Probabilistic Semantics for RoboChart
A Weakest Completion Approach

Jim Woodcock[0000−0001−7955−2702]1, Ana Cavalcanti[0000−0002−0831−1976]1,
Simon Foster[0000−0002−9889−9514]1, Alexandre Mota[0000−0003−4416−8123]2, and
Kangfeng Ye[0000−0000−0000−0000]1

[1] University of York
{jim.woodcock,ana.cavalcanti,simon.foster,kangfeng.ye}@york.ac.uk
[2] Federal University of Pernambuco
acm@cin.ufpe.br

Abstract. We outline a probabilistic denotational semantics for the
RoboChart language, a diagrammatic, domain-specific notation for de-
scribing robotic controllers with their hardware platforms and operating
environments. We do this using a powerful (but perhaps not so well
known) semantic technique: He, Morgan, and McIver's *weakest comple-
tion semantics*, which is based on Hoare and He's Unifying Theories
of Programming. In this approach, we do the following: (1) start with
the standard semantics for a nondeterministic programming language;
(2) propose a new probabilistic semantic domain; (3) propose a forgetful
function from the probabilistic semantic domain to the standard semantic
domain; (4) use the converse of the forgetful function to embed the stan-
dard semantic domain in the probabilistic semantic domain; (5) demon-
strate that this embedding preserves program structure; (6) define the
probabilistic choice operator. Weakest completion semantics guides the
semantic definition of new languages by building on existing semantics
and, in this case, tackling a notoriously thorny issue: the relationship
between demonic and probabilistic choice. Consistency ensures that pro-
gramming intuitions, development techniques, and proof methods can
be carried over from the standard language to the probabilistic one. We
largely follow He et al., our contribution being an explication of the tech-
nique with meticulous proofs suitable for mechanisation in Isabelle/UTP.

Keywords: RoboChart language · Robotic controllers · Statecharts · Prob-
abilistic semantics · Relational calculus · Unifying Theories of Programming
(UTP) · Weakest completion semantics

1 Introduction

Modern robotics simulators enable fast prototyping of robots, using a virtual
simulation environment as a software creation and design tool. They provide re-
alistic, computer gaming-style, 3-D rendering of robots and environments with

physics engines to animate their movements authentically in automatically generated movies. Examples of such simulators include the Virtual Robot Experimentation Platform (V-REP) [54] and Webots [56].

One drawback of using these simulators as part of a principled development process is a lack of tool interoperability, with each simulator depending on a customised programming language or API. As a result of this, there are few possibilities for reuse of specifications and algorithms, and software development starts at a low-level with few abstractions. A notable exception to this is the Robot Modeling Language, ROBOTML [7], which targets the design of robotic applications, their simulation, and their deployment to multiple target execution platforms. The motivation for ROBOTML is to encourage a more abstract design process with explicit architectures, but there is no support for formal methods for verifying properties of these designs and architectures.

The RoboStar programme is developing a framework for modelling and simulating mobile and autonomous robots [49].[3] An early product of the research is the RoboChart language, a graphical domain-specific notation with a code generator that automatically produces mathematical models [41, 48, 42] in the notations of Communicating Sequential Processes (CSP) [52]. This enables the analysis of structural properties of RoboCharts: freedom from deadlock, livelock, and nondeterminism; it also supports the verification of more general untimed and timed properties by refinement checking [51]. RoboChart has an associated Eclipse-based development support environment, RoboTool [40], that enables graphical modelling and automatic generation of CSP scripts, and is integrates CSP's refinement model checker FDR4 [9].

RoboCalc's RoboSim language [4] provides a second graphical notation for developing simulations. A novel feature of RoboSim is the ability to verify simulations against their abstract RoboChart models. This ensures that the combination of models, simulations, deployed controllers, and hardware platforms refine the properties verified and validated by analysis and simulation. RoboChart and RoboSim support real time, discrete, continuous, and probabilistic properties; we consider only discrete probabilistic semantics in this paper. Our probabilistic models are essentially Markov Decision Processes (MDPs).

RoboChart and RoboSim have strong mathematical foundations, but they are also practical for industrial-strength robotic software engineering. This requires that they be attractive to practising engineers, but with additional powers to enable formal verification. The state of the art in industry is to use modelling techniques to specify the behaviours of robot controllers, but not the robotic hardware platform or the operating environment. Even at their most advanced, current industrial techniques use only simple state machines without formal semantics [3, 7, 47, 55]. Any abstract descriptions that are used guide simulation development, but without any relationship between abstract descriptions and implemented code. There is often a so-called "reality gap" between the state machine and simulation on the one hand, and the hardware platform on the

---

[3] The RoboStar programme includes a number of individual projects, including RoboCalc, which is developing a calculus of software engineering for robotic controllers.

other, and ad hoc adjustments must be made to get the robot working. It is for this reason that we have developed RoboChart with high-fidelity modelling capabilities, including continuous time and probabilism [42].[4] There is little motivation to keep the abstract state machine in line with these changes.

RoboChart has a probabilistic choice operator, but this cannot be supported by the translation into CSP, because the standard semantics and tools are not probabilistic. This paper presents an approach to developing a suitable imperative, reactive, probabilistic semantics for RoboChart. The method chosen to develop this semantics is the weakest completion semantics [25] approach based on Unifying Theories of Programming [32]. In this paper, we consider a semantics for the imperative action language for RoboChart. Elsewhere, we consider the use of this semantics to produce a sound translation from diagrams to mathematics, suitable for analysis by verification tools [10].

Our contribution is an explication of the weakest completion approach: a detailed analysis of this principle for developing semantics, enabling future application to RoboChart [42], a complex language with events, timed primitives, rich data types, a concurrency model based on synchronous and asynchronous communications, and shared variables. The inspiration for our work is precisely that of He, Morgan, and McIver [25]; but it is not straightforward to take their informal proof outlines and use them directly in a mechanical theorem prover: they are inspirational, but essentially informal. We present an abbreviation of our proof due to space limitations, but our proof steps are based on explicit axioms, lemmas, theorems, and inferences.

This paper has the following structure. In Sect. 2, we describe a few elements of the RoboChart language. In Sect. 3, we give an overview of Unifying Theories of Programming. In Sect. 4, we provide an interlude, where we discuss two predicate transformers: weakest preconditions and weakest prespecifications. In Sect. 5, we describe the technique of weakest completion semantics. In Sect. 6, we present a nondeterministic probabilistic programming language and its semantic domain. In Sect. 7, we describe the semantics of probabilistic choice and discuss how to combine distributions. In Sect. 8, we provide a detailed example: embedding nondeterministic choice in the probabilistic domain. In Sect. 9, we discuss related work on formalising probabilistic RoboCharts. Finally, in Sect. 10, we draw some conclusions from this research in progress and discuss future work.

2   RoboChart

We model robot controllers using RoboChart [42], a UML profile [21]. RoboChart models are Statecharts, a diagrammatic notation for defining behaviour [23]. State machines are part of the fabric of computing, recognisable in many forms

---

[4] The difficulty of transferring simulated experience into the real world, often called the "reality gap" [33], is a subtle but important discrepancy between reality and simulation that prevents simulated robotic experience from directly enabling effective real-world performance [2].

(including, for example, Mealy and Moore automata [38]) and they are widely accepted in the embedded-software industry as a design notation. Statecharts [23] extend these familiar diagrams with two features: hierarchy and orthogonal regions. Hierarchy is provided by allowing states themselves to contain state machines, where control flow resides in exactly one state: the so-called OR-decomposition of behaviour; orthogonal regions provide a complementary AND-decomposition, where control-flow can simultaneously reside in one position in each orthogonal region, fully independently. This parallel decomposition can lead to a reduction in complexity (the "conjunction as composition" paradigm [63]). RoboChart inherits both features: it has hierarchical state machines (which encourage modularity and reuse) and parallelism, arising for components defined by several state machines and for composite states, including durative actions.

In RoboChart, the behaviour of a robot is characterised by a state, in which it may execute a particular operation and react to events from its environment. RoboChart includes structures for describing robotic platforms and their controllers, with CSP-style synchronous communication between controllers and asynchronous communication between controllers and their hardware. It has constructs to specify time properties: budgets and deadlines for operations and events. Here, we consider only the probabilistic aspects of RoboChart. We isolate a language subset of flat nondeterministic state machines with a probabilistic choice node.

RoboTool [40] provides a graphical editor for RoboChart models and automatically generates mathematical definitions in CSP that precisely define their behaviour. RoboTool is closely coupled with the FDR model checker [9] to analyse these definitions.

We present two RoboChart models to illustrate the language. The first model is part of the controller for a tele-operated robot used to search an arena for evidence of a harmful chemical, using the receptor density algorithm [29]. The RoboChart is depicted in Fig. 1, which is a screen-shot from a RoboTool session. The robot controller uses a sensor to detect changes in the chemical composition of air over time. It reacts to gas anomalies depending on their nature and composition: with a yellow or a red light, a siren, and a flag to mark the location. The hardware includes a robot body, wheels, and motor, a main processor to detect gas and accept movement commands from an operator, and a microcontroller to manage the light, siren, and flag.

