Compositional Assume-Guarantee Reasoning of Control Law Diagrams using UTP

Kangfeng Ye Simon Foster
Jim Woodcock
University of York, UK

{kangfeng.ye,simon.foster,jim.woodcock}@york.ac.uk

March 4, 2020

Abstract

This report is a summary of our work for the VeTSS funded project "Mechanised Assume-Guarantee Reasoning for Control Law Diagrams via Circus". Our Assume-Guarantee (AG) reasoning of control law diagrams is based on Hoare and He's Unifying Theories of Programming and their theory of designs. In this report, we present developed theories and laws to map discrete-time Simulink block diagrams to designs in UTP, calculate assumptions and guarantees, and verify properties for modelled systems. A practical application of our AG reasoning to an aircraft cabin pressure control subsystem is also presented. In addition, all mechanised theories in Isabelle/UTP are attached in Appendices. In the end of this report, we summarise current progress for each work package.

Contents

Acknowledgements.

A Probabilistic Designs

This is the mechanisation of *probabilistic designs* [1, 2] in Isabelle/UTP.

theory *utp-prob-des*
imports *UTP−Calculi.utp-wprespec UTP−Designs.utp-designs HOL−Probability.Probability-Mass-Function HOL−Probability.SPMF*
begin recall-syntax

purge-notation *inner* (**infix** · *70*)

declare [[*coercion pmf*]]

alphabet *'s prss* =
prob :: *'s pmf*

If the probabilities of two disjoint sample sets sums up to 1, then the probability of the first set is equal to 1 minus the probability of the second set.

lemma *pmf-disj-set*:
 assumes $X \cap Y = \{\}$
 shows $((\sum_a i{\in}(X \cup Y).\ pmf\ M\ i) = 1) = ((\sum_a i{\in}X.\ pmf\ M\ i) = 1 - (\sum_a i{\in}Y.\ pmf\ M\ i))$
 by (*metis assms diff-eq-eq infsetsum-Un-disjoint pmf-abs-summable*)

no-utp-lift *ndesign wprespec uwp*

Probabilistic designs $(('s, 's)\ rel\text{-}pdes)$, that map the standard state space to the probabilistic state space, are heterogeneous.

type-synonym $('a, 'b)\ rel\text{-}pdes = ('a, 'b\ prss)\ rel\text{-}des$
type-synonym $'s\ hrel\text{-}pdes = ('s, 's)\ rel\text{-}pdes$
type-synonym $'s\ hrel\text{-}hpdes = ('s\ prss, 's\ prss)\ rel\text{-}des$

translations
 $(type)\ ('a, 'b)\ rel\text{-}pdes <= (type)\ ('a, 'b\ prss)\ rel\text{-}des$

forget-prob is a non-homogeneous design as a forgetful function that maps a discrete probability distribution $\boldsymbol{U}(\$prob)$ at initial observation to a final state.

definition *forget-prob* :: $('s\ prss, 's)\ rel\text{-}des$ (**fp**) **where**
[*upred-defs*]: $forget\text{-}prob = U(true \vdash_n (\$prob(\$\mathbf{v}') > 0))$

The weakest prespecification of a standard design D wrt **fp** is the weakest probabilistic design, as an embedding of D in the probabilistic world through $\mathcal{K}$.

definition *pemb* :: $('a, 'b)\ rel\text{-}des \Rightarrow ('a, 'b)\ rel\text{-}pdes\ (\mathcal{K})$
 where [*upred-defs*]: $pemb\ D = \mathbf{fp} \setminus D$

lemma *pemb-mono*: $P \sqsubseteq Q \Longrightarrow \mathcal{K}(P) \sqsubseteq \mathcal{K}(Q)$
 by (*metis (mono-tags, lifting) dual-order.trans order-refl pemb-def wprespec*)

lemma *wdprespec*: $(true \vdash_n R) \setminus (p \vdash_n Q) = (p \vdash_n (R \setminus Q))$
 by (*rel-auto*)

declare [[*show-types*]]

