Probabilistic Relations in Isabelle/UTP

Kangfeng Ye Simon Foster
Jim Woodcock
University of York, UK

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

February 17, 2023

Abstract

This document presents our theory of probabilistic relations, based on Hehner's predicative probabilistic programming [1], for reasoning about imperative probabilistic programs.

Contents

1 Laws related to *infsum*

theory *infsum-laws*
imports
HOL−Analysis.Infinite-Sum
UTP2.utp

begin
unbundle *UTP-Syntax*

1.1 Useful lemmas

lemma *finite-image-set2′*:
assumes *finite A finite B*
shows *finite* $\{(a, b).\ a \in A \wedge b \in B\}$
apply (*rule finite-subset* [**where** $B = \bigcup x \in A.\ \bigcup y \in B.\ \{(x,y)\}$])
apply *auto*
using *assms*(*1*) *assms*(*2*) **by** *blast*

lemma *finite-image-set2″*:
assumes *finite B* $\forall x.$ *finite* $(A\ x)$
shows *finite* $\{(a, b).\ b \in B \wedge a \in A\ b\}$
apply (*rule finite-subset* [**where** $B = \bigcup y \in B.\ \bigcup x \in A\ y.\ \{(x,y)\}$])
apply *auto*
using *assms*(*1*) *assms*(*2*) **by** *blast*

lemma *card-1-singleton*:
  assumes $\exists!x.\ P\ x$
  shows $card\ \{x.\ P\ x\} = Suc\ (0::\mathbb{N})$
  using *assms card-1-singleton-iff* **by** *fastforce*

lemma *card-0-singleton*:
  assumes $\neg(\exists\, x.\ P\ x)$
  shows $card\ \{x.\ P\ x\} = (0::\mathbb{N})$
  using *assms* **by** *auto*

lemma *card-0-false*:
  shows $card\ \{x.\ False\} = (0::\mathbb{R})$
  by *simp*

lemma *conditional-conds-conj*:
  $\forall\, s.\ (if\ b_1\ \ s\ then\ (1::\mathbb{R})\ else\ (0::\mathbb{R})) * (if\ b_2\ \ s\ then\ (1::\mathbb{R})\ else\ (0::\mathbb{R})) =$
    $(if\ b_1\ \ s \wedge b_2\ \ s\ then\ 1\ else\ 0)$
  apply (*rule allI*)
  by *force*

lemma *conditional-conds-conj'*:
  $\forall\, s.\ (if\ b_1\ \ s\ then\ (m::\mathbb{R})\ else\ (0::\mathbb{R})) * (if\ b_2\ \ s\ then\ (p::\mathbb{R})\ else\ (0::\mathbb{R})) =$
    $(if\ b_1\ \ s \wedge b_2\ \ s\ then\ m * p\ else\ 0)$
  apply (*rule allI*)
  by *simp*

lemma *conditional-cmult*: $\forall\, s.\ (if\ b_1\ \ s\ then\ (m::\mathbb{R})\ else\ (0::\mathbb{R})) * c =$
    $((if\ b_1\ \ s\ then\ (m::\mathbb{R}) * c\ else\ (0::\mathbb{R})))$
  apply (*rule allI*)
  by *force*

lemma *conditional-cmult-1*: $\forall\, s.\ (if\ b_1\ \ s\ then\ (1::\mathbb{R})\ else\ (0::\mathbb{R})) * c =$
    $((if\ b_1\ \ s\ then\ c\ else\ (0::\mathbb{R})))$
  apply (*rule allI*)
  by *force*

1.2 Laws of *infsum*

lemma *infset-0-not-summable-or-sum-to-zero*:
  assumes $infsum\ f\ A = 0$
  shows $(f\ summable\text{-}on\ A \wedge has\text{-}sum\ f\ A\ \ 0) \vee \neg\ f\ summable\text{-}on\ A$
  by (*simp add*: *assms summable-iff-has-sum-infsum*)

lemma *infset-0-not-summable-or-zero*:
  assumes $\forall\, s.\ f\ s \geq (0::\mathbb{R})$
  assumes $infsum\ f\ A = 0$
  shows $(\forall\, s \in A.\ f\ s = 0) \vee \neg\ f\ summable\text{-}on\ A$
proof (*rule ccontr*)
  assume $a1$: $\neg\ ((\forall\, s{\in}A.\ f\ s = (0)) \vee \neg\ f\ summable\text{-}on\ A)$
  then have $f1$: $(\neg\ (\forall\, s{\in}A.\ f\ s = (0))) \wedge f\ summable\text{-}on\ A$
    by *linarith*
  then have $\exists\, x \in A.\ f\ x > 0$
    apply (*simp add*: *Bex-def*)
    apply (*auto*)
    apply (*rule-tac* $x = x$ **in** *exI*)

apply (*simp*)
using *assms*(*1*) **by** (*metis order-le-less*)

have *ind-ge-0*: *infsum f* {(*SOME x. x* ∈*A* ∧ *f x* > *0*)} > *0*
using *a1 assms*(*1*) *assms*(*2*) *nonneg-infsum-le-0D* **by** *force*

have *infsum f* {(*SOME x. x* ∈*A* ∧ *f x* > *0*)} ≤ *infsum f A*
apply (*rule infsum-mono2*)
apply *simp*
using *f1* **apply** *blast*
using *a1 assms*(*1*) *assms*(*2*) *nonneg-infsum-le-0D* **apply** *force*
using *assms*(*1*) **by** *blast*
then have *infsum f A* > *0*
using *ind-ge-0* **by** *linarith*
then show *False*
using *assms*(*2*) **by** *simp*
qed

lemma *has-sum-cdiv-left*:
fixes *f* :: *′a* ⇒ ℝ
assumes ‹*has-sum f A a*›
shows *has-sum* (λ*x. f x* / *c*) *A* (*a* / *c*)
apply (*simp only* : *divide-inverse*)
using *assms has-sum-cmult-left* **by** *blast*

lemma *infsum-cdiv-left*:
fixes *f* :: *′a* ⇒ ℝ
assumes ‹*c* ≠ *0* ⟹ *f summable-on A*›
shows *infsum* (λ*x. f x* / *c*) *A* = *infsum f A* / *c*
apply (*simp only* : *divide-inverse*)
using *infsum-cmult-left′* **by** *blast*

lemma *summable-on-cdiv-left*:
fixes *f* :: *′a* ⇒ ℝ
assumes ‹*f summable-on A*›
shows (λ*x. f x* / *c*) *summable-on A*
using *assms summable-on-def has-sum-cdiv-left* **by** *blast*

lemma *summable-on-cdiv-left′*:
fixes *f* :: *′a* ⇒ ℝ
assumes ‹*c* ≠ *0*›
shows (λ*x. f x* / *c*) *summable-on A* ⟷ *f summable-on A*
apply (*simp only* : *divide-inverse*)
by (*simp add*: *assms summable-on-cmult-left′*)

lemma *not-summable-on-cdiv-left′*:
fixes *f* :: *′a* ⇒ ℝ
assumes ‹*c* ≠ *0*›
shows ¬(λ*x. f x* / *c*) *summable-on A* ⟷ ¬*f summable-on A*
apply (*simp only* : *divide-inverse*)
by (*simp add*: *assms summable-on-cmult-left′*)

lemma *summable-on-minus*:
fixes *f g* :: *′a* ⇒ ℝ
assumes ‹*f summable-on A*›

assumes ‹*g summable-on A*›
shows ‹$(\lambda x.\ f\ x - g\ x)$ *summable-on A*›
apply (*subst add-uminus-conv-diff*[*symmetric*])
apply (*subst summable-on-add*)
using *assms*(*1*) **apply** *blast*
by (*simp add*: *assms*(*2*) *summable-on-uminus*)+

lemma *infsum-geq-element*:
fixes $f :: 'a \Rightarrow \mathbb{R}$
assumes $\forall s.\ f\ s \geq 0$
assumes *f summable-on A*
assumes $s \in A$
shows $f\ s \leq$ *infsum f A*
proof −
have *f0*: *infsum f* $(A - \{s\}) \geq 0$
by (*simp add*: *assms*(*1*) *infsum-nonneg*)
have *f1*: *infsum f A* = *infsum f* $(\{s\} \cup (A - \{s\}))$
using *assms*(*3*) *insert-Diff* **by** *force*
also have *f2*: ... = *infsum f* $\{s\}$ + *infsum f* $(A - \{s\})$
apply (*subst infsum-Un-disjoint*)
apply (*simp-all*)
by (*simp add*: *assms*(*2*) *summable-on-cofin-subset*)
show *?thesis*
using *f0 f1 f2* **by** *auto*
qed

lemma *infsum-geq-element'*:
fixes $f :: 'a \Rightarrow \mathbb{R}$
assumes $\forall s.\ f\ s \geq 0$
assumes *f summable-on A*
assumes $s \in A$
assumes *infsum f A* = *x*
shows $f\ s \leq x$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *infsum-geq-element*)

lemma *infsum-on-singleton*:
$(\sum_\infty s \in \{x\}.\ f\ s) = f\ x$
apply (*rule infsumI*)
apply (*simp add*: *has-sum-def*)
apply (*subst topological-tendstoI*)
apply (*auto*)
apply (*simp add*: *eventually-finite-subsets-at-top*)
apply (*rule-tac* $x = \{x\}$ **in** *exI*)
by (*metis add.right-neutral finite.emptyI finite-insert insert-absorb insert-not-empty subset-antisym subset-singleton-iff sum.empty sum.insert*)

lemma *infsum-singleton*:
$(\sum_\infty v_0::'a.\ (\textit{if}\ c = v_0\ \textit{then}\ (m::\mathbb{R})\ \textit{else}\ 0)) = m$
apply (*rule infsumI*)
apply (*simp add*: *has-sum-def*)
apply (*subst topological-tendstoI*)
apply (*auto*)
apply (*simp add*: *eventually-finite-subsets-at-top*)
apply (*rule-tac* $x = \{c\}$ **in** *exI*)
by (*auto*)

lemma *infsum-singleton-summable*:
  assumes $m \neq 0$
  shows $(\lambda v_0.\ (\mathit{if}\ c = v_0\ \mathit{then}\ (m{::}\mathbb{R})\ \mathit{else}\ 0))$ *summable-on UNIV*
proof (*rule ccontr*)
  assume *a1*: $\neg\ (\lambda v_0{::}'a.\ \mathit{if}\ c = v_0\ \mathit{then}\ m\ \mathit{else}\ (0{::}\mathbb{R}))$ *summable-on UNIV*
  from *a1* **have** $(\sum_\infty v_0{::}'a.\ (\mathit{if}\ c = v_0\ \mathit{then}\ (m{::}\mathbb{R})\ \mathit{else}\ 0)) = (0{::}\mathbb{R})$
    by (*simp add*: *infsum-def*)
  then show *False*
    by (*simp add*: *infsum-singleton assms*)
qed

lemma *infsum-singleton-1*:
  $(\sum_\infty v_0{::}'a.\ (\mathit{if}\ v_0 = c\ \mathit{then}\ (m{::}\mathbb{R})\ \mathit{else}\ 0)) = m$
  by (*smt* (*verit*, *del-insts*) *infsum-cong infsum-singleton*)

lemma *infsum-cond-finite-states*:
  assumes *finite* $\{s.\ b\ s\}$
  shows $(\sum_\infty v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ (0{::}\mathbb{R}))) = (\sum v_0 \in \{s.\ b\ s\}.\ f\ v_0)$
proof −
  have $(\sum_\infty v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ 0)) = (\sum_\infty v_0 \in \{s.\ b\ s\} \cup (-\{s.\ b\ s\}).\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ 0))$
    by *auto*
  moreover have ... $= (\sum_\infty v_0 \in \{s.\ b\ s\}.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ 0))$
    apply (*subst infsum-Un-disjoint*)
    apply (*simp add*: *assms*)
    apply (*smt* (*verit*, *ccfv-threshold*) *ComplD mem-Collect-eq summable-on-0*)
    apply *simp*
    by (*smt* (*verit*, *best*) *ComplD infsum-0 mem-Collect-eq*)
  moreover have ... $= (\sum v_0 \in \{s.\ b\ s\}.\ f\ v_0)$
    using *assms* **by** *force*
  ultimately show *?thesis*
    by *presburger*
qed

lemma *infsum-cond-finite-states-summable*:
  assumes *finite* $\{s.\ b\ s\}$
  shows $(\lambda v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ (0{::}\mathbb{R})))$ *summable-on UNIV*
proof −
  have $((\lambda v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ (0{::}\mathbb{R})))$ *summable-on UNIV*$)$ =
      $((\lambda v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ f\ v_0\ \mathit{else}\ (0{::}\mathbb{R})))$ *summable-on* $(\{s.\ b\ s\} \cup -\{s.\ b\ s\}))$
    by *auto*
  moreover have ...
    apply (*rule summable-on-Un-disjoint*)
    apply (*simp add*: *assms*)
    apply (*smt* (*verit*, *ccfv-threshold*) *ComplD mem-Collect-eq summable-on-0*)
    by *simp*
  ultimately show *?thesis*
    by *presburger*
qed

lemma *infsum-constant-finite-states*:
  assumes *finite* $\{s.\ b\ s\}$
  shows $(\sum_\infty v_0{::}'a.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ (m{::}\mathbb{R})\ \mathit{else}\ 0)) = m * \mathit{card}\ \{s.\ b\ s\}$
  apply (*rule infsumI*)
  apply (*simp add*: *has-sum-def*)

```
  apply (subst topological-tendstoI)
  apply (auto)
  apply (simp add: eventually-finite-subsets-at-top)
  apply (rule-tac x = {v. b v} in exI)
  apply (auto)
  using assms apply force
proof −
  fix S::ℙ ℝ and Y::ℙ 'a
  assume a1: m ∗ real (card (Collect b)) ∈ S
  assume a2: finite Y
  assume a3: {v::'a. b v} ⊆ Y
  have (∑v0::'a∈Y. if b v0 then m else (0::ℝ)) = (∑v0::'a∈{v::'a. b v}. if b v0 then m else (0::ℝ))
    by (smt (verit, best) DiffD2 a2 a3 mem-Collect-eq sum.mono-neutral-cong-right)
  moreover have ... = m ∗ card {s. b s}
    by auto
  ultimately show (∑v0::'a∈Y. if b v0 then m else (0::ℝ)) ∈ S
    using a1 by presburger
qed

lemma infsum-constant-finite-states-summable:
  assumes finite {s. b s}
  shows (λv0::'a. (if b v0 then (m::ℝ) else 0)) summable-on UNIV
  apply (simp add: summable-on-def)
  apply (rule-tac x = m ∗ card {s. b s} in exI)
  apply (simp add: has-sum-def)
  apply (subst topological-tendstoI)
  apply (auto)
  apply (simp add: eventually-finite-subsets-at-top)
  apply (rule-tac x = {v. b v} in exI)
  apply (auto)
  using assms apply force
proof −
  fix S::ℙ ℝ and Y::ℙ 'a
  assume a1: m ∗ real (card (Collect b)) ∈ S
  assume a2: finite Y
  assume a3: {v::'a. b v} ⊆ Y
  have (∑v0::'a∈Y. if b v0 then m else (0::ℝ)) = (∑v0::'a∈{v::'a. b v}. if b v0 then m else (0::ℝ))
    by (smt (verit, best) DiffD2 a2 a3 mem-Collect-eq sum.mono-neutral-cong-right)
  moreover have ... = m ∗ card {s. b s}
    by auto
  ultimately show (∑v0::'a∈Y. if b v0 then m else (0::ℝ)) ∈ S
    using a1 by presburger
qed

lemma infsum-constant-finite-states-summable-2:
  assumes finite {s. b1 s} finite {s. b2 s}
  shows (λv0::'a. (if b1 v0 then (m::ℝ) else 0) +
          (if b2 v0 then (n::ℝ) else 0)) summable-on UNIV
  apply (subst summable-on-add)
  apply (simp add: assms(1) infsum-constant-finite-states-summable)
  by (simp add: assms(2) infsum-constant-finite-states-summable)+

lemma infsum-constant-finite-states-summable-3:
  assumes finite {s. b1 s} finite {s. b2 s} finite {s. b3 s}
  shows (λv0::'a. (if b1 v0 then (m::ℝ) else 0) +
```

$(if\ b_2\ v_0\ then\ (n{::}\mathbb{R})\ else\ 0)\ +$
$(if\ b_3\ v_0\ then\ (p{::}\mathbb{R})\ else\ 0))\ summable\text{-}on\ UNIV$
apply (*subst summable-on-add*)+
apply (*simp add*: *assms*(*1*) *infsum-constant-finite-states-summable*)
apply (*simp add*: *assms*(*2*) *infsum-constant-finite-states-summable*)+
by (*simp add*: *assms*(*3*) *infsum-constant-finite-states-summable*)+

lemma *infsum-constant-finite-states-summable-cmult-1*:
assumes *finite* $\{s.\ b_1\ s\}$
shows $(\lambda v_0{::}'a.\ (if\ b_1\ v_0\ then\ (m{::}\mathbb{R})\ else\ 0) * c_1)\ summable\text{-}on\ UNIV$
apply (*rule summable-on-cmult-left*)
by (*simp add*: *assms*(*1*) *infsum-constant-finite-states-summable*)

lemma *infsum-constant-finite-states-cmult-1*:
assumes *finite* $\{s.\ b_1\ s\}$
shows $(\sum_\infty v_0{::}'a.\ (if\ b_1\ v_0\ then\ (m{::}\mathbb{R})\ else\ 0) * c_1) = m * card\ \{s.\ b_1\ s\} * c_1$
apply (*subst infsum-cmult-left*)
using *assms infsum-constant-finite-states-summable* **apply** *blast*
apply (*subst infsum-constant-finite-states*)
using *assms* **apply** *blast*
by *auto*

lemma *infsum-constant-finite-states-summable-cmult-2*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$
shows $(\lambda v_0{::}'a.\ (if\ b_1\ v_0\ then\ (m{::}\mathbb{R})\ else\ 0) * c_1\ +$
$(if\ b_2\ v_0\ then\ (n{::}\mathbb{R})\ else\ 0) * c_2$
$)\ summable\text{-}on\ UNIV$
apply (*subst summable-on-add*)
apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*1*) *infsum-constant-finite-states-summable*)
apply (*rule summable-on-cmult-left*)
by (*simp add*: *assms*(*2*) *infsum-constant-finite-states-summable*)+

lemma *infsum-constant-finite-states-cmult-2*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$
shows $(\sum_\infty v_0{::}'a.$
$(if\ b_1\ v_0\ then\ (m{::}\mathbb{R})\ else\ 0) * c_1\ +$
$(if\ b_2\ v_0\ then\ (n{::}\mathbb{R})\ else\ 0) * c_2)$
$= m * card\ \{s.\ b_1\ s\} * c_1 + n * card\ \{s.\ b_2\ s\} * c_2$
apply (*subst infsum-add*)
using *assms*(*1*) *infsum-constant-finite-states-summable-cmult-1* **apply** *blast*
using *assms*(*2*) *infsum-constant-finite-states-summable-cmult-1* **apply** *blast*
apply (*subst infsum-constant-finite-states-cmult-1*)
using *assms*(*1*) **apply** *blast*
apply (*subst infsum-constant-finite-states-cmult-1*)
using *assms*(*2*) **apply** *blast*
by *blast*

lemma *infsum-constant-finite-states-summable-cmult-3*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$ *finite* $\{s.\ b_3\ s\}$
shows $(\lambda v_0{::}'a.\ (if\ b_1\ v_0\ then\ (m{::}\mathbb{R})\ else\ 0) * c_1\ +$
$(if\ b_2\ v_0\ then\ (n{::}\mathbb{R})\ else\ 0) * c_2\ +$
$(if\ b_3\ v_0\ then\ (p{::}\mathbb{R})\ else\ 0) * c_3$
$)\ summable\text{-}on\ UNIV$
apply (*subst summable-on-add*)+

apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*1*) *infsum-constant-finite-states-summable*)
apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*2*) *infsum-constant-finite-states-summable*)+
apply (*rule summable-on-cmult-left*)
by (*simp add*: *assms*(*3*) *infsum-constant-finite-states-summable*)+

lemma *infsum-constant-finite-states-cmult-3*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$ *finite* $\{s.\ b_3\ s\}$
shows $(\sum_\infty v_0::'a.$
$(\mathit{if}\ b_1\ v_0\ \mathit{then}\ (m::\mathbb{R})\ \mathit{else}\ 0) * c_1 +$
$(\mathit{if}\ b_2\ v_0\ \mathit{then}\ (n::\mathbb{R})\ \mathit{else}\ 0) * c_2 +$
$(\mathit{if}\ b_3\ v_0\ \mathit{then}\ (p::\mathbb{R})\ \mathit{else}\ 0) * c_3)$
$= m * \mathit{card}\ \{s.\ b_1\ s\} * c_1 + n * \mathit{card}\ \{s.\ b_2\ s\} * c_2 + p * \mathit{card}\ \{s.\ b_3\ s\} * c_3$
apply (*subst infsum-add*)
using *assms*(*1*) *assms*(*2*) **apply** (*rule infsum-constant-finite-states-summable-cmult-2*)
using *assms*(*3*) **apply** (*rule infsum-constant-finite-states-summable-cmult-1*)
apply (*subst infsum-constant-finite-states-cmult-1*)
using *assms*(*3*) **apply** *blast*
apply (*subst infsum-constant-finite-states-cmult-2*)
using *assms*(*1*) *assms*(*2*) **by** *blast*+

lemma *infsum-constant-finite-states-summable-cmult-4*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$ *finite* $\{s.\ b_3\ s\}$ *finite* $\{s.\ b_4\ s\}$
shows $(\lambda v_0::'a.\ (\mathit{if}\ b_1\ v_0\ \mathit{then}\ (m::\mathbb{R})\ \mathit{else}\ 0) * c_1 +$
$(\mathit{if}\ b_2\ v_0\ \mathit{then}\ (n::\mathbb{R})\ \mathit{else}\ 0) * c_2 +$
$(\mathit{if}\ b_3\ v_0\ \mathit{then}\ (p::\mathbb{R})\ \mathit{else}\ 0) * c_3 +$
$(\mathit{if}\ b_4\ v_0\ \mathit{then}\ (q::\mathbb{R})\ \mathit{else}\ 0) * c_4$
) *summable-on UNIV*
apply (*subst summable-on-add*)+
apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*1*) *infsum-constant-finite-states-summable*)
apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*2*) *infsum-constant-finite-states-summable*)+
apply (*rule summable-on-cmult-left*)
apply (*simp add*: *assms*(*3*) *infsum-constant-finite-states-summable*)+
apply (*rule summable-on-cmult-left*)
by (*simp add*: *assms*(*4*) *infsum-constant-finite-states-summable*)+

lemma *infsum-constant-finite-states-4*:
assumes *finite* $\{s.\ b_1\ s\}$ *finite* $\{s.\ b_2\ s\}$ *finite* $\{s.\ b_3\ s\}$ *finite* $\{s.\ b_4\ s\}$
shows $(\sum_\infty v_0::'a.$
$(\mathit{if}\ b_1\ v_0\ \mathit{then}\ (m::\mathbb{R})\ \mathit{else}\ 0) * c_1 +$
$(\mathit{if}\ b_2\ v_0\ \mathit{then}\ (n::\mathbb{R})\ \mathit{else}\ 0) * c_2 +$
$(\mathit{if}\ b_3\ v_0\ \mathit{then}\ (p::\mathbb{R})\ \mathit{else}\ 0) * c_3+$
$(\mathit{if}\ b_4\ v_0\ \mathit{then}\ (q::\mathbb{R})\ \mathit{else}\ 0) * c_4)$
$= m * \mathit{card}\ \{s.\ b_1\ s\} * c_1 + n * \mathit{card}\ \{s.\ b_2\ s\} * c_2 + p * \mathit{card}\ \{s.\ b_3\ s\} * c_3 + q * \mathit{card}\ \{s.\ b_4\ s\}$
$* c_4$
apply (*subst infsum-add*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) **apply** (*rule infsum-constant-finite-states-summable-cmult-3*)
using *assms*(*4*) **apply** (*rule infsum-constant-finite-states-summable-cmult-1*)
apply (*subst infsum-constant-finite-states-cmult-1*)
using *assms*(*4*) **apply** *blast*
apply (*subst infsum-constant-finite-states-cmult-3*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*+

lemma *infsum-singleton-cond-unique*:
  assumes $\exists!\ v.\ b\ v$
  shows $(\sum_\infty v_0{::}'a.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ (m{::}\mathbb{R})\ \mathit{else}\ 0)) = m$
  apply (*rule infsumI*)
  apply (*simp add*: *has-sum-def*)
  apply (*subst topological-tendstoI*)
  apply (*auto*)
  apply (*simp add*: *eventually-finite-subsets-at-top*)
  apply (*rule-tac* $x = \{\mathit{THE}\ v.\ b\ v\}$ **in** *exI*)
  apply (*auto*)
  by (*smt* (*verit*, *ccfv-SIG*) *assms finite-insert mk-disjoint-insert sum.insert sum-nonneg sum-nonpos theI*)

lemma *infsum-mult-singleton-left*:
  $(\sum_\infty v_0{::}'a.\ ((\mathit{if}\ v_0 = c\ \mathit{then}\ (1{::}\mathbb{R})\ \mathit{else}\ 0) * (P\ v_0))) = P\ c$
  apply (*rule infsumI*)
  apply (*simp add*: *has-sum-def*)
  apply (*subst topological-tendstoI*)
  apply (*auto*)
  apply (*simp add*: *eventually-finite-subsets-at-top*)
  apply (*rule-tac* $x = \{c\}$ **in** *exI*)
  apply (*auto*)
  by (*simp add*: *sum.remove*)

lemma *infsum-mult-singleton-right*:
  $(\sum_\infty v_0{::}'a.\ ((P\ v_0) * (\mathit{if}\ v_0 = c\ \mathit{then}\ (1{::}\mathbb{R})\ \mathit{else}\ 0))) = P\ c$
  using *infsum-mult-singleton-left*
  by (*metis* (*no-types*, *lifting*) *infsum-cong mult.commute*)