Our second example is of part of the controller for a foraging robot [57] (the corresponding RoboChart model and the results of its analysis are available online [50]).[5] The robot has an idealised randomising device with two states that are equally likely to occur; the device generates an outcome from a flip event in every time step.[6] The robot uses the device to decide whether to terminate or to

---

[5] The Statechart in this example is originally due to Jansen [35], but has been reinterpreted here as a robotics example.

[6] The semantics in this paper does not capture the real-time behaviour of RoboChart; however, every transition in an MDP takes unit time. When we develop the real-time probabilistic model, these two notions of time will be complementary, allowing
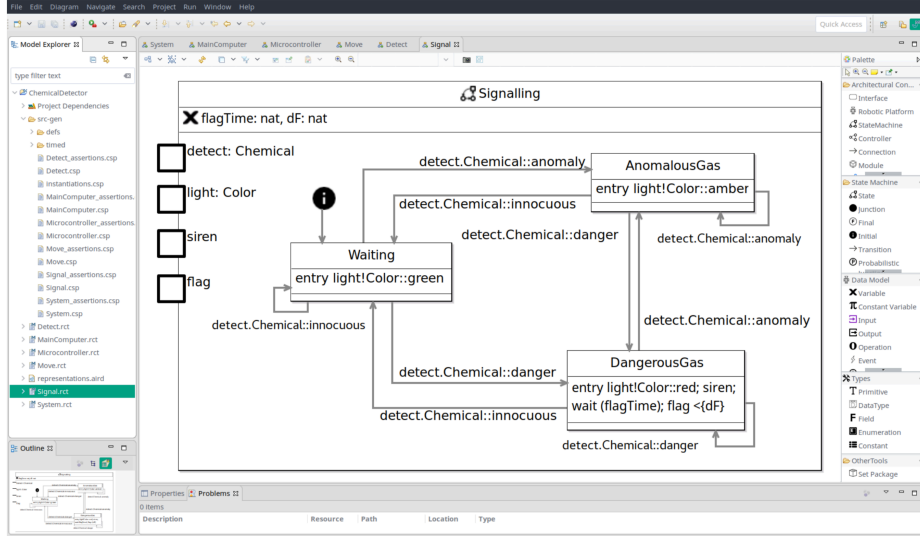
Fig. 1. Signalling state machine for Chemical Detector Robot (taken from [41]).

continue a particular activity (here, foraging for energy). The robot may choose to ignore the outcome of the device. Finally, the robot considers only a limited number of times whether to continue foraging. We call this number N and leave it loosely defined. Our simple modelling objective is to explore different values for N that give us a high probability of terminating.

We specify the behaviour of the device as a RoboChart model in Figure 2. One possibility in the FORAGE state is for the flip event to occur and the robot to remain in the FORAGE state; this models the robot ignoring the randomising device. The other possibility is available only if the number of choices has not been exhausted ($flips < N$). In this case, the robot controller proceeds to a probabilistic choice between two equally likely alternatives. One alternative is to move into the STOP state, which it signals with the stop event; the other alternative is to return to the FORAGE state, signalling this with the forage event. In both cases, the controller keeps track of the number of choices made. Note that, if in the FORAGE state $flips < N$, then the behaviour is nondeterministic: the robot controller might take either flip alternative. In the STOP state, only the flip event is possible, with a self-loop acting as a sink. A well-formed MDP must be free from deadlock (every state must have at least one outgoing transition), and anyway, this transition is needed because of the requirement that flip must occur in every time step, even when the controller has terminated.

Analysis of the generated model using PRISM [36] shows that the model is deadlock free, but that the STOP state is not always possible (the minimum probability of finally reaching STOP is zero) because the model could stay in the FORAGE state forever. Additionally, using experiment for N ranging from

---

events to be simultaneous with respect to the real-time clock, but ordered at the MDP level: super-dense time.
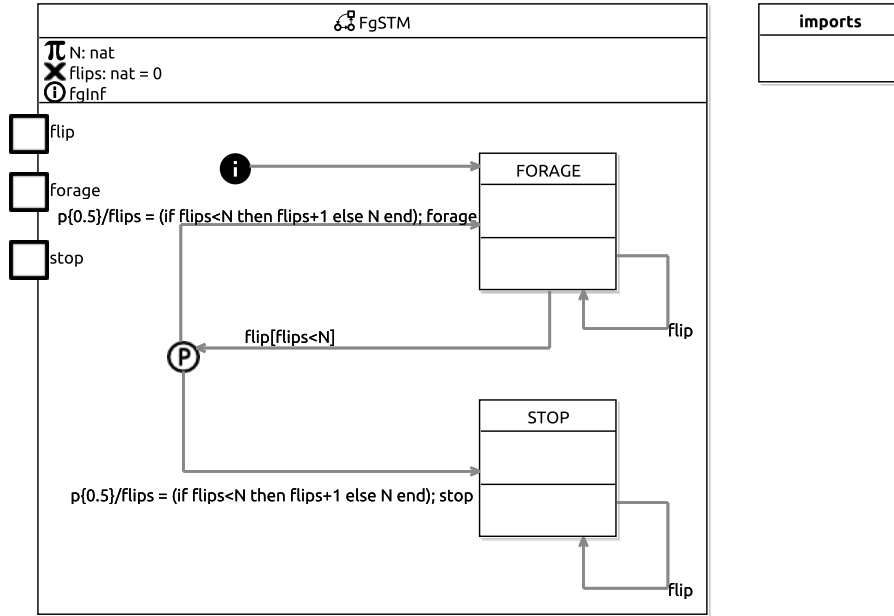
Fig. 2. RoboChart model of a foraging robot.

1 to 20, we can obtain the probability of finally reaching STOP, as shown in
Figure 3. For $N \geq 6$ the device will terminate with probability greater than 0.98.

3   Unifying Theories of Programming

There are tutorial introductions to UTP's theory of designs [59, 60], CSP [6], and
the use of Galois connections to link these theories [61]. UTP embodies Hehner's
predicative semantic paradigm [26–28], where programs are predicates [30]: a
program is identified with its meaning as a predicate, expressed pointwise. The-
ories describe the meaning of a computation as a relation between a before-state
and an after-state, and these relations form complete lattices ordered by refine-
ment. Several basic UTP theories are relevant to this paper.

1. A relational theory of a nondeterministic programming language (basically,
   Dijkstra's guarded command language (GCL)) supports reasoning about
   partial correctness [32, Chap.2].
2. A theory of designs, pre- and postcondition pairs, and an associated version
   of GCL supports reasoning about total correctness [32, Chap.3].
3. A theory of reactive processes with communication and concurrency [32,
   Chap.8].
4. A theory of CSP, essentially a predicative version of CSP's failures-divergences
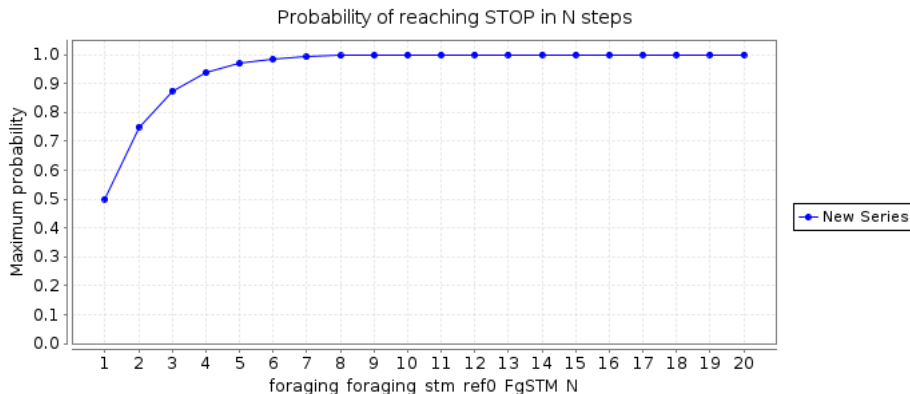   semantics [32, Chap.8].

Fig. 3. Model checking experiment for foraging robot.

5. Circus, a combination of CSP and Z [45, 46].

UTP has been used in a wide variety of applications, from specifying and reasoning about difficult program features [24], to specifying the semantic interfaces in a cyber-physical systems tool chain [12, 37].

A core concept is the embedding of the pre- and postconditions of designs in other semantic domains. For example, the theory of reactive designs [6] is an embedding of designs in the theory of reactive processes, which brings the familiar techniques of assertional reasoning and design calculi to reactive programming, allowing the creation of a reactive Hoare logic and a reactive weakest precondition calculus.

Unification in UTP is in three dimensions:

1. Programming paradigms: comparing and combining different language features in a coherent way.
2. Levels of abstraction: refining different design concepts.
3. Methods of presentation: moving between denotational, algebraic, and operational semantics.

There are four principal mechanisms for unification:

1. Subset embeddings, e.g., total and partial correctness (designs and relations) [58].
2. Weakest completion semantics, e.g., probabilistic and standard programs, as explained in Sect. 5.
3. Galois connections, e.g., imperative programs and reactive processes [5, 58].
4. Parametrised theories, e.g., reactive processes and hybrid processes [15].

We have implemented UTP in the Isabelle/HOL theorem prover [43]. The resulting proof tool is Isabelle/UTP [18, 16, 19, 17, 13]. Our research aim is a sound automated theorem prover, built in Isabelle/UTP, for diagrammatic descriptions

of reactive, timed, probabilistic controllers for robotics and autonomous systems. We note that it is not straightforward to take the informal proof outlines in [25] and use them directly in a mechanical theorem prover. In this paper, our objective is to explicate the weakest completion semantic technique and in doing so, to explore how to mechanise it.