lemma *pemb-form*:
 fixes R :: $('a, 'b)\ urel$
 shows $\boldsymbol{U}((\$prob(\$\mathbf{v}') > 0) \setminus R) = \boldsymbol{U}((\sum_a i{\in}\{s'.(R\ wp\ (\&\mathbf{v} = s'))^<\}.\ \$prob'\ i) = 1)$ (**is** *?lhs* = *?rhs*)
proof −
 have $?lhs = \boldsymbol{U}((\neg (\neg R) ;; (0 < \$prob'\$\mathbf{v})))$
  by (*rel-auto*)
 also have $... = \boldsymbol{U}((\sum_a i{\in}\{s'.(R\ wp\ (\&\mathbf{v} = s'))^<\}.\ \$prob'\ i) = 1)$
  apply (*rel-auto*)
  apply (*metis (no-types, lifting) infsetsum-pmf-eq-1 mem-Collect-eq pmf-positive subset-eq*)
  apply (*metis AE-measure-pmf-iff UNIV-I measure-pmf.prob-eq-1 measure-pmf-conv-infsetsum mem-Collect-eq set-pmf-eq' sets-measure-pmf*)
  done
 finally show *?thesis* **.**
qed

Embedded standard designs are probabilistic designs [2, Theorem 1] and [1, Theorem 3.6].

lemma *prob-lift* [*ndes-simp*]:
fixes $R :: ('a, 'b)\ urel$ **and** $p :: 'a\ upred$
shows $\mathcal{K}(p \vdash_n R) = \boldsymbol{U}(p \vdash_n ((\sum_a i{\in}\{s'.(R\ wp\ (\&\mathbf{v} = s'))^{<}\}.\ \$prob\acute{}\ i) = 1))$
proof −
have $1{:}\mathcal{K}(p \vdash_n R) = \boldsymbol{U}(p \vdash_n ((\$prob(\$\mathbf{v}\acute{}) > 0) \setminus R))$
by (*rel-auto*)
have $2{:}\boldsymbol{U}((\$prob(\$\mathbf{v}\acute{}) > 0) \setminus R) = \boldsymbol{U}((\sum_a i{\in}\{s'.(R\ wp\ (\&\mathbf{v} = s'))^{<}\}.\ \$prob\acute{}\ i) = 1)$
by (*simp add*: *pemb-form*)
show *?thesis*
by (*simp add*: *1 2*)
qed

Inverse of $\mathcal{K}$ [1, Corollary 3.7]: embedding a standard design (P) in the probabilistic world then forgetting its probability distribution is equal to P itself.

lemma *pemb-inv*:
assumes *P is* **N**
shows $\mathcal{K}(P)$; ; **fp** $= P$
proof −
obtain $pre_p\ post_p$
where $p{:}P = (pre_p \vdash_n post_p)$
using *assms* **by** (*metis ndesign-form*)
have *f1*: $\mathcal{K}(pre_p \vdash_n post_p)$; ; **fp** $= (pre_p \vdash_n post_p)$
apply (*simp add*: *prob-lift forget-prob-def*)
apply (*ndes-simp*)
apply (*rel-auto*)
proof −
fix $ok_v{::}bool$ **and** $more{::}'a$ **and** $ok_v'{::}bool$ **and** $morea{::}'b$ **and** $prob_v{::}'b\ pmf$
assume *a1*: $(\sum_a x{::}'b \mid [\![post_p]\!]_e\ (more, x).\ pmf\ prob_v\ x) = (1{::}real)$
assume *a2*: $(0{::}real) < pmf\ prob_v\ morea$
show $[\![post_p]\!]_e\ (more, morea)$
proof (*rule ccontr*)
assume *aa1*: $\neg\ [\![post_p]\!]_e\ (more, morea)$
have *f1*: $(\sum_a x{::}'b \in \{x.\ [\![post_p]\!]_e\ (more, x)\} \cup \{morea\}.\ pmf\ prob_v\ x) =$
$(\sum_a x{::}'b \in \{x.\ [\![post_p]\!]_e\ (more, x)\}.\ pmf\ prob_v\ x) +$
$(\sum_a x{::}'b \in \{morea\}.\ pmf\ prob_v\ x)$
unfolding *infsetsum-altdef abs-summable-on-altdef*
apply (*subst set-integral-Un*, *auto*)
using *aa1* **apply** (*simp*)
using *abs-summable-on-altdef assms* **apply** *fastforce*
using *abs-summable-on-altdef* **by** *blast*
then have *f2*: $... = 1 + pmf\ prob_v\ morea$
using *a1* **by** *auto*
then have *f3*: $... > 1$
using *a2* **by** *linarith*
show *False*
using *f1 f2 f3*
by (*metis f1 f2 measure-pmf.prob-le-1 measure-pmf-conv-infsetsum not-le*)
qed
next
fix $ok_v{::}bool$ **and** $more{::}'a$ **and** $ok_v'{::}bool$ **and** $morea{::}'b$
assume *a1*: $[\![post_p]\!]_e\ (more, morea)$
have *f1*: $\forall x.\ (pmf\ (pmf\text{-}of\text{-}list\ [(morea, 1{::}real)])\ x) = (if\ x = morea\ then\ (1{::}real)\ else\ 0)$
by (*simp add*: *pmf-of-list-wf-def pmf-pmf-of-list*)
have *f2*: $(\sum_a x{::}'b \mid [\![post_p]\!]_e\ (more, x).\ pmf\ (pmf\text{-}of\text{-}list\ [(morea, 1{::}real)])\ x) =$
$(\sum_a x{::}'b \mid [\![post_p]\!]_e\ (more, x).\ (if\ x = morea\ then\ (1{::}real)\ else\ 0))$