lemma *infsum-mult-singleton-left-1*:
  $(\sum_\infty v_0{::}'a.\ ((\mathit{if}\ c = v_0\ \mathit{then}\ (1{::}\mathbb{R})\ \mathit{else}\ 0) * (P\ v_0))) = P\ c$
  using *infsum-mult-singleton-left*
  by (*smt* (*verit*) *infsum-cong*)

lemma *infsum-mult-singleton-right-1*:
  $(\sum_\infty v_0{::}'a.\ ((P\ v_0) * (\mathit{if}\ c = v_0\ \mathit{then}\ (1{::}\mathbb{R})\ \mathit{else}\ 0))) = P\ c$
  using *infsum-mult-singleton-right*
  by (*smt* (*verit*) *infsum-cong*)

lemma *infsum-mult-singleton-1*:
  $(\sum_\infty s{::}'a.$
    $(\sum_\infty v_0{::}'a.\ (\mathit{if}\ c = v_0\ \mathit{then}\ 1{::}\mathbb{R}\ \mathit{else}\ (0{::}\mathbb{R}))$
      $* (\mathit{if}\ f\ v_0 = s\ \mathit{then}\ 1{::}\mathbb{R}\ \mathit{else}\ (0{::}\mathbb{R})))$
  $) = (1{::}\mathbb{R})$
  apply (*rule infsumI*)
  apply (*simp add*: *has-sum-def*)
  apply (*subst topological-tendstoI*)
  apply (*auto*)
  apply (*simp add*: *eventually-finite-subsets-at-top*)
  apply (*rule-tac* $x=\{f\ c\}$ **in** *exI*)
  apply (*auto*)
  apply (*subgoal-tac* $(\sum s{::}'a{\in}Y.\ \sum_\infty v_0{::}'a.\ (\mathit{if}\ c = v_0\ \mathit{then}\ 1{::}\mathbb{R}\ \mathit{else}\ (0{::}\mathbb{R})) *$
  $(\mathit{if}\ f\ v_0 = s\ \mathit{then}\ 1{::}\mathbb{R}\ \mathit{else}\ (0{::}\mathbb{R})))$
  $= 1$)

apply *presburger*
apply (*simp add*: *sum.remove*)
apply (*subgoal-tac* ($\sum s$::$'a\in Y - \{f\ c\}$. $\sum_\infty v_0$::$'a$. (*if* $c = v_0$ *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* $f\ v_0 = s$ *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)))
$= 0$)
prefer *2*
apply (*subst sum-nonneg-eq-0-iff*)
apply (*simp*)+
apply (*simp add*: *infsum-nonneg*)
apply (*smt* (*verit*, *best*) *Diff-iff infsum-0 insert-iff mult-not-zero*)
apply (*simp*)
apply (*rule infsumI*)
apply (*simp add*: *has-sum-def*)
apply (*subst topological-tendstoI*)
apply (*auto*)
apply (*simp add*: *eventually-finite-subsets-at-top*)
apply (*rule-tac* $x = \{c\}$ **in** *exI*)
apply (*auto*)
apply (*subgoal-tac* ($\sum v_0$::$'a\in Ya$.
(*if* $c = v_0$ *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* $f\ v_0 = f\ c$ *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)))
$= 1$)
apply *simp*
apply (*simp add*: *sum.remove*)
by (*smt* (*verit*, *ccfv-SIG*) *Diff-insert-absorb mk-disjoint-insert mult-cancel-left1*
sum.not-neutral-contains-not-neutral)

lemma *infsum-mult-subset-left*:
($\sum_\infty v_0$::$'a$. ((*if* $b\ v_0$ *then* (1::$\mathbb{R}$) *else* 0) $*$ ($P\ v_0$))) = ($\sum_\infty v_0$::$'a \in \{v_0.\ b\ v_0\}$. ($P\ v_0$))
apply (*rule infsum-cong-neutral*)
by *simp*+

lemma *infsum-mult-subset-left-summable*:
((λv_0::$'a$. (*if* $b\ v_0$ *then* (1::$\mathbb{R}$) *else* 0) $*$ ($P\ v_0$)) *summable-on UNIV*) =
((λv_0::$'a$. ($P\ v_0$)) *summable-on* $\{v_0.\ b\ v_0\}$)
apply (*rule summable-on-cong-neutral*)
apply *simp*
by *simp*+

lemma *infsum-mult-subset-right*:
($\sum_\infty v_0$::$'a$. (($P\ v_0$) $*$ (*if* $b\ v_0$ *then* (1::$\mathbb{R}$) *else* 0))) = ($\sum_\infty v_0$::$'a \in \{v_0.\ b\ v_0\}$. ($P\ v_0$))
apply (*rule infsum-cong-neutral*)
by *simp*+

lemma *infsum-not-zero-summable*:
assumes *infsum f A* $= x$
assumes $x \neq 0$
shows *f summable-on A*
using *assms*(*1*) *assms*(*2*) *infsum-not-exists* **by** *blast*

lemma *infsum-not-zero-is-summable*:
assumes *infsum f A* $\neq 0$
shows *f summable-on A*
using *assms infsum-not-exists* **by** *blast*

lemma *infsum-mult-subset-left-summable'*:
assumes *P summable-on UNIV*
shows $(\lambda v_0::'a.\ ((\mathit{if}\ b\ v_0\ \mathit{then}\ (m::\mathbb{R})\ \mathit{else}\ 0) * (P\ v_0)))$ *summable-on UNIV*
apply (*subgoal-tac* $(\lambda v_0.\ (\mathit{if}\ b\ v_0\ \mathit{then}\ (m::\mathbb{R})\ \mathit{else}\ 0) * (P\ v_0))$ *summable-on UNIV*
$\longleftrightarrow (\lambda x::'a.\ m * P\ x)$ *summable-on* $\{x.\ b\ x\}$)
apply (*metis assms subset-UNIV summable-on-cmult-right summable-on-subset-banach*)
apply (*rule summable-on-cong-neutral*)
apply *blast*
apply *simp*
by *auto*

lemma *infsum-mono-strict*:
fixes $f :: 'a \Rightarrow \mathbb{R}$
assumes *f summable-on A* **and** *g summable-on A*
assumes $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x < g\ x \rangle$
assumes $A \neq \{\}$
shows *infsum f A* $<$ *infsum g A*
proof −
have *f0*: $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x \rangle$
using *assms*(*3*) *nless-le* **by** *blast*
then have *f1*: *infsum f A* $\leq$ *infsum g A*
by (*simp add*: *assms*(*1*) *assms*(*2*) *infsum-mono*)
have *f2*: *infsum g A* $=$ *infsum* $(\lambda x.\ (g\ x - f\ x) + f\ x)\ A$
by *auto*
also have *f3*: ... $=$ *infsum* $(\lambda x.\ (g\ x - f\ x))\ A$ $+$ *infsum f A*
apply (*subst infsum-add*)
using *summable-on-minus assms*(*1*) *assms*(*2*) **apply** *blast*
apply (*simp add*: *assms*(*1*))
by *simp*
obtain *x* **where** *P-x*: $x \in A$
using *assms*(*4*) **by** *blast*
have *f4*: $\bigwedge x.\ x \in A \Longrightarrow (g\ x - f\ x) > 0$
using *assms*(*3*) **by** *auto*
have *f5*: *infsum* $(\lambda x.\ (g\ x - f\ x))\ ((A - \{x\}) \cup \{x\}) =$ *infsum* $(\lambda x.\ (g\ x - f\ x))\ (A - \{x\}) +$ *infsum* $(\lambda x.\ (g\ x - f\ x))\ \{x\}$
apply (*subst infsum-Un-disjoint*)
apply (*simp add*: *P-x assms*(*1*) *assms*(*2*) *summable-on-Diff summable-on-minus*)
apply *simp*
apply *blast*
by (*simp*)
have *f6*: ... $\geq$ *infsum* $(\lambda x.\ (g\ x - f\ x))\ \{x\}$
by (*smt* (*verit*) *DiffD1 f0 infsum-nonneg*)
have *f7*: ... > 0
using *f4 P-x f6* **by** *fastforce*
have *f8*: *infsum* $(\lambda x.\ (g\ x - f\ x))\ A > 0$
by (*metis P-x Un-commute f5 f7 insert-Diff insert-is-Un*)
then have *infsum f A* $\neq$ *infsum g A*
using *f2 f3* **by** *linarith*
then show *infsum f A* $<$ *infsum g A*
using *f1 nless-le* **by** *blast*
qed

end

2 Iverson Bracket

theory *utp-iverson-bracket*
  imports *UTP2.utp*
    infsum-laws
begin

unbundle *UTP-Syntax*
print-bundles

bundle *no-UTP-lattice-syntax*
begin

no-notation
  bot (⊤) **and**
  top (⊥) **and**
  inf (**infixl** ⊔ *70*) **and**
  sup (**infixl** ⊓ *65*) **and**
  Inf (⨆ - [*900*] *900*) **and**
  Sup (⨅ - [*900*] *900*)

no-syntax
  -INF1 :: *pttrns* ⇒ *'b* ⇒ *'b* ((*3*⨆ -./ -) [*0*, *10*] *10*)
  -INF :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* ((*3*⨆ -∈-./ -) [*0*, *0*, *10*] *10*)
  -SUP1 :: *pttrns* ⇒ *'b* ⇒ *'b* ((*3*⨅ -./ -) [*0*, *10*] *10*)
  -SUP :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* ((*3*⨅ -∈-./ -) [*0*, *0*, *10*] *10*)
end
unbundle *no-UTP-lattice-syntax*

print-bundles

unbundle *lattice-syntax*
term ⊥

2.1 Iverson Bracket

definition *iverson-bracket* :: *'s pred* ⇒ (*'s* ⇒ ℝ) **where**
[*expr-defs*]: *iverson-bracket P* = (*if P then 1 else 0*)$_e$

syntax
  -e-iverson-bracket :: *logic* ⇒ *logic* (⟦-⟧$_{\mathcal{I}e}$ *150*)
  -iverson-bracket :: *logic* ⇒ *logic* (⟦-⟧$_{\mathcal{I}}$ *150*)

translations
  -e-iverson-bracket P == *CONST iverson-bracket* (*P*)$_e$
  -iverson-bracket P == *CONST iverson-bracket P*

expr-constructor *iverson-bracket*

lemma *iverson-bracket-true*: ⟦*true*⟧$_{\mathcal{I}}$ = (*1*)$_e$
  apply (*simp add*: *iverson-bracket-def*)

by (*simp add*: *true-pred-def*)

lemma *iverson-bracket-false*: $[\![false]\!]_{\mathcal{I}} = (0)_e$
apply (*simp add*: *iverson-bracket-def*)
by (*simp add*: *false-pred-def*)

lemma *iverson-bracket-mono*: $[\![\ (P) \sqsupseteq (Q)\ ]\!] \Longrightarrow [\![P]\!]_{\mathcal{I}} \leq [\![Q]\!]_{\mathcal{I}}$
apply (*simp add*: *ref-by-pred-is-leq*)
apply (*simp add*: *iverson-bracket-def*)
apply (*intro le-funI*)
by *auto*

lemma *iverson-bracket-conj*: $[\![P \wedge Q]\!]_{\mathcal{I}e} = ([\![P]\!]_{\mathcal{I}e} * [\![Q]\!]_{\mathcal{I}e})_e$
by (*expr-auto*)

lemma *iverson-bracket-conj1* : $[\![\lambda s.\ (a \leq s \wedge s \leq b)]\!]_{\mathcal{I}} = ([\![\lambda s.\ a \leq s]\!]_{\mathcal{I}} * [\![\lambda s.\ s \leq b]\!]_{\mathcal{I}})_e$
by (*expr-auto*)

lemma *iverson-bracket-disj*: $[\![P \vee Q]\!]_{\mathcal{I}e} = ([\![P]\!]_{\mathcal{I}e} + [\![Q]\!]_{\mathcal{I}e} - ([\![P]\!]_{\mathcal{I}e} * [\![Q]\!]_{\mathcal{I}e}))_e$
by (*expr-auto*)

lemma *iverson-bracket-not*: $[\![\neg P]\!]_{\mathcal{I}e} = (1 - [\![P]\!]_{\mathcal{I}e})_e$
by (*expr-auto*)

lemma *iverson-bracket-plus*: $([\![\lambda s.\ s \in A]\!]_{\mathcal{I}} + [\![\lambda s.\ s \in B]\!]_{\mathcal{I}})_e = ([\![\lambda s.\ s \in A \cap B]\!]_{\mathcal{I}} + [\![\lambda s.\ s \in A \cup B]\!]_{\mathcal{I}})_e$
by (*expr-auto*)

lemma *iverson-bracket-inter* : $[\![\lambda s.\ s \in A \cap B]\!]_{\mathcal{I}} = ([\![\lambda s.\ s \in A]\!]_{\mathcal{I}} * [\![\lambda s.\ s \in B]\!]_{\mathcal{I}})_e$
by (*expr-auto*)

lemma *infinite-prod-is-1*:
fixes $P::'b \Rightarrow \mathbb{R}$
assumes $\neg$ *finite* (*UNIV*::$'b$ *set*)
shows $(\prod\ m|\mathit{True}.\ (P\ m)) = (1::\mathbb{R})$
using *assms* **by** *force*

lemma *infinite-sum-is-0*:
fixes $P::'b \Rightarrow \mathbb{R}$
assumes $\neg$ *finite* (*UNIV*::$'b$ *set*)
shows $(\sum\ m|\mathit{True}.\ (P\ m)) = (0::\mathbb{R})$
using *assms* **by** *auto*

lemma *iverson-bracket-forall-prod*:
fixes $P::'a \Rightarrow\ 'b \Rightarrow \mathit{bool}$
assumes *finite* (*UNIV*::$'b$ *set*)
shows $[\![(\forall\, m.\ P\ m)]\!]_{\mathcal{I}e} = (\prod\ m|\mathit{True}.\ ([\![(P\ «m»)]\!]_{\mathcal{I}e}))_e$
apply (*expr-auto*)
proof −
fix $x::'a$ **and** $xa::'b$
assume *a1*: $\neg\ P\ x\ xa$
show $(\prod m::'b{\in}\mathit{UNIV}.\ \mathit{if}\ P\ x\ m\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) = (0::\mathbb{R})$
apply (*rule prod-zero*)

apply (*simp add*: *assms*)
using *a1* **by** *auto*
qed

We use $\sum_\infty$ (*infsum*) to take into account infinite sets that satisfy P. For this case, the summation is just equal to 0. Then this lemma is not true, and so we have added a finite assumption.

lemma *iverson-bracket-exist-sum*:
fixes $P{::}'a \Rightarrow {}'b \Rightarrow bool$
assumes ‘*finite* $\{m.\ P\ m\}$‘
shows $[\![(\exists\, m.\ P\ m)]\!]_{\mathcal{I}e} = (\lambda s.\ (min\ (1{::}\mathbb{R})\ ((\sum_\infty m.\ ([\![(P\ «m»)]\!]_{\mathcal{I}e}))_e\ s)))$
apply (*expr-auto*)
apply (*subst infsum-constant-finite-states*)
using *assms* **apply** (*simp add*: *taut-def*)
by (*smt* (*verit*, *del-insts*) *assms SEXP-def taut-def mem-Collect-eq real-of-card sum-nonneg-leq-bound*)

lemma *iverson-bracket-exist-sum-1*:
fixes $P{::}'a \Rightarrow {}'b \Rightarrow bool$
assumes *finite* $(UNIV{::}'b\ set)$
shows $[\![(\exists\, m.\ P\ m)]\!]_{\mathcal{I}e} = (1 - (\prod\ m|\ True.\ ([\![(\neg P\ «m»)]\!]_{\mathcal{I}e})))_e$
apply (*expr-auto*)
using *assms* **by** *auto*

lemma *iverson-bracket-card*:
fixes $P{::}'a \Rightarrow {}'b \Rightarrow bool$
assumes ‘*finite* $(\{m{::}'b.\ P\ m\})$‘
shows $(card\ \{m.\ P\ m\})_e = (\sum_\infty m.\ ([\![(P\ «m»)]\!]_{\mathcal{I}e}))_e$
apply (*expr-auto*)
apply (*subst infsum-constant-finite-states*)
using *assms* **apply** (*simp add*: *taut-def*)
by *force*

With the Iverson bracket, summation with index (LHS) can be defined without its index (RHS). As Donald E. Knuth mentioned in “Two Notes on Notation”, the summation without indices (or limits) is better (not easily make a mistake when dealing with its index).

lemma *iverson-bracket-summation*:
fixes $P{::}'s \Rightarrow bool$ **and** $f :: {}'s \Rightarrow \mathbb{R}$
shows $(\sum_\infty k|P\ k.\ (f)_e\ k) = (\sum_\infty k.\ (f * [\![P]\!]_{\mathcal{I}})_e\ k)$
by (*simp add*: *infsum-mult-subset-right iverson-bracket-def*)

definition *nat-of-real-1* $:: \mathbb{R} \Rightarrow nat$ **where**
nat-of-real-1 $r = (if\ r = (1{::}\mathbb{R})\ then\ (1)\ else\ 0)$

lemma *iverson-bracket-product*:
fixes $P{::}'s \Rightarrow bool$
assumes *finite* $(UNIV{::}'s\ set)$
shows $(\prod\ m|P\ m.\ f\ m) = (\prod\ m|\ True.\ (f\ \hat{}\ («nat\text{-}of\text{-}real\text{-}1»\ ([\![P]\!]_{\mathcal{I}e})))_e\ m)$
proof −
let $?P = \lambda m.\ (if\ P\ m\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?Q = \lambda r.\ (if\ r = (1{::}\mathbb{R})\ then\ 1{::}\mathbb{N}\ else\ (0{::}\mathbb{N}))$
have *f1*: $(\prod m{::}'s{\in}UNIV.\ f\ m\ \hat{}\ (?Q\ (?P\ m))) = (\prod m{::}'s{\in}\{m.\ \neg\ P\ m\} \cup \{m.\ P\ m\}.\ f\ m\ \hat{}\ (?Q\ (?P\ m)))$
by (*simp add*: *Un-def*)
have *f2*: $... = (\prod m{::}'s{\in}\{m.\ \neg\ P\ m\}.\ f\ m\ \hat{}\ (?Q\ (?P\ m))) * (\prod m{::}'s{\in}\{m.\ P\ m\}.\ f\ m\ \hat{}\ (?Q\ (?P\ m)))$
apply (*subst prod.union-inter-neutral*)

apply (*meson assms rev-finite-subset subset-UNIV*)
apply (*meson assms rev-finite-subset subset-UNIV*)
apply *force*
by *auto*
show *?thesis*
apply (*simp add*: *expr-defs*)
apply (*simp add*: *nat-of-real-1-def*)
using *f1 f2* **by** *auto*
qed

lemma *max-iverson-bracket*:
$(max\ x\ y)_e = (x * ([\![x > y]\!]_{\mathcal{I}e}) + y * ([\![x \leq y]\!]_{\mathcal{I}e}))_e$
by (*expr-auto*)

lemma *min-iverson-bracket*:
$(min\ x\ y)_e = (x * ([\![x \leq y]\!]_{\mathcal{I}e}) + y * ([\![x > y]\!]_{\mathcal{I}e}))_e$
by (*expr-auto*)

lemma *floor-iverson-bracket*:
$(real\text{-}of\text{-}int\ \lfloor x \rfloor)_e = (\sum_\infty\ n.\ n * [\![((real\text{-}of\text{-}int)\ «n» \leq x \wedge x < (real\text{-}of\text{-}int)\ («n» + 1))]\!]_{\mathcal{I}e})_e$
apply (*expr-auto*)
apply (*subst infsum-mult-subset-right*)
proof −
fix *xa*
have $\{v_0{::}\mathbb{Z}.\ real\text{-}of\text{-}int\ v_0 \leq x\ xa \wedge x\ xa < real\text{-}of\text{-}int\ v_0 + (1{::}\mathbb{R})\} = \{\lfloor x\ xa \rfloor\}$
by (*smt* (*verit*) *Collect-cong floor-split singleton-conv*)
then show $real\text{-}of\text{-}int\ \lfloor x\ xa \rfloor =$
$infsum\ real\text{-}of\text{-}int\ \{v_0{::}\mathbb{Z}.\ real\text{-}of\text{-}int\ v_0 \leq x\ xa \wedge x\ xa < real\text{-}of\text{-}int\ v_0 + (1{::}\mathbb{R})\}$
by *simp*
qed

lemma *ceiling-iverson-bracket*:
$(real\text{-}of\text{-}int\ \lceil x \rceil)_e = (\sum_\infty\ n.\ n * [\![((real\text{-}of\text{-}int)\ «n - 1» < x \wedge x \leq (real\text{-}of\text{-}int)\ («n»))]\!]_{\mathcal{I}e})_e$
apply (*expr-auto*)
apply (*subst infsum-mult-subset-right*)
proof −
fix *xa*
have $\{v_0{::}\mathbb{Z}.\ real\text{-}of\text{-}int\ v_0 - (1{::}\mathbb{R}) < x\ xa \wedge x\ xa \leq real\text{-}of\text{-}int\ v_0\} = \{\lceil x\ xa \rceil\}$
by (*smt* (*verit*) *Collect-cong ceiling-split singleton-conv*)
then show $real\text{-}of\text{-}int\ \lceil x\ xa \rceil =$
$infsum\ real\text{-}of\text{-}int\ \{v_0{::}\mathbb{Z}.\ real\text{-}of\text{-}int\ v_0 - (1{::}\mathbb{R}) < x\ xa \wedge x\ xa \leq real\text{-}of\text{-}int\ v_0\}$
by *simp*
qed

2.2 Inverse Iverson Bracket

axiomatization *iverson-bracket-inv* :: $('s \Rightarrow \mathbb{R}) \Rightarrow 's\ pred\ (\langle\text{-}\rangle_{\mathcal{I}})$ **where**
iverson-bracket-inv-def: $(\langle N \rangle_{\mathcal{I}} \sqsupseteq (P)) = `(N \leq [\![P]\!]_{\mathcal{I}e})`$

expr-constructor *iverson-bracket-inv*

lemma *false-0*: $[\![false]\!]_{\mathcal{I}} = (0)_e$
by (*pred-simp*)

lemma *iverson-bracket-inv-1*: $\langle (1)_e \rangle_{\mathcal{I}} = true$
by (*smt* (*verit*, *best*) *SEXP-def false-pred-def iverson-bracket-def iverson-bracket-inv-def le-funI*

le-fun-def order-antisym-conv pred-ba.order-eq-iff pred-ba.order-refl ref-by-fun-def ref-lattice.bot-least ref-lattice.top-greatest ref-preorder.order-refl taut-True taut-def true-pred-def zero-neq-one)

lemma *iverson-bracket-inv-0*: $\langle (0)_e \rangle_{\mathcal{I}} = \mathit{false}$
by (*smt* (*verit*, *ccfv-SIG*) *SEXP-def false-0 iverson-bracket-inv-def pred-ba.bot.extremum pred-ba.order-eq-iff taut-def*)

lemma *iverson-bracket-approximate-inverse*: ‘$N \leq [\![\langle N \rangle_{\mathcal{I}}]\!]_{\mathcal{I}e}$‘
by (*metis SEXP-def iverson-bracket-inv-def pred-ba.order-refl*)

lemma *iverson-bracket-inv-approximate-inverse*: $\langle [\![P]\!]_{\mathcal{I}} \rangle_{\mathcal{I}} \sqsupseteq P$
using *iverson-bracket-inv-def* **by** (*smt* (*verit*, *ccfv-SIG*) *SEXP-def taut-def*)

lemma *iverson-bracket-inv-N-0*:
assumes ‘$N \geq 0$‘
shows ‘$\neg(\langle N \rangle_{\mathcal{I}})$‘ = ‘$N = 0$‘
by (*smt* (*verit*, *best*) *SEXP-def assms false-pred-def iverson-bracket-approximate-inverse iverson-bracket-def iverson-bracket-inv-def order-antisym-conv pred-ba.bot.extremum-unique taut-def*)

lemma *iverson-bracket-inv-mono*: $[\![$ ‘$(M \leq N)$‘ $]\!] \Longrightarrow \langle M \rangle_{\mathcal{I}} \sqsupseteq \langle N \rangle_{\mathcal{I}}$
by (*smt* (*verit*) *SEXP-def dual-order.trans iverson-bracket-approximate-inverse iverson-bracket-inv-def taut-def*)

end

3 Probabilistic distributions

theory *utp-distribution*
imports
HOL.Series
utp-iverson-bracket
begin

unbundle *UTP-Syntax*
print-bundles

named-theorems *dist-defs*

3.1 Probability and distributions

definition *is-nonneg*:: (*real*, *'s*) *expr* ⇒ *bool* **where**
[*dist-defs*]: *is-nonneg e* = ‘$0 \leq e$‘

definition *is-prob*:: (*real*, *'s*) *expr* ⇒ *bool* **where**
[*dist-defs*]: *is-prob e* = ‘$0 \leq e \wedge e \leq 1$‘

definition *is-sum-1*:: (*real*, *'s*) *expr* ⇒ *bool* **where**
[*dist-defs*]: *is-sum-1 e* = $((\sum_\infty s.\ e\ s) = (1::\mathbb{R}))$

We treat a real function whose probability is always zero for any state as not a subdistribution, which allows us to conclude this function is summable or convergent.