4 Weakest Preconditions and Prespecifications

In this section, we review Dijkstra's weakest precondition predicate transformer [8] and its generalisation, the weakest prespecification [31].

A typical stage in program development is to prove that a program meets its specification. Schematically, this is a problem in three variables: the program and its specification, which is a precondition and a postcondition. The weakest precondition calculus fixes two of these variables, the program and the postcondition, and calculates the third, the precondition, as an extreme value.

$$
\begin{aligned}
& [\, P \Rightarrow s \Rightarrow q' \,] \\
=\ & [\, s \Rightarrow P \Rightarrow q' \,] \\
=\ & [\, s \Rightarrow \forall\, v' \bullet P \Rightarrow q' \,] \\
=\ & [\, s \Rightarrow \neg\, \exists\, v' \bullet P \wedge \neg\, q' \,] \\
=\ & [\, s \Rightarrow \neg\, \exists\, v_0 \bullet P[v_0/v'] \wedge \neg\, q_0 \,] \\
=\ & [\, s \Rightarrow \neg\, (P \ ;\ \neg\, q) \,]
\end{aligned}
$$

(This derivation is a small variation on that in [32, Chap.2].) UTP's relational calculus is alphabetised: names are an important part of the meaning. Where we think that it might help, we have emphasised which names occur in each predicate by using parameters. This also streamlines substitution.

Formally, given program P and postcondition q, the problem is to find the weakest precondition s (in terms of P and q) such that P refines $(s \Rightarrow q')$. P refines S, written $P \sqsupseteq S$, just in case $\forall\, v, v' \bullet P \Rightarrow S$, where v and v' denote the before and after states. In UTP, universal closure over an alphabet is abbreviated by brackets, so refinement is defined as $[\, P \Rightarrow S \,]$.

So the predicate $\neg\, (P \ ;\ \neg\, q)$ is the weakest precondition for execution of P to guarantee postcondition q (written as $P\ \mathbf{wp}\ q$). Here, $P \ ;\ Q$ is the relational composition of P and Q, [32, Chap.2] defined by $P(s, t') \ ;\ Q(t, u') \ =\ \exists\, t_0 \bullet P(s, t_0) \wedge Q(t_0, u')$. Our minor generalisation accounts for its use with non-homogeneous relations later in the paper. Note that there is a modality here, between necessity and possibility. Compare the definition of weakest precondition with its dual, the conjugate weakest precondition [62]: $P\ \overline{\mathbf{wp}}\ q = \neg\, (P\ \mathbf{wp}\ \neg\, q)$.

$$
\begin{aligned}
& P\ \overline{\mathbf{wp}}\ q \\
=\ & \neg\, (P\ \mathbf{wp}\ \neg\, q) \\
=\ & \neg\,\neg\, (P \ ;\ \neg\,\neg\, q) \\
=\ & P \ ;\ q
\end{aligned}
$$

$$= \quad \exists\, v' \bullet P \wedge q'$$

During the derivation of weakest precondition, we see that $(P\ \mathbf{wp}\ q) = \forall\, v' \bullet P \Rightarrow q'$. This has universal force: every final state v' of the program P must satisfy q. Its conjugate has existential force: some execution of P satisfies q.

Now we move on to a generalisation of weakest precondition: the weakest prespecification. First, we define relational converse $P^{\smile}(s, t') = P(s', t)$. For example, the converse of an assignment is calculated as follows:

$$\begin{aligned}
& (x := x + 1)^{\smile} \\
=\ & (x' = x + 1)^{\smile} \\
=\ & (x = x' + 1) \\
=\ & (x' = x - 1) \\
=\ & x := x - 1
\end{aligned}$$

Weakest prespecifications generalise weakest preconditions from conditions to relations: given specifications Y and K, find the weakest specification X (in terms of Y and K), such that Y is refined by $X\ ;\ K$. We proceed in a similar way to our previous calculation for the weakest precondition: first, isolate X on the stronger side of the refinement relation, so that we can conclude we have a weakest solution; then rewrite the other side of the relation so that we can use the definition of sequential composition. Our derivation is (as far as we know) novel in the literature. There is a strong analogy between the weakest precondition and weakest prespecification predicate transformers; see Appendix A for further motivation.

$$\begin{aligned}
& X\ ;\ K\ \sqsupseteq\ Y \\
=\ & \{\,\text{law of refinement: } (P\ ;\ Q \sqsupseteq R) = (P[x_0/x'] \wedge Q[x_0/x] \sqsupseteq R)\,\} \\
& X[s_0/s'] \wedge K[s_0/s]\ \sqsupseteq\ Y \\
=\ & \{\,\text{law of refinement: } (P \wedge Q \sqsupseteq R) = (P \sqsupseteq Q \Rightarrow R)\,\} \\
& X[s_0/s']\ \sqsupseteq\ K[s_0/s] \Rightarrow Y \\
=\ & \{\,\text{change of variables: } s_0, s' \mapsto s', s_0\,\} \\
& X\ \sqsupseteq\ K[s', s_0/s, s'] \Rightarrow Y[s_0/s'] \\
=\ & \{\,\text{propositional calculus: contraposition}\,\} \\
& X\ \sqsupseteq\ \neg\, Y[s_0/s'] \Rightarrow \neg\, K[s', s_0/s, s'] \\
=\ & \{\,\text{definition of converse}\,\} \\
& X\ \sqsupseteq\ \neg\, Y[s_0/s'] \Rightarrow \neg\, K^{\smile}[s_0/s] \\
=\ & \{\,\text{predicate calculus: narrow scope of } s_0\,\} \\
& X\ \sqsupseteq\ \forall\, s_0 \bullet \neg\, Y[s_0/s'] \Rightarrow \neg\, K^{\smile}[s_0/s] \\
=\ & \{\,\text{predicate calculus: De Morgan}\,\} \\
& X\ \sqsupseteq\ \neg\, \exists\, s_0 \bullet \neg\, Y[s_0/s'] \wedge K^{\smile}[s_0/s] \\
=\ & \{\,\text{definition sequential composition}\,\}
\end{aligned}$$

$$X \;\sqsupseteq\; \neg\,(\neg\, Y \;;\; K^{\smallsmile})$$

So, X must be at least as strong as $\neg\,(\neg\, Y \;;\; K^{\smallsmile})$. We read this as "The weakest prespecification of K through Y", and denote it by Y/K (the weak inverse of the function $\lambda\, X \bullet X \;;\; K$). The weakest prespecification forms one adjoint of a Galois connection, with sequential composition as the other adjoint; that is: $(X \;;\; K \sqsupseteq Y) \;=\; (X \sqsupseteq Y/K)$. We give an example of calculating a leading assignment: we want to implement the assignment $x := 2$ as the sequential composition $(X;\; x := x + 1)$. That is, X is the weakest prespecification of $x := x + 1$ through $x := 2$.

$$
\begin{aligned}
& x := 2 / x := x + 1 \\
=\ & \neg\,(\neg\, x := 2 \;;\; (x := x + 1)^{\smallsmile}) \\
=\ & \neg\,(x' \neq 2 \;;\; (x' = x + 1)^{\smallsmile}) \\
=\ & \neg\,(x' \neq 2 \;;\; x = x' + 1) \\
=\ & \neg\,(\exists\, x_0 \bullet x_0 \neq 2 \wedge x_0 = x' + 1) \\
=\ & \neg\,(x' + 1 \neq 2) \\
=\ & x' = 1 \\
=\ & x := 1
\end{aligned}
$$

5   Weakest Completion Semantics

We now turn to weakest completion semantics [25], where we lift standard designs to probabilistic designs. Our objective is to give semantics to a nondeterministic probabilistic programming language that is consistent with a similar standard programming language: the only difference being the presence or absence of a probabilistic choice operator. Consistency is important to make sure that programming intuitions, development techniques, and proof methods can be carried over, as far as possible, from the standard language to the probabilistic one.

One way to achieve consistency is to extend the standard semantics to the probabilistic one in a controlled way. He et al.'s work [25] develops a semantic method to extend theories of programming automatically, as far as possible. Their method is to make only a few explicit assumptions and then generate a semantics by following a set of principles. They have applied their technique to two semantics for the nondeterministic programming language: a relational semantics and a predicate-transformer one.

He et al. propose the following procedure:

1. Start from the semantics for the nondeterministic programming language.
2. Propose a probabilistic semantic domain.
3. Propose a mapping from the probabilistic semantics to the standard semantics to relate computations of probabilistic programs to computations of standard programs.

4. Use this mapping to induce automatically an embedding of programs over the standard semantics: the technique is to consider the weakest completion of a sub-commuting diagram expressing refinement, rather than equality.
5. Determine its defining algebraic characteristics of the new language.