using *f1* **by** *simp*
have *f3*: ... = (*1*::*real*)
proof −
have $(\sum_a x{::}'b \mid [\![post_p]\!]_e\ (more,\ x).\ \mathit{if}\ x = morea\ \mathit{then}\ 1{::}real\ \mathit{else}\ (0{::}real)) =$
$(\sum_a x{::}'b \in \{morea\} \cup \{t.\ [\![post_p]\!]_e\ (more,\ t) \wedge t \neq morea\}.$
$\mathit{if}\ x = morea\ \mathit{then}\ 1{::}real\ \mathit{else}\ (0{::}real))$
proof −
have $\{t.\ [\![post_p]\!]_e\ (more,\ t)\} = \{morea\} \cup \{t.\ [\![post_p]\!]_e\ (more,\ t) \wedge t \neq morea\}$
using *a1* **by** *blast*
then show *?thesis*
by *presburger*
qed
also have ... $= (\sum_a x{::}'b \in \{morea\}.\ \mathit{if}\ x = morea\ \mathit{then}\ 1{::}real\ \mathit{else}\ (0{::}real)) +$
$(\sum_a x{::}'b \in \{t.\ [\![post_p]\!]_e\ (more,\ t) \wedge t \neq morea\}.\ \mathit{if}\ x = morea\ \mathit{then}\ 1{::}real\ \mathit{else}\ (0{::}real))$
unfolding *infsetsum-altdef abs-summable-on-altdef*
apply (*subst set-integral-Un*, *auto*)
using *abs-summable-on-altdef* **apply** *fastforce*
using *abs-summable-on-altdef* **by** (*smt abs-summable-on-0 abs-summable-on-cong mem-Collect-eq*)
also have ... = (*1*::*real*) +
$(\sum_a x{::}'b \in \{t.\ [\![post_p]\!]_e\ (more,\ t) \wedge t \neq morea\}.\ \mathit{if}\ x = morea\ \mathit{then}\ 1{::}real\ \mathit{else}\ (0{::}real))$
by *simp*
also have ... = (*1*::*real*)
by (*smt add-cancel-left-right infsetsum-all-0 mem-Collect-eq*)
then show *?thesis*
by (*simp add*: *calculation*)
qed
show $\exists\, prob_v{::}'b\ pmf.$
$(\sum_a x{::}'b \mid [\![post_p]\!]_e\ (more,\ x).\ pmf\ prob_v\ x) = (1{::}real) \wedge (0{::}real) < pmf\ prob_v\ morea$
apply (*rule-tac x = pmf-of-list* [(*morea*, *1.0*)] **in** *exI*)
apply (*auto*)
apply (*simp add*: *f1 f2 f3*)
by (*simp add*: *pmf-of-list-wf-def pmf-pmf-of-list*)
qed
show *?thesis*
using *f1* **by** (*simp add*: *p*)
qed

no-utp-lift *usubst* (*0*) *subst* (*1*)

A.1 wplus

Two pmfs can be joined into one by their corresponding weights via $P +_w Q$ where w is the weight of P.