definition *is-sum-leq-1*:: (*real*, *'s*) *expr* ⇒ *bool* **where**
[*dist-defs*]: *is-sum-leq-1 e* = $(((\sum_\infty s.\ e\ s) \leq (1::\mathbb{R})) \wedge ((\sum_\infty s.\ e\ s) > (0::\mathbb{R})))$

definition *is-dist*:: (*real*, *'s*) *expr* $\Rightarrow$ *bool* **where**
[*dist-defs*]: *is-dist e* = (*is-prob e* $\wedge$ *is-sum-1 e*)

definition *is-sub-dist*:: (*real*, *'s*) *expr* $\Rightarrow$ *bool* **where**
[*dist-defs*]: *is-sub-dist e* = (*is-prob e* $\wedge$ *is-sum-leq-1 e*)

abbreviation *is-final-distribution f* $\equiv$ ($\forall\, s_1$::$'s_1$. *is-dist* ((*curry f*) s_1))
abbreviation *is-final-sub-dist f* $\equiv$ ($\forall\, s_1$::$'s_1$. *is-sub-dist* ((*curry f*) s_1))
abbreviation *is-final-prob f* $\equiv$ ($\forall\, s_1$::$'s_1$. *is-prob* ((*curry f*) s_1))

full-exprs

3.2 Normalisaiton

Normalisation of a real-valued expression. If p is not summable, the infinite summation ($\sum_\infty$) will be equal to 0 based on the definition of infsum, then this definition here will have a problem (divide-by-zero). We need to make sure that p is summable.

definition *dist-norm*::(*real*, *'s*) *expr* $\Rightarrow$ (*real*, *'s*) *expr* (**N** -) **where**
[*dist-defs*]: *dist-norm p* = (p / ($\sum_\infty$ s. «p» s))$_e$

definition *dist-norm-final* ::(*real*, $'s_1 \times 's_2$) *expr* $\Rightarrow$ (*real*, $'s_1 \times 's_2$) *expr* ($\mathbf{N}_f$ -) **where**
[*dist-defs*]: *dist-norm-final P* = (P / ($\sum_\infty$ v_0. ([$\mathbf{v}^>$ $\rightsquigarrow$ «v_0»] † P)))$_e$

thm *dist-norm-final-def*

definition *dist-norm-alpha*::($'v \Longrightarrow 's_2$) $\Rightarrow$ (*real*, $'s_1 \times 's_2$) *expr* $\Rightarrow$ (*real*, $'s_1 \times 's_2$) *expr* ($\mathbf{N}_\alpha$ - -) **where**
[*dist-defs*]: *dist-norm-alpha x P* = (P / ($\sum_\infty$ v. ([$x^>$ $\rightsquigarrow$ «v»] † P)))$_e$

thm *dist-norm-alpha-def*
expr-constructor *dist-norm-alpha dist-norm*

definition *uniform-dist*:: ($'b \Longrightarrow 's$) $\Rightarrow$ $\mathbb{P}$ $'b$ $\Rightarrow$ (*real*, $'s \times 's$) *expr* (**infix** $\mathcal{U}$ *60*) **where**
[*dist-defs*]: *uniform-dist x A* = $\mathbf{N}_\alpha$ x ($[\![\bigsqcup$ $v \in$ «A». x := «v»$]\!]_{\mathcal{I}e}$)

lemma ($\bigsqcup$ $v \in \{\}$. x := «v») = *false*
  by (*pred-auto*)

3.3 Laws

lemma *is-prob-ibracket*: *is-prob* ($[\![p]\!]_{\mathcal{I}e}$)
  by (*simp add*: *is-prob-def expr-defs*)

lemma *is-dist-subdist*: $[\![$*is-dist p*$]\!] \Longrightarrow$ *is-sub-dist p*
  by (*simp add*: *dist-defs*)

lemma *is-final-distribution-prob*:
  assumes *is-final-distribution f*
  shows *is-final-prob f*
  using *assms is-dist-def* **by** *blast*

lemma *is-final-prob-prob*:

assumes *is-final-prob f*
shows *is-prob f*
by (*smt* (*verit*, *best*) *SEXP-def assms curry-conv is-prob-def prod.collapse taut-def*)

lemma *is-prob-final-prob*: $[\![$*is-prob P*$]\!] \Longrightarrow$ *is-final-prob P*
by (*simp add*: *is-prob-def taut-def*)

lemma *is-prob*: $[\![$*is-prob P*$]\!] \Longrightarrow (\forall s.\ P\ s \geq 0 \wedge P\ s \leq 1)$
by (*simp add*: *is-prob-def taut-def*)

lemma *is-final-prob-altdef*:
assumes *is-final-prob f*
shows $\forall s\ s'.\ f\ (s,\ s') \geq 0 \wedge f\ (s,\ s') \leq 1$
by (*metis* (*mono-tags*, *lifting*) *SEXP-def assms curry-conv is-prob-def taut-def*)

lemma *is-final-dist-subdist*:
assumes *is-final-distribution f*
shows *is-final-sub-dist f*
apply (*simp add*: *dist-defs*)
by (*smt* (*z3*) *SEXP-def assms cond-case-prod-eta curry-case-prod is-dist-def is-prob-def is-sum-1-def order.refl taut-def*)

lemma *is-final-sub-dist-prob*:
assumes *is-final-sub-dist f*
shows *is-final-prob f*
apply (*simp add*: *dist-defs*)
by (*metis* (*mono-tags*, *lifting*) *SEXP-def assms curry-def is-prob is-sub-dist-def tautI*)

lemma *is-nonneg*: (*is-nonneg e*) $\longleftrightarrow (\forall s.\ e\ s \geq 0)$
apply (*auto*)
by (*simp add*: *is-nonneg-def taut-def*)+

lemma *is-nonneg2*: $[\![$*is-nonneg p*; *is-nonneg q*$]\!] \Longrightarrow$ *is-nonneg* $(p{*}q)_e$
by (*simp add*: *is-nonneg-def taut-def*)+

lemma *dist-norm-is-prob*:
assumes *is-nonneg e*
assumes *infsum e UNIV* > 0
shows *is-prob* ($\mathbf{N}$ *e*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule allI*, *rule conjI*)
apply (*meson assms*(*1*) *assms*(*2*) *divide-nonneg-pos is-nonneg*)
apply (*insert infsum-geq-element*[**where** $f = e$])
by (*metis UNIV-I assms*(*1*) *assms*(*2*) *divide-le-eq-1-pos division-ring-divide-zero infsum-not-exists is-nonneg linordered-nonzero-semiring-class.zero-le-one*)
end

4 Probabilistic relation programming

theory *utp-prob-rel-lattice*
imports
HOL−Analysis.Infinite-Sum
HOL−Library.Complete-Partial-Order2
utp-iverson-bracket
utp-distribution

begin

unbundle *UTP-Syntax*

named-theorems *pfun-defs* **and** *ureal-defs* **and** *chains-defs*

4.1 Unit real interval *ureal*

typedef *ureal* = $\{(0::ereal)..1\}$
  morphisms *ureal2ereal ereal2ureal′*
  apply (*rule-tac* $x = 0$ **in** *exI*)
  by *auto*

find-theorems *name*:*ureal*

definition *ereal2ureal* **where**
[*ureal-defs*]: *ereal2ureal x* = *ereal2ureal′* (*min* (*max 0 x*) *1*)

definition *real2ureal* **where**
[*ureal-defs*]: *real2ureal x* = *ereal2ureal* (*ereal x*)

definition *ureal2real* **where**
[*ureal-defs*]: *ureal2real x* = (*real-of-ereal* ∘ *ureal2ereal*) *x*

lemma *enn2ereal-range*: *ereal2ureal* ‘ $\{0..1\}$ = *UNIV*
proof −
  have $\exists\, y \in \{0..1\}$. *x* = *ereal2ureal y* **for** *x*
    apply (*auto simp*: *ereal2ureal-def max-absorb2*)
    by (*meson ereal2ureal′-cases*)
  then show *?thesis*
    by (*auto simp*: *image-iff Bex-def*)
qed

lemma *type-definition-ureal′*: *type-definition ureal2ereal ereal2ureal* $\{x.\ 0 \le x \wedge x \le 1\}$
  using *type-definition-ureal*
  by (*auto simp*: *type-definition-def ereal2ureal-def max-absorb2*)

setup-lifting *type-definition-ureal′*

declare [[*coercion ereal2ureal*]]

term *a*::(*′a*::*linorder*)
instantiation *ureal* :: *complete-linorder*
begin

lift-definition *top-ureal* :: *ureal* **is** *1* **by** *simp*
lift-definition *bot-ureal* :: *ureal* **is** *0* **by** *simp*
lift-definition *sup-ureal* :: *ureal* ⇒ *ureal* ⇒ *ureal* **is** *sup* **by** (*metis le-supI le-supI1*)
lift-definition *inf-ureal* :: *ureal* ⇒ *ureal* ⇒ *ureal* **is** *inf* **by** (*metis le-infI le-infI1*)

lift-definition *Inf-ureal* :: *ureal set* ⇒ *ureal* **is** *inf 1* ∘ *Inf*
  by (*simp add*: *le-Inf-iff*)

lift-definition *Sup-ureal* :: *ureal set* ⇒ *ureal* **is** *sup 0* ∘ *Sup*
  by (*metis Sup-le-iff comp-apply sup.absorb-iff2 sup.boundedI sup.left-idem zero-less-one-ereal*)

lift-definition *less-eq-ureal* :: *ureal* ⇒ *ureal* ⇒ *bool* **is** (≤) **.**
lift-definition *less-ureal* :: *ureal* ⇒ *ureal* ⇒ *bool* **is** (<) **.**

instance
  apply *standard*
  using *less-eq-ureal.rep-eq less-ureal.rep-eq* **apply** *force*
  apply (*simp add*: *less-eq-ureal.rep-eq*)
  using *less-eq-ureal.rep-eq* **apply** *auto*[*1*]
  apply (*simp add*: *less-eq-ureal.rep-eq ureal2ereal-inject*)
  apply (*simp add*: *inf-ureal.rep-eq less-eq-ureal.rep-eq*)+
  apply (*simp add*: *sup-ureal.rep-eq*)
  apply (*simp add*: *less-eq-ureal.rep-eq sup-ureal.rep-eq*)
  apply (*simp add*: *less-eq-ureal.rep-eq sup-ureal.rep-eq*)
  apply (*smt* (*verit*) *INF-lower Inf-ureal.rep-eq le-inf-iff less-eq-ureal.rep-eq nle-le*)
  using *INF-greatest Inf-ureal.rep-eq less-eq-ureal.rep-eq ureal2ereal* **apply** *auto*[*1*]
  apply (*metis Sup-le-iff Sup-ureal.rep-eq image-eqI inf-sup-ord*(*4*) *less-eq-ureal.rep-eq*)
  using *SUP-least Sup-ureal.rep-eq less-eq-ureal.rep-eq ureal2ereal* **apply** *auto*[*1*]
  apply (*smt* (*verit*, *best*) *Inf-ureal.rep-eq ccInf-empty image-empty inf-top.right-neutral top-ureal.rep-eq ureal2ereal-inverse*)
  apply (*smt* (*verit*) *Sup-ureal.rep-eq bot-ureal.rep-eq ccSup-empty image-empty sup-bot.right-neutral ureal2ereal-inverse*)
  using *less-eq-ureal.rep-eq* **by** *force*
end

instantiation *ureal* :: {*one*,*zero*,*plus*,*times*,*mult-zero*, *zero-neq-one*, *semigroup-mult*, *semigroup-add*, *ab-semigroup-mult*, *ab-semigroup-add*, *monoid-add*, *monoid-mult*, *comm-monoid-add*}
begin

lift-definition *one-ureal* :: *ureal* **is** *1* **by** *simp*
lift-definition *zero-ureal* :: *ureal* **is** *0* **by** *simp*
lift-definition *plus-ureal* :: *ureal* ⇒ *ureal* ⇒ *ureal* **is** λ*a b*. *min 1* (*a* + *b*)
  by *simp*
lift-definition *times-ureal* :: *ureal* ⇒ *ureal* ⇒ *ureal* **is** (∗)
  by (*metis ereal-mult-mono ereal-zero-le-0-iff mult.comm-neutral*)

instance
  apply *standard*
  apply (*smt* (*verit*, *best*) *monoid.right-neutral mult.left-commute mult.monoid-axioms times-ureal.rep-eq ureal2ereal-inverse*)
  apply (*metis mult.commute times-ureal.rep-eq ureal2ereal-inverse*)
  apply *transfer*
  apply (*smt* (*verit*, *ccfv-threshold*) *add.commute add.left-commute ereal-le-add-mono2 min.absorb1 min.absorb2 nle-le*)
  apply (*metis add.commute plus-ureal.rep-eq ureal2ereal-inject*)
  apply (*smt* (*verit*, *best*) *atLeastAtMost-iff comm-monoid-add-class.add-0 min.absorb2 plus-ureal.rep-eq*

    ureal2ereal ureal2ereal-inject zero-ureal.rep-eq)
  using *one-ureal.rep-eq times-ureal.rep-eq ureal2ereal-inject* **apply** *force*
  using *one-ureal.rep-eq times-ureal.rep-eq ureal2ereal-inject* **apply** *force*
  using *times-ureal.rep-eq ureal2ereal-inject zero-ureal.rep-eq* **apply** *force*
  using *times-ureal.rep-eq ureal2ereal-inject zero-ureal.rep-eq* **apply** *force*
  using *one-ureal.rep-eq zero-ureal.rep-eq* **by** *fastforce*
end

instantiation *ureal* :: *minus*

begin

lift-definition *minus-ureal* :: *ureal* ⇒ *ureal* ⇒ *ureal* **is** $\lambda a\ b.\ max\ 0\ (a - b)$
  by (*simp add*: *ereal-diff-le-mono-left*)

instance **..**

end

instance *ureal* :: *numeral* **..**

instantiation *ureal* :: *linear-continuum-topology*
begin

definition *open-ureal* :: *ureal set* ⇒ *bool*
  where (*open* :: *ureal set* ⇒ *bool*) = *generate-topology* (*range lessThan* ∪ *range greaterThan*)

instance
proof
  show $\exists\, a\ b{::}ureal.\ a \neq b$
    using *zero-neq-one* **by** (*intro exI*)
  show $\bigwedge(x{::}ureal)\ y{::}ureal.\ x < y \Longrightarrow \exists\, z{::}ureal.\ x < z \wedge z < y$
  proof *transfer*
    fix *x y* :: *ereal*
    assume *a1*: $(0{::}ereal) \leq x \wedge x \leq (1{::}ereal)$
    assume *a2*: $(0{::}ereal) \leq y \wedge y \leq (1{::}ereal)$
    assume *a3*: $x < y$
    from *dense*[*OF this*] **obtain** *z* **where** $x < z \wedge z < y$ **..**
    with *a1 a2* **show** $\exists\, z{::}ereal{\in}\{x{::}ereal.\ (0{::}ereal) \leq x \wedge x \leq (1{::}ereal)\}.\ x < z \wedge z < y$
      by (*intro bexI*[*of - z*]) (*auto*)
  qed
qed (*rule open-ureal-def*)

end

instance *ureal* :: *ordered-comm-monoid-add*
proof
  fix *a b c*::*ureal*
  assume ∗: $a \leq b$
  then show $c + a \leq c + b$
   by (*smt* (*verit*, *best*) *Orderings.order-eq-iff ereal-add-le-add-iff less-eq-ureal.rep-eq min.mono plus-ureal.rep-eq*)
  qed

4.2 Probability functions

type-synonym $('s_1, 's_2)$ *rvfun* = $(\mathbb{R}, 's_1 \times 's_2)$ *expr*
type-synonym *'s rvhfun* = (*'s*, *'s*) *rvfun*

type-synonym $('s_1, 's_2)$ *prfun* = $(ureal, 's_1 \times 's_2)$ *expr*
type-synonym *'s prhfun* = (*'s*, *'s*) *prfun*

definition *prfun-of-rvfun*:: $('s_1, 's_2)$ *rvfun* ⇒ $('s_1, 's_2)$ *prfun* **where**
[*ureal-defs*]: *prfun-of-rvfun f* = $(real2ureal\ f)_e$

definition *rvfun-of-prfun* **where**
[*ureal-defs*]: *rvfun-of-prfun f* = $(ureal2real\ f)_e$

4.2.1 Characterise an expression over the final state

abbreviation *summable-on-final* :: $('s_1, 's_2)$ *rvfun* $\Rightarrow \mathbb{B}$ **where**
summable-on-final $p \equiv (\forall s.\ (\lambda s'.\ p\ (s,s'))$ *summable-on UNIV*$)$

abbreviation *summable-on-final2* :: $('s_1, 's_2)$ *rvfun* $\Rightarrow ('s_1, 's_2)$ *rvfun* $\Rightarrow \mathbb{B}$ **where**
summable-on-final2 $p\ q \equiv (\forall s.\ (\lambda s'.\ p(s,s') * q(s,s'))$ *summable-on UNIV*$)$

abbreviation *final-reachable* :: $('s_1, 's_2)$ *rvfun* $\Rightarrow \mathbb{B}$ **where**
final-reachable $p \equiv (\forall s.\ \exists s'.\ p\ (s, s') > 0)$

abbreviation *final-reachable2* :: $('s_1, 's_2)$ *rvfun* $\Rightarrow ('s_1, 's_2)$ *rvfun* $\Rightarrow \mathbb{B}$ **where**
final-reachable2 $p\ q \equiv (\forall s.\ \exists s'.\ p\ (s, s') > 0 \wedge q\ (s, s') > 0)$

4.3 Syntax

abbreviation *one-r* (1_R) **where**
one-r $\equiv (\lambda s.\ 1{::}real)$

abbreviation *zero-r* (0_R) **where**
zero-r $\equiv (\lambda s.\ 0{::}real)$

abbreviation *one-f* $(\mathbf{1})$ **where**
one-f $\equiv (\lambda s.\ 1{::}ureal)$

abbreviation *zero-f* $(\mathbf{0})$ **where**
zero-f $\equiv (\lambda s.\ 0{::}ureal)$

definition *pzero* :: $('s_1, 's_2)$ *prfun* (0_p) **where**
[*pfun-defs*]: *pzero* = *zero-f*

definition *pone* :: $('s_1, 's_2)$ *prfun* (1_p) **where**
[*pfun-defs*]: *pone* = *one-f*

4.3.1 Skip

abbreviation *pskip-f* (II_f) **where**
pskip-f $\equiv [\![\ II\ ]\!]_{\mathcal{I}}$

definition *pskip* :: *'s prhfun* (II_p) **where**
[*pfun-defs*]: *pskip* = *prfun-of-rvfun* (*pskip-f*)

adhoc-overloading
uskip pskip

term *II*::*'s hrel*
term *II*::*'s prhfun*
term $x := (\$x + 1)$
term $x^{>} := (\$x^{<} + 1)$

4.3.2 Assignment

abbreviation *passigns-f* **where**
passigns-f $\sigma \equiv [\![\ \langle\sigma\rangle_a\ ]\!]_{\mathcal{I}}$

definition *passigns* :: $('a, 'b)$ *psubst* $\Rightarrow ('a, 'b)$ *prfun* **where**
[*pfun-defs*]: *passigns* σ = *prfun-of-rvfun* (*passigns-f* σ)

adhoc-overloading
uassigns passigns

term $(s := e)::'s\ prhfun$
term $(s := e)::'s\ hrel$

4.3.3 Probabilistic choice

abbreviation $pchoice\text{-}f :: ('s_1,\ 's_2)\ rvfun \Rightarrow ('s_1,\ 's_2)\ rvfun \Rightarrow ('s_1,\ 's_2)\ rvfun \Rightarrow ('s_1,\ 's_2)\ rvfun$
$((\text{-} \oplus_{f\text{-}}\ \text{-})\ [61,\ 0,\ 60]\ 60)$ **where**
$pchoice\text{-}f\ P\ r\ Q \equiv (r * P + (1 - r) * Q)_e$

definition $pchoice :: ('s_1,\ 's_2)\ prfun \Rightarrow ('s_1,\ 's_2)\ prfun \Rightarrow ('s_1,\ 's_2)\ prfun \Rightarrow ('s_1,\ 's_2)\ prfun$
$((\text{-} \oplus_{\text{-}}\ \text{-})\ [61,\ 0,\ 60]\ 60)$ **where**
$[pfun\text{-}defs]$: $pchoice\ P\ r\ Q = prfun\text{-}of\text{-}rvfun\ (pchoice\text{-}f\ (rvfun\text{-}of\text{-}prfun\ P)\ (rvfun\text{-}of\text{-}prfun\ r)\ (rvfun\text{-}of\text{-}prfun\ Q))$

syntax
$\text{-}pchoice :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic\ ((if_p\ (\text{-})/\ then\ (\text{-})/\ else\ (\text{-}))\ [0,\ 61,\ 60]\ 60)$

translations
$\text{-}pchoice\ r\ P\ Q == CONST\ pchoice\ P\ (r)_e\ Q$
$\text{-}pchoice\ r\ P\ Q <= \text{-}pchoice\ (r)_e\ P\ Q$

term $if_p\ 0.5\ then\ P\ else\ Q$
term $if_p\ R\ then\ P\ else\ Q$
term $if_p\ R\ then\ P\ else\ Q = if_p\ R\ then\ P\ else\ Q$

The definition *lift-pre* below lifts a real-valued function r over the initial state to over the initial and final states. In the definition of *pchoice*, we use a general function for the weight r, which is $'s \times 's \Rightarrow \mathbb{R}$. However, now we only consider the probabilistic choice whose weight is only over the initial state. Then *lift-pre* is useful to lift a such function to a more general function used in *pchoice*.

abbreviation *lift-pre* **where** $lift\text{-}pre\ r \equiv (\lambda(s,\ s').\ r\ s)$
notation $lift\text{-}pre\ (\text{-}^{\Uparrow})$
expr-constructor *lift-pre*

4.3.4 Conditional choice

abbreviation $pcond\text{-}f :: ('s_1,\ 's_2)\ rvfun \Rightarrow ('s_1,\ 's_2)\ urel \Rightarrow ('s_1,\ 's_2)\ rvfun \Rightarrow ('s_1,\ 's_2)\ rvfun$
$((3\text{-} \lhd_f\ \text{-} \rhd/\ \text{-})\ [61,0,60]\ 60)$ **where**
$pcond\text{-}f\ P\ b\ Q \equiv (if\ b\ then\ P\ else\ Q)_e$

definition $pcond :: ('s_1,\ 's_2)\ urel \Rightarrow ('s_1,\ 's_2)\ prfun \Rightarrow ('s_1,\ 's_2)\ prfun \Rightarrow ('s_1,\ 's_2)\ prfun$ **where**
$[pfun\text{-}defs]$: $pcond\ b\ P\ Q \equiv prfun\text{-}of\text{-}rvfun\ (pcond\text{-}f\ (rvfun\text{-}of\text{-}prfun\ P)\ b\ (rvfun\text{-}of\text{-}prfun\ Q))$

syntax
$\text{-}pcond :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic\ ((if_c\ (\text{-})/\ then\ (\text{-})/\ else\ (\text{-}))\ [0,\ 61,\ 60]\ 60)$

translations
$\text{-}pcond\ b\ P\ Q == CONST\ pcond\ (b)_e\ P\ Q$
$\text{-}pcond\ b\ P\ Q <= \text{-}pcond\ (b)_e\ P\ Q$

term $if_c\ True\ then\ P\ else\ Q$

4.3.5 Sequential composition

abbreviation *pseqcomp-f* :: $'s$ *rvhfun* ⇒ $'s$ *rvhfun* ⇒ $'s$ *rvhfun* (**infixl** $;_f$ *59*) **where**
pseqcomp-f P Q ≡ $(\sum_\infty v_0.$ ([$\mathbf{v}^>$ ⇝ «v_0»] † P) ∗ ([$\mathbf{v}^<$ ⇝ «v_0»] † Q)$)_e$

definition *pseqcomp* :: $'s$ *prhfun* ⇒ $'s$ *prhfun* ⇒ $'s$ *prhfun* **where**
[*pfun-defs*]: *pseqcomp* P Q = *prfun-of-rvfun* (*pseqcomp-f* (*rvfun-of-prfun* P) (*rvfun-of-prfun* Q))

consts
pseqcomp-c :: $'a$ ⇒ $'a$ ⇒ $'a$ (**infixl** ; *59*)
adhoc-overloading
pseqcomp-c *pseqcomp-f* **and**
pseqcomp-c *pseqcomp*

term (P::($'s$, $'s$) *rvfun*) ; Q
term (P::$'s$ *prhfun*) ; Q

4.3.6 Parallel composition

abbreviation *pparallel-f* :: ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *rvfun* (**infixl** $\|_f$ *58*)
where *pparallel-f* P Q ≡ $(\mathbf{N}_f\ (P * Q)_e)$

abbreviation *pparallel-f′* :: ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *rvfun*
where *pparallel-f′* P Q ≡ ((P ∗ Q) / $(\sum_\infty s'.$ ([$\mathbf{v}^>$ ⇝ «s'»] † P) ∗ ([$\mathbf{v}^>$ ⇝ «s'»] † Q)$))_e$

lemma *pparallel-f-eq*: *pparallel-f* P Q = *pparallel-f′* P Q
apply (*simp add*: *dist-defs*)
by (*expr-auto*)

We provide four variants (different combinations of types for their parameters) of parallel composition for convenience and they use a same notation ‖. All of them defines probabilistic programs of type ($'a_1$, $'a_2$) *prfun*.