6 Probabilistic Programs

Our standard and our probabilistic programming languages have identical syntax, except that the latter has the addition of a probabilistic choice operator: $P \oplus_r Q$. This is a choice between P with probability r, and Q with probability $1 - r$. The syntax of this language is given in

$$
\begin{array}{llll}
P & ::= & \bot & \text{abort} \\
  & \mid & \mathbb{II} & \text{skip} \\
  & \mid & x := e & \text{assignment} \\
  & \mid & P \lhd b \rhd Q & \text{conditional} \\
  & \mid & P \sqcap Q & \text{nondeterminism} \\
  & \mid & P \oplus_r Q & \text{probabilism} \\
  & \mid & P \ ; \ Q & \text{sequence} \\
  & \mid & \mu X \bullet P(X) & \text{recursion}
\end{array}
$$

This nondeterministic probabilistic language is a suitable target for probabilistic RoboChart [10]. The semantic domain for the language without probabilistic choice is the UTP theory of designs. This theory allows the boolean observation of a program starting (ok) and of it terminating (ok'). A design with precondition $p(s)$ and postcondition $R(s, s')$ is a pair of predicates $(p(s) \vdash R(s, s'))$, which is defined as the single relation $ok \wedge p(s) \Rightarrow ok' \wedge R(s, s')$. This is a statement of total correctness: if the program is started in a state satisfying its precondition, then it will terminate and when it does, its postcondition will be satisfied. The vectors of variables $s, s' : S$ represent the initial and final states of ordinary program variables, which are modelled as mappings from the names of program variables to their values. The UTP semantics for this nondeterministic programming language is well known [32, Chap.3].

$$
\begin{aligned}
\bot &= (\textbf{false} \vdash \textbf{true}) \\
\mathbb{II} &= (\textbf{true} \vdash s' = s) \\
x := e &= (\textbf{true} \vdash s' = s[e/x]) \\
P \sqcap Q &= P \vee Q \\
P \lhd b \rhd Q &= (b \wedge P) \vee (\neg \, b \wedge Q) \\
P \ ; \ Q &= \exists \, ok_0, s_0 \bullet P[ok_0, s_0/ok', s'] \wedge Q[ok_0, s_0/ok, s] \\
\mu X \bullet P(X) &= \bigsqcap \{ X \mid X \sqsupseteq P(X) \}
\end{aligned}
$$

Next, we consider the probabilistic semantic domain. Let the state space be S. Let the set of probabilistic distributions over S be the set of total functions to

probabilities: $PROB = S \rightarrow [0, 1]$. The probabilities in a discrete distribution f sum to 1: $(\sum s : S \bullet f(s)) = 1$, for $f \in PROB$.

A probabilistic design is defined as $p \vdash Q$, where the alphabet of p is $\{s\}$ and the alphabet of Q is $\{s, prob'\}$, for $s \in S$ and $prob' \in PROB$.

The relationship between standard and probabilistic programs is most easily understood as an abstraction from the probabilistic semantic domain: a mapping ρ that forgets the probabilities and replaces them by possibilities. We define ρ as a design with a non-homogeneous alphabet: $\{ok, prob, ok', s'\}$, where ok and ok' design observations about initiation and termination, $prob : PROB$ is a discrete probability distribution, and $s' : S$.

$$\rho \; \widehat{=} \; (\textbf{true} \vdash prob(s') > 0)$$

This non-homogeneous design is a forgetful function: the *probability* of arriving in state s' is $prob(s')$; this is replaced by the *possibility* of arriving in that state: $prob(s') > 0$.

Note now that $P \; ; \; \rho$ is a standard design if P is a probabilistic design.

Using this idea, for probabilistic design P and standard design D, we construct the following sub-commuting diagram

where $P \; ; \; \rho \sqsupseteq D$. This is an inequality in three variables, two of which we already know: D and ρ. So, we calculate P using the weakest prespecification of D wrt ρ. The result is the weakest probabilistic design related to the standard design D. We introduce the following definition: for any standard design D, define $\mathcal{K}(D) \widehat{=} D/\rho$ as its embedding in the probabilistic world.

We need to prove that this embedding really does produce probabilistic designs, which we do in the following theorem. For any subset X of S, define $f(X) = \sum s : X \bullet f(s)$, for any probability distribution function f. Furthermore, for any relation R with alphabet $\{s, s'\}$ (both in S), define $f(R) = f(\{ s' \mid R \})$.[7] If X and Y are disjoint sets then

$$(f(X \cup Y) = 1) \; = \; (f(X) = 1 - f(Y))$$

A corollary is that

$$(f(R) = 1) \; = \; (f(\neg R) = 0)$$

---

[7] Note that if f is a probability distribution function, then lifting f from states to a relation on states results in an alphabetised definition: $f(R)$ has s as a free variable (s' is bound by the set comprehension). If we now fix s, then we get the probability sum for the image of s through R. Note that $prob'(R)$ is also an alphabetised expression, this time with alphabet $\{s, prob'\}$. Thus $prob'(R) = 1$, which encounter next, is a suitable candidate for the postcondition of a probabilistic design.

Theorem 1 (Embedded standard designs are probabilistic designs).

$$\mathcal{K}(p(s) \vdash R(s, s')) \;=\; (p(s) \vdash prob'(R) = 1)$$

Proof. Start by simplifying the definition of $\mathcal{K}$ by pushing the weakest prespecification operator into the postcondition. Note that the law we use requires that the design is **H3** *healthy [32, Chap.3]: its precondition must not mention any variables from the after-state.[8] This assumption is discharged here. Our account of this law is novel, but we do not present it in this paper.*

$$\mathcal{K}(p \vdash R)$$
$=$ { definition of $\mathcal{K}$ }
$$(p \vdash R)/(\mathbf{true} \vdash prob(s') > 0)$$
$=$ $\left\{ \begin{array}{l} \text{weakest design prespecification,} \\ P \vdash R \text{ is } \mathbf{H3} \text{ implies } (P \vdash Q)/(\mathbf{true} \vdash R) = (P \vdash Q/R) \end{array} \right\}$
$$p \vdash R/(prob(s') > 0)$$

Now show that $R/(prob(s') > 0) = (prob'(R) = 1)$

$$R/(prob(s') > 0)$$
$=$ { definition weakest prespecification }
$$\neg\, (\neg\, R \; ; \; (prob(s') > 0)\,\breve{}\,)$$
$=$ { converse }
$$\neg\, (\neg\, R \; ; \; prob'(s) > 0)$$
$=$ { definition sequential composition }
$$\neg\, (\exists\, s_0 \bullet \neg\, R[s_0/s'] \wedge prob'(s_0) > 0)$$
$=$ { predicate calculus }
$$\forall\, s' \bullet \neg\, R \Rightarrow prob'(s') = 0$$
$=$ { property of lifted probability distribution function }
$$prob'(\neg\, R) = 0$$
$=$ { property of lifted probability distribution function }
$$prob'(R) = 1$$

    Our next task is to prove that the embedding is a homomorphism on the structure of standard programs. As a result, most of the algebraic laws that hold

---

[8] This subclass of specification contracts is sometimes known as "normal" designs [22, 14]. The theory of reactive designs [6], mentioned on page 7, is not an embedding of normal designs, since a reactive design can mention the after-value of the trace variable in its precondition. To see this, consider the precondition in the reactive design for the CSP process $a \to CHAOS$. This process can diverge, but only after an a-event. The process's precondition records the circumstances under which the process will not diverge: $\neg\, tr \,^\frown\, \langle a \rangle \leq tr'$. In words: "Don't press the a button, or else we crash!"

in the standard semantic framework remain valid in the probabilistic model. We give two example cases in the proof of the homomorphism: the embedding of assignment (here) and nondeterminism (in Sect. 8).

Lemma 1 (Embedded assignment).

$$\mathcal{K}(x := e) \;=\; (\mathbf{true} \vdash prob'(s[e/x]) = 1)$$

Proof.

$$\begin{aligned}
&\mathcal{K}(x := e) \\
=\; &\{\,\text{semantics of standard assignment}\,\} \\
&\mathcal{K}(\mathbf{true} \vdash s' = s[e/x]) \\
=\; &\{\,\text{theorem 1}\,\} \\
&\mathbf{true} \vdash prob'(s' = s[e/x]) = 1 \\
=\; &\{\,\text{function lifted to relation: } prob(R(s,s')) = prob(\{\,s' \mid R(s,s')\,\})\,\} \\
&\mathbf{true} \vdash prob'(\{\,s' \mid s' = s[e/x]\,\}) = 1 \\
=\; &\{\,\text{function lifted to set: } prob(X) = \sum s : X \bullet prob(s)\,\} \\
&\mathbf{true} \vdash (\sum s : \{\,s' \mid s' = s[e/x]\,\} \bullet prob'(s)) = 1 \\
=\; &\{\,\text{set one-point rule: } \{\,x \mid x = e\,\} = \{e\}\,\} \\
&\mathbf{true} \vdash (\sum s : \{s[e/x]\} \bullet prob'(s)) = 1 \\
=\; &\{\,\text{singleton sum: } (\sum x : \{e\}) = e\,\} \\
&\mathbf{true} \vdash prob'(s[e/x]) = 1
\end{aligned}$$

In the next section, we consider how to combine probability distributions in order to support probabilistic and nondeterministic choice operators.