definition *wplus* :: $'a\ pmf \Rightarrow real \Rightarrow\ 'a\ pmf \Rightarrow\ 'a\ pmf$ ((- +- -) [*64*, *0*, *65*] *64*) **where**
$wplus\ P\ w\ Q = join\text{-}pmf\ (pmf\text{-}of\text{-}list\ [(P,\ w),\ (Q,\ 1 - w)])$

Query of the probability value of a state i in a joined probability distribution is just the summation of the query of i in P by its weight w and the query of i in Q by its weight $(1 - w)$.

lemma *pmf-wplus*:
assumes $w \in \{0..1\}$
shows $pmf\ (P +_w Q)\ i = pmf\ P\ i * w + pmf\ Q\ i * (1 - w)$
proof −
from *assms* **have** *pmf-wf-list*: $pmf\text{-}of\text{-}list\text{-}wf\ [(P,\ w),\ (Q,\ 1 - w)]$

by (*auto intro*!: *pmf-of-list-wfI*)
show *?thesis*
proof (*cases* $w \in \{0<..<1\}$)
case *True*
hence *set-pmf*:*set-pmf* (*pmf-of-list* $[(P, w), (Q, 1 - w)]$) $= \{P, Q\}$
by (*subst set-pmf-of-list-eq*, *auto simp add*: *pmf-wf-list*)
thus *?thesis*
proof (*cases* $P = Q$)
case *True*
from *assms* **show** *?thesis*
apply (*auto simp add*: *wplus-def join-pmf-def pmf-bind*)
apply (*subst integral-measure-pmf*[*of* $\{P, Q\}$])
apply (*auto simp add*: *set-pmf-of-list pmf-wf-list set-pmf pmf-pmf-of-list*)
apply (*simp add*: *True*)
apply (*metis distrib-right eq-iff-diff-eq-0 le-add-diff-inverse mult.commute mult-cancel-left1*)
done
next
case *False*
then show *?thesis*
apply (*auto simp add*: *wplus-def join-pmf-def pmf-bind*)
apply (*subst integral-measure-pmf*[*of* $\{P, Q\}$])
apply (*auto simp add*: *set-pmf-of-list pmf-wf-list set-pmf pmf-pmf-of-list*)
done
qed
next
case *False*
thm *disjE*
with *assms* **have** $w = 0 \vee w = 1$
by (*auto*)
with *assms* **show** *?thesis*
proof (*erule-tac disjE*, *simp-all*)
assume *w*: $w = 0$
with *pmf-wf-list* **have** *set-pmf* (*pmf-of-list* $[(P, w), (Q, 1 - w)]$) $= \{Q\}$
apply (*simp add*: *pmf-of-list-remove-zeros*(*2*)[*THEN sym*])
apply (*subst set-pmf-of-list-eq*, *auto simp add*: *pmf-of-list-wf-def*)
done
with *w* **show** *pmf* $(P +_0 Q)$ $i =$ *pmf* Q i
apply (*auto simp add*: *wplus-def join-pmf-def pmf-bind pmf-wf-list pmf-of-list-remove-zeros*(*2*)[*THEN sym*])
apply (*subst integral-measure-pmf*[*of* $\{Q\}$])
apply (*simp-all add*: *set-pmf-of-list-eq pmf-pmf-of-list pmf-of-list-wf-def*)
done
next
assume *w*: $w = 1$
with *pmf-wf-list* **have** *set-pmf* (*pmf-of-list* $[(P, w), (Q, 1 - w)]$) $= \{P\}$
apply (*simp add*: *pmf-of-list-remove-zeros*(*2*)[*THEN sym*])
apply (*subst set-pmf-of-list-eq*, *auto simp add*: *pmf-of-list-wf-def*)
done
with *w* **show** *pmf* $(P +_1 Q)$ $i =$ *pmf* P i
apply (*auto simp add*: *wplus-def join-pmf-def pmf-bind pmf-wf-list pmf-of-list-remove-zeros*(*2*)[*THEN sym*])
apply (*subst integral-measure-pmf*[*of* $\{P\}$])
apply (*simp-all add*: *set-pmf-of-list-eq pmf-pmf-of-list pmf-of-list-wf-def*)
done
qed

```
  qed
qed

lemma wplus-commute:
  assumes w ∈{0..1}
  shows P +_w Q = Q +_(1 − w) P
  using assms by (auto intro: pmf-eqI simp add: pmf-wplus)

lemma wplus-idem:
  assumes w ∈{0..1}
  shows P +_w P = P
  using assms
  apply (rule-tac pmf-eqI)
  apply (simp add: pmf-wplus)
  by (metis le-add-diff-inverse mult.commute mult-cancel-left2 ring-class.ring-distribs(2))

lemma wplus-zero: P +_0 Q = Q
  by (auto intro: pmf-eqI simp add: pmf-wplus)

lemma wplus-one: P +_1 Q = P
  by (auto intro: pmf-eqI simp add: pmf-wplus)
```