definition *pparallel* :: ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *prfun* (**infixl** $\|_p$ *58*) **where**
[*pfun-defs*]: *pparallel* P Q = *prfun-of-rvfun* (*pparallel-f* P Q)

definition *pparallel-pp* :: ($'s_1$, $'s_2$) *prfun* ⇒ ($'s_1$, $'s_2$) *prfun* ⇒ ($'s_1$, $'s_2$) *prfun* **where**
[*pfun-defs*]: *pparallel-pp* P Q = *pparallel* (*rvfun-of-prfun* P) (*rvfun-of-prfun* Q)

definition *pparallel-fp* :: ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *prfun* ⇒ ($'s_1$, $'s_2$) *prfun* **where**
[*pfun-defs*]: *pparallel-fp* P Q = *pparallel* P (*rvfun-of-prfun* Q)

definition *pparallel-pf* :: ($'s_1$, $'s_2$) *prfun* ⇒ ($'s_1$, $'s_2$) *rvfun* ⇒ ($'s_1$, $'s_2$) *prfun* **where**
[*pfun-defs*]: *pparallel-pf* P Q = *pparallel* (*rvfun-of-prfun* P) Q

no-notation *Sublist.parallel* (**infixl** ‖ *50*)
consts
parallel-c :: $'a$ ⇒ $'b$ ⇒ $'c$ (**infixl** ‖ *58*)

adhoc-overloading
parallel-c *pparallel* **and**
parallel-c *pparallel-pp* **and**
parallel-c *pparallel-fp* **and**
parallel-c *pparallel-pf* **and**
parallel-c *Sublist.parallel*

term ((P::($'s$, $'s$) *rvfun*) ‖ (Q::($'s$, $'s$) *rvfun*))

term $((P{::}('s,\ 's)\ rvfun) \parallel (Q{::}('s,\ 's)\ prfun))$
term $((P{::}('s,\ 's)\ prfun) \parallel (Q{::}('s,\ 's)\ rvfun))$
term $((P{::}('s,\ 's)\ prfun) \parallel (Q{::}('s,\ 's)\ prfun))$
term $((P{::}'s\ list) \parallel Q)$
term $([] \parallel [a])$

4.3.7 Recursion

alphabet *time* =
  $t :: enat$

In UTP, μ and ν are the weakest and strongest fix point, but there are μ and ν in Isabelle (see *utp-pred.thy*). Here, we use the same order as Isabelle, the opposite of UTP. So we define μ_p for the least fix point (also ν in Isabelle).

notation *lfp* (μ_p)
notation *gfp* (ν_p)

syntax
  -mu :: $pttrn \Rightarrow logic \Rightarrow logic\ (\mu_p\ \text{-} \cdot \text{-}\ [0,\ 10]\ 10)$
  -nu :: $pttrn \Rightarrow logic \Rightarrow logic\ (\nu_p\ \text{-} \cdot \text{-}\ [0,\ 10]\ 10)$

translations
  $\nu_p\ X \cdot P == CONST\ gfp\ (\lambda\ X.\ P)$
  $\mu_p\ X \cdot P == CONST\ lfp\ (\lambda\ X.\ P)$

term $\mu_p\ X\ \cdot\ (X{::}'s\ prhfun)$
term $lfp\ (\lambda X.\ (P{::}'s\ prhfun))$

4.4 Chains

There are similar definitions *incseq* and *decseq* in the topological space. Our definition here is more restricted to complete lattices instead of general partial order *order*, and so we can prove more specific laws with it.

definition *increasing-chain* :: $(nat \Rightarrow 'a{::}complete\text{-}lattice) \Rightarrow bool$ **where**
[*chains-defs*]: *increasing-chain* $f = (\forall\, m.\ \forall\, n.\ m \le n \longrightarrow f\ m \le f\ n)$

definition *decreasing-chain* :: $(nat \Rightarrow 'a{::}complete\text{-}lattice) \Rightarrow bool$ **where**
[*chains-defs*]: *decreasing-chain* $f = (\forall\, m.\ \forall\, n.\ m \le n \longrightarrow f\ m \ge f\ n)$

abbreviation *finite-state-incseq* $(\mathcal{FS})$ **where**
finite-state-incseq $f \equiv$ *finite* $\{s.\ ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ s) > ureal2real\ (f\ 0\ s)\}$

abbreviation *finite-state-decseq* $(\mathcal{FS})$ **where**
finite-state-decseq $f \equiv$ *finite* $\{s.\ ureal2real\ (\bigsqcap\ n{::}\mathbb{N}.\ f\ n\ s) < ureal2real\ (f\ 0\ s)\}$

4.5 While loops

definition *loopfunc* :: $('a \times 'a)\ pred \Rightarrow 'a\ prhfun \Rightarrow 'a\ prhfun \Rightarrow 'a\ prhfun\ (\mathcal{F})$ **where**
[*pfun-defs*]: *loopfunc* $b\ P\ X\ \equiv (if_c\ b\ then\ (P\ ;\ X)\ else\ II)$

definition *pwhile* :: $('a \times 'a)\ pred \Rightarrow 'a\ prhfun \Rightarrow 'a\ prhfun\ (while_p\ \text{-}\ do\ \text{-}\ od)$ **where**
[*pfun-defs*]: *pwhile* $b\ P = (\mu_p\ X \cdot \mathcal{F}\ b\ P\ X)$

definition *pwhile-top* :: $('a \times 'a)\ pred \Rightarrow 'a\ prhfun \Rightarrow 'a\ prhfun\ (while_p{}^{\top}\ \text{-}\ do\ \text{-}\ od)$ **where**

[*pfun-defs*]: *pwhile-top b P* = (ν_p *X* · $\mathcal{F}$ *b P X*)

primrec *iterate* :: ℕ ⇒ ('*a* × '*a*) *pred* ⇒ '*a prhfun* ⇒ '*a prhfun* ⇒ '*a prhfun* ($iter_p$) **where**
  iterate 0 b P X = *X*
 | *iterate* (*Suc n*) *b P X* = ($\mathcal{F}$ *b P* (*iterate n b P X*))

iterdiff constructs a form *P* ; (*P* ; ... ; (*P* ; *X*)). This particularly is used for *X* being 1_p.

primrec *iterdiff* :: ℕ ⇒ ('*a* × '*a*) *pred* ⇒ '*a prhfun* ⇒ '*a prhfun* ⇒ '*a prhfun* ($iter_d$) **where**
  iterdiff 0 b P X = *X*
 | *iterdiff* (*Suc n*) *b P X* = (if_c *b then* (*P* ; (*iterdiff n b P X*)) *else* 0_p)

definition *Pt* (*P*::'*a time-scheme prhfun*) ≡ (*P* ; *t* := $*t* + *1*)

definition *ptwhile* :: ('*a time-scheme* × '*a time-scheme*) *pred* ⇒ '*a time-scheme prhfun* ⇒ '*a time-scheme prhfun*
($while_{pt}$ - *do* - *od*) **where**
[*pfun-defs*]: *ptwhile b P* = *pwhile b* (*Pt P*)

abbreviation *iteratet* ($iterate_t$) **where** *iteratet n b P X* ≡ *iterate n b* (*Pt P*) *X*
term $iterate_t$ *0 b P* **0** = **0**

end

5 Probabilistic relation programming laws

theory *utp-prob-rel-lattice-laws*
 imports

  HOL.Series
  utp-prob-rel-lattice
begin

This version of *expr-simp* removes (*?f* = *?g*) = (∀ *x*. *?f x* = *?g x*) because this method will prevent the application of *apply* (*rule HOL.arg-cong*[*where f*=*prfun-of-rvfun*] to prove subgoals similar to *prfun-of-rvfun A* = *prfun-of-rvfun B*. This method will simplify it to something like ⋀ *s s'*. (*prfun-of-rvfun A*) (*s*, *s'*) = (*prfun-of-rvfun B*) (*s*, *s'*). Then *apply* (*rule HOL.arg-cong*[*where f*=*prfun-of-rvfun*] cannot be applied.

Our intention is to simplify *A* and *B* only with *expr-simp-1*

method *expr-simp-1* **uses** *add* =
 ((*simp add*: *expr-simps*)*?* — Perform any possible simplifications retaining the lens structure
  ; ((*simp add*: *prod.case-eq-if alpha-splits expr-defs lens-defs add*) ; — Explode the rest
   (*simp add*: *expr-defs lens-defs add*)*?*))

5.1 *ureal* laws

lemma *real-1*: *real-of-ereal* (*ureal2ereal* (*ereal2ureal'* (*ereal* (*1*::ℝ)))) = *1*
 by (*simp add*: *ereal2ureal'-inverse*)

lemma *real-1'*: *real-of-ereal* (*ureal2ereal* (*1*::*ureal*)) = *1*
 by (*simp add*: *one-ureal.rep-eq*)

lemma *ureal2ereal-mono*:
 ⟦*a* < *b*⟧ ⟹ *ureal2ereal a* < *ureal2ereal b*
 by (*simp add*: *less-ureal.rep-eq*)

lemma *ureal2real-mono*:
  assumes $a \le b$
  shows *ureal2real* $a \le$ *ureal2real* b
  apply (*simp add*: *ureal-defs*)
  by (*metis assms atLeastAtMost-iff dual-order.eq-iff ereal-less-eq*(*1*) *ereal-times*(*2*)
    less-eq-ureal.rep-eq real-of-ereal-positive-mono ureal2ereal)

lemma *ureal2real-mono-strict*:
  assumes $a < b$
  shows *ureal2real* $a <$ *ureal2real* b
  apply (*simp add*: *ureal-defs*)
  by (*metis abs-ereal-ge0 assms atLeastAtMost-iff ereal-infty-less*(*1*) *ereal-less-real-iff ereal-real*
    ereal-times(*1*) *linorder-not-less ureal2ereal ureal2ereal-mono*)

lemma *real2ureal-mono*:
  assumes $a \le b$
  shows *real2ureal* $a \le$ *real2ureal* b
  apply (*simp add*: *ureal-defs*)
  by (*smt* (*verit*) *assms atLeastAtMost-iff ereal2ureal′-inverse ereal-min less-eq-ureal.rep-eq*
    max.orderI max-def min.absorb1 min.absorb2 min.boundedE)

lemma *ureal-lower-bound*: *ureal2real* $x \ge 0$
  using *real-of-ereal-pos ureal2ereal ureal2real-def* **by** *auto*

lemma *ureal-upper-bound*: *ureal2real* $x \le 1$
  using *real-of-ereal-le-1 ureal2ereal ureal2real-def* **by** *auto*

lemma *ureal-minus-larger-zero*:
  assumes $a \le (e{::}ureal)$
  shows $a - e = 0$
  apply (*simp add*: *minus-ureal-def*)
  apply (*simp add*: *less-ureal-def ureal-defs*)
  by (*metis assms atLeastAtMost-iff ereal-0-le-uminus-iff ereal-diff-nonpos ereal-minus-eq-PInfty-iff*
    ereal-times(*1*) *less-eq-ureal.rep-eq max.absorb1 min-def ureal2ereal ureal2ereal-inverse*
    zero-ureal.rep-eq)

lemma *ureal-minus-less*:
  assumes $e > (0{::}ureal)$ $a > 0$
  shows $a - e < a$
  apply (*simp add*: *minus-ureal-def*)
  apply (*simp add*: *less-ureal-def ureal-defs*)
  by (*smt* (*verit*, *del-insts*) *assms*(*1*) *assms*(*2*) *atLeastAtMost-iff ereal2ureal′-inverse ereal-between*(*1*)
    ereal-less-PInfty ereal-times(*1*) *ereal-x-minus-x less-ureal.rep-eq linorder-not-less max-def*
    min.absorb1 minus-ureal.rep-eq nle-le ureal2ereal)

lemma *ureal-larger-minus-greater*:
  assumes $a \ge (e{::}ureal)$ $a < b$
  shows $a - e < b - e$
  apply (*simp add*: *minus-ureal-def less-ureal-def ureal-defs*)
  by (*smt* (*z3*) *antisym-conv2 assms*(*1*) *assms*(*2*) *atLeastAtMost-iff diff-add-eq-ereal*
    ereal2ureal′-inverse ereal-diff-gr0 ereal-diff-le-mono-left ereal-diff-positive
    ereal-minus(*7*) *ereal-minus-le-iff ereal-minus-minus ereal-minus-mono ereal-times*(*2*)

less-eq-ureal.rep-eq less-le-not-le linorder-not-le max.boundedI max-absorb1 max-absorb2 min-absorb1 order.trans order-eq-refl ureal2ereal ureal2ereal-inject)

lemma *ureal-minus-larger-less*:
assumes $(e::ureal) > d$ $a \geq e$
shows $a - e < a - d$
apply (*simp add*: *minus-ureal-def*)
apply (*simp add*: *less-ureal-def ureal-defs*)
by (*smt* (*verit*, *best*) *assms*(*1*) *assms*(*2*) *atLeastAtMost-iff ereal2ureal′-inverse ereal-diff-le-mono-left ereal-diff-positive ereal-less-PInfty ereal-mono-minus-cancel ereal-times*(*1*) *less-eq-ureal.rep-eq linorder-not-less max-def min-def order-le-less-trans order-less-imp-le ureal2ereal*)

lemma *ureal-plus-larger-greater*:
assumes $(e::ureal) < d$ $a + d < 1$
shows $a + e < a + d$
apply (*simp add*: *plus-ureal-def less-ureal-def ureal-defs*)
by (*smt* (*z3*) *abs-ereal-ge0 assms*(*1*) *assms*(*2*) *atLeastAtMost-iff ereal-less-PInfty ereal-less-add ereal-times*(*1*) *less-ureal.rep-eq max-def min-def not-less-iff-gr-or-eq order-le-less-trans plus-ureal.rep-eq ureal2ereal ureal2ereal-inverse*)

lemma *ureal-minus-larger-zero-unit*:
assumes $a \leq (e::ureal)$
shows $a - (a - e) = a$
apply (*simp add*: *minus-ureal-def*)
apply (*simp add*: *less-ureal-def ureal-defs*)
by (*metis assms atLeastAtMost-iff ereal-diff-nonpos ereal-minus*(*7*) *ereal-minus-eq-PInfty-iff less-eq-ureal.rep-eq max.absorb1 max-def min-def ureal2ereal ureal2ereal-inverse zero-ureal.rep-eq*)

lemma *ureal-minus-larger-zero-less*:
assumes $a \leq (e::ureal)$
shows $a - (a - e) \leq e$
by (*simp add*: *ureal-minus-larger-zero-unit assms*)

lemma *ureal-minus-less-assoc*:
assumes $a \geq (e::ureal)$
shows $a - (a - e) = a - a + e$
apply (*simp add*: *minus-ureal-def*)
apply (*simp add*: *less-ureal-def ureal-defs*)
by (*smt* (*z3*) *Orderings.order-eq-iff abs-ereal-one assms atLeastAtMost-iff diff-add-eq-ereal ereal2ureal′-inverse ereal-diff-positive ereal-minus-eq-PInfty-iff ereal-minus-minus ereal-x-minus-x less-eq-ureal.rep-eq max-absorb2 min.commute min-absorb1 minus-ureal.rep-eq one-ureal.rep-eq plus-ureal.rep-eq ureal2ereal ureal2ereal-inject ureal-minus-larger-zero*)

lemma *ureal-minus-less-diff*:
assumes $a \geq (e::ureal)$
shows $a - (a - e) = e$
apply (*simp add*: *ureal-minus-less-assoc assms*)
by (*simp add*: *ureal-minus-larger-zero*)

lemma *ureal-plus-less-1-unit*:
assumes $a + (e::ureal) < 1$
shows $a + e - a = e$
by (*smt* (*z3*) *assms atLeastAtMost-iff ereal-0-le-uminus-iff ereal-diff-add-inverse ereal-diff-positive ereal-le-add-self ereal-minus-le-iff max.absorb1 max-absorb2 min-def*

minus-ureal.rep-eq not-less-iff-gr-or-eq one-ureal.rep-eq plus-ureal.rep-eq ureal2ereal ureal2ereal-inverse)

lemma *ureal-plus-eq-1-minus-eq*:
assumes $a + (e{::}ureal) \geq 1$
shows $a + e - a = 1 - a$
by (*metis assms atLeastAtMost-iff less-ureal.rep-eq linorder-not-le one-ureal.rep-eq ureal2ereal verit-la-disequality*)

lemma *ureal-plus-eq-1-minus-less*:
assumes $a + (e{::}ureal) \geq 1$
shows $a + e - a \leq e$
by (*smt* (*verit, ccfv-SIG*) *add.commute assms atLeastAtMost-iff ereal-diff-positive ereal-minus-le-iff ereal-times*(*1*) *less-eq-ureal.rep-eq max-absorb2 min-def minus-ureal.rep-eq one-ureal.rep-eq plus-ureal.rep-eq ureal2ereal*)

lemma *ureal2ereal-add-dist*:
assumes *ureal2ereal a* + *ureal2ereal b* ≤ 1
shows *ureal2ereal* (*a* + *b*) = *ureal2ereal a* + *ureal2ereal b*
by (*simp add*: *assms plus-ureal.rep-eq*)

lemma *ureal2real-add-dist*:
assumes *ureal2real a* + *ureal2real b* ≤ 1
shows *ureal2real* (*a* + *b*) = *ureal2real a* + *ureal2real b*
by (*smt* (*verit, del-insts*) *abs-ereal-ge0 add-nonneg-nonneg assms atLeastAtMost-iff ereal-diff-add-inverse ereal-minus-eq-PInfty-iff ereal-minus-le-iff ereal-times*(*1*) *o-def one-ereal-def real-le-ereal-iff real-of-ereal-minus ureal2ereal ureal2ereal-add-dist ureal2real-def*)

lemma *ureal2real-add-dist-ureal2ereal*:
assumes *ureal2real* (*a* + *b*) = *ureal2real a* + *ureal2real b*
shows *ureal2ereal* (*a* + *b*) = *ureal2ereal a* + *ureal2ereal b*
apply (*rule ureal2ereal-add-dist*)
by (*smt* (*verit, del-insts*) *abs-ereal-ge0 add-nonneg-nonneg assms atLeastAtMost-iff ereal-diff-add-inverse ereal-minus-eq-PInfty-iff o-def one-ereal-def real-le-ereal-iff real-of-ereal-add ureal2ereal ureal2real-def*)

lemma *ureal2real-add-leq-1-ureal2ereal*:
assumes *ureal2real a* + *ureal2real b* ≤ 1
shows *ureal2ereal a* + *ureal2ereal b* ≤ 1
by (*metis assms atLeastAtMost-iff ureal2ereal ureal2real-add-dist ureal2real-add-dist-ureal2ereal*)

lemma *real2ureal-add-dist*:
assumes $a \geq 0$ $b \geq 0$ $a + b \leq 1$
shows *real2ureal* (*a* + *b*) = *real2ureal a* + *real2ureal b*
apply (*simp add*: *ureal-defs*)
by (*smt* (*verit*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *atLeastAtMost-iff ereal2ureal′-inverse ereal-less-eq*(*5*) *ereal-less-eq*(*6*) *max-absorb2 min.commute min-absorb1 plus-ereal.simps*(*1*) *plus-ureal.rep-eq ureal2ereal-inject*)

lemma *ureal-real2ureal-smaller*:
assumes $r \geq 0$
shows *ureal2real* (*real2ureal r*) $\leq r$
apply (*simp add*: *ureal-defs*)
by (*simp add*: *assms ereal2ureal′-inverse real-le-ereal-iff*)

lemma *ureal-minus-larger-than-real-minus*:

```
  shows ureal2real a − ureal2real e ≤ ureal2real (a − e)
  apply (simp add: ureal-defs minus-ureal-def)
  by (smt (verit, del-insts) abs-ereal-ge0 atLeastAtMost-iff ereal2ureal′-inverse ereal-less-eq(1)
      max-def min-def nle-le real-ereal-1 real-of-ereal-le-0 real-of-ereal-le-1 real-of-ereal-minus
      real-of-ereal-pos ureal2ereal)

lemma ureal-plus-greater:
  assumes e > (0::ureal) a < (1::ureal)
  shows a + e > a
  apply (simp add: plus-ureal-def )
  apply (simp add: less-ureal-def ureal-defs)
  by (smt (verit, del-insts) abs-ereal-zero add-nonneg-nonneg assms(1) assms(2) atLeastAtMost-iff
      ereal2ureal′-inverse ereal-between(2) ereal-eq-0(1) ereal-le-add-self ereal-less-PInfty
      ereal-real less-ureal.rep-eq linorder-not-less max.absorb1 max.cobounded1 max-def min.absorb3
      min-def one-ureal.rep-eq real-of-ereal-le-0 zero-less-one-ereal zero-ureal.rep-eq)

lemma ureal-gt-zero:
  assumes a > (0::ℝ)
  shows real2ureal a > 0
  apply (simp add: ureal-defs)
  using assms ereal2ureal′-inverse less-ureal.rep-eq zero-ureal.rep-eq by auto

lemma ureal2real-eq:
  assumes ureal2real a = ureal2real b
  shows a = b
  by (metis assms linorder-neq-iff ureal2real-mono-strict)

lemma ureal-1-minus-1-minus-r-r:
  ((1::ℝ) − rvfun-of-prfun (λs::′a × ′b. (1::ureal) − r s) (a, b)) = rvfun-of-prfun r (a, b)
  apply (simp add: ureal-defs)
  by (smt (verit, ccfv-threshold) Orderings.order-eq-iff abs-ereal-ge0 atLeastAtMost-iff
      ereal-diff-positive ereal-less-eq(1) ereal-times(1) max-def minus-ureal.rep-eq one-ureal.rep-eq
      real-ereal-1 real-of-ereal-minus ureal2ereal)

lemma ureal-1-minus-real:
  ureal2real ((1::ureal) − s) = 1 − ureal2real s
  apply (simp add: ureal-defs)
  by (metis abs-ereal-ge0 atLeastAtMost-iff ereal-diff-positive ereal-less-eq(1) ereal-times(1)
      max-def min.absorb2 min-def minus-ureal.rep-eq one-ureal.rep-eq real-ereal-1
      real-of-ereal-minus ureal2ereal)

lemma ureal-zero-0: real-of-ereal (ureal2ereal (0::ureal)) = 0
  by (simp add: zero-ureal.rep-eq)

lemma ureal-one-1: real-of-ereal (ureal2ereal (1::ureal)) = 1
  by (simp add: one-ureal.rep-eq)

lemma ureal2real-distr:
  assumes a ≥ b
  shows ureal2real (a − b) = ureal2real a − ureal2real b
  by (smt (verit) assms ereal-diff-positive less-eq-ureal.rep-eq max-def minus-ureal.rep-eq o-apply
      real-of-ereal-minus ureal2real-def ureal2real-mono ureal-minus-larger-than-real-minus)

lemma ureal2real-mult-strict-left-mono:
  assumes p > 0 c ≥ 0 c < d
```

shows (*ureal2real p*) ∗ *c* < *ureal2real p* ∗ *d*
by (*smt* (*verit*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *mult-le-less-imp-less ureal2real-mono-strict ureal-lower-bound*)

lemma *ereal-1-div*:
assumes *n* ≠ *0*
shows (*1*::*ereal*) / *ereal* (*n*::ℝ) = *ereal* (*1*/*n*)
by (*simp add*: *one-ereal-def assms*)

lemma *ereal-div*:
assumes *n* ≠ *0 m* ≠ *PInfty m* ≠ *MInfty*
shows (*m*::*ereal*) / *ereal* (*n*::ℝ) = *ereal* (*real-of-ereal m*/*n*)
apply (*simp add*: *divide-ereal-def*)
apply (*auto*)
using *assms* **apply** *blast*
by (*metis assms*(*2*) *assms*(*3*) *divide-inverse real-of-ereal.simps*(*1*) *times-ereal.simps*(*1*) *uminus-ereal.cases*)

lemma *real2uereal-inverse*:
assumes *r* ≥ *0 r* ≤ *1*
shows *real-of-ereal* (*ureal2ereal* (*ereal2ureal′ r*)) = *real-of-ereal r*
apply (*subst ereal2ureal′-inverse*)
apply (*simp add*: *atLeastAtMost-def*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *divide-le-eq-1 order-less-le*)
by (*auto*)

lemma *real2uereal-inverse′*:
assumes *r* ≥ *0 r* ≤ *1*
shows *real-of-ereal* (*ureal2ereal* (*ereal2ureal′* (*ereal r*))) = *r*
by (*simp add*: *real2uereal-inverse assms*)

lemma *real2uereal-min-inverse′*:
assumes *r* ≥ *0 r* ≤ *1*
shows *real-of-ereal* (*ureal2ereal* (*ereal2ureal′* (*min* (*ereal r*) (*1*::*ereal*)))) = *r*
by (*simp add*: *assms*(*1*) *assms*(*2*) *real2uereal-inverse′*)

lemma *ureal2rereal-inverse*: *ereal2ureal* (*ereal* (*ureal2real u*)) = *u*
apply (*simp add*: *ureal-defs*)
by (*smt* (*verit*, *best*) *Orderings.order-eq-iff atLeastAtMost-iff ereal-less*(*2*) *ereal-less-eq*(*1*) *ereal-max ereal-real ereal-times*(*1*) *min-def real-of-ereal-le-0 type-definition.Rep-inverse type-definition-ureal ureal2ereal*)

lemma *ereal2real-inverse*:
fixes *e*::*ereal*
assumes *0* ≤ *e e* ≤ (*1*::*ereal*)
shows *ureal2real* (*ereal2ureal e*) = *real-of-ereal e*
apply (*simp add*: *ureal-defs*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *real2uereal-inverse*)

lemma *real2eureal-inverse*:
assumes *0* ≤ *e e* ≤ *1*
shows *ureal2real* (*ereal2ureal* (*ereal e*)) = *e*
apply (*simp add*: *ureal-defs*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *real2uereal-inverse′*)

lemma *real2ureal-inverse*:
assumes *r* ≥ *0 r* ≤ *1*

shows *ureal2real* (*real2ureal r*) = *r*
apply (*simp add*: *ureal-defs*)
by (*simp add*: *assms ereal2ureal′-inverse real-le-ereal-iff*)

lemma *ureal2real-inverse*:
real2ureal (*ureal2real u*) = *u*
apply (*simp add*: *ureal-defs*)
by (*metis abs-ereal-ge0 atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-real ereal-times*(*1*) *max.absorb2 min.commute min.orderE ureal2ereal ureal2ereal-inverse*)

lemma *rvfun-of-prfun-simp*: *rvfun-of-prfun* $[\lambda s::'a \times 'a.\ u]_e = (\lambda s.\ ureal2real\ u)$
by (*simp add*: *SEXP-def rvfun-of-prfun-def*)

lemma *ureal2real-mult-dist*: *ureal2real* $(a * b)$ = *ureal2real* a ∗ *ureal2real* b
apply (*simp add*: *ureal-defs*)
by (*simp add*: *times-ureal.rep-eq*)

lemma *ureal2real-power-dist*: *ureal2real* $(u \widehat{}\ n)$ = (*ureal2real* u) $\widehat{}\ n$
apply (*induction n*)
apply (*simp add*: *one-ureal.rep-eq ureal2real-def*)
apply (*simp*)
using *ureal2real-mult-dist* **by** *presburger*

