge University Press (1991)
166. Woodcock, J.: An introduction to refinement in Z. In: Prehn, S., Toetenel, W.J. (eds.) VDM '91 - Formal Software Development, 4th International Symposium of VDM Europe, Noordwijkerhout, The Netherlands, October 21-25, 1991, Proceedings, Volume 2: Tutorials. Lecture Notes in Computer Science, vol. 552, pp. 96–117. Springer (1991)
167. Woodcock, J.: A tutorial on the refinement calculus. In: Prehn, S., Toetenel, W.J. (eds.) VDM '91 - Formal Software Development, 4th International Symposium of VDM Europe, Noordwijkerhout, The Netherlands, October 21-25, 1991, Proceedings, Volume 2: Tutorials. Lecture Notes in Computer Science, vol. 552, pp. 79–140. Springer (1991). 10.1007/BFB0019996
168. Woodcock, J.: Using circus for safety-critical applications. In: Cavalcanti, A., Machado, P.D.L. (eds.) Proceedings of the 6th Brazilian Workshop on Formal Methods, WMF 2003, Campina Grande, Brazil, October 12-14, 2003. Electronic Notes in Theoretical Computer Science, vol. 95, pp. 3–22. Elsevier (2003). 10.1016/J.ENTCS.2004.04.003
169. Woodcock, J.: The miracle of reactive programming. In: Butterfield, A. (ed.) Unifying Theories of Programming, Second International Symposium, UTP 2008, Dublin, Ireland, September 8-10, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5713, pp. 202–217. Springer (2008). 10.1007/978-3-642-14521-6_12
170. Woodcock, J.: Engineering utopia - formal semantics for CML. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8442, pp. 22–41. Springer (2014). 10.1007/978-3-319-06410-9_3
171. Woodcock, J., Cavalcanti, A.: A Tutorial Introduction to Designs in Unifying Theories of Programming. In: Boiten, E.A., Derrick, J., Smith, G. (eds.) Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK, April 4-7, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2999, pp. 40–66. Springer (2004). 10.1007/978-3-540-24756-2_4
172. Woodcock, J., Cavalcanti, A.: A tutorial introduction to designs in unifying theories of programming. In: Boiten, E.A., Derrick, J., Smith, G. (eds.) Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK, April 4-7, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2999, pp. 40–66. Springer (2004). 10.1007/978-3-540-24756-2_4
173. Woodcock, J., Cavalcanti, A., Foster, S., Mota, A., Ye, K.: Probabilistic Semantics for RoboChart. In: Ribeiro, P., Sampaio, A. (eds.) Unifying Theories of Programming. pp. 80–105. Springer International Publishing, Cham (2019). 10.1007/978-3-030-31038-7_5
174. Woodcock, J., Cavalcanti, A., Foster, S., Oliveira, M., Sampaio, A., Zeyda, F.: UTP, Circus, and Isabelle. In: Bowen, J.P., Li, Q., Xu, Q. (eds.) Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion

of His 80th Birthday. Lecture Notes in Computer Science, vol. 14080, pp. 19–51. Springer (2023). 10.1007/978-3-031-40436-8_2

175. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Comput. Surv. **41**(4), 19:1–19:36 (2009). 10.1145/1592434.1592436
176. Woodcock, J., Stepney, S., Cooper, D., Clark, J.A., Jacob, J.: The certification of the mondex electronic purse to ITSEC level E6. Formal Aspects Comput. **20**(1), 5–19 (2008). 10.1007/S00165-007-0060-5
177. Wu, S., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic i/o automata. Theoretical Computer Science **176**(1), 1–38 (1997). 10.1016/S0304-3975(97)00056-X
178. Xia, L.y., Zakowski, Y., He, P., Hur, C.K., Malecha, G., Pierce, B.C., Zdancewic, S.: Interaction trees: representing recursive and impure programs in coq. Proc. ACM Program. Lang. **4**(POPL) (dec 2019). 10.1145/3371119
179. Ye, K., Cavalcanti, A., Foster, S., Miyazawa, A., Woodcock, J.: Probabilistic modelling and verification using RoboChart and PRISM. Softw. Syst. Model. **21**(2), 667–716 (2022). 10.1007/s10270-021-00916-8
180. Ye, K., Foster, S., Woodcock, J.: Automated reasoning for probabilistic sequential programs with theorem proving. In: Fahrenberg, U., Gehrke, M., Santocanale, L., Winter, M. (eds.) Relational and Algebraic Methods in Computer Science. pp. 465–482. Springer International Publishing, Cham (2021). 10.1007/978-3-030-88701-8_28
181. Ye, K., Foster, S., Woodcock, J.: Formally verified animation for robochart using interaction trees. Journal of Logical and Algebraic Methods in Programming **137**, 100940 (2024). 10.1016/j.jlamp.2023.100940
182. Ye, K., Woodcock, J.: RoboCertProb: Property Specification for Probabilistic RoboChart Models (2024), `https://arxiv.org/abs/2403.08136`
183. Ye, K., Woodcock, J., Foster, S.: Probabilistic relations for modelling epistemic and aleatoric uncertainty: semantics and automated reasoning with theorem proving. CoRR **abs/2303.09692** (2023). 10.48550/ARXIV.2303.09692
184. Yi, W., Larsen, K.G.: Testing probabilistic and nondeterministic processes. In: Proceedings of the IFIP TC6/WG6.1 Twelth International Symposium on Protocol Specification, Testing and Verification XII. p. 47–61. North-Holland Publishing Co., NLD (1992)
185. Zabih, R., McAllester, D.A., Chapman, D.: Non-deterministic Lisp with dependency-directed backtracking. In: Forbus, K.D., Shrobe, H.E. (eds.) Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987. pp. 59–65. Morgan Kaufmann (1987), `http://www.aaai.org/Library/AAAI/1987/aaai87-011.php`
186. Zeyda, F.: Reversible Computations in B. Ph.D. thesis, University of Teesside, Middlesbrough, Tees Valley, TS1 3BX, UK (Jul 2007)
187. Zeyda, F., Stoddart, B., Dunne, S.: A Prospective-Value Semantics for the GSL. In: ZB 2005: Formal Specification and Development in Z and B. LNCS, vol. 3455, pp. 187–202. Springer (Apr 2005). 10.1007/11415787_12