This is used to prove the associativity of probabilistic choice: *prob-choice-assoc*.

```
lemma wplus-assoc:
  assumes w₁ ∈ {0..1} w₂ ∈ {0..1}
  assumes (1−w₁)∗(1−w₂)=(1−r₂) w₁=r₁∗r₂
  shows P +_w₁ (Q +_w₂ R) = (P +_r₁ Q) +_r₂ R
proof (cases w₁ = 0 ∧ w₂ = 0)
  case True
  then show ?thesis
    proof −
      from assms(3−4) have t1: r₂=0
        by (simp add: True)
      then show ?thesis
        by (simp add: wplus-zero True t1)
    qed
next
  case False
  from assms(3) have f1: r₂ = w₁+w₂−w₁∗w₂
    proof −
      have f1: ∀ r ra. (ra::real) + − r = 0 ∨ ¬ ra = r
        by simp
      have f2: ∀ r ra rb rc. (rc::real) · rb + − (ra · r) = rc · (rb + − r) + (rc + − ra) · r
        by (simp add: mult-diff-mult)
      have f3: ∀ r ra. (ra::real) + (r + − ra) = r + 0
        by fastforce
      have f4: ∀ r ra. (ra::real) + ra · r = ra · (1 + r)
        by (simp add: distrib-left)
      have f5: ∀ r ra. (ra::real) + − r + 0 = ra + − r
        by linarith
      have f6: ∀ r ra. (0::real) + (ra + − r) = ra + − r
        by simp
      have 1 + − w₂ + − (w₁ · (1 + − w₂)) = 1 + (0 + − r₂)
      using f2 f1 by (metis (no-types) add.left-commute add-uminus-conv-diff assms(3) mult.left-neutral)
      then have 1 + (w₁ + w₁ · − w₂ + − r₂) = 1 + − w₂
```

```
      using f6 f5 f4 f3 by (metis (no-types) add.left-commute)
    then show ?thesis
    by linarith
    qed
  then have f2: r2 ∈ {0..1}
    using assms(1−2) by (smt assms(3) atLeastAtMost-iff mult-le-one sum-le-prod1)
  from f1 have f2': (w1+w2−w1∗w2) ≥ w1
    using assms(1) assms(2) mult-left-le-one-le by auto
  from f1 have f3: r1 = w1/(w1+w2−w1∗w2)
    by (metis False add.commute add-diff-eq assms(4) diff-add-cancel
        mult-zero-left mult-zero-right nonzero-eq-divide-eq)
  show ?thesis
  proof (cases w1 = 0)
    case True
    from f3 have ft1: r1 = 0
      by (simp add: True)
    from f1 have ft2: r2 = w2
      by (simp add: True)
    then show ?thesis
      using ft1 ft2 assms(1−2)
      by (simp add: True wplus-zero)
  next
    case False
    from f3 f2' have ff1: r1 ≤ 1
      using False
       by (metis assms(4) atLeastAtMost-iff eq-iff f1 f2 le-cases le-numeral-extra(4) mult-cancel-right2
mult-right-mono)
    have ff2: r1 ≥ 0
      by (smt False assms(1) assms(4) atLeastAtMost-iff f2 mult-not-zero zero-le-mult-iff)
    from ff1 and ff2 have ff3: r1 ∈ {0..1}
      by simp
    have ff4: w2 ∗ (1 − w1) = (1 − r1) ∗ r2
      using f1 f3 False assms
      by (metis (no-types, hide-lams) add-diff-eq diff-add-eq-diff-diff-swap diff-diff-add
          diff-diff-eq2 eq-iff-diff-eq-0 mult.commute mult.right-neutral right-diff-distrib' right-minus-eq)
    then show ?thesis
      using assms(1−2) f2 ff3 apply (rule-tac pmf-eqI)
      apply (simp add: assms(1−2) f2 ff3 pmf-wplus)
      using assms(3−4) ff4
      by (metis (no-types, hide-lams) add.commute add.left-commute mult.assoc mult.commute)
  qed
qed
```