5.2 Infinite summation

lemma *rvfun-prob-sum1-summable*:
assumes *is-final-distribution p*
shows $\forall s.\ 0 \le p\ s \wedge p\ s \le 1$
$(\sum_\infty s.\ p\ (s_1, s)) = (1::\mathbb{R})$
$(\lambda s.\ p\ (s_1, s))$ *summable-on UNIV*
$\exists s'.\ p\ (s_1, s') > 0$
using *assms* **apply** (*simp add*: *dist-defs expr-defs*)
using *assms is-dist-def is-sum-1-def* **apply** (*metis* (*no-types*, *lifting*) *curry-conv infsum-cong*)
proof (*rule ccontr*)
assume *a1*: ¬ $(\lambda s.\ p\ (s_1, s))$ *summable-on UNIV*
from *a1* **have** *f1*: $(\sum_\infty s.\ p\ (s_1, s)) = (0::\mathbb{R})$
by (*simp add*: *infsum-def*)
then show *False*
by (*metis assms case-prod-eta curry-case-prod is-dist-def is-sum-1-def zero-neq-one*)
next
show $\exists s'::'b.\ (0::\mathbb{R}) < p\ (s_1, s')$
apply (*rule ccontr*)
proof −
assume *a1*: $\neg (\exists s'::'b.\ (0::\mathbb{R}) < p\ (s_1, s'))$
then have $\forall s'.\ (0::\mathbb{R}) = p\ (s_1, s')$
by (*meson assms is-final-distribution-prob is-final-prob-altdef order-neq-le-trans*)
then have $(\sum_\infty s.\ p\ (s_1, s)) = 0$
by *simp*
then show *False*
by (*smt* (*verit*, *best*) *assms curry-conv infsum-cong is-dist-def is-sum-1-def*)
qed
qed

lemma *rvfun-prob-sum1-summable′*:
assumes *is-final-distribution p*
shows *is-prob*(*p*)

$(\sum_\infty\ s.\ p\ (s_1,\ s)) = (1::\mathbb{R})$
summable-on-final p
final-reachable p
apply (*metis assms is-dist-def is-final-prob-prob*)
apply (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))
apply (*simp add*: *assms rvfun-prob-sum1-summable*(*3*))
by (*simp add*: *assms rvfun-prob-sum1-summable*(*4*))

lemma *rvfun-prob-sum-leq-1-summable*:
assumes *is-final-sub-dist p*
shows $\forall s.\ 0 \leq p\ s \wedge p\ s \leq 1$
$(\sum_\infty\ s.\ p\ (s_1,\ s)) \leq (1::\mathbb{R})$
$(\sum_\infty\ s.\ p\ (s_1,\ s)) > (0::\mathbb{R})$
$(\lambda s.\ p\ (s_1,\ s))$ *summable-on UNIV*
$(\lambda s.\ p\ (s_1,\ s))$ *summable-on A*
using *assms* **apply** (*simp add*: *dist-defs expr-defs*)
using *assms is-sub-dist-def is-sum-leq-1-def* **apply** (*metis* (*no-types*, *lifting*) *curry-conv infsum-cong*)
using *assms is-sub-dist-def is-sum-leq-1-def* **apply** (*metis case-prod-eta curry-case-prod*)
proof (*rule ccontr*)
assume *a1*: $\neg\ (\lambda s.\ p\ (s_1,\ s))$ *summable-on UNIV*
from *a1* **have** *f1*: $(\sum_\infty s.\ p\ (s_1,\ s)) = (0::\mathbb{R})$
by (*simp add*: *infsum-def*)
have *f2*: $(\sum_\infty s.\ p\ (s_1,\ s)) > (0::\mathbb{R})$
using *assms case-prod-eta curry-case-prod is-sub-dist-def is-sum-leq-1-def*
by (*metis a1 infsum-not-zero-is-summable*)
then show *False*
by (*simp add*: *f1*)
next
show $(\lambda s::'b.\ p\ (s_1,\ s))$ *summable-on A*
by (*smt* (*verit*, *best*) *UNIV-I assms curry-conv infsum-not-exists is-sub-dist-def is-sum-leq-1-def subsetI summable-on-cong summable-on-subset-banach*)
qed

lemma *rvfun-prob-sum-leq-1-summable′*:
assumes *is-final-sub-dist p*
shows $\forall s.\ 0 \leq p\ s \wedge p\ s \leq 1$
$(\sum_\infty\ s.\ p\ (s_1,\ s)) \leq (1::\mathbb{R})$
$(\sum_\infty\ s.\ p\ (s_1,\ s)) > (0::\mathbb{R})$
summable-on-final p
final-reachable p
using *assms rvfun-prob-sum-leq-1-summable*(*1*) **apply** *blast*
apply (*simp add*: *assms rvfun-prob-sum-leq-1-summable*(*2*))
apply (*simp add*: *assms rvfun-prob-sum-leq-1-summable*(*3*))
apply (*simp add*: *assms rvfun-prob-sum-leq-1-summable*(*4*))
apply (*auto*, *rule ccontr*)
proof −
fix *s*
assume *a1*: $\neg\ (\exists\, s'::'b.\ (0::\mathbb{R}) < p\ (s,\ s'))$
then have $\forall\, s'.\ (0::\mathbb{R}) = p\ (s,\ s')$
by (*meson assms linorder-not-le nle-le rvfun-prob-sum-leq-1-summable*(*1*))
then have $(\sum_\infty\ s'.\ p\ (s,\ s')) = 0$
by *simp*
then show *False*
by (*metis assms order-less-irrefl rvfun-prob-sum-leq-1-summable*(*3*))
qed

A probability distribution function is probabilistic, whose final states forms a distribution, and summable (convergent).

lemma *pdrfun-prob-sum1-summable*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
  shows $\forall s.\ 0 \leq f\ s \wedge f\ s \leq 1$
    $\forall s.\ 0 \leq$ *ureal2real* ($f\ s$) $\wedge$ *ureal2real* ($f\ s$) ≤ 1
    ($\sum_\infty s.$ *ureal2real* (f (s_1, s))) = (1::$\mathbb{R}$)
    ($\lambda s.$ *ureal2real* (f (s_1, s))) *summable-on UNIV*
  using *assms* **apply** (*simp add*: *dist-defs expr-defs*)
  apply (*simp add*: *ureal-defs*)
   apply (*auto*)
  using *less-eq-ureal.rep-eq ureal2ereal zero-ureal.rep-eq* **apply** *force*
  apply (*metis one-ureal.rep-eq top-greatest top-ureal.rep-eq ureal2ereal-inject*)
  using *real-of-ereal-pos ureal2ereal ureal2real-def* **apply** *auto*[*1*]
   apply (*simp add*: *ureal-upper-bound*)
proof −
  have $\forall s_1$::$'s_1$. ($\sum_\infty s.$ ((*curry* (*rvfun-of-prfun* f)) s_1) s) = 1
   using *assms* **by** (*simp add*: *is-dist-def is-sum-1-def*)
  then show *dist*: ($\sum_\infty s$::$'s_2$. *ureal2real* (f (s_1, s))) = (1::$\mathbb{R}$)
   by (*simp add*: *ureal-defs*)
  show (λs::$'s_2$. *ureal2real* (f (s_1, s))) *summable-on UNIV*
   apply (*rule ccontr*)
   by (*metis dist infsum-not-exists zero-neq-one*)
qed

lemma *pdrfun-prob-sum1-summable$'$*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
  shows $\forall s.\ 0 \leq f\ s \wedge f\ s \leq 1$
    $\forall s.\ 0 \leq$ *rvfun-of-prfun* $f\ s$ $\wedge$ *rvfun-of-prfun* $f\ s \leq 1$
    ($\sum_\infty s.$ *rvfun-of-prfun* f (s_1, s)) = (1::$\mathbb{R}$)
    ($\lambda s.$ *rvfun-of-prfun* f (s_1, s)) *summable-on UNIV*
  apply (*simp add*: *assms pdrfun-prob-sum1-summable*(*1*))
  using *assms rvfun-prob-sum1-summable*(*1*) **apply** *blast*
  apply (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))
  by (*simp add*: *assms rvfun-prob-sum1-summable*(*3*))

lemma *pdrfun-product-summable*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
  shows ($\lambda s.$ (*ureal2real* (f (s_1, s))) $*$ (*ureal2real* (g (s_1, s)))) *summable-on UNIV*
  apply (*subst summable-on-iff-abs-summable-on-real*)
  apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s.$ (*ureal2real* (f (s_1, s)))])
  apply (*metis assms infsum-not-exists pdrfun-prob-sum1-summable*(*3*)
    summable-on-iff-abs-summable-on-real zero-neq-one)
  by (*simp add*: *mult-right-le-one-le ureal-lower-bound ureal-upper-bound*)

lemma *pdrfun-product-summable-1*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
  assumes *is-prob* ($\lambda s.$ g(s_1, s))
  shows ($\lambda s.$ (*ureal2real* (f (s_1, s))) $*$ (g (s_1, s))) *summable-on UNIV*
  apply (*subst summable-on-iff-abs-summable-on-real*)
  apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s.$ (*ureal2real* (f (s_1, s)))])
  apply (*metis assms infsum-not-exists pdrfun-prob-sum1-summable*(*3*)
    summable-on-iff-abs-summable-on-real zero-neq-one)
  by (*smt* (*verit*, *del-insts*) *assms*(*2*) *is-prob mult-commute-abs mult-left-le-one-le mult-nonneg-nonneg real-norm-def ureal-lower-bound*)

lemma *pdrfun-product-summable-swap*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
shows (λs. (*ureal2real* (g (s_1, s))) $*$ (*ureal2real* (f (s_1, s)))) *summable-on UNIV*
using *pdrfun-product-summable* **by** (*smt* (*verit*, *ccfv-threshold*) *assms mult-commute-abs summable-on-cong*)

lemma *pdrfun-product-summable-1-swap*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
assumes *is-prob* (λs. g(s_1, s))
shows (λs. (g (s_1, s)) $*$ (*ureal2real* (f (s_1, s)))) *summable-on UNIV*
apply (*subst mult.commute*)
using *pdrfun-product-summable-1 assms*(*1*) *assms*(*2*) **by** *fastforce*

lemma *pdrfun-product-summable$'$*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
shows (λs. (*ureal2real* (f (s_1, s))) $*$ (*ureal2real* (g (s, s')))) *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** g = λs. (*ureal2real* (f (s_1, s)))])
apply (*metis assms infsum-not-exists pdrfun-prob-sum1-summable*(*3*)
summable-on-iff-abs-summable-on-real zero-neq-one)
by (*simp add*: *mult-right-le-one-le ureal-lower-bound ureal-upper-bound*)

lemma *pdrfun-product-summable$'$-1*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
assumes *is-prob* (λs. g(s, s'))
shows (λs. (*ureal2real* (f (s_1, s))) $*$ (g (s, s'))) *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** g = λs. (*ureal2real* (f (s_1, s)))])
apply (*metis assms*(*1*) *pdrfun-prob-sum1-summable*(*4*) *summable-on-iff-abs-summable-on-real*)
by (*smt* (*verit*, *del-insts*) *assms*(*2*) *is-prob mult-commute-abs mult-left-le-one-le mult-nonneg-nonneg real-norm-def ureal-lower-bound*)

lemma *pdrfun-product-summable$'$-swap*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
shows (λs. (*ureal2real* (g (s, s'))) $*$ (*ureal2real* (f (s_1, s)))) *summable-on UNIV*
using *pdrfun-product-summable$'$* **by** (*smt* (*verit*, *ccfv-threshold*) *assms mult-commute-abs summable-on-cong*)

lemma *ureal2real-summable-eq*:
assumes (λs. *ureal2real* (f (s_1, s))) *summable-on UNIV*
shows (λs. *real-of-ereal* (*ureal2ereal* (f (s_1, s)))) *summable-on UNIV*
using *assms ureal-defs* **by** *auto*

lemma *pdrfun-product-summable$''$*:
assumes *is-final-distribution* (*rvfun-of-prfun* (f::($'s_1$, $'s_2$) *prfun*))
shows (λs. *real-of-ereal* (*ureal2ereal* (f (s_1, s))) $*$ *real-of-ereal* (*ureal2ereal* (g (s, s'))))
summable-on UNIV
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** g = λs. *real-of-ereal* (*ureal2ereal* (f (s_1, s)))])
using *ureal2real-summable-eq* **apply** (*metis assms infsum-not-exists pdrfun-prob-sum1-summable*(*3*)
summable-on-iff-abs-summable-on-real zero-neq-one)
by (*smt* (*z3*) *atLeastAtMost-iff mult-nonneg-nonneg mult-right-le-one-le real-norm-def*
real-of-ereal-le-1 real-of-ereal-pos ureal2ereal)

lemma *summable-on-ureal-product*:
assumes *P-summable*: (λv_0. *real-of-ereal* (*ureal2ereal* (P (s, v_0)))) *summable-on UNIV*

shows $(\lambda v_0::'c$ *time-scheme. real-of-ereal* (*ureal2ereal* $(P\ (s,\ v_0)))\ *$
real-of-ereal (*ureal2ereal* $(x\ (v_0,\ b))))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda x.$ *real-of-ereal* (*ureal2ereal* $(P\ (s,\ x)))$])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms* **apply** *blast*
by (*smt* (*verit*) *atLeastAtMost-iff mult-nonneg-nonneg mult-right-le-one-le real-norm-def real-of-ereal-le-1 real-of-ereal-pos ureal2ereal*)

5.3 *is-prob*

lemma *ureal-is-prob*: *is-prob* (*rvfun-of-prfun P*)
by (*simp add*: *is-prob-def rvfun-of-prfun-def ureal-lower-bound ureal-upper-bound*)

lemma *ureal-1-minus-is-prob*: *is-prob* $((1)_e -$ *rvfun-of-prfun P*)
by (*simp add*: *is-prob-def rvfun-of-prfun-def ureal-lower-bound ureal-upper-bound*)

5.4 Inverse between *rvfun* and *prfun*

lemma *rvfun-inverse*:
assumes *is-prob P*
shows *rvfun-of-prfun* (*prfun-of-rvfun P*) = *P*
apply (*simp add*: *ureal-defs*)
apply (*expr-auto*)
proof −
fix *a b*
have $\forall s.\ P\ s \geq 0 \land P\ s \leq 1$
by (*metis* (*mono-tags*, *lifting*) *SEXP-def assms is-prob-def taut-def*)
then show *real-of-ereal* (*ureal2ereal* (*ereal2ureal′* (*min* (*max* (*0*::*ereal*) (*ereal* (*P* (*a*, *b*)))) (*1*::*ereal*)))) = *P* (*a*, *b*)
by (*simp add*: *ereal2ureal′-inverse*)
qed

lemma *prfun-inverse*:
shows *prfun-of-rvfun* (*rvfun-of-prfun P*) = *P*
apply (*simp add*: *ureal-defs*)
apply (*expr-auto*)
by (*smt* (*verit*, *best*) *atLeastAtMost-iff ereal-le-real-iff ereal-less-eq*(*1*) *ereal-real′ ereal-times*(*2*) *max.bounded-iff min-absorb1 nle-le real-of-ereal-le-0 type-definition.Rep-inverse type-definition-ureal ureal2ereal zero-ereal-def*)

lemma *rvfun-inverse-ibracket*: *rvfun-of-prfun* (*prfun-of-rvfun* ($[\![p]\!]_{\mathcal{I}}$)) = $[\![p]\!]_{\mathcal{I}}$
by (*simp add*: *is-prob-def iverson-bracket-def rvfun-inverse*)

5.5 *rvfun* laws

lemma *Sigma-Un-distrib2*:
shows *Sigma A* ($\lambda s.$ *B s*) $\cup$ *Sigma A* ($\lambda s.$ *C s*) = *Sigma A* ($\lambda s.$ (*B s* $\cup$ *C s*))
apply (*simp add*: *Sigma-def*)
by (*auto*)

lemma *prel-Sigma-UNIV-divide*:
assumes *is-final-distribution q*
shows *Sigma* (*UNIV*) ($\lambda v_0.$ $\{s'.\ q(v_0,\ s') > (0::real)\}$) $\cup$ (*Sigma* (*UNIV*) ($\lambda v_0.$ $\{s'.\ q(v_0,\ s') = (0::real)\}$))
= *Sigma* (*UNIV*) ($\lambda v_0.$ *UNIV*)

apply (*simp add*: *Sigma-Un-distrib2*)
apply (*auto*)
by (*metis antisym-conv2 assms rvfun-prob-sum1-summable*(*1*))

lemma *rvfun-infsum-1-finite-subset*:
assumes *is-final-distribution p*
shows $\forall S{::}\mathbb{P}\ \mathbb{R}.\ open\ S \longrightarrow (1{::}\mathbb{R}) \in S \longrightarrow$
$(\exists X{::}\mathbb{P}\ 'a.\ finite\ X \wedge (\forall Y{::}\mathbb{P}\ 'a.\ finite\ Y \wedge X \subseteq Y \longrightarrow (\sum s{::}'a{\in}Y.\ p\ (s_1,\ s)) \in S))$
proof −
have $(\sum_\infty s{::}'a.\ p\ (s_1,\ s)) = (1{::}\mathbb{R})$
by (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*2*))
then have *has-sum* $(\lambda s{::}'a.\ p\ (s_1,\ s))$ *UNIV* $(1{::}\mathbb{R})$
by (*metis has-sum-infsum infsum-not-exists zero-neq-one*)
then have $(sum\ (\lambda s{::}'a.\ p\ (s_1,\ s)) \longrightarrow (1{::}\mathbb{R}))$ (*finite-subsets-at-top UNIV*)
using *has-sum-def* **by** *blast*
then have $\forall S{::}\mathbb{P}\ \mathbb{R}.\ open\ S \longrightarrow (1{::}\mathbb{R}) \in S \longrightarrow (\forall_F\ x{::}\mathbb{P}\ 'a\ in\ finite\text{-}subsets\text{-}at\text{-}top\ UNIV.\ (\sum s{::}'a{\in}x.\ p\ (s_1,\ s)) \in S)$
by (*simp add*: *tendsto-def*)
thus *?thesis*
by (*simp add*: *eventually-finite-subsets-at-top*)
qed

lemma *rvfun-product-summable-subdist*:
assumes *is-final-sub-dist p is-prob q*
shows $(\lambda s{::}'a.\ p\ (x,\ s) * q\ (s,\ y))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s{::}'a.\ p\ (x,\ s)$])
apply (*metis assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*4*) *summable-on-iff-abs-summable-on-real*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob mult-left-le rvfun-prob-sum-leq-1-summable*(*1*))

lemma *rvfun-product-summable-dist*:
assumes *is-final-distribution p*
assumes $\forall s.\ q\ s \leq 1 \wedge q\ s \geq 0$
shows $(\lambda s{::}'a.\ p\ (x,\ s) * q\ (s,\ y))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s{::}'a.\ p\ (x,\ s)$])
apply (*metis assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-iff-abs-summable-on-real*)
using *assms*(*2*) **by** (*smt* (*verit*) *SEXP-def mult-right-le-one-le norm-mult real-norm-def*)

lemma *rvfun-product-prob-dist-leq-1*:
assumes *is-final-distribution p*
assumes *is-prob q*
shows $(\sum_\infty s{::}'a.\ p\ (x,\ s) * q\ (s,\ y)) \leq (1{::}\mathbb{R})$
proof −
have $(\sum_\infty s{::}'a.\ p\ (x,\ s) * q\ (s,\ y)) \leq (\sum_\infty s{::}'a.\ p\ (x,\ s))$
apply (*subst infsum-mono*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob rvfun-product-summable-dist*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob mult-right-le-one-le rvfun-prob-sum1-summable*(*1*))
by *simp*
also have ... = *1*
by (*metis assms*(*1*) *rvfun-prob-sum1-summable*(*2*))
then show *?thesis*
using *calculation* **by** *presburger*
qed

lemma *rvfun-product-prob-sub-dist-leq-1*:
  assumes *is-final-sub-dist p*
  assumes *is-prob q*
  shows $(\sum_\infty s::'a.\ p\ (x,\ s) * q\ (s,\ y)) \leq (1::\mathbb{R})$
proof –
  have $(\sum_\infty s::'a.\ p\ (x,\ s) * q\ (s,\ y)) \leq (\sum_\infty s::'a.\ p\ (x,\ s))$
    apply (*subst infsum-mono*)
    apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob rvfun-product-summable-subdist*)
    apply (*simp add*: *assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*4*))
    apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob mult-right-le-one-le rvfun-prob-sum-leq-1-summable*(*1*))
    by *simp*
  also have ... $\leq$ *1*
    by (*metis assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*2*))
  then show *?thesis*
    using *calculation* **by** *linarith*
qed

lemma *rvfun-product-summable-summable*:
  assumes $\forall x.$ ((*curry p*) *x*) *summable-on UNIV*
  assumes *is-prob p is-prob q*
  shows $(\lambda s::'a.\ p\ (x,\ s) * q\ (s,\ y))$ *summable-on UNIV*
  apply (*subst summable-on-iff-abs-summable-on-real*)
  apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s::'a.\ p\ (x,\ s)$])
  apply (*subst summable-on-iff-abs-summable-on-real*)
  apply (*smt* (*verit*, *del-insts*) *abs-of-nonneg assms*(*1*) *assms*(*2*) *curry-conv is-prob real-norm-def summable-on-cong*)
  by (*simp add*: *assms*(*2*) *assms*(*3*) *is-prob mult-left-le*)

lemma *rvfun-product-summable'*:
  assumes *is-final-distribution p*
  assumes *is-final-distribution q*
  shows $(\lambda s::'a.\ p\ (x,\ s) * q\ (s,\ y))$ *summable-on UNIV*
  apply (*rule rvfun-product-summable-dist*)
  apply (*simp add*: *assms*(*1*))
  using *assms*(*2*) *rvfun-prob-sum1-summable*(*1*) **by** *blast*

lemma *rvfun-joint-prob-summable-on-product*:
  assumes *is-final-prob p*
  assumes *is-final-prob q*
  assumes *summable-on-final p* $\vee$ *summable-on-final q*
  shows *summable-on-final2 p q*
  apply (*auto*)
proof –
  fix *s*
  show $(\lambda s'::'b.\ p\ (s,\ s') * q\ (s,\ s'))$ *summable-on UNIV*
  proof (*cases summable-on-final p*)
    case *True*
    then show *?thesis*
      apply (*subst summable-on-iff-abs-summable-on-real*)
      apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s'.\ p\ (s,\ s')$])
      apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
      using *assms*(*3*) **apply** *blast*
      apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-final-prob-altdef*)
      by (*simp add*: *assms*(*1*) *assms*(*2*) *is-final-prob-altdef mult-right-le-one-le*)
  next

case *False*
then have $(\lambda s'.\ q\ (s,\ s'))$ *summable-on UNIV*
using *assms*(*3*) **by** *blast*
then show *?thesis*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s'.\ q\ (s,\ s')$])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms*(*3*) **apply** *blast*
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-final-prob-altdef*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *is-final-prob-altdef mult-left-le-one-le*)
qed
qed

lemma *rvfun-joint-prob-summable-on-product-dist*:
assumes *is-final-distribution p*
assumes *is-prob q*
shows $(\lambda s{::}'a.\ p\ (x,\ s) * q\ (x,\ s))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s{::}'a.\ p\ (x,\ s)$])
apply (*metis assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-iff-abs-summable-on-real*)
using *assms*(*2*) **by** (*smt* (*verit*) *is-prob SEXP-def mult-right-le-one-le norm-mult real-norm-def*)

lemma *rvfun-joint-prob-summable-on-product-dist'*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda s{::}'a.\ p\ (x,\ s) * q\ (x,\ s))$ *summable-on UNIV*
apply (*rule rvfun-joint-prob-summable-on-product-dist*)
apply (*simp add*: *assms*(*1*))
using *assms*(*2*) *rvfun-prob-sum1-summable*(*1*) **by** (*simp add*: *is-dist-def is-final-prob-prob*)

lemma *rvfun-joint-prob-sum-ge-zero*:
assumes $\forall s.\ P\ s \geq (0{::}\mathbb{R})\ \forall s.\ Q\ s \geq 0$
$\forall s_1.\ (\lambda s'.\ P\ (s_1,\ s') * Q\ (s_1,\ s'))$ *summable-on UNIV*
$\forall s_1.\ \exists s'.\ P\ (s_1,\ s') > 0 \wedge Q\ (s_1,\ s') > 0$
shows $\forall s_1.\ ((\sum_\infty s'.\ P\ (s_1,\ s') * Q\ (s_1,\ s')) > 0)$
proof (*rule allI*)
fix s_1
let $?P = \lambda s'.\ P\ (s_1,\ s') > 0 \wedge Q\ (s_1,\ s') > 0$
have *f1*: $?P\ (SOME\ s'.\ ?P\ s')$
apply (*rule someI-ex*[**where** $P{=}?P$])
using *assms* **by** *blast*
have *f2*: $(\lambda s.\ P\ (s_1,\ s) * Q\ (s_1,\ s))\ (SOME\ s'.\ ?P\ s') \leq (\sum_\infty s'.\ P\ (s_1,\ s') * Q\ (s_1,\ s'))$
apply (*rule infsum-geq-element*)
apply (*simp add*: *assms*(*1*−*2*))
apply (*simp add*: *assms*(*3*))
by *auto*
also have *f3*: ... > 0
by (*smt* (*verit*, *ccfv-threshold*) *f1 f2 mult-pos-pos*)
then show $(0{::}\mathbb{R}) < (\sum_\infty s'{::}'b.\ P\ (s_1,\ s') * Q\ (s_1,\ s'))$
by *linarith*
qed