7 Probabilistic Choice and Combining Distributions

We start with a motivating example of combining probability distributions: expressing multiway probabilistic choice as a combination of binary probabilistic choices. This leads us to propose a semantics for probabilistic choice in the spirit of UTP's parallel-by-merge operator. We consider how to decompose a probability distribution into two distributions combined by probabilistic choice. This leads to two projection functions, one for each operand. We conclude the section with three lemmas that will be used in the case for nondeterministic choice in the proof of $\mathcal{K}$ being a homomorphism. These lemmas provide witnesses for the decomposition required.

Consider a multiway probabilistic choice, as found in the Reactive Modules formalism, [1] used by the probabilistic model checker PRISM [36]:

$$\alpha : (s := 0) \;+\; (1 - (\alpha + \beta)) : (s := 1) \;+\; \beta : (s := 2)$$

Here, each assignment is labelled by a probability and these probabilities sum to 1. How can we express this using binary probabilistic choice? One simple solution uses two operators:

$$(s := 0 \oplus_{\alpha/(1-\beta)} s := 1) \oplus_{1-\beta} s := 2$$

To show that this is a solution, note that the assignment $s := 0$ is chosen with probability $(1 - \beta) \times (\alpha/(1 - \beta)) = \alpha$; $s := 2$ is chosen with probability β; and $s := 1$ must be chosen with the remaining probability, which is $1 - (\alpha + \beta)$. A slightly more complicated solution uses three operators:

$$(s := 0 \oplus_{\alpha+\beta} s := 1) \oplus_{\alpha/(\alpha+\beta)} (s := 1 \oplus_{1-(\alpha+\beta)} s := 2)$$

Analysing probabilities once more gives us $(\alpha/(\alpha + \beta)) \times (\alpha + \beta)$ for $s := 0$; $(1 - (\alpha/(\alpha + \beta))) \times (\alpha + \beta)$ for $s := 2$; and $1 - (\alpha + \beta)$ for $s := 1$. Simple arithmetic proves that we got this right.

These examples show how distributions are combined as we move the binary operator to its multi-way cousin. In the first example, we are combining the following two distributions:[9]

$$0.prob = \{(s = 0) \mapsto \alpha/(1 - \beta), (s = 1) \mapsto 1 - (\alpha/(1 - \beta))\}$$
$$1.prob = \{(s = 2) \mapsto 1\}$$

and we are combining them in the ratio given by the outermost choice operator: $1 - \beta$:

$$
\begin{aligned}
& prob' \\
= \quad & (1 - \beta) \times 0.prob + (1 - (1 - \beta)) \times 1.prob \\
= \quad & (1 - \beta) \times 0.prob + \beta \times 1.prob \\
= \quad & (1 - \beta) \times \{(s = 0) \mapsto \alpha/(1 - \beta), (s = 1) \mapsto 1 - (\alpha/(1 - \beta))\} \\
& + \beta \times \{(s = 2) \mapsto 1\} \\
= \quad & \{(s = 0) \mapsto (1 - \beta) \times (\alpha/(1 - \beta)), (s = 1) \mapsto (1 - \beta) \times (1 - (\alpha/(1 - \beta)))\} \\
& + \{(s = 2) \mapsto \beta \times 1\} \\
= \quad & \{(s = 0) \mapsto \alpha, (s = 1) \mapsto 1 - (\alpha + \beta), (s = 2) \mapsto \beta\}
\end{aligned}
$$

To formalise this, define the merge of two distributions, $0.prob$ and $1.prob$, to form distribution $prob'$ as: $M_r = (prob' = r \times 0.prob + (1 - r) \times 1.prob)$, for some probability ratio r. We use this in the definition of an operator inspired by UTP's parallel-by-merge [32, Chap.7] to combine the probability distributions described by two postconditions:

$$P(prob') \parallel_{M_r} Q(prob') = P(0.prob') \wedge Q(1.prob') \; ; \; M_r$$

---

[9] The notation $0.prob$ and $1.prob$ come from the separating simulation operator in UTP's parallel-by-merge [32, Sect.7.2], which is being used here to combine probability distributions.

This operator may be applied equally well to a design, rather than an individual postcondition, without any confusion.

With this operator, we now have a semantics for probabilistic choice:

$$P \oplus_r Q \;=\; P \parallel_{M_r} Q$$

The meaning of probabilistic choice is clearly compositional: if we have the meaning of P and Q, then we can find the meaning of $P \oplus_r Q$. But we can also think about the decomposition of a probabilistic program into the probabilistic choice between two subprograms. Suppose that we have two sets of states A and B, such that $A \cup B = S$ and a probabilistic ratio $0 < r < 1$ (to ensure $1/r$ and $1/(1-r)$ are well defined).[10] In this case we can unravel the merge of two distributions if $0.prob(A) = 1$ and $1.prob(B) = 1$. To do this, we define the projections.[11]

$$\mathcal{F}(prob', A, B, r) = (1/r) \times ((A \setminus B) \triangleleft prob') + ((A \cap B) \triangleleft prob')$$
$$\mathcal{G}(prob', A, B, r) = (1/(1-r)) \times ((B \setminus A) \triangleleft prob') + ((A \cap B) \triangleleft prob')$$

For $\mathcal{F}(prob', A, B, r)$ to be a distribution, we need its domain to sum to unity; that is, $\mathcal{F}(prob', A, B, r)(A) = 1$. These projections satisfy our merge predicate, and in that sense provide a joint witness.

Lemma 2 (Merge witnesses). *For $0 < r < 1$, $\mathcal{F}(prob', A, B, r)(A) = 1$, and $\mathcal{G}(prob', A, B, r)(B) = 1$,*

$$M_r[\mathcal{F}(prob', A, B, r), \mathcal{G}(prob', A, B, r)/0.prob, 1.prob]$$

Proof.

$$M_r[\mathcal{F}(prob', A, B, r), \mathcal{G}(prob', A, B, r)/0.prob, 1.prob]$$

$=$ { definition of M_r }

$$\left(prob' = r \times 0.prob + (1-r) \times 1.prob\right) \left[\begin{matrix} \mathcal{F}(prob', A, B, r)/0.prob \\ \mathcal{G}(prob', A, B, r)/1.prob \end{matrix}\right]$$

$=$ { substitution }

$$prob' = r \times \mathcal{F}(prob', A, B, r) + (1-r) \times \mathcal{G}(prob', A, B, r)$$

$=$ { definitions of $\mathcal{F}$ and $\mathcal{G}$ }

$$prob' = r \times ((1/r) \times ((A \setminus B) \triangleleft prob') + ((A \cap B) \triangleleft prob'))$$
$$+(1-r) \times ((1/(1-r)) \times ((B \setminus A) \triangleleft prob') + ((A \cap B) \triangleleft prob'))$$

$=$ { function scaling: $x \times (f + g) = x \times f + x \times g$, arithmetic }

$$prob' = ((A \setminus B) \triangleleft prob') + r \times ((A \cap B) \triangleleft prob')$$
$$+((B \setminus A) \triangleleft prob') + (1-r) \times ((A \cap B) \triangleleft prob')$$

---

[10] This case analysis is present in [25], although its purpose is not explained there).

[11] The expression $S \triangleleft R$ is Z's domain restriction operator [53, p.98]: the domain restriction $S \triangleleft R$ of a relation R to a set S relates x to y if and only if R relates x to y and x is a member of S.

$$
\begin{aligned}
=\ & \{\,\text{function scaling: } (x+y) \times f = x \times f + y \times f,\ \text{arithmetic}\,\} \\
& prob' = ((A \setminus B) \lhd prob') + ((A \cap B) \lhd prob') + ((B \setminus A) \lhd prob') \\
=\ & \{\,\text{function addition: } X \cap Y = \varnothing \ \Rightarrow\ (X \cup Y) \lhd f = (X \lhd f) + (Y \lhd f)\,\} \\
& prob' = ((A \setminus B) \cup (A \cap B) \cup (B \setminus A)) \lhd prob' \\
=\ & \{\,\text{assumption: } A \cup B = S\,\} \\
& true
\end{aligned}
$$

Now we state two lemmas that ensure that our two projections are probability distributions that sum to unity.

Lemma 3 (Total witness 1). *Let $p(A \setminus B) = \alpha$ and $p(A \cap B) = 1 - (\alpha + \beta)$; then*

$$
\mathcal{F}(p, A, B, \alpha/(\alpha + \beta))(A) = 1
$$

Proof.

$$
\begin{aligned}
& \mathcal{F}(p, A, B, \alpha/(\alpha + \beta))(A) \\
=\ & \{\,\text{definition } \mathcal{F}\,\} \\
& ((1/(\alpha/(\alpha + \beta)))) \times ((A \setminus B) \lhd p) + (A \cap B) \lhd p)(A) \\
=\ & \{\,\text{arithmetic, function scaling: } (f + g)(X) = f(X) + g(X)\,\} \\
& ((\alpha + \beta)/\alpha) \times (A \setminus B) \lhd p(A) + (A \cap B) \lhd p(A) \\
=\ & \{\,\text{functions: } X \subseteq Y \ \Rightarrow\ X \lhd f(Y) = f(X)\,\} \\
& ((\alpha + \beta)/\alpha) \times p(A \setminus B) + p(A \cap B) \\
=\ & \{\,\text{assumptions: } p(A \setminus B) = \alpha \text{ and } p(A \cap B) = 1 - (\alpha + \beta)\,\} \\
& ((\alpha + \beta)/\alpha) \times \alpha + 1 - (\alpha + \beta) \\
=\ & \{\,\text{arithmetic}\,\} \\
& \alpha + \beta + 1 - \alpha - \beta \\
=\ & \{\,\text{arithmetic}\,\} \\
& 1
\end{aligned}
$$

Lemma 4 (Total witness 2). *Let $p(B \setminus A) = \beta$ and $p(A \cap B) = 1 - (\alpha + \beta)$; then*

$$
\mathcal{G}(p, A, B, \alpha/(\alpha + \beta))(B) = 1
$$

Proof. Similar to Lemma 3.