A.2 Probabilistic Choice

We use parallel-by-merge in UTP to define the probabilistic choice operator. The merge predicate is the join of two distributions by their weights.

```
definition prob-merge :: real ⇒ (('s, 's prss, 's prss) mrg, 's prss) urel (PM_) where
[upred-defs]: prob-merge r = U($prob´ = $0:prob +«r» $1:prob)

lemma swap-prob-merge:
  assumes r ∈ {0..1}
  shows swap_m ;; PM_r = PM_(1 − r)
  by (rel-auto, (metis assms wplus-commute)+)
```

abbreviation *prob-des-merge* :: *real* ⇒ (('*s des*, '*s prss des*, '*s prss des*) *mrg*, '*s prss des*) *urel* ($\mathbf{PDM}_{\_}$)
where
$\mathbf{PDM}_r \equiv \mathbf{DM}(\mathbf{PM}_r)$

lemma *swap-prob-des-merge*:
  assumes $r \in \{0..1\}$
  shows $swap_m \;;\; \mathbf{PDM}_r = \mathbf{PDM}_{1\ -\ r}$
  by (*metis assms swap-des-merge swap-prob-merge*)

The probabilistic choice operator is defined conditionally in order to satisfy unit and zero laws (*prob-choice-one* and *prob-choice-zero*::'*a*) below. The definition of the operator follows [1, Definition 3.14]. Actually use of $P \parallel^D{}_{\mathbf{PM}_r} Q$ directly for (r = 0) or (r = 1) cannot get the desired result (P or Q) as the precondition of merged designs cannot be discharged to the precondition of P or Q simply.

definition *prob-choice* :: '*s hrel-pdes* ⇒ *real* ⇒ '*s hrel-pdes* ⇒ '*s hrel-pdes* ((- $\oplus_{\_}$ -) [*164*, *0*, *165*] *164*)
  where [*upred-defs*]:
prob-choice P r Q ≡
  if $r \in \{0{<}..{<}1\}$
  then $P \parallel^D{}_{\mathbf{PM}_r} Q$
  else (*if* $r = 0$
      then Q
      else (*if* $r = 1$
          then P
          else $\top_D$))

The r in $P \oplus_r Q$ is a real number (HOL terms). Sometimes, however, we want a similar operator of which the weight is a UTP expression (therefore it depends on the values of state variables). For example, $P \oplus_{U(1/real\ («N»-i))} Q$ in a uniform selection algorithms where $\&i$ is a state variable. Hence, $(P \oplus_{eE} Q)$ is defined below, which is inspired by Morgan's logical constant [3].

definition *prob-choice-r* :: ('*a*, '*a*) *rel-pdes* ⇒ (*real*, '*a*) *uexpr* ⇒ ('*a*, '*a*) *rel-pdes* ⇒ ('*a*, '*a*) *rel-pdes*
  ((- $\oplus_{e\_}$ -) [*164*, *0*, *165*] *164*)
where [*upred-defs*]:
prob-choice-r P E Q ≡ $(con_D\ R \cdot (II_D \triangleleft U(«R» = E) \triangleright_D \bot_D) \;;\; (P \oplus_R Q))$

lemma *prob-choice-commute*: $r \in \{0..1\} \Longrightarrow P \oplus_r Q = Q \oplus_{1\ -\ r} P$
  by (*simp add*: *prob-choice-def swap-prob-des-merge*[*THEN sym*], *metis par-by-merge-commute-swap*)

lemma *prob-choice-one*:
  $P \oplus_1 Q = P$
  by (*simp add*: *prob-choice-def*)

lemma *prob-choice-zero*:
  $P \oplus_0 Q = Q$
  by (*simp add*: *prob-choice-def*)

lemma *prob-choice-r*:
  $r \in \{0{<}..{<}1\} \Longrightarrow P \oplus_r Q = P \parallel^D{}_{\mathbf{PM}_r} Q$
  by (*simp add*: *prob-choice-def*)

lemma *prob-choice-inf-simp*:
  $(\bigsqcap\ r \in \{0{<}..{<}1\} \cdot (P \oplus_r Q)) = (\bigsqcap\ r \in \{0{<}..{<}1\} \cdot\ P \parallel^D{}_{\mathbf{PM}_r} Q)$
  using *prob-choice-r*
  apply (*simp add*: *prob-choice-def*)
  by (*simp add*: *UINF-as-Sup-collect image-def*)