lemma *prfun-in-0-1*: $(curry\ (rvfun\text{-}of\text{-}prfun\ Q))\ x\ y \geq 0 \wedge (curry\ (rvfun\text{-}of\text{-}prfun\ Q))\ x\ y \leq 1$
by (*simp add*: *is-prob ureal-is-prob*)

lemma *prfun-in-0-1'*: (*rvfun-of-prfun Q*) *s* ≥ *0* ∧ (*rvfun-of-prfun Q*) *s* ≤ *1*
  apply (*simp add*: *ureal-defs*)
  apply (*auto*)
  using *real-of-ereal-pos ureal2ereal* **apply** *fastforce*
  using *ureal2real-def ureal-upper-bound* **by** *auto*

lemma *prfun-infsum-over-pair-fst-discard*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::*'a prhfun*))
  shows ($\sum_\infty$ (*s*::*'a*, v_0::*'a*) ∈ {(*s*::*'a*, v_0::*'a*) | *s* v_0. put_x v_0 (*e* v_0) = *s*}. *rvfun-of-prfun P* (s_1, v_0)) =
  ($\sum_\infty$ v_0::*'a*. *rvfun-of-prfun P* (s_1, v_0))
  apply (*simp add*: *pdrfun-prob-sum1-summable' assms*)
  — Definition of *infsum*
  apply (*rule infsumI*)
  apply (*simp add*: *has-sum-def*)
  apply (*subst topological-tendstoI*)
  apply (*auto*)
  apply (*simp add*: *eventually-finite-subsets-at-top*)
proof −
  fix *S*::ℙ ℝ
  assume *a1*: *open S*
  assume *a2*: (*1*::ℝ) ∈ *S*
  — How to improve this proof? Forward proof. Focus on the goal f0 9 lines below
  have ($\sum_\infty$*s*::*'a*. *rvfun-of-prfun P* (s_1, *s*)) = (*1*::ℝ)
    by (*simp add*: *pdrfun-prob-sum1-summable' assms*)
  then have *has-sum* (λ*s*::*'a*. *rvfun-of-prfun P* (s_1, *s*)) *UNIV* (*1*::ℝ)
    by (*metis has-sum-infsum infsum-not-exists zero-neq-one*)
  then have (*sum* (λ*s*::*'a*. *rvfun-of-prfun P* (s_1, *s*)) ⟶ (*1*::ℝ)) (*finite-subsets-at-top UNIV*)
    using *has-sum-def* **by** *blast*
  then have $\forall_F$ *x*::ℙ *'a in finite-subsets-at-top UNIV*. (∑ *s*::*'a*∈*x*. *rvfun-of-prfun P* (s_1, *s*)) ∈ *S*
    using *a1 a2 tendsto-def* **by** *blast*
  then have *f0*: ∃ *X*::ℙ *'a*. *finite X* ∧ (∀ *Y*::ℙ *'a*. *finite Y* ∧ *X* ⊆ *Y* ⟶
      (∑ *s*::*'a*∈*Y*. *rvfun-of-prfun P* (s_1, *s*)) ∈ *S*)
    by (*simp add*: *eventually-finite-subsets-at-top*)
  then show ∃ *X*::*'a rel*. *finite X* ∧ *X* ⊆ {*uu*::*'a* × *'a*. ∃ v_0::*'a*. *uu* = (put_x v_0 (*e* v_0), v_0)} ∧
        (∀ *Y*::*'a rel*.
            finite Y ∧ *X* ⊆ *Y* ∧ *Y* ⊆ {*uu*::*'a* × *'a*. ∃ v_0::*'a*. *uu* = (put_x v_0 (*e* v_0), v_0)} ⟶
            (∑ *x*::*'a* × *'a*∈*Y*. *case x of* (*s*::*'a*, v_0::*'a*) ⇒ *rvfun-of-prfun P* (s_1, v_0)) ∈ *S*)
  proof −
    assume *a11*: ∃ *X*::ℙ *'a*. *finite X* ∧ (∀ *Y*::ℙ *'a*. *finite Y* ∧ *X* ⊆ *Y* ⟶
      (∑ *s*::*'a*∈*Y*. *rvfun-of-prfun P* (s_1, *s*)) ∈ *S*)
    have *f1*: *finite*
      {*uu*::*'a* × *'a*. ∃ v_0::*'a*. v_0 ∈ (*SOME X*::ℙ *'a*.
        finite X ∧ (∀ *Y*::ℙ *'a*. *finite Y* ∧ *X* ⊆ *Y* ⟶ (∑ *s*::*'a*∈*Y*. *rvfun-of-prfun P* (s_1, *s*)) ∈ *S*))
      ∧ *uu* = (put_x v_0 (*e* v_0), v_0)}
      apply (*subst finite-Collect-bounded-ex*)
      apply (*smt* (*verit*, *ccfv-threshold*) *CollectD a11 rev-finite-subset someI-ex subset-iff*)
      by (*auto*)
    show *?thesis*

      apply (*rule-tac x* = {(put_x v_0 (*e* v_0), v_0) | v_0 .
        v_0 ∈ (*SOME X*::ℙ *'a*. *finite X* ∧ (∀ *Y*::ℙ *'a*. *finite Y* ∧ *X* ⊆ *Y* ⟶
        (∑ *s*::*'a*∈*Y*. *rvfun-of-prfun P* (s_1, *s*)) ∈ *S*))} **in** *exI*)
    apply (*rule conjI*)

using *f1* **apply** (*smt* (*verit*, *best*) *Collect-mono rev-finite-subset*)
apply (*auto*)
proof −
fix $Y::'a\ rel$
assume *a111*: *finite* Y
assume *a112*: $\{uu::'a \times 'a.$
$\exists\, v_0::'a.$
$uu = (put_x\ v_0\ (e\ v_0),\ v_0) \wedge$
$v_0 \in (SOME\ X::\mathbb{P}\ 'a.\ finite\ X \wedge (\forall\ Y::\mathbb{P}\ 'a.\ finite\ Y \wedge X \subseteq Y \longrightarrow (\sum s::'a{\in}Y.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1,\ s)) \in S))\}$
$\subseteq Y$
assume *a113*: $Y \subseteq \{uu::'a \times 'a.\ \exists\, v_0::'a.\ uu = (put_x\ v_0\ (e\ v_0),\ v_0)\}$
have *f11*: $(\sum s::'a{\in}Range\ Y.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1,\ s)) \in S$
using *a11 a111 a112*
by (*smt* (*verit*, *del-insts*) *Range-iff finite-Range mem-Collect-eq subset-iff verit-sko-ex-indirect*)
have *f12*: *inj-on* $(\lambda v_0.\ (put_x\ v_0\ (e\ v_0),\ v_0))$ (*Range* Y)
using *inj-on-def* **by** *blast*
have *f13*: $(\sum x::'a \times 'a{\in}Y.\ case\ x\ of\ (s::'a,\ v_0::'a) \Rightarrow rvfun\text{-}of\text{-}prfun\ P\ (s_1,\ v_0)) =$
$(\sum s::'a{\in}Range\ Y.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1,\ s))$
apply (*rule sum.reindex-cong*[**where** $l = (\lambda v_0.\ (put_x\ v_0\ (e\ v_0),\ v_0))$ **and** $B = Range\ Y$])
apply (*simp add*: *f12*)
using *a113* **by** (*auto*)
show $(\sum x::'a \times 'a{\in}Y.\ case\ x\ of\ (s::'a,\ v_0::'a) \Rightarrow rvfun\text{-}of\text{-}prfun\ P\ (s_1,\ v_0)) \in S$
using *f11 f13* **by** *presburger*
qed
qed
qed

lemma *prfun-minus-distribution*:
fixes $X\ Y$ :: $'a\ prhfun$
assumes $X \geq Y$
shows $rvfun\text{-}of\text{-}prfun\ X - rvfun\text{-}of\text{-}prfun\ Y = rvfun\text{-}of\text{-}prfun\ (X - Y)$
apply (*subst fun-eq-iff*)
apply (*rule allI*)
apply (*simp add*: *ureal-defs*)
by (*smt* (*verit*, *del-insts*) *abs-ereal-ge0 assms atLeastAtMost-iff ereal-diff-positive ereal-less-eq*(*1*) *ereal-times*(*1*) *le-fun-def less-eq-ureal.rep-eq max-def minus-ureal.rep-eq nle-le real-of-ereal-minus ureal2ereal*)

5.6 Probabilistic programs

5.6.1 Bottom and Top

We are not able to use ⊥ for bot because this notation has been used in UTP as top.

lemma *ureal-bot-zero*: $\bot = \mathbf{0}$
by (*metis bot-apply bot-ureal.rep-eq ureal2ereal-inject zero-ureal.rep-eq*)

lemma *ureal-top-one*: $\top = \mathbf{1}$
by (*metis one-ureal.rep-eq top-apply top-ureal.rep-eq ureal2ereal-inject*)

lemma *ureal-zero*: $rvfun\text{-}of\text{-}prfun\ \mathbf{0} = (0)_e$
apply (*simp add*: *ureal-defs*)
by (*simp add*: *zero-ureal.rep-eq*)

lemma *ureal-zero'*: $prfun\text{-}of\text{-}rvfun\ (0)_e = \mathbf{0}$

apply (*simp add*: *ureal-defs*)
by (*metis SEXP-apply ureal2ereal-inverse zero-ureal.rep-eq*)

lemma *ureal-one*: *rvfun-of-prfun* $\mathbf{1}$ = $(1)_e$
apply (*simp add*: *ureal-defs*)
by (*simp add*: *one-ureal.rep-eq*)

lemma *ureal-one'*: *prfun-of-rvfun* $(1)_e = \mathbf{1}$
apply (*simp add*: *ureal-defs*)
by (*metis SEXP-def one-ereal-def one-ureal.rep-eq ureal2ereal-inverse*)

lemma *ureal-bottom-least*: $\mathbf{0} \leq P$
apply (*simp add*: *le-fun-def pfun-defs ureal-defs*)
apply (*auto*)
by (*metis bot.extremum bot-ureal.rep-eq ureal2ereal-inject zero-ureal.rep-eq*)

lemma *ureal-bottom-least'*: $0_p \leq P$
apply (*simp add*: *pfun-defs*)
by (*rule ureal-bottom-least*)

lemma *ureal-top-greatest*: $P \leq \mathbf{1}$
apply (*simp add*: *le-fun-def pfun-defs ureal-defs*)
apply (*auto*)
using *less-eq-ureal.rep-eq one-ureal.rep-eq ureal2ereal* **by** *auto*

lemma *ureal-top-greatest'*: $P \leq 1_p$
by (*metis le-fun-def one-ureal.rep-eq pone-def top-greatest top-ureal.rep-eq ureal2ereal-inject*)

lemma *ureal-rzero-0*: $[0_R]_e\ s = 0$
by *simp*

5.6.2 Skip

lemma *rvfun-skip-f-is-prob*: *is-prob* II_f
by (*simp add*: *is-prob-def iverson-bracket-def*)

lemma *rvfun-skip-f-is-dist*: *is-final-distribution* II_f
apply (*simp add*: *dist-defs expr-defs*)
by (*simp add*: *infsum-singleton-1 skip-def*)

lemma *rvfun-skip-inverse*: *rvfun-of-prfun* (*prfun-of-rvfun* II_f) = II_f
by (*simp add*: *is-prob-def iverson-bracket-def rvfun-inverse*)

lemma *rvfun-skip_f-simp*: $II_f = (\lambda(s, s').\ if\ s = s'\ then\ 1\ else\ 0)$
by (*expr-auto add*: *skip-def*)

theorem *prfun-skip*:
assumes *wb-lens x*
shows (*II*::*'a prhfun*) = ($x := \$x$)
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *expr-defs skip-def*)
by (*simp add*: *assigns-r-def assms*)

theorem *prfun-skip'*:
shows *rvfun-of-prfun* (*II*) = *pskip_f*

apply (*simp add*: *pfun-defs*)
using *rvfun-skip-inverse* **by** *blast*

lemma *prfun-skip-id*: II_p $(s,\ s) = 1$
apply (*simp add*: *pfun-defs ureal-defs*)
by (*simp add*: *ereal2ureal-def iverson-bracket-def one-ereal-def one-ureal-def skip-def*)

lemma *prfun-skip-not-id*:
assumes $s \neq s'$
shows II_p $(s,\ s') = 0$
apply (*simp add*: *pfun-defs ureal-defs skip-def*)
by (*smt* (*verit*, *ccfv-SIG*) *SEXP-def assms case-prod-conv ereal2ureal-def iverson-bracket-def zero-ereal-def zero-ureal-def*)

5.6.3 Assignment

lemma *rvfun-assignment-is-prob*: *is-prob* (*passigns-f* σ)
by (*simp add*: *is-prob-def iverson-bracket-def*)

lemma *rvfun-assignment-is-dist*: *is-final-distribution* (*passigns-f* σ)
apply (*simp add*: *dist-defs expr-defs*)
by (*simp add*: *infsum-singleton-1 assigns-r-def*)

lemma *rvfun-assignment-inverse*: *rvfun-of-prfun* (*prfun-of-rvfun* (*passigns-f* σ)) = (*passigns-f* σ)
by (*simp add*: *is-prob-def iverson-bracket-def rvfun-inverse*)

5.6.4 Probabilistic choice

term (*rvfun-of-prfun r*)$^{\Uparrow}$
lemma *rvfun-pchoice-is-prob*:
assumes *is-prob P is-prob Q*
shows *is-prob* ($P \oplus_{f_{(rvfun\text{-}of\text{-}prfun\ r)^{\Uparrow}}} Q$)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob prfun-in-0-1'*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-final-prob-altdef is-prob-final-prob prfun-in-0-1'*)

lemma *rvfun-pchoice-is-prob'*:
assumes *is-prob P is-prob Q*
shows *is-prob* ($P \oplus_{f_{(\lambda s.\ ureal2real\ r)}} Q$)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob ureal-lower-bound ureal-upper-bound*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-final-prob-altdef is-prob-final-prob ureal-lower-bound ureal-upper-bound*)

lemma *rvfun-pchoice-is-dist*:
assumes *is-final-distribution P is-final-distribution Q*
shows *is-final-distribution* ($P \oplus_{f_{(rvfun\text{-}of\text{-}prfun\ r)^{\Uparrow}}} Q$)
apply (*simp add*: *dist-defs expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *prfun-in-0-1' rvfun-prob-sum1-summable*(*1*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le prfun-in-0-1' rvfun-prob-sum1-summable*(*1*))
apply (*subst infsum-add*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst summable-on-cmult-right*)

apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*))+
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-prob-sum1-summable*(*2*))

lemma *rvfun-pchoice-is-dist′*:
assumes *is-final-distribution P is-final-distribution Q*
shows *is-final-distribution* ($P \oplus_{f(\lambda s.\ ureal2real\ r)} Q$)
apply (*simp add*: *dist-defs expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-prob-sum1-summable*(*1*) *ureal-lower-bound ureal-upper-bound*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le rvfun-prob-sum1-summable*(*1*) *ureal-lower-bound ureal-upper-bound*)
apply (*subst infsum-add*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst summable-on-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*))+
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-prob-sum1-summable*(*2*))

lemma *rvfun-pchoice-is-dist-c*:
assumes *is-final-distribution P is-final-distribution Q*
$r \geq 0$ $r \leq 1$
shows *is-final-distribution* ($P \oplus_{f(\lambda s.\ r)} Q$)
apply (*simp add*: *dist-defs expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *rvfun-prob-sum1-summable*(*1*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *convex-bound-le rvfun-prob-sum1-summable*(*1*))
apply (*subst infsum-add*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst summable-on-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*))+
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-prob-sum1-summable*(*2*))

lemma *rvfun-pchoice-is-dist-c′*:
assumes *is-final-distribution P is-final-distribution Q*
$r \geq 0$ $r \leq 1$
shows *is-final-distribution* ($P \oplus_{f[(\lambda s.\ r)]_e} Q$)
apply (*simp add*: *dist-defs expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *rvfun-prob-sum1-summable*(*1*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *convex-bound-le rvfun-prob-sum1-summable*(*1*))
apply (*subst infsum-add*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst summable-on-cmult-right*)
apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*))+
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
apply (*subst infsum-cmult-right*)

apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *rvfun-prob-sum1-summable*(*2*))

lemma *rvfun-pchoice-inverse*:
assumes *is-prob P is-prob Q*
shows *rvfun-of-prfun* (*prfun-of-rvfun* ($P \oplus_{f(rvfun\text{-}of\text{-}prfun\ r)}$ Q)) = ($P \oplus_{f(rvfun\text{-}of\text{-}prfun\ r)}$ Q)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob prfun-in-0-1′*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-prob prfun-in-0-1′*)

lemma *rvfun-pchoice-inverse-pre*:
assumes *is-prob P is-prob Q*
shows *rvfun-of-prfun* (*prfun-of-rvfun* ($P \oplus_{f(rvfun\text{-}of\text{-}prfun\ r)^{\Uparrow}}$ Q)) = ($P \oplus_{f(rvfun\text{-}of\text{-}prfun\ r)^{\Uparrow}}$ Q)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob prfun-in-0-1′*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-prob prfun-in-0-1′*)

lemma *rvfun-pchoice-inverse-pre′*:
assumes *is-prob P is-prob Q*
shows *rvfun-of-prfun* (*prfun-of-rvfun* (*pchoice-f P* [(*rvfun-of-prfun r*)$^{\Uparrow}$]$_e$ *Q*)) = *pchoice-f P* [(*rvfun-of-prfun r*)$^{\Uparrow}$]$_e$ *Q*
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob prfun-in-0-1′*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-prob prfun-in-0-1′*)

lemma *rvfun-pchoice-inverse-c*:
assumes *is-prob P is-prob Q*
shows *rvfun-of-prfun* (*prfun-of-rvfun* ($P \oplus_{f(\lambda s.\ ureal2real\ r)}$ Q)) = ($P \oplus_{f(\lambda s.\ ureal2real\ r)}$ Q)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob ureal-lower-bound ureal-upper-bound*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-final-prob-altdef is-prob-final-prob ureal-lower-bound ureal-upper-bound*)

lemma *rvfun-pchoice-inverse-c′*:
assumes *is-prob P is-prob Q*
assumes $0 \leq r \wedge r \leq (1{::}ureal)$
shows *rvfun-of-prfun* (*prfun-of-rvfun* (*pchoice-f P* [($\lambda s.$ *ureal2real r*)]$_e$ *Q*)) = (*pchoice-f P* [($\lambda s.$ *ureal2real r*)]$_e$ *Q*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob ureal-lower-bound ureal-upper-bound*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-bound-le is-final-prob-altdef is-prob-final-prob ureal-lower-bound ureal-upper-bound*)

lemma *rvfun-pchoice-inverse-c′′*:
assumes *is-prob P is-prob Q*

assumes $0 \leq r \wedge r \leq (1::\mathbb{R})$
shows *rvfun-of-prfun* (*prfun-of-rvfun* (*pchoice-f P* $[(\lambda s.\ r)]_e$ *Q*)) = (*pchoice-f P* $[(\lambda s.\ r)]_e$ *Q*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*rule rvfun-inverse*)
apply (*simp add*: *is-prob-def expr-defs*, *auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *is-prob*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *convex-bound-le is-prob*)

theorem *prfun-pchoice-altdef*:
if_p *r then P else Q*
= *prfun-of-rvfun* ($\bullet$(*rvfun-of-prfun r*) $*$ $\bullet$(*rvfun-of-prfun P*) + (*1* − $\bullet$(*rvfun-of-prfun* (*r*))) $*$ $\bullet$(*rvfun-of-prfun Q*))$_e$
by (*simp add*: *pfun-defs ureal-defs*)

theorem *prfun-pchoice-commute*: if_p *r then P else Q* = if_p *1* − *r then Q else P*
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*expr-auto*)
apply (*simp add*: *ureal-1-minus-1-minus-r-r*)
apply (*simp add*: *ureal-defs*)
apply (*rule disjI2*)
by (*metis Orderings.order-eq-iff abs-ereal-ge0 atLeastAtMost-iff ereal-diff-positive ereal-less-eq*(*1*) *ereal-times*(*1*) *max.absorb2 minus-ureal.rep-eq one-ureal.rep-eq real-ereal-1 real-of-ereal-minus ureal2ereal*)

theorem *prfun-pchoice-zero*: if_p *0 then P else Q* = *Q*
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *ureal-defs*)
apply (*simp add*: *ureal-zero-0*)
apply (*subst fun-eq-iff*, *auto*)
by (*metis abs-ereal-ge0 add-0 atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-real ereal-times*(*1*) *max.absorb2 max-min-same*(*1*) *min.commute plus-ureal.rep-eq ureal2ereal ureal2ereal-inverse zero-ureal.rep-eq*)

theorem *prfun-pchoice-one*: if_p *1 then P else Q* = *P*
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *ureal-defs*)
apply (*simp add*: *ureal-one-1*)
apply (*subst fun-eq-iff*, *auto*)
by (*metis abs-ereal-ge0 add-0 atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-real ereal-times*(*1*) *max.absorb2 max-min-same*(*1*) *min.commute plus-ureal.rep-eq ureal2ereal ureal2ereal-inverse zero-ureal.rep-eq*)

theorem *prfun-pchoice-zero'*:
fixes w_1 :: *'a* ⇒ *ureal*
assumes ‘w_1 = *0*‘
shows $P \oplus_{w_1{}^\Uparrow} Q = Q$
apply (*simp add*: *pfun-defs*)
proof −
have *f1*: *rvfun-of-prfun* ($w_1{}^\Uparrow$) = (*0*)$_e$
apply (*simp add*: *ureal-defs*)
apply (*subst fun-eq-iff*, *auto*)
by (*metis* (*mono-tags*, *lifting*) *SEXP-def assms real-of-ereal-0 taut-def zero-ureal.rep-eq*)
show *prfun-of-rvfun* (*pchoice-f* (*rvfun-of-prfun P*) (*rvfun-of-prfun* ($w_1{}^\Uparrow$)) (*rvfun-of-prfun Q*)) = *Q*
apply (*simp add*: *f1 SEXP-def*)

by (*simp add*: *prfun-inverse*)
qed

lemma *prfun-condition-pre*: (*rvfun-of-prfun r*)$^{\Uparrow}$ (*a*, *b*) = *ureal2real* (*r a*)
by (*simp add*: *rvfun-of-prfun-def*)

theorem *prfun-pchoice-assoc*:
fixes w_1 :: *'a* ⇒ *ureal*
assumes ∀ *s*. ((*1* − *ureal2real* (w_1 *s*)) ∗ (*1* − *ureal2real* (w_2 *s*))) = (*1* − *ureal2real* (r_2 *s*))
assumes ∀ *s*. (*ureal2real* (w_1 *s*)) = (*ureal2real* (r_1 *s*) ∗ *ureal2real* (r_2 *s*))
shows $P \oplus_{w_1{}^{\Uparrow}} (Q \oplus_{(w_2{}^{\Uparrow})} R) = (P \oplus_{r_1{}^{\Uparrow}} Q) \oplus_{r_2{}^{\Uparrow}} R$ (**is** *?lhs* = *?rhs*)
proof −
have *f0*: ∀ *s*. ((*1* − *ureal2real* (w_1 *s*)) ∗ (*1* − *ureal2real* (w_2 *s*))) =
(*1* − *ureal2real* (w_1 *s*) − *ureal2real* (w_2 *s*) + *ureal2real* (w_1 *s*) ∗ *ureal2real* (w_2 *s*))
by (*metis diff-add-eq diff-diff-eq2 left-diff-distrib mult.commute mult-1*)
then have *f1*: ∀ *s*. (*1* − *ureal2real* (w_1 *s*) − *ureal2real* (w_2 *s*) + *ureal2real* (w_1 *s*) ∗ *ureal2real* (w_2 *s*))
= ((*1* − *ureal2real* (r_2 *s*)))
using *assms*(*1*) **by** *presburger*
then have *f2*: ∀ *s*. (*ureal2real* (r_2 *s*)) = (*ureal2real* (w_1 *s*) + *ureal2real* (w_2 *s*) − *ureal2real* (w_1 *s*) ∗ *ureal2real* (w_2 *s*))
by (*smt* (*verit*, *del-insts*) *SEXP-apply*)
have *f3*: ∀ *s*. (*ureal2real* (w_1 *s*)) = (*ureal2real* (r_1 *s*) ∗ (*ureal2real* (w_1 *s*) + *ureal2real* (w_2 *s*) − *ureal2real* (w_1 *s*) ∗ *ureal2real* (w_2 *s*)))
using *assms*(*2*) *f2* **by** (*simp*)
have *P-eq*: ∀ *a b*. ((*rvfun-of-prfun* w_1)$^{\Uparrow}$ (*a*, *b*) ∗ (*rvfun-of-prfun P*) (*a*, *b*) =
((*rvfun-of-prfun* r_2)$^{\Uparrow}$ (*a*, *b*) ∗ ((*rvfun-of-prfun* r_1)$^{\Uparrow}$ (*a*, *b*) ∗ (*rvfun-of-prfun P*) (*a*, *b*))))
apply (*auto*)
by (*simp add*: *assms*(*2*) *rvfun-of-prfun-def*)
have *Q-eq*: ∀ *a b*. ((((*1*::ℝ) − (*rvfun-of-prfun* w_1)$^{\Uparrow}$ (*a*, *b*)) ∗ ((*rvfun-of-prfun* w_2)$^{\Uparrow}$ (*a*, *b*) ∗ (*rvfun-of-prfun Q*) (*a*, *b*)))
= ((*rvfun-of-prfun* r_2)$^{\Uparrow}$ (*a*, *b*) ∗ (((*1*::ℝ) − (*rvfun-of-prfun* r_1)$^{\Uparrow}$ (*a*, *b*)) ∗ (*rvfun-of-prfun Q*) (*a*, *b*))))
apply (*simp add*: *prfun-condition-pre*)
apply (*rule allI*)
apply (*rule disjI2*)
proof −
fix *a*
have *rvfun-of-prfun* r_2 *a* ∗ ((*1*::ℝ) − *rvfun-of-prfun* r_1 *a*) = *rvfun-of-prfun* r_2 *a* − *rvfun-of-prfun* r_2 *a* ∗ *rvfun-of-prfun* r_1 *a*
by (*simp add*: *right-diff-distrib*)
also have ... = *rvfun-of-prfun* r_2 *a* − *rvfun-of-prfun* w_1 *a*
by (*simp add*: *assms*(*2*) *rvfun-of-prfun-def*)
also have ... = *rvfun-of-prfun* w_2 *a* − *rvfun-of-prfun* w_1 *a* ∗ *rvfun-of-prfun* w_2 *a*
using *f2* **by** (*simp add*: *rvfun-of-prfun-def*)
then show ((*1*::ℝ) − *rvfun-of-prfun* w_1 *a*) ∗ *rvfun-of-prfun* w_2 *a* = *rvfun-of-prfun* r_2 *a* ∗ ((*1*::ℝ) − *rvfun-of-prfun* r_1 *a*)
by (*simp add*: *calculation left-diff-distrib*)
qed
have *R-eq*: ∀ *a b*. ((((*1*::ℝ) − (*rvfun-of-prfun* w_1)$^{\Uparrow}$ (*a*, *b*)) ∗ (((*1*::ℝ) − (*rvfun-of-prfun* w_2)$^{\Uparrow}$ (*a*, *b*)) ∗ (*rvfun-of-prfun R*) (*a*, *b*)))
= (((*1*::ℝ) − (*rvfun-of-prfun* r_2)$^{\Uparrow}$ (*a*, *b*)) ∗ (*rvfun-of-prfun R*) (*a*, *b*)))
apply (*simp add*: *prfun-condition-pre*)
apply (*rule allI*)
apply (*rule disjI2*)
by (*simp add*: *assms*(*1*) *rvfun-of-prfun-def*)