The main result that we want to present in this paper is stated and proved in the next section.

8 Nondeterministic Choice

In this section, we prove the case for nondeterministic choice in the homomorphism theorem. Nondeterministic choice can be used in the top-down develop-

ment of a program to abstract from detail, including specific details of a probabilistic choice. So $\mathcal{K}$ should distribute through nondeterministic choice in the following way:

$$\mathcal{K}(D_0 \sqcap D_1) \;=\; \exists\, r : [0,1] \bullet (\,\mathcal{K}(D_0) \parallel_{M_r} \mathcal{K}(D_1)\,)$$

Refinement to a particular probabilistic choice $\oplus_\alpha$ would then follow by strengthening the result, choosing α for r. In this section, we prove one half of this result, omitting the other half only because we lack space.

The next lemma simplifies the embedding of nondeterministic choice.

Lemma 5 (Embedded nondeterministic choice).

$$\mathcal{K}((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) \;=\; (p_0 \wedge p_1 \vdash prob'(Q_0 \vee Q_1) = 1)$$

Proof.

$$
\begin{aligned}
&\mathcal{K}((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) \\
=\;&
\left\{
\begin{array}{l}
\text{designs closed under nondeterministic choice:} \\
((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) = (p_0 \wedge p_1 \vdash Q_0 \vee Q_1)
\end{array}
\right\} \\
&\mathcal{K}(p_0 \wedge p_1 \vdash Q_0 \vee Q_1) \\
=\;& \{\, \text{definition of } \mathcal{K}\!: \mathcal{K}(p \vdash Q) = (p \vdash prob'(Q) = 1) \,\} \\
&p_0 \wedge p_1 \vdash prob'(Q_0 \vee Q_1) = 1
\end{aligned}
$$

Now we show half of our result: that the embedding is a weakening homomorphism for nondeterministic choice. This means that as $\mathcal{K}$ distributes through nondeterminism, it produces a weaker predicate.

Theorem 2 (Nondeterminism embedding weakening).

$$\mathcal{K}((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) \;\sqsupseteq\; \exists\, r : [0,1] \bullet (\,\mathcal{K}(p_0 \vdash Q_0) \parallel_{M_r} \mathcal{K}(p_1 \vdash Q_1)\,)$$

Proof.

$$
\begin{aligned}
&\mathcal{K}((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) \\
=\;&
\left\{
\begin{array}{l}
\text{lemma 5: embedded nondeterministic choice:} \\
\mathcal{K}((p_0 \vdash Q_0) \sqcap (p_1 \vdash Q_1)) \;=\; p_0 \wedge p_1 \vdash prob'(Q_0 \vee Q_1) = 1
\end{array}
\right\} \\
&p_0 \wedge p_1 \vdash prob'(Q_0 \vee Q_1) = 1 \\
\Rightarrow\;&
\left\{
\begin{array}{l}
\alpha := p(Q_0 \setminus Q_1) \wedge \beta := p(Q_1 \setminus Q_0) \Rightarrow p(Q_0 \cap Q_1) = 1 - (\alpha + \beta) \\
\text{lemma 3: total witness 1, lemma 4: total witness 2}
\end{array}
\right\} \\
&p_0 \wedge p_1 \vdash \mathcal{F}(prob', Q_0, Q_1, \alpha/(\alpha+\beta))(Q_0) = 1 \\
&\qquad\qquad \wedge \mathcal{G}(prob', Q_0, Q_1, \alpha/(\alpha+\beta))(Q_1) = 1 \\
=\;& \{\, \text{lemma 2: merge witnesses} \,\} \\
&p_0 \wedge p_1 \vdash \mathcal{F}(prob', Q_0, Q_1, \alpha/(\alpha+\beta))(Q_0) = 1 \\
&\qquad\qquad \wedge \mathcal{G}(prob', Q_0, Q_1, \alpha/(\alpha+\beta))(Q_1) = 1 \\
&\qquad\qquad \wedge M_{\alpha/(\alpha+\beta)}
\begin{bmatrix}
\mathcal{F}(prob', A, B, \alpha/(\alpha+\beta))/0.prob \\
\mathcal{G}(prob', A, B, \alpha/(\alpha+\beta))/1.prob
\end{bmatrix}
\end{aligned}
$$

$$= \left\{ \begin{array}{l} \text{existential introduction: } r := \alpha/(\alpha + \beta), \\ 0.prob_0 := \mathcal{F}(prob', A, B, r), \; 1.prob_0 := \mathcal{G}(prob', A, B, r) \end{array} \right\}$$

$$p_0 \wedge p_1 \vdash \exists\, r : [0, 1]; \; 0.prob_0, 1.prob_0 : PROB \bullet$$
$$0.prob_0(Q_0) = 1 \wedge 1.prob_0(Q_1) = 1$$
$$\wedge\, M_r[0.prob_0, 1.prob_0/0.prob, 1.prob]$$

$$= \quad \{\, \text{sequential composition} \,\}$$
$$\exists\, r : [0, 1] \bullet (p_0 \wedge p_1 \vdash 0.prob'(Q_0) = 1 \wedge 1.prob(Q_1) = 1 \; ; \; M_r)$$

$$= \quad \{\, \text{definition merge operator} \,\}$$
$$\exists\, r : [0, 1] \bullet (p_0 \vdash prob'(Q_0) = 1) \,\|_{M_r}\, (p_1 \vdash prob'(Q_1) = 1)$$

$$= \quad \{\, \text{definition } \mathcal{K} \,\}$$
$$\exists\, r : [0, 1] \bullet (\mathcal{K}(p_0 \vdash Q_0) \,\|_{M_r}\, \mathcal{K}(p_1 \vdash Q_1))$$

We omit the (easier) proof that the embedding is a strengthening homomorphism for nondeterministic choice: as $\mathcal{K}$ distributes through nondeterminism we obtain a stronger predicate.

This concludes our presentation of the semantics for the nondeterministic probabilistic programming language that serves as the textual version of RoboChart diagrams with discrete probabilistic behaviour. We have described the semantic domain and an embedding function from standard programs to probabilistic ones. We have shown just two cases for the proof that the embedding is a homomorphism. This has guided the definition of individual program operators. For example, we have

$$\mathcal{K}(D_0 \sqcap D_1) \;=\; \mathcal{K}(D_0) \vee \mathcal{K}(D_1) \vee \bigvee\nolimits_{0 < r < 1}(\mathcal{K}(D_0) \,\|_{M_r}\, \mathcal{K}(D_1))$$

This definition is supported by Theorem 2 and a matching proof for the strengthening homomorphism (omitted in this paper). The proof identified the need for the two special cases in the semantics of nondeterminism: $r = 0$ and $r = 1$.

9 Related Work

Jansen et al. propose a probabilistic extension to UML [34, 35]. They add to UML's basic Statecharts a probabilistic choice node whose out-edges are annotated with probabilities. They identify interferences between Statechart transition priorities and the order of resolving nondeterministic and probabilistic choice. Verification is performed using the PRISM probabilistic model checker, with the probabilistic logic PCTL specifying properties over Statecharts. They describe the operational semantics of step execution. This is then embedded in a finite Markov Decision Process specified as a probabilistic Kripke system.

Nokovic and Sekerinski [44] propose pCharts, another variation on Statecharts, but extended with timed transitions, probabilistic transitions, costs and rewards, and state invariants. They present a translation scheme from untimed pCharts to Markov Decision Processes (MDPs), from timed pCharts to probabilistic timed automata (PTA), and from pCharts to executable C code. Everything is implemented in the pState tool. MDPs are used to verify probabilistic

and nondeterministic behaviour. PTAs are used to verify additional real-time constraints, such as the maximum or minimum probability of reaching a state within a given time and the maximum expected time to reach that state (its deadline). pCharts can be augmented with quantitative information for costs and rewards for both transitions and states: priced PTAs. This permits analysis of the maximum or minimum expected time before a transition takes place, or the number of expected steps to reach a particular state. Translation rules deal with hierarchy and orthogonality.

Both Jansen's and Nokovic's work is similar to He et al.'s [25], and therefore ours, in constructing a conservative extension of standard Statecharts. Both of them go further in dealing with hierarchy and orthogonality. This differs from our work in several ways. We focus on producing a semantics that can be combined with other UTP theories. Both Jansen and Nokovic focus on model checking, and therefore have a closed-world assumption and restrict variables to bound integers. We are interested in both model checking and theorem proving.

In 2004, Goldsmith reported an experiment [20] to extend the input language for FDR2 to accept a probabilistic choice construct with added functionality, to produce models suitable for analysis by Prism [36]. Goldsmith describes some encouraging results, but also warns about various drawbacks in the work: the loss of regularity in code emitted from FDR2 that would lead to Prism exploiting symmetries in its model checking; and that the transformation scheme does not support CSP's full failures-divergences model. The probabilistic functionality in FDR2 was lost when development moved to FDR3 in 2012 and remained lost with the move to FDR4 in 2017.