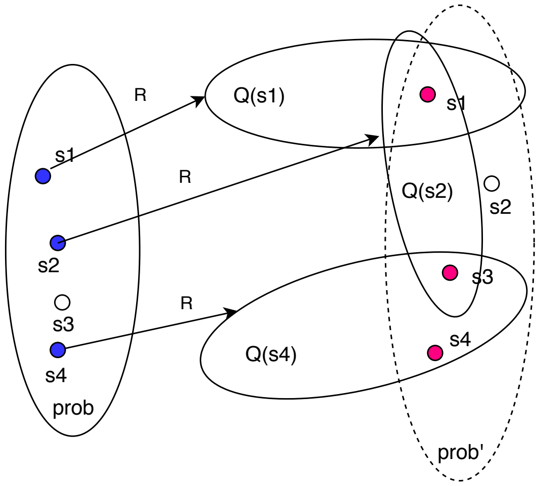

Figure 1: Illustration of Kleisli lifting

inf-is-exists helps to establish the fact that our theorem regarding nondeterminism [2, Sect. 8] is the same as He's [1, Theorem 3.10].

lemma *inf-is-exists*:
$(\bigsqcap\ r \in \{0{<}..{<}1\} \cdot\ (p \vdash_n P) \parallel^{D}\mathbf{PM}_r\ (q \vdash_n Q))$
$= (\exists\ r \in \boldsymbol{U}(\{0{<}..{<}1\}) \cdot\ (p \vdash_n P) \parallel^{D}\mathbf{PM}_r\ (q \vdash_n Q))$
by (*pred-auto*)

A.3 Kleisli Lifting and Sequential Composition

utp-lit-vars

The Kleisli lifting operator maps a probabilistic design $(p \vdash_n R)$ into a "lifted" design that maps from *prob* to *prob*. Therefore, one probabilistic design can be composed sequentially with another lifted design. The precondition of the definition specifies that all states of the initial distribution satisfy the predicate p. The postcondition specifies that there exists a function Q, that maps states to distributions, such that

- for any state s, if its probability in the initial distribution is larger than 0, then R(s, Q(s)) must be held;
- any state ss in final distribution $\$prob´$ is equal to summation of all paths from any state t in its initial distribution to ss via $Q\ t$.

Figure 1 illustrates the lifting operation, provided that there are four states in the state space. The blue states in $\$prob$ denotes their initial probabilities are larger than 0, and the red states in $\$prob´$ denotes their final probabilities are larger than 0. Q is defined as

$$\{(s_1, Q(s_1)), (s_2, Q(s_2)), (s_4, Q(s_4))\}$$

and the relation between s_i and $Q(s_i)$ is established by R. In addition, the probability of s_1 in $Q(s_1)$ is larger than 0, that of s_1 and s_3 in $Q(s_2)$, and that of s_3 and s_4 in $Q(s_4)$. Finally, the finally distribution is given below.

$$prob'(s_1) = prob(s_1) * Q(s_1)(s_1) + prob(s_2) * Q(s_2)(s_1)$$
$$prob'(s_3) = prob(s_2) * Q(s_2)(s_3) + prob(s_4) * Q(s_4)(s_3)$$
$$prob'(s_4) = prob(s_2) * Q(s_2)(s_4) + prob(s_4) * Q(s_4)(s_4)$$