show *?thesis*
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *dist-defs expr-defs*)
apply (*subst rvfun-inverse*)
apply (*smt* (*verit*, *del-insts*) *SEXP-apply is-prob-def mult-nonneg-nonneg mult-right-le-one-le prfun-in-0-1′ taut-def*)
apply (*subst rvfun-inverse*)
apply (*smt* (*verit*, *del-insts*) *SEXP-apply is-prob-def mult-nonneg-nonneg mult-right-le-one-le prfun-in-0-1′ taut-def*)
apply (*subst fun-eq-iff*)
apply (*auto*)
apply (*subst distrib-left*)+
using *P-eq Q-eq R-eq* **by** (*smt* (*verit*, *ccfv-SIG*) *SEXP-def prod.simps*(*2*) *rvfun-of-prfun-def*)
qed

theorem *prfun-pchoice-assigns*:
$(if_p\ r\ then\ x := e\ else\ y := f) =$
$prfun\text{-}of\text{-}rvfun\ (\bullet(rvfun\text{-}of\text{-}prfun\ r) * [\![x := e]\!]_{\mathcal{I}e} + (1 - \bullet(rvfun\text{-}of\text{-}prfun\ r)) * [\![y := f]\!]_{\mathcal{I}e})_e$
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *rvfun-assignment-inverse*)
by (*expr-auto*)

thm *rvfun-pchoice-inverse*
lemma *prfun-pchoice-assigns-inverse*:
shows $rvfun\text{-}of\text{-}prfun\ ((x := e) \oplus_{r^{\Uparrow}} (y := f))$
$= (pchoice\text{-}f\ ([\![x := e]\!]_{\mathcal{I}})\ ((rvfun\text{-}of\text{-}prfun\ r)^{\Uparrow})_e\ ([\![y := f]\!]_{\mathcal{I}}))$
apply (*simp only*: *passigns-def pchoice-def*)
apply (*simp add*: *rvfun-assignment-inverse*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *is-prob-def prfun-in-0-1′*)
apply (*subst fun-eq-iff*)
apply (*auto*)
by (*simp add*: *rvfun-of-prfun-def*)+

lemma *prfun-pchoice-assigns-inverse-c*:
shows $rvfun\text{-}of\text{-}prfun\ ((x := e) \oplus_{(\lambda s.\ r)} (y := f))$
$= (pchoice\text{-}f\ ([\![x := e]\!]_{\mathcal{I}e})\ (ureal2real\ «r»)_e\ ([\![y := f]\!]_{\mathcal{I}e}))$
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *rvfun-assignment-inverse*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *is-prob-def prfun-in-0-1′*)
apply (*subst fun-eq-iff*)
apply (*auto*)
apply (*simp add*: *rvfun-of-prfun-def*)
by (*simp add*: *rvfun-of-prfun-def*)

lemma *prfun-pchoice-assigns-inverse-c′*:
shows $rvfun\text{-}of\text{-}prfun\ ((x := e) \oplus_{[(\lambda s.\ r)]_e} (y := f))$
$= (pchoice\text{-}f\ ([\![x := e]\!]_{\mathcal{I}e})\ (ureal2real\ «r»)_e\ ([\![y := f]\!]_{\mathcal{I}e}))$
using *prfun-pchoice-assigns-inverse-c SEXP-def* **by** *metis*

5.6.5 Conditional choice

lemma *rvfun-pcond-is-prob*:
  assumes *is-prob P is-prob Q*
  shows *is-prob* $(P \triangleleft_f\ b \triangleright Q)$
  by (*smt* (*verit*, *best*) *SEXP-def assms*(*1*) *assms*(*2*) *is-prob-def taut-def*)

lemma *rvfun-pcond-altdef*: $(P \triangleleft_f\ b \triangleright Q) = ([\![b]\!]_{\mathcal{I}} * P + [\![\neg b]\!]_{\mathcal{I}e} * Q)_e$
  by (*expr-auto*)

lemma *rvfun-pcond-is-dist*:
  assumes *is-final-distribution P is-final-distribution Q*
  shows *is-final-distribution* $(P \triangleleft_f (b^{\Uparrow}) \triangleright Q)$
  apply (*simp add*: *dist-defs expr-defs*, *auto*)
  apply (*simp add*: *assms is-final-distribution-prob is-final-prob-altdef*)+
  by (*smt* (*verit*, *best*) *assms*(*1*) *assms*(*2*) *curry-conv infsum-cong is-dist-def is-sum-1-def*)

lemma *rvfun-pcond-is-dist'*:
  assumes *is-final-distribution P is-final-distribution Q*
    $\forall s\ s_1\ s_2.\ b\ (s, s_1) = b\ (s, s_2)$
  shows *is-final-distribution* $(P \triangleleft_f (b) \triangleright Q)$
  apply (*simp add*: *dist-defs expr-defs*, *auto*)
  apply (*simp add*: *assms is-final-distribution-prob is-final-prob-altdef*)+
proof −
  fix s_1
  show $(\sum_\infty s::'b.\ \mathit{if}\ b\ (s_1, s)\ \mathit{then}\ P\ (s_1, s)\ \mathit{else}\ Q\ (s_1, s)) = (1::\mathbb{R})$
  proof (*cases* $\forall s.\ b\ (s_1, s)$)
    case *True*
    then show *?thesis*
      by (*smt* (*verit*, *best*) *assms*(*1*) *curry-conv infsum-cong is-dist-def is-sum-1-def*)
  next
    case *False*
    then have $\forall s.\ b\ (s_1, s) = \mathit{False}$
      using *assms*(*3*) **by** *blast*
    then show *?thesis*
      by (*smt* (*verit*, *best*) *assms*(*2*) *curry-conv infsum-cong is-dist-def is-sum-1-def*)
  qed
qed

lemma *rvfun-pcond-inverse*:
  assumes *is-prob P is-prob Q*
  shows *rvfun-of-prfun* (*prfun-of-rvfun* $(P \triangleleft_f\ b \triangleright Q)) = (P \triangleleft_f\ b \triangleright Q)$
  by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-inverse rvfun-pcond-is-prob*)

lemma *prfun-pcond-altdef*:
  shows $\mathit{if}_c\ b$ *then P else Q* = *prfun-of-rvfun* $([\![b]\!]_{\mathcal{I}} * \bullet($*rvfun-of-prfun P*$) + [\![\neg b]\!]_{\mathcal{I}e} * \bullet($*rvfun-of-prfun*
$Q))_e$
  apply (*simp add*: *pfun-defs*)
  apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
  by (*expr-auto*)

lemma *prfun-pcond-id*:
  shows ($\mathit{if}_c\ b$ *then P else P*) = *P*
  apply (*simp add*: *pfun-defs*)
  apply (*expr-auto*)
  by (*simp add*: *prfun-inverse*)

lemma *prfun-pcond-pchoice-eq*:
shows $if_c\ b\ then\ P\ else\ Q = (if_p\ [\![b]\!]_{\mathcal{I}}\ then\ P\ else\ Q)$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*simp add*: *rvfun-pcond-altdef*)
proof −
have *f0*: $rvfun\text{-}of\text{-}prfun\ (\lambda x::'a \times 'b.\ ereal2ureal\ (ereal\ (([\![b]\!]_{\mathcal{I}})\ x))) = [\![b]\!]_{\mathcal{I}}$
apply (*simp add*: *ureal-defs*)
apply (*simp add*: *expr-defs*)
by (*simp add*: *ereal2ureal′-inverse*)
show $[\lambda s::'a \times 'b.\ ([\![b]\!]_{\mathcal{I}})\ s * rvfun\text{-}of\text{-}prfun\ P\ s + ([\![[\lambda s::'a \times 'b.\ \neg\ b\ s]_e]\!]_{\mathcal{I}})\ s * rvfun\text{-}of\text{-}prfun\ Q\ s]_e = rvfun\text{-}of\text{-}prfun\ P \oplus_{f\ rvfun\text{-}of\text{-}prfun\ (\lambda x::'a \times 'b.\ ereal2ureal\ (ereal\ (([\![b]\!]_{\mathcal{I}})\ x)))}\ rvfun\text{-}of\text{-}prfun\ Q$
apply (*simp add*: *f0*)
apply (*subst fun-eq-iff*)
apply (*auto*)
by (*metis SEXP-def iverson-bracket-not*)
qed

lemma *prfun-pcond-mono*: $[\![\ P_1 \le P_2;\ \ Q_1 \le Q_2\ ]\!] \Longrightarrow (if_c\ b\ then\ P_1\ else\ Q_1) \le (if_c\ b\ then\ P_2\ else\ Q_2)$
apply (*simp add*: *pcond-def ureal-defs*)
apply (*simp add*: *le-fun-def*)
apply (*auto*)
apply (*simp add*: *ureal-defs*)
apply (*smt* (*z3*) *atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-less-eq*(*4*) *ereal-real ereal-times*(*1*) *max.absorb1 max.absorb2 min.absorb1 real-of-ereal-le-0 ureal2ereal ureal2ereal-inverse*)
apply (*simp add*: *ureal-defs*)
by (*smt* (*z3*) *atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-less-eq*(*4*) *ereal-real ereal-times*(*1*) *max.absorb1 max.absorb2 min.absorb1 real-of-ereal-le-0 ureal2ereal ureal2ereal-inverse*)

5.6.6 Sequential composition

lemma *rvfun-seqcomp-dist-is-prob*:
assumes *is-final-distribution p is-prob q*
shows *is-prob* (*pseqcomp-f p q*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *infsum-nonneg is-prob rvfun-prob-sum1-summable*(*1*))
proof −
fix *a b*
have $(\sum_\infty v_0::'a.\ p\ (a,\ v_0) * q\ (v_0,\ b)) \le (\sum_\infty v_0::'a.\ p\ (a,\ v_0))$
apply (*subst infsum-mono*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob rvfun-product-summable-dist*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob mult-right-le-one-le rvfun-prob-sum1-summable*(*1*))
by *simp*
also have ... = *1*
by (*metis assms*(*1*) *rvfun-prob-sum1-summable*(*2*))
then show $(\sum_\infty v_0::'a.\ p\ (a,\ v_0) * q\ (v_0,\ b)) \le (1::\mathbb{R})$
using *calculation* **by** *presburger*
qed

lemma *rvfun-seqcomp-subdist-is-prob*:
assumes *is-final-sub-dist p is-prob q*

shows *is-prob* (*pseqcomp-f p q*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *infsum-nonneg is-prob rvfun-prob-sum-leq-1-summable*(*1*))
proof −
fix *a b*
have $(\sum_\infty v_0::'a.\ p\ (a,\ v_0) * q\ (v_0,\ b)) \leq (\sum_\infty v_0::'a.\ p\ (a,\ v_0))$
apply (*subst infsum-mono*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob rvfun-product-summable-subdist*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*4*))
apply (*simp add*: *assms*(*1*) *assms*(*2*) *is-prob mult-right-le-one-le rvfun-prob-sum-leq-1-summable*(*1*))
by *simp*
then show $(\sum_\infty v_0::'a.\ p\ (a,\ v_0) * q\ (v_0,\ b)) \leq (1::\mathbb{R})$
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *curry-conv dual-order.refl infsum-cong is-sub-dist-def is-sum-leq-1-def*)
qed

lemma *rvfun-seqcomp-ibracket*: $[\![p]\!]_\mathcal{I}\ ;_f\ [\![q]\!]_\mathcal{I} = (\sum_\infty\ v_0.\ [\![([\ \mathbf{v}^> \rightsquigarrow «v_0»\ ]\ \dagger\ p) \wedge ([\ \mathbf{v}^< \rightsquigarrow «v_0»\ ]\ \dagger\ q)]\!]_{\mathcal{I}e})_e$
apply (*pred-auto*)
by (*smt* (*verit*, *ccfv-threshold*) *infsum-cong mult-cancel-left1 mult-cancel-right1*)

lemma *prfun-seqcomp-ibracket*: ((*prfun-of-rvfun* ($[\![p]\!]_\mathcal{I}$)) ; (*prfun-of-rvfun* ($[\![q]\!]_\mathcal{I}$))) =
prfun-of-rvfun $(\sum_\infty\ v_0.\ [\![([\ \mathbf{v}^> \rightsquigarrow «v_0»\ ]\ \dagger\ p) \wedge ([\ \mathbf{v}^< \rightsquigarrow «v_0»\ ]\ \dagger\ q)]\!]_{\mathcal{I}e})_e$
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *rvfun-inverse-ibracket*)
by (*simp add*: *rvfun-seqcomp-ibracket*)

lemma *rvfun-seqcomp-ibracket-contra*:
assumes $(c_1::'a) \neq c_2$
shows $[\![x^> = «c_1»]\!]_{\mathcal{I}e}\ ;_f\ [\![x^< = «c_2»]\!]_{\mathcal{I}e} = 0_R$
apply (*simp add*: *rvfun-seqcomp-ibracket*)
apply (*pred-auto*)
by (*simp add*: *assms infsum-0*)

lemma *prfun-seqcomp-ibracket-contra*:
assumes $(c_1::'a) \neq c_2$
shows (*prfun-of-rvfun* ($[\![x^> = «c_1»]\!]_{\mathcal{I}e}$)) ; (*prfun-of-rvfun* ($[\![x^< = «c_2»]\!]_{\mathcal{I}e}$)) = 0_p
apply (*simp add*: *prfun-seqcomp-ibracket*)
apply (*simp add*: *pfun-defs ureal-defs*)
apply (*pred-auto*)
by (*simp add*: *assms ereal2ureal-def infsum-0 zero-ureal-def*)

lemma *rvfun-seqcomp-ibracket-onepoint*:
assumes *vwb-lens x*
shows $(([\![\$x^< = «c_0» \wedge (x := «c_1»)]\!]_{\mathcal{I}e})_e\ ;_f\ [\![\$x^< = «c_1»]\!]_{\mathcal{I}e}) = [\![\$x^< = «c_0»]\!]_{\mathcal{I}e}$

apply (*simp add*: *rvfun-seqcomp-ibracket*)
apply (*pred-auto*)
proof −
fix *a*
have $get_x\ (put_x\ a\ c_1) = c_1$
by (*meson assms mwb-lens-weak vwb-lens-iff-mwb-UNIV-src weak-lens.put-get*)
then have *f1*: $\forall\, v_0.\ (v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1) = (\ v_0 = put_x\ a\ c_1)$
by (*auto*)

have *f2*: $(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R})) =$
$(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R}))$
using *f1* **by** *force*
also have ... = *1*
using *infsum-singleton-1* **by** *fastforce*
finally show $(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R})) = (\mathit{1}{::}\mathbb{R})$
by *presburger*
qed

lemma *rvfun-seqcomp-ibracket-onepoint'*:
assumes *vwb-lens x*
shows $((\llbracket \$x^< = «c_0» \wedge (x := «c_1»)\rrbracket_{\mathcal{I}e})_e\ ;_f\ \llbracket \$x^< = «c_1» \wedge (x := «c_2»)\rrbracket_{\mathcal{I}e}) = \llbracket \$x^< = «c_0» \wedge (x := «c_2»)\rrbracket_{\mathcal{I}e}$
proof −
have $\forall\, a.\ get_x\ (put_x\ a\ c_1) = c_1$
by (*meson assms mwb-lens-weak vwb-lens-iff-mwb-UNIV-src weak-lens.put-get*)
then have *f1*: $\forall\, a.\ \forall\, v_0.\ (v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1) = (\ v_0 = put_x\ a\ c_1)$
by (*auto*)
have *f2*: $\forall\, a.\ \forall\, v_0.\ (v_0 = put_x\ a\ c_1 \wedge put_x\ a\ c_2 = put_x\ v_0\ c_2) = (\ v_0 = put_x\ a\ c_1)$
apply (*auto*)
by (*metis assms mwb-lens.put-put vwb-lens-mwb*)

show *?thesis*
apply (*simp add*: *rvfun-seqcomp-ibracket*)
apply (*pred-auto*)
proof −
fix *a*
have *f3*: $(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1 \wedge put_x\ a\ c_2 = put_x\ v_0\ c_2\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R})) =$
$(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1 \wedge put_x\ a\ c_2 = put_x\ v_0\ c_2\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R}))$
using *f1* **by** *meson*
also have ... = *1*
apply (*simp add*: *f2*)
using *infsum-singleton-1* **by** *fastforce*
finally show $(\sum_\infty v_0{::}'b.\ \mathit{if}\ v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1 \wedge put_x\ a\ c_2 = put_x\ v_0\ c_2\ \mathit{then}$
$\mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R})) = (\mathit{1}{::}\mathbb{R})$
by *presburger*
next
fix *a b*
assume *a1*: $\neg\ b = put_x\ a\ c_2$
show $(\sum_\infty v_0{::}'b.\ \mathit{if}\ get_x\ a = c_0 \wedge v_0 = put_x\ a\ c_1 \wedge get_x\ v_0 = c_1 \wedge b = put_x\ v_0\ c_2\ \mathit{then}\ \mathit{1}{::}\mathbb{R}\ \mathit{else}\ (\mathit{0}{::}\mathbb{R})) = (\mathit{0}{::}\mathbb{R})$
by (*smt* (*verit*, *best*) *a1 f2 infsum-0*)
qed
qed

lemma *rvfun-cond-prob-abs-summable-on-product*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda(v_0{::}'a,\ s{::}'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$ *abs-summable-on*
Sigma (*UNIV*) $(\lambda v_0.\ \{s'.\ q(v_0,\ s') > (0{::}real)\})$
apply (*subst abs-summable-on-Sigma-iff*)
apply (*rule conjI*)

```
  apply (auto)
proof -
  fix x::'a

  have f1: (λxa::'a. |p (s₁, x) * q (x, xa)|) = (λxa::'a. p (s₁, x) * q (x, xa))
    apply (subst abs-of-nonneg)
    by (simp add: assms(1) assms(2) rvfun-prob-sum1-summable(1))+
  have f2: (λxa::'a. p (s₁, x) * q (x, xa)) summable-on {s'::'a. (0::ℝ) < q (x, s')}
    apply (rule summable-on-cmult-right)
    apply (rule summable-on-subset-banach[where A=UNIV])
    using assms(1) assms(2) rvfun-prob-sum1-summable(3) apply metis
    by (simp)
  show (λxa::'a. |p (s₁, x) * q (x, xa)|) summable-on {s'::'a. (0::ℝ) < q (x, s')}
    using f1 f2 by presburger
next
  have f1: (λx::'a. |∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. |p (s₁, x) * q (x, y)||) =
      (λx::'a. ∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. p (s₁, x) * q (x, y))
    apply (subst abs-of-nonneg)
    apply (subst abs-of-nonneg)
    apply (simp add: assms(1) assms(2) rvfun-prob-sum1-summable(1))+
     apply (simp add: assms(1) assms(2) infsum-nonneg rvfun-prob-sum1-summable(1))
    apply (subst abs-of-nonneg)
    by (simp add: assms(1) assms(2) rvfun-prob-sum1-summable(1))+
  then have f2: ... = (λx::'a. p (s₁, x) * (∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. q (x, y)))
    using infsum-cmult-right' by fastforce
  have f3: ... = (λx::'a. p (s₁, x))
    apply (rule ext)
    proof -
      fix x
      have f31: (∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. q (x, y)) =
        (∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')} ∪ {s'::'a. (0::ℝ) = q (x, s')}. q (x, y))
        apply (rule infsum-cong-neutral)
        by force+
      then have f32: ... = (∑∞y::'a. q (x, y))
        by (smt (verit, del-insts) assms(2) infsum-cong infsum-mult-subset-right mult-cancel-left1
              rvfun-prob-sum1-summable(1))
      then have f33: ... = 1
        by (simp add: assms(2) rvfun-prob-sum1-summable(2))
      then show p (s₁, x) * (∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. q (x, y)) = p (s₁, x)
        using f31 f32 by auto
    qed
  have f4: infsum (λx::'a. ∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. p (s₁, x) * q (x, y)) UNIV =
      infsum (λx::'a. p (s₁, x)) UNIV
    using f2 f3 by presburger
  then have f5: ... = 1
    by (simp add: assms(1) rvfun-prob-sum1-summable(2))

  have f6: (λx::'a. ∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. p (s₁, x) * q (x, y)) summable-on UNIV
    using f4 f5 infsum-not-exists by fastforce

  show (λx::'a. |∑∞y::'a∈{s'::'a. (0::ℝ) < q (x, s')}. |p (s₁, x) * q (x, y)||) summable-on UNIV
    using f1 f6 by presburger
qed

lemma rvfun-cond-prob-product-summable-on-sigma-possible-sets:
```

assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda(v_0::'a,\ s::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$ *summable-on*
Sigma (*UNIV*) $(\lambda v_0.\ \{s'.\ q(v_0,\ s') > (0::real)\})$
apply (*subst summable-on-iff-abs-summable-on-real*)
using *rvfun-cond-prob-abs-summable-on-product assms*(*1*) *assms*(*2*) **by** *fastforce*

lemma *rvfun-cond-prob-product-summable-on-sigma-impossible-sets*:
shows $(\lambda(v_0::'a,\ s::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$ *summable-on* (*Sigma* (*UNIV*) $(\lambda v_0.\ \{s'.\ q(v_0,\ s') = (0::real)\}))$
apply (*simp add*: *summable-on-def*)
apply (*rule-tac* $x = 0$ **in** *exI*)
apply (*rule has-sum-0*)
by *force*

lemma *rvfun-cond-prob-product-summable-on-UNIV*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda(v_0::'a,\ s::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$ *summable-on Sigma* (*UNIV*) $(\lambda v_0.\ UNIV)$
proof –
let *?A1* = *Sigma* (*UNIV*) $(\lambda v_0.\ \{s'.\ q(v_0,\ s') > (0::real)\})$
let *?A2* = *Sigma* (*UNIV*) $(\lambda v_0.\ \{s'.\ q(v_0,\ s') = (0::real)\})$
let *?f* = $(\lambda(v_0::'a,\ s::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$

have *?f summable-on* (*?A1* ∪ *?A2*)
apply (*rule summable-on-Un-disjoint*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-cond-prob-product-summable-on-sigma-possible-sets*)
apply (*simp add*: *rvfun-cond-prob-product-summable-on-sigma-impossible-sets*)
by *fastforce*
then show *?thesis*
by (*simp add*: *assms*(*2*) *prel-Sigma-UNIV-divide*)
qed

lemma *rvfun-cond-prob-product-summable-on-UNIV-2*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda(s::'a,\ v_0::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$ *summable-on UNIV* × *UNIV*
apply (*subst product-swap*[*symmetric*])
apply (*subst summable-on-reindex*)
apply *simp*
proof –
have *f0*: $(\lambda(s::'a,\ v_0::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \circ$ *prod.swap* = $(\lambda(v_0::'a,\ s::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$
by (*simp add*: *comp-def*)
show $(\lambda(s::'a,\ v_0::'a).\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \circ$ *prod.swap summable-on UNIV* × *UNIV*
using *assms*(*1*) *assms*(*2*) *f0 rvfun-cond-prob-product-summable-on-UNIV* **by** *fastforce*
qed

lemma *rvfun-cond-prob-infsum-pcomp-swap*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\sum_\infty s::'a.\ \sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) = (\sum_\infty v_0::'a.\ \sum_\infty s::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s))$
apply (*rule infsum-swap-banach*)
using *assms*(*1*) *assms*(*2*) *rvfun-cond-prob-product-summable-on-UNIV-2* **by** *fastforce*

lemma *rvfun-infsum-pcomp-sum-1*:

assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\sum_\infty s::'a.\ \sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) = 1$
apply (*simp add*: *assms rvfun-cond-prob-infsum-pcomp-swap*)
apply (*simp add*: *infsum-cmult-right'*)
by (*simp add*: *assms rvfun-prob-sum1-summable*)

lemma *rvfun-infsum-pcomp-summable*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $(\lambda s::'a.\ (\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)))$ *summable-on UNIV*
apply (*rule infsum-not-zero-is-summable*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-infsum-pcomp-sum-1*)