Mota et al. [10] rediscovered the functionality in FDR2 (as well as legacy copies of the tool) in their work on analysing probability in RoboChart. They define the semantics of the RoboChart probabilistic choice operator in terms of CSP's probabilistic operator. They show how this augmented CSP semantics for RoboChart can be translated into the Prism's Reactive Modules input language to check stochastic properties of RoboChart.

Zhao et al. [65] describe mapping rules between UML state diagrams and probabilistic Kripke structure semantics. They present an asynchronous parallel language based on discrete time Markov chains. Non-functional properties of systems specified using PCTL, with verification provided by the Prism model checker. Interactive theorem proving is also supported and linked to experimental results. Interestingly, the mapping rules are provided as a bidirectional transformation.

Zhang et al. [64] address the formal verification of dynamic behaviour of UML diagrams. They automatically verify UML state machine models by translating UML models to the input language of the PAT model checker in such a way as to be transparent for users. They can check safety and liveness properties with fairness assumptions using the PAT model checker [39].

10 Conclusions and Future Work

We have presented an overview of our ongoing work in giving a probabilistic semantics to RoboChart. We have concentrated on the imperative, sequential action language for RoboChart, using the weakest completion semantics approach. The result is a programming theory that can now be combined with other programming paradigms, using UTP's unification techniques explained in Sect. 3. The next step for us is to lift the current semantics into UTP's reactive theory to produce a theory of reactive probabilistic designs.

We have explicated the weakest completion approach, showing how proof outlines in [25] can be turned into near formal proofs suitable for implementation in a theorem prover. In doing this, we spent a surprising amount of time understanding the structure of He et al.'s proof, especially the nondeterminism case for the proof that the embedding function $\mathcal{K}$ is a homomorphism. This led us to investigate the weakest prespecification operator for the design theory in some detail, coming up with what we believe to be a novel derivation of the operator that echoes Hoare and He's derivation of the weakest precondition operator [32]. We observe that a law quoted in the proof of this case in [25] requires a side condition that the design to which it is applied satisfies the **H3** healthiness condition [32, Chap.3]. This is the case in the proof where it is used, but it does raise an interesting question for our lifting the current semantics to the reactive world. We found a small number of inconsistencies in the proof outlines, but these have not affected the validity of the lemmas and theorems in [25].

Our future work consists of the following:

1. Complete the rest of the proof that $\mathcal{K}$ is a homomorphism (essentially, the Kleisli lifting needed for sequential composition).
2. Implement our proofs in the Isabelle/UTP theorem prover [19].[12]
3. Lift the semantics to the reactive theory.
4. Use our semantics to verify the soundness of a translation from RoboChart to Reactive Modules, so that PRISM can be used to analyse probabilistic RoboCharts.
5. Tackle a range of different examples using both model checking and theorem proving to challenge our work. We have in our sights various probabilistic robotic control algorithms.

Examples include verifying robot localisation algorithms, such as the Random Sample Consensus algorithm RANSAC that is frequently used in robotic control [11]; providing bounds for the battery life required for coverage using random walks and arena-mapping techniques by autonomous robotic cleaners and searchers; and verifying learning algorithms for robots in uncertain environments.

---

[12] We have already begun work on the mechanisation of the proofs in Isabelle/UTP. Early indications show that the meticulous detail in the hand-written proofs is very helpful in the mechanisation.

References

1. Alur, R., Henzinger, T.A.: Reactive Modules. Formal Methods in System Design **15**(1), 7–48 (1999)
2. Bousmalis, K.: Closing the simulation-to-reality gap for deep robotic learning. `ai.googleblog.com/2017/10/closing-simulation-to-reality-gap-for.html` (2019), Google AI Blog
3. Brunner, S.G., Steinmetz, F., Belder, R., Dömel, A.: RAFCON: A graphical tool for engineering complex, robotic tasks. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, 9–14 October, 2016. pp. 3283–3290 (2016)
4. Cavalcanti, A., Ribeiro, P., Miyazawa, A., Sampaio, A., Filho, M.C., Didier, A.: RoboSim: Reference Manual (2019), www.cs.york.ac.uk/robostar/robosim/robosim-reference.pdf
5. Cavalcanti, A., Sampaio, A., Woodcock, J.: Refinement of actions in Circus. Electr. Notes Theor. Comput. Sci. **70**(3), 132–162 (2002)
6. 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, 23 November–5 December 2004, Revised Lectures. Lecture Notes in Computer Science, vol. 3167, pp. 220–268. Springer (2006)
7. Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M.: RobotML, a domain-specific language to design, simulate and deploy robotic applications. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) Simulation, Modeling, and Programming for Autonomous Robots—Third International Conference, SIMPAR 2012, Tsukuba, Japan, 5–8 November, 2012. Lecture Notes in Computer Science, vol. 7628, pp. 149–160. Springer (2012)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
9. FDR: Failures-Divergences Refinement. `www.cs.ox.ac.uk/projects/fdr/`

10. Filho, M.S.C., Marinho, R., Mota, A., Woodcock, J.: Analysing RoboChart with probabilities. In: Massoni, T., Mousavi, M.R. (eds.) Formal Methods: Foundations and Applications — 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, 26–30 November 2018. Lecture Notes in Computer Science, vol. 11254, pp. 198–214. Springer (2018)

11. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)

12. Fitzgerald, J.S., Gamble, C., Larsen, P.G., Pierce, K., Woodcock, J.: Cyber-physical systems design: Formal foundations, methods and integrated tool chains. In: Gnesi, S., Plat, N. (eds.) 3rd IEEE/ACM FME Workshop on Formal Methods in Software Engineering, FormaliSE 2015, Florence, 18 May, 2015. pp. 40–46. IEEE Computer Society (2015)

13. Foster, S., Baxter, J., Cavalcanti, A., Miyazawa, A., Woodcock, J.: Automating verification of state machines with reactive designs and Isabelle/UTP. In: Bae, K., Ölveczky, P.C. (eds.) Formal Aspects of Component Software — 15th International Conference, FACS 2018, Pohang, South Korea, 10–12 October 2018. Lecture Notes in Computer Science, vol. 11222, pp. 137–155. Springer (2018)

14. Foster, S., Cavalcanti, A., Canham, S., Woodcock, J., Zeyda, F.: Unifying theories of reactive design contracts. CoRR **abs/1712.10233** (2017)

15. Foster, S., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying theories of time with generalised reactive processes. Inf. Process. Lett. **135**, 47–52 (2018)

16. Foster, S., Woodcock, J.: Unifying theories of programming in Isabelle. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Unifying Theories of Programming and Formal Engineering Methods — International Training School on Software Engineering, Held at ICTAC 2013, Shanghai, 26–30 August 2013, Advanced Lectures. Lecture Notes in Computer Science, vol. 8050, pp. 109–155. Springer (2013)

17. Foster, S., Woodcock, J.: Towards verification of cyber-physical systems with UTP and Isabelle/HOL. In: Gibson-Robinson, T., Hopcroft, P.J., Lazic, R. (eds.) Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10160, pp. 39–64. Springer (2017)

18. Foster, S., Zeyda, F., Nemouchi, Y., Ribeiro, P., Wolff, B.: Isabelle/UTP: Mechanised theory engineering for unifying theories of programming. Archive of Formal Proofs (Feb 2019)

19. Foster, S., Zeyda, F., Woodcock, J.: Isabelle/UTP: A mechanised theory engineering framework. In: Naumann, D. (ed.) Unifying Theories of Programming — 5th International Symposium, UTP 2014, Singapore, 13 May 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8963, pp. 21–41. Springer (2015)

20. Goldsmith, M.: CSP: The best concurrent-system description language in the world—probably! In: Communicating Process Architectures. pp. 227–232 (2004)

21. Group, O.M.: OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1

22. Guttmann, W., Möller, B.: Normal design algebra. J. Log. Algebr. Program. **79**(2), 144–173 (2010)

23. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)

24. Harwood, W., Cavalcanti, A., Woodcock, J.: A theory of pointers for the UTP. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigün, H. (eds.) Theoretical Aspects of Computing - ICTAC 2008, 5th International Colloquium, Istanbul, 1–3 September,

2008. Proceedings. Lecture Notes in Computer Science, vol. 5160, pp. 141–155. Springer (2008)

25. He, J., Morgan, C., McIver, A.: Deriving probabilistic semantics via the 'weakest completion'. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004, 6th International Conference on Formal Engineering Methods, Seattle, 8–12 November 2004. Lecture Notes in Computer Science, vol. 3308, pp. 131–145. Springer (2004)

26. Hehner, E.C.R.: Predicative programming, part I. Commun. ACM **27**(2), 134–143 (1984)

27. Hehner, E.C.R.: Predicative programming, part II. Commun. ACM **27**(2), 144–151 (1984)

28. Hehner, E.C.R., Gupta, L.E., Malton, A.J.: Predicative methodology. Acta Inf. **23**(5), 487–505 (1986)

29. Hilder, J.A., Owens, N.D.L., Neal, M.J., Hickey, P.J., Cairns, S.N., Kilgour, D.P.A., Timmis, J., Tyrrell, A.M.: Chemical detection using the receptor density algorithm. IEEE Trans. Systems, Man, and Cybernetics, Part C **42**(6), 1730–1741 (2012)