definition *kleisli-lift2*:: $'a\ upred \Rightarrow ('a,\ 'a\ prss)\ urel \Rightarrow ('a\ prss,\ 'a\ prss)\ rel\text{-}des$
where *kleisli-lift2 p R* =
($\boldsymbol{U}((\sum_a\ i{\in}[\![p]\!]_p.\ \$prob\ i) = 1)$
$\vdash_r$
$(\exists\ Q \cdot ($
$(\forall\ ss \cdot U((\$prob\acute{}\ ss) = (\sum_a\ t.\ ((\$prob\ t) * (pmf\ (Q\ t)\ ss))))) \wedge$
$(\forall\ s \cdot (\neg(U(\$prob\ \$\mathbf{v}\acute{} > 0 \wedge \$\mathbf{v}\acute{} = s)\ ;;$
$((((\neg R)\ ;;\ (\forall\ t \cdot U((\$prob\ t) = (pmf\ (Q\ s)\ t)))))))$
))
)))

named-theorems *kleisli-lift*

Alternatively, we can define the lifting operator as a normal design, instead of a design in previous definition.

definition *kleisli-lift2'*:: $'a\ upred \Rightarrow ('a,\ 'a\ prss)\ urel \Rightarrow ('a\ prss,\ 'a\ prss)\ rel\text{-}des$ **where**
[*kleisli-lift*]: *kleisli-lift2' p R* =
($\boldsymbol{U}((\sum_a\ i{\in}[\![p]\!]_p.\ \&prob\ i) = 1)$
$\vdash_n$
$(\exists\ Q \cdot ($
$(\forall\ ss \cdot U((\$prob\acute{}\ ss) = (\sum_a\ t.\ ((\$prob\ t) * (pmf\ (Q\ t)\ ss))))) \wedge$
$(\forall\ s \cdot (\neg(U(\$prob\ \$\mathbf{v}\acute{} > 0 \wedge \$\mathbf{v}\acute{} = s)\ ;;$
$((\neg R)\ ;;\ (\forall\ t \cdot U((\$prob\ t) = (pmf\ (Q\ s)\ t)))))$
))
)))

Two definitions actually are equal.

lemma *kleisli-lift2-eq*: *kleisli-lift2' p R* = *kleisli-lift2 p R*
apply (*simp add*: *kleisli-lift2-def*)
apply (*simp add*: *utp-prob-des.kleisli-lift2'-def*)
by (*rel-auto*)

utp-expr-vars

Then the lifting operator ↑ is defined upon *kleisli-lift2*.

definition *kleisli-lift* (↑) **where**
kleisli-lift P = *kleisli-lift2* $(\lfloor pre_D(P)\rfloor_<)\ (pre_D(P) \wedge post_D(P))$

The alternative definition of the lifting operator ↑ is based on *kleisli-lift2'*.

lemma *kleisli-lift-alt-def*:
kleisli-lift P = *kleisli-lift2'* $(\lfloor pre_D(P)\rfloor_<)\ (pre_D(P) \wedge post_D(P))$
by (*simp add*: *kleisli-lift-def kleisli-lift2-eq*)

Sequential composition of two probabilistic designs (P and Q) is composition of P with the lifted Q through the Kleisli lifting operator.

abbreviation *pseqr* :: $('b,\ 'b)\ rel\text{-}pdes \Rightarrow ('b,\ 'b)\ rel\text{-}pdes \Rightarrow ('b,\ 'b)\ rel\text{-}pdes$ (**infix** $;;_p$ *60*) **where**
pseqr P Q ≡ $(P\ ;;\ (\uparrow Q))$

II_p is the identity of sequence of probabilistic designs.

abbreviation *skip-p* (II_p) **where**
skip-p ≡ $\mathcal{K}(II_D)$

The top of probabilistic designs is still the top of designs.

abbreviation *falsep* :: $('b,\ 'b)\ rel\text{-}pdes$ ($false_p$) **where**

$falsep \equiv false$

end

References

[1] J. He, C. Morgan, and A. McIver, "Deriving probabilistic semantics via the 'weakest completion'," in *Formal Methods and Software Engineering*, J. Davies, W. Schulte, and M. Barnett, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 131–145.

[2] J. C. P. Woodcock, A. L. C. Cavalcanti, S. Foster, A. Mota, and K. Ye, "Probabilistic semantics for RoboChart: A weakest completion approach," in *Unifying Theories of Programming*, ser. Lecture Notes in Computer Science. Springer, 2019, p. to appear.

[3] C. C. Morgan, *Programming from Specifications.* Prentice-Hall, 1990.