lemma *rvfun-infsum-pcomp-lessthan-1*:
assumes *is-final-distribution p*
assumes *is-final-distribution q*
shows $\forall\, s::'a.\ (\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \le 1$
proof (*rule allI*, *rule ccontr*)
fix $s::'a$
assume *a1*: $\neg\ ((\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \le 1)$
then have *f0*: $(\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) > 1$
by *simp*
have $(\sum_\infty s::'a.\ \sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) = (\sum_\infty s::'a{\in}\{s\}{\cup}(-\{s\}).\ \sum_\infty v_0::'a.\ p\ (s_1,\ v_0) *$
$q\ (v_0,\ s))$
by *force*
also have $... = (\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) + (\sum_\infty s::'a{\in}(-\{s\}).\ \sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,$
$s))$
apply (*subst infsum-Un-disjoint*)
apply *simp*
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
by (*simp-all add*: *rvfun-infsum-pcomp-summable assms*(*1*) *assms*(*2*))
also have $... > 1$
by (*smt* (*verit*, *del-insts*) *assms*(*1*) *assms*(*2*) *f0 infsum-nonneg mult-nonneg-nonneg rvfun-prob-sum1-summable*(*1*))
then show *False*
using *rvfun-infsum-pcomp-sum-1 assms*(*1*) *assms*(*2*) *calculation* **by** *fastforce*
qed

lemma *rvfun-infsum-pcomp-lessthan-1-subdist*:
assumes *is-final-sub-dist p*
assumes *is-final-sub-dist q*
shows $\forall\, s::'a.\ (\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \le 1$
proof
fix *s*
have *f0*: $\forall\, v_0.\ p\ (s_1,\ v_0) * q\ (v_0,\ s) \le p\ (s_1,\ v_0)$
by (*simp add*: *assms mult-left-le rvfun-prob-sum-leq-1-summable*(*1*))
then have $(\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \le (\sum_\infty v_0::'a.\ p\ (s_1,\ v_0))$
apply (*subst infsum-mono*)
apply (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *is-final-prob-prob is-final-sub-dist-prob rvfun-product-summable-subd*
summable-on-cong)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*4*))
apply *blast*
by *simp*
then show $(\sum_\infty v_0::'a.\ p\ (s_1,\ v_0) * q\ (v_0,\ s)) \le (1::\mathbb{R})$
by (*simp add*: *assms*(*1*) *assms*(*2*) *is-final-prob-prob is-final-sub-dist-prob rvfun-product-prob-sub-dist-leq-1*)

qed

lemma *rvfun-seqcomp-is-dist*:
  assumes *is-final-distribution p*
  assumes *is-final-distribution q*
  shows *is-final-distribution* (*pseqcomp-f p q*)
  apply (*simp add*: *dist-defs expr-defs*, *auto*)
  apply (*simp add*: *assms*(*1*) *assms*(*2*) *infsum-nonneg rvfun-prob-sum1-summable*(*1*))
  defer
  apply (*simp-all add*: *lens-defs*)
  apply (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-infsum-pcomp-sum-1*)
proof (*rule ccontr*)
  fix s_1::'*a* **and** *s*::'*a*
  let *?f* = $\lambda s.$ ($\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *s*))
  assume *a1*: ¬ ($\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *s*)) ≤ (*1*::ℝ)
  then have *f0*: ($\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *s*)) > *1*
    by *force*
  have *f1*: (λs::'*a*. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *s*)) *summable-on UNIV*
    apply (*rule infsum-not-zero-summable*[**where** *x* = *1*])
    by (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-infsum-pcomp-sum-1*)+
  have *f2*: ($\sum_\infty$*ss*::'*a*. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*)) =
    ($\sum_\infty$*ss*::'*a*∈{*s*} ∪ {*ss*. *ss* ≠ *s*}. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*))
    by (*metis* (*mono-tags*, *lifting*) *CollectI DiffD2 UNIV-I UnCI infsum-cong-neutral insert-iff*)
  also have *f3*: ... = ($\sum_\infty$*ss*::'*a*∈{*s*}. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*)) +
    ($\sum_\infty$*ss*::'*a*∈{*ss*. *ss* ≠ *s*}. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*))
    apply (*rule infsum-Un-disjoint*)
    apply *simp*
    using *f1 summable-on-subset-banach* **apply** *blast*
    by *simp*
  also have *f4*: ... = ($\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *s*)) +
    ($\sum_\infty$*ss*::'*a*∈{*ss*. *ss* ≠ *s*}. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*))
    by *simp*
  also have *f5*: ... > *1*
    by (*smt* (*verit*, *del-insts*) *a1 assms*(*1*) *assms*(*2*) *infsum-nonneg mult-nonneg-nonneg*
        rvfun-prob-sum1-summable(*1*))
  have *f6*: ($\sum_\infty$*ss*::'*a*. $\sum_\infty v_0$::'*a*. *p* (s_1, v_0) ∗ *q* (v_0, *ss*)) > *1*
    using *calculation f5* **by** *presburger*
  show *False*
    using *rvfun-infsum-pcomp-sum-1 f6 assms*(*1*) *assms*(*2*) **by** *fastforce*
qed

lemma *rvfun-seqcomp-inverse*:
  assumes *is-final-distribution p*
  assumes *is-prob q*
  shows *rvfun-of-prfun* (*prfun-of-rvfun* (*pseqcomp-f p q*)) = *pseqcomp-f p q*
  apply (*subst rvfun-inverse*)
  apply (*simp add*: *assms rvfun-seqcomp-dist-is-prob*)
  using *assms*(*1*) *assms*(*2*) *rvfun-seqcomp-is-dist* **by** *blast*

lemma *rvfun-seqcomp-inverse-subdist*:
  assumes *is-final-sub-dist p*
  assumes *is-prob q*
  shows *rvfun-of-prfun* (*prfun-of-rvfun* (*pseqcomp-f p q*)) = *pseqcomp-f p q*
  apply (*subst rvfun-inverse*)
  apply (*simp add*: *assms rvfun-seqcomp-subdist-is-prob*)

using *assms*(*1*) *assms*(*2*) *rvfun-seqcomp-is-dist* **by** *blast*

lemma *prfun-zero-right*: P ; $\mathbf{0} = \mathbf{0}$
apply (*simp add*: *pfun-defs ureal-zero*)
apply (*simp add*: *ureal-defs*)
by (*simp add*: *SEXP-def ereal2ureal-def zero-ureal-def subst-app-def*)

lemma *prfun-zero-right'*: P ; $\mathit{0}_p = \mathit{0}_p$
by (*simp add*: *prfun-zero-right pzero-def*)

lemma *prfun-zero-left*: $\mathbf{0}$; $P = \mathbf{0}$
apply (*simp add*: *pfun-defs ureal-zero*)
apply (*simp add*: *ureal-defs*)
by (*simp add*: *SEXP-def ereal2ureal-def subst-app-def zero-ureal-def*)

lemma *prfun-zero-left'*: $\mathit{0}_p$; $P = \mathit{0}_p$
by (*simp add*: *prfun-zero-left pzero-def*)

lemma *prfun-pseqcomp-mono*:
fixes P_1 :: *'s prhfun*
assumes $\forall a\ b.\ (\lambda v_0$::*'s. real-of-ereal*
(*ureal2ereal* (P_1 (a, v_0))) ∗ *real-of-ereal* (*ureal2ereal* (Q_1 (v_0, b)))) *summable-on UNIV*
assumes $\forall a\ b.\ (\lambda v_0$::*'s. real-of-ereal*
(*ureal2ereal* (P_2 (a, v_0))) ∗ *real-of-ereal* (*ureal2ereal* (Q_2 (v_0, b)))) *summable-on UNIV*
shows $[\![\ P_1 \le P_2;\ Q_1 \le Q_2\ ]\!] \Longrightarrow (P_1\ ;\ Q_1) \le (P_2\ ;\ Q_2)$
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *le-fun-def*)
apply (*simp add*: *ureal-defs*)
apply (*expr-auto*)
proof −
fix a b :: *'s*
assume *a1*: ∀ (*a*::*'s*) *b*::*'s*. P_1 (a, b) ≤ P_2 (a, b)
assume *a2*: ∀ (*a*::*'s*) *b*::*'s*. Q_1 (a, b) ≤ Q_2 (a, b)
let *?lhs* = ($\sum_\infty v_0$::*'s*.
real-of-ereal (*ureal2ereal* (P_1 (a, v_0))) ∗ *real-of-ereal* (*ureal2ereal* (Q_1 (v_0, b))))
let *?rhs* = ($\sum_\infty v_0$::*'s*.
real-of-ereal (*ureal2ereal* (P_2 (a, v_0))) ∗ *real-of-ereal* (*ureal2ereal* (Q_2 (v_0, b))))

have *?lhs* ≤ *?rhs*
apply (*rule infsum-mono*)
apply (*simp add*: *assms*(*1*))
apply (*simp add*: *assms*(*2*))
by (*metis a1 a2 atLeastAtMost-iff ereal-less-PInfty ereal-times*(*1*) *less-eq-ureal.rep-eq linorder-not-less mult-mono real-of-ereal-pos real-of-ereal-positive-mono ureal2ereal*)
then show *ereal2ureal'* (*min* (*max* (*0*::*ereal*) (*ereal ?lhs*)) (*1*::*ereal*)) ≤
ereal2ureal' (*min* (*max* (*0*::*ereal*) (*ereal ?rhs*)) (*1*::*ereal*))
by (*smt* (*z3*) *atLeastAtMost-iff ereal2ureal'-inverse ereal-less-eq*(*3*) *ereal-less-eq*(*4*) *ereal-less-eq*(*7*) *ereal-max-0 less-eq-ureal.rep-eq linorder-le-cases max.absorb-iff2 min.absorb1 min.absorb2*)
qed

lemma *prfun-pseqcomp-mono'*:
fixes P_1 :: *'s prhfun*

assumes $\forall\, a\ b.\ (\lambda v_0{::}'s.\ \textit{ureal2real}\ (P_1\ (a,\ v_0)) * \textit{ureal2real}\ (Q_1\ (v_0,\ b)))$ *summable-on UNIV*
assumes $\forall\, a\ b.\ (\lambda v_0{::}'s.\ \textit{ureal2real}\ (P_2\ (a,\ v_0)) * \textit{ureal2real}\ (Q_2\ (v_0,\ b)))$ *summable-on UNIV*
shows $[\![\ P_1 \le P_2;\ \ Q_1 \le Q_2\ ]\!] \Longrightarrow (P_1\ ;\ \ Q_1) \le (P_2\ ;\ \ Q_2)$
apply (*subst prfun-pseqcomp-mono*)
using *assms*(*1*) *ureal2real-def* **apply** *auto*[*1*]
using *assms*(*2*) *ureal2real-def* **apply** *auto*[*1*]
by *simp+*

theorem *prfun-seqcomp-left-unit*: $II\ ;\ \ (P{::}'a\ \textit{prhfun}) = P$
apply (*simp add*: *pseqcomp-def pskip-def*)
apply (*simp add*: *rvfun-skip-inverse*)
apply (*expr-auto add*: *skip-def*)
apply (*simp add*: *infsum-mult-singleton-left*)
by (*simp add*: *prfun-inverse*)

theorem *prfun-seqcomp-right-unit*: $(P{::}'a\ \textit{prhfun})\ ;\ \ II = P$
apply (*simp add*: *pseqcomp-def pskip-def*)
apply (*simp add*: *rvfun-skip-inverse*)
apply (*expr-auto add*: *skip-def*)
apply (*simp add*: *infsum-mult-singleton-right-1*)
by (*simp add*: *prfun-inverse*)

theorem *prfun-seqcomp-one*:
assumes *is-final-distribution* (*rvfun-of-prfun* ($P{::}'a$ *prhfun*))
shows $(P{::}'a\ \textit{prhfun})\ ;\ \ \mathit{1}_p = \mathit{1}_p$
apply (*simp add*: *pseqcomp-def pskip-def*)
apply (*simp add*: *ureal-defs pfun-defs*)
apply (*pred-auto*)
apply (*simp add*: *one-ureal.rep-eq*)
apply (*subst rvfun-prob-sum1-summable*(*2*))
apply (*smt* (*verit*, *best*) *SEXP-def assms case-prod-curry cond-case-prod-eta curry-conv o-apply rvfun-of-prfun-def ureal2real-def*)
using *ereal2ureal-def one-ereal-def one-ureal.abs-eq* **by** *presburger*

lemma *prfun-passign-simp*: $(x := e) = \textit{prfun-of-rvfun}\ ([\![\ x := e\ ]\!]_{\mathcal{I}})$
by (*simp add*: *pfun-defs expr-defs*)

theorem *prfun-passign-comp*:

shows $(x := e)\ ;\ \ (y := f) = \textit{prfun-of-rvfun}\ ([\![\ (x := e)\ ;;\ \ (y := f)\ ]\!]_{\mathcal{I}})$
apply (*simp add*: *pseqcomp-def passigns-def*)
apply (*simp add*: *rvfun-assignment-inverse*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*pred-auto*)
apply (*subst infsum-mult-singleton-left*)
apply *simp*
by (*smt* (*verit*, *best*) *infsum-0 mult-cancel-left1 mult-cancel-right1*)

lemma *prfun-prob-choice-is-sum-1*:
assumes $\mathit{0} \le r \wedge r \le \mathit{1}$
assumes *is-final-distribution* (*rvfun-of-prfun* ($P{::}'a$ *prhfun*))
assumes *is-final-distribution* (*rvfun-of-prfun Q*)
shows $(\sum_\infty s{::}'a.\ r * \textit{rvfun-of-prfun}\ P\ (s_1,\ s) + ((\mathit{1}{::}\mathbb{R}) - r\ ) * \textit{rvfun-of-prfun}\ Q\ (s_1,\ s)) = (\mathit{1}{::}\mathbb{R})$
proof –
have *f1*: $(\sum_\infty s{::}'a.\ r\ * \textit{rvfun-of-prfun}\ P\ (s_1,\ s) + ((\mathit{1}{::}\mathbb{R}) - r\ ) * \textit{rvfun-of-prfun}\ Q\ (s_1,\ s)) =$

$(\sum_\infty s::'a.\ r * \mathit{rvfun\text{-}of\text{-}prfun}\ P\ (s_1,\ s)) + (\sum_\infty s::'a.\ ((1::\mathbb{R}) - r\ ) * \mathit{rvfun\text{-}of\text{-}prfun}\ Q\ (s_1,\ s))$
    apply (*rule infsum-add*)
    apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    by (*simp add*: *assms*(*3*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
  also have *f2*: ... $= r * (\sum_\infty s::'a.\ \mathit{rvfun\text{-}of\text{-}prfun}\ P\ (s_1,\ s)) +$
      $(1 - r) * (\sum_\infty s::'a.\ \mathit{rvfun\text{-}of\text{-}prfun}\ Q\ (s_1,\ s))$
    apply (*subst infsum-cmult-right*)
    apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    apply (*subst infsum-cmult-right*)
    apply (*simp add*: *assms*(*3*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    by *simp*
  show *?thesis*
    apply (*simp add*: *f1 f2*)
    by (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))
qed

lemma *prfun-prob-choice-is-sum-1'*:
  assumes $0 \le r \wedge r \le 1$
  assumes *is-final-distribution* (p)
  assumes *is-final-distribution* (q)
  shows $(\sum_\infty s::'a.\ r * p\ (s_1,\ s) + ((1::\mathbb{R}) - r\ ) * q\ (s_1,\ s)) = (1::\mathbb{R})$
proof −
  have *f1*: $(\sum_\infty s::'a.\ r\ * p\ (s_1,\ s) + ((1::\mathbb{R}) - r\ ) * q\ (s_1,\ s)) =$
    $(\sum_\infty s::'a.\ r * p\ (s_1,\ s)) + (\sum_\infty s::'a.\ ((1::\mathbb{R}) - r\ ) * q\ (s_1,\ s))$
    apply (*rule infsum-add*)
    apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    by (*simp add*: *assms*(*3*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
  also have *f2*: ... $= r * (\sum_\infty s::'a.\ p\ (s_1,\ s)) +$
      $(1 - r) * (\sum_\infty s::'a.\ q\ (s_1,\ s))$
    apply (*subst infsum-cmult-right*)
    apply (*simp add*: *assms*(*2*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    apply (*subst infsum-cmult-right*)
    apply (*simp add*: *assms*(*3*) *rvfun-prob-sum1-summable*(*3*) *summable-on-cmult-right*)
    by *simp*
  show *?thesis*
    apply (*simp add*: *f1 f2*)
    by (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))
qed

theorem *prfun-seqcomp-left-one-point*: $x := e\ ;\ \ P = \mathit{prfun\text{-}of\text{-}rvfun}\ (([\ x^< \rightsquigarrow e^<\ ]\ \dagger \bullet(\mathit{rvfun\text{-}of\text{-}prfun}\ P)))_e$
  apply (*simp add*: *pfun-defs expr-defs*)
  apply (*subst rvfun-inverse*)
  apply (*simp add*: *dist-defs expr-defs*)
  apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
  apply (*pred-auto*)
  by (*simp add*: *infsum-mult-singleton-left*)

lemma *prfun-infsum-over-pair-subset-1*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (P::'a *prhfun*))
  shows $(\sum_\infty\ (s::'a,\ v_0::'a).\ \mathit{rvfun\text{-}of\text{-}prfun}\ P\ (s_1,\ v_0) * (\mathit{if}\ \mathit{put}_x\ v_0\ (e\ v_0) = s\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R}))) = 1$
proof −
  have *f1*: $(\sum_\infty\ (s::'a,\ v_0::'a).\ \mathit{rvfun\text{-}of\text{-}prfun}\ P\ (s_1,\ v_0) * (\mathit{if}\ \mathit{put}_x\ v_0\ (e\ v_0) = s\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R}))) =$

$(\sum_\infty (s::'a, v_0::'a) \in \{(s::'a, v_0::'a) \mid s\ v_0.\ put_x\ v_0\ (e\ v_0) = s\}.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0))$
apply (*rule infsum-cong-neutral*)
apply *force*
using *DiffD2 prod.collapse* **apply** *fastforce*
by *force*
have *f2*: $(\sum_\infty (s::'a, v_0::'a) \in \{(s::'a, v_0::'a) \mid s\ v_0.\ put_x\ v_0\ (e\ v_0) = s\}.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0)) = 1$
apply (*subst prfun-infsum-over-pair-fst-discard*)
apply (*simp add*: *assms*)
by (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))
show *?thesis*
using *f1 f2* **by** *presburger*
qed

lemma *prfun-infsum-swap*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::'*a prhfun*))
shows $(\sum_\infty s::'a.\ \sum_\infty v_0::'a.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$ =
$(\sum_\infty v_0::'a.\ \sum_\infty s::'a.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$
apply (*rule infsum-swap-banach*)
apply (*simp add*: *summable-on-def*)
apply (*rule-tac* $x = 1$ **in** *exI*)
by (*smt* (*verit*, *best*) *assms has-sum-infsum infsum-cong infsum-not-exists prfun-infsum-over-pair-subset-1 split-cong*)

lemma *prfun-infsum-infsum-subset-1*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::'*a prhfun*))
shows $(\sum_\infty s::'a.\ \sum_\infty v_0::'a.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$ =
$(1::\mathbb{R})$
apply (*simp add*: *assms prfun-infsum-swap*)
proof −
have *f0*: $(\sum_\infty v_0::'a.\ (\sum_\infty s::'a.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))))$
$= (\sum_\infty v_0::'a.\ (rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (\sum_\infty s::'a.\ (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))))$
apply (*subst infsum-cmult-right*)
apply (*simp add*: *infsum-singleton-summable*)
by (*simp*)
then have *f1*: ... $= (\sum_\infty v_0::'a.\ (rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * 1))$
by (*simp add*: *infsum-singleton*)
then show $(\sum_\infty v_0::'a.\ \sum_\infty s::'a.\ rvfun\text{-}of\text{-}prfun\ P\ (s_1, v_0) * (if\ put_x\ v_0\ (e\ v_0) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))) = (1::\mathbb{R})$
using *f0 assms rvfun-prob-sum1-summable*(*2*) **by** *force*
qed

theorem *prfun-seqcomp-assoc*:
assumes *is-final-distribution* (*rvfun-of-prfun P*)
is-final-distribution (*rvfun-of-prfun Q*)
is-final-distribution (*rvfun-of-prfun R*)
shows (P::'*a prhfun*) ; (Q ; R) = (P ; Q) ; R
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*simp add*: *infsum-nonneg is-prob ureal-is-prob*)

```
   apply (subst rvfun-infsum-pcomp-lessthan-1)
  apply (simp add: assms)+
  apply (subst rvfun-inverse)
  using assms(1) rvfun-seqcomp-dist-is-prob ureal-is-prob apply blast
  apply (expr-auto)
proof −
  fix a and b :: 'a
  let ?q = λ(v0, b). (∑∞v0'::'a. rvfun-of-prfun Q (v0, v0') ∗ rvfun-of-prfun R (v0', b))
  let ?lhs = (∑∞v0::'a. rvfun-of-prfun P (a, v0) ∗
          (∑∞v0'::'a. rvfun-of-prfun Q (v0, v0') ∗ rvfun-of-prfun R (v0', b)))
  let ?lhs' = (∑∞v0::'a.(∑∞v0'::'a.
      rvfun-of-prfun P (a, v0) ∗ rvfun-of-prfun Q (v0, v0') ∗ rvfun-of-prfun R (v0', b)))
  let ?rhs = (∑∞v0::'a.
          (∑∞v0'::'a. rvfun-of-prfun P (a, v0') ∗ rvfun-of-prfun Q (v0', v0))
          ∗ rvfun-of-prfun R (v0, b))
  let ?rhs' = (∑∞v0::'a. (∑∞v0'::'a.
          rvfun-of-prfun P (a, v0') ∗ rvfun-of-prfun Q (v0', v0) ∗ rvfun-of-prfun R (v0, b)))

  have lhs-1: (∀ v0::'a. rvfun-of-prfun P (a, v0) ∗
          (∑∞v0'::'a. rvfun-of-prfun Q (v0, v0') ∗ rvfun-of-prfun R (v0', b))
      = (∑∞v0'::'a.
          rvfun-of-prfun P (a, v0) ∗ rvfun-of-prfun Q (v0, v0') ∗ rvfun-of-prfun R (v0', b)))
    apply (rule allI)
    by (metis (no-types, lifting) ab-semigroup-mult-class.mult-ac(1) infsum-cmult-right' infsum-cong)
  then have lhs-eq: ?lhs = ?lhs'
    by presburger

  have rhs-1: (∀ v0::'a. (∑∞v0'::'a. rvfun-of-prfun P (a, v0') ∗ rvfun-of-prfun Q (v0', v0))
          ∗ rvfun-of-prfun R (v0, b)
      = (∑∞v0'::'a.
          rvfun-of-prfun P (a, v0') ∗ rvfun-of-prfun Q (v0', v0) ∗ rvfun-of-prfun R (v0, b)))
    apply (rule allI)
    by (metis (mono-tags, lifting) infsum-cmult-left' infsum-cong)
  then have rhs-eq: ?rhs = ?rhs'
    by presburger

  have lhs-rhs-eq: ?lhs' = ?rhs'
    apply (rule infsum-swap-banach)
    apply (subst summable-on-iff-abs-summable-on-real)
    apply (subst abs-summable-on-Sigma-iff)
    apply (rule conjI)
    apply (auto)
    apply (subst abs-of-nonneg)
    apply (simp add: is-prob ureal-is-prob)
    apply (subst mult.assoc)
    apply (rule summable-on-cmult-right)
    apply (rule rvfun-product-summable')
    apply (simp add: assms)+
    apply (subst abs-of-nonneg)
    apply (subst abs-of-nonneg)
    apply (simp add: is-prob ureal-is-prob)
    apply (simp add: assms(1) assms(2) assms(3) infsum-nonneg rvfun-prob-sum1-summable(1))
    apply (subst abs-of-nonneg)
    apply (simp add: is-prob ureal-is-prob)
    apply (subst mult.assoc)
```

apply (*subst infsum-cmult-right*)
apply (*rule rvfun-product-summable'*)
apply (*simp add*: *assms*)+
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s$::'*a*. *rvfun-of-prfun P* (*a*, *s*)])
using *assms*(*1*) *summable-on-iff-abs-summable-on-real* **apply** (*metis pdrfun-prob-sum1-summable'*(*4*))
apply (*subgoal-tac* ($\sum_\infty y$::'*a*. *rvfun-of-prfun Q* (*x*, *y*) $*$ *rvfun-of-prfun R* (*y*, *b*)) $\leq$ *1*)
using *infsum-nonneg mult-right-le-one-le prfun-in-0-1'*
apply (*smt* (*verit*, *ccfv-SIG*) *mult-nonneg-nonneg real-norm-def*)
apply (*subst rvfun-infsum-pcomp-lessthan-1*)
by (*simp add*: *assms*)+

then show *?lhs* = *?rhs*
using *lhs-eq rhs-eq* **by** *presburger*
qed

theorem *prfun-seqcomp-assoc-subdist*:
assumes *is-final-sub-dist* (*rvfun-of-prfun P*)
is-final-sub-dist (*rvfun-of-prfun Q*)
is-final-sub-dist (*rvfun-of-prfun R*)
shows (*P*::'*a prhfun*) ; (*Q* ; *R*) = (*P* ; *Q*) ; *R*
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*simp add*: *infsum-nonneg is-prob ureal-is-prob*)
apply (*subst rvfun-infsum-pcomp-lessthan-1-subdist*)
apply (*simp add*: *assms*)+
apply (*subst rvfun-inverse*)
using *assms*(*1*) *rvfun-seqcomp-subdist-is-prob ureal-is-prob* **apply** *blast*
apply (*expr-auto*)
proof −
fix *a* **and** *b* :: '*a*
let *?q* = $\lambda(v_0, b)$. ($\sum_\infty v_0'$::'*a*. *rvfun-of-prfun Q* (v_0, v_0') $*$ *rvfun-of-prfun R* (v_0', *b*))
let *?lhs* = ($\sum_\infty v_0$::'*a*. *rvfun-of-prfun P* (*a*, v_0) $*$
($\sum_\