30. Hoare, C.A.R.: Programs are predicates. In: FGCS. pp. 211–218 (1992)

31. Hoare, C.A.R., He, J.: The weakest prespecification. Inf. Process. Lett. **24**(2), 127–132 (1987)

32. Hoare, C.A.R., He, J.: Unifying theories of programming. Prentice Hall (1998)

33. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: Morán, F., Moreno, A., Merelo Guerv'os, J.J., Chacón, P. (eds.) Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain, 4–6 June 1995. Lecture Notes in Computer Science, vol. 929, pp. 704–720. Springer (1995)

34. Jansen, D.N., Hermanns, H., Katoen, J.: A probabilistic extension of UML State-Charts. In: Damm, W., Olderog, E. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, FTRTFT 2002, Co-sponsored by IFIP WG 2.2, Oldenburg, 9–12 September 2002. Lecture Notes in Computer Science, vol. 2469, pp. 355–374. Springer (2002)

35. Jansen, D.: Extensions of Statecharts with probability, time, and stochastic timing. Ph.D. thesis, University of Twente (10 2003)

36. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J.T., Harder, U. (eds.) Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, London, 14–17 April 2002. Lecture Notes in Computer Science, vol. 2324, pp. 200–204. Springer (2002)

37. Larsen, P.G., Fitzgerald, J.S., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project. In: 2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS, CPS Data 2016, Vienna, 11 April, 2016. pp. 1–6. IEEE Computer Society (2016)

38. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems: A Cyber-Physical Systems Approach. The MIT Press, 2nd edn. (2016)

39. Liu, Y., Sun, J., Dong, J.S.: PAT 3: An extensible architecture for building multi-domain model checkers. In: Dohi, T., Cukic, B. (eds.) IEEE 22nd International Symposium on Software Reliability Engineering, ISSRE 2011, Hiroshima, 29 November–2 December 2011. pp. 190–199. IEEE Computer Society (2011)

40. Miyazawa, A.: RoboTool: RoboChart Tool Manual (2018), www.cs.york.ac.uk/-circus/publications/techreports/reports/robotool-manual.pdf

41. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J.: Automatic property checking of robotic applications. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, 24–28 September, 2017. pp. 3869–3876 (2017)

42. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: RoboChart: modelling and verification of the functional behaviour of robotic applications. Software & Systems Modeling (Jan 2019)

43. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)

44. Nokovic, B., Sekerinski, E.: Verification and code generation for timed transitions in pCharts. In: Desai, B.C. (ed.) International C* Conference on Computer Science & Software Engineering, C3S2E '14, Montreal, 3–5 August 2014. pp. 3:1–3:10. ACM (2014)

45. Oliveira, M., Cavalcanti, A., Woodcock, J.: A denotational semantics for Circus. Electr. Notes Theor. Comput. Sci. **187**, 107–123 (2007)

46. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for *Circus*. Formal Asp. Comput. **21**(1-2), 3–32 (2009)

47. Pembeci, I., Nilsson, H., Hager, G.D.: Functional reactive robotics: an exercise in principled integration of domain-specific languages. In: Proceedings of the 4th international ACM SIGPLAN conference on Principles and practice of declarative programming, 6–8 October, 2002, Pittsburgh, (Affiliated with PLI 2002). pp. 168–179 (2002)

48. Ribeiro, P., Miyazawa, A., Li, W., Cavalcanti, A., Timmis, J.: Modelling and verification of timed robotic controllers. In: Polikarpova, N., Schneider, S. (eds.) IFM 2017, Integrated Formal Methods — 13th International Conference, Turin, 20–22 September 2017. Lecture Notes in Computer Science, vol. 10510, pp. 18–33. Springer (2017)

49. RoboCalc: `www.cs.york.ac.uk/circus/RoboCalc`

50. RoboCalc Project: The foraging robot example (2019), `www.cs.york.ac.uk/-robostar/prob_case_studies/foraging_robot`

51. Roscoe, A.W.: On the expressive power of CSP refinement. Formal Asp. Comput. **17**(2), 93–112 (2005)

52. Roscoe, A.W.: Understanding Concurrent Systems. Texts in Computer Science, Springer (2010)

53. Spivey, J.: The Z Notation: A Reference Manual. Prentice-Hall., 2nd edn. (1989)

54. V-REP: Virtual Robot Experimentation Platform, User Manual, Version 3.6.1. `www.coppeliarobotics.com/helpFiles/en/importExport.htm`

55. Wächter, M., Ottenhaus, S., Kröhnert, M., Vahrenkamp, N., Asfour, T.: The ArmarX Statechart concept: Graphical programming of robot behavior. Front. Robotics and AI (2016)

56. Webots: Reference Manual, Rel. R2019a. `www.cyberbotics.com/doc/reference/`

57. Winfield, A.F.T.: Foraging robots. In: Meyers, R.A. (ed.) Encyclopedia of Complexity and Systems Science, pp. 3682–3700. Springer (2009)

58. 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, 12–16 May, 2014. Lecture Notes in Computer Science, vol. 8442, pp. 22–41. Springer (2014)

59. 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, 4–7 April 2004. Lecture Notes in Computer Science, vol. 2999, pp. 40–66. Springer (2004)

60. Woodcock, J., Foster, S.: UTP by example: Designs. In: Bowen, J.P., Liu, Z., Zhang, Z. (eds.) Engineering Trustworthy Software Systems — Second International School, SETSS 2016, Chongqing, China, 28 March – 2 April 2016, Tutorial Lectures. Lecture Notes in Computer Science, vol. 10215, pp. 16–50 (2017)
61. Woodcock, J., Foster, S., Butterfield, A.: Heterogeneous semantics and unifying theories. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques — 7th International Symposium, ISoLA 2016, Imperial, Corfu, 1–14 October 2016, Proceedings Part I. Lecture Notes in Computer Science, vol. 9952, pp. 374–394 (2016)
62. Woodcock, J., Morgan, C.: Refinement of state-based concurrent systems. In: Bjørner, D., Hoare, C.A.R., Langmaack, H. (eds.) VDM '90, VDM and Z—Formal Methods in Software Development, Third International Symposium of VDM Europe, Kiel, 17–21 April 1990. Lecture Notes in Computer Science, vol. 428, pp. 340–351. Springer (1990)
63. Zave, P., Jackson, M.: Conjunction as composition. ACM Trans. Softw. Eng. Methodol. **2**(4), 379–411 (1993)
64. Zhang, S.J., Liu, Y.: An automatic approach to model checking UML state machines. In: Fourth International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2010, Singapore, 9–11 June 2010. pp. 1–6. IEEE Computer Society (2010)
65. Zhao, Y., Yang, Z., Xie, J., Liu, Q.: Quantitative analysis of system based on extended UML state diagrams and probabilistic model checking. JSW **5**(7), 793–800 (2010)

A Connecting Weakest Preconditions and Prespecifications

Weakest preconditions and prespecifications each arise as the weakest solution of an inequality in three variables. Both have a conjunction on the implementation side. The inequality for the weakest precondition in stated as $P \sqsupseteq s \Rightarrow q$, but this is equivalent to $s \wedge P \sqsupseteq q$ (1). The inequality for the weakest prespecification is stated as $X \; ; \; K \sqsupseteq Y$, but this is equivalent to $X[v_0/v'] \wedge K[v_0/v] \sqsupseteq Y$ (2). The two inequalities have the same essential structure. Hoare & He go further and note as a conjecture that the two predicate transformers are almost identical when the first argument mentions only dashed variables: $r'/K = (K \; \mathbf{wp} \; r)'$. The conjecture is easily proved.

$$r'/K$$
$$= \quad \{\, \text{dashing a condition: } c' = c[v'/v] \,\}$$
$$r[v'/v]/K$$
$$= \quad \{\, \text{definition of weakest prespecification} \,\}$$
$$\neg \, (\neg \, r[v'/v] \; ; \; K^{\smile})$$
$$= \quad \{\, \text{definition of relational converse} \,\}$$
$$\neg \, (\neg \, r[v'/v] \; ; \; K[v', v/v, v']))$$
$$= \quad \{\, \text{definition of sequential composition} \,\}$$

$$\neg\ \mathit{exists}v_0 \bullet \neg\ r[v_0/v] \wedge K[v', v_0/v, v'])$$

$=$ { propositional calculus }

$$\neg\ \mathit{exists}v_0 \bullet K[v', v_0/v, v'] \wedge \neg\ r[v_0/v])$$

$=$ { dashing a relation: $R' = R[v'/v]$ }

$$\neg\ (\mathit{exists}v_0 \bullet K[v, v_0/v, v'] \wedge \neg\ r[v_0/v])'$$

$=$ { definition of relational converse }

$$\neg\ (K\ ;\ \neg\ r)'$$

$=$ { definition weakest precondition }

$$(K\ \textbf{wp}\ r)'$$

This result means that the weakest prespecification subsumes the weakest precondition and so could be used to give its definition: $K\ \textbf{wp}\ r \mathrel{\widehat{=}} (r'/K)[v/v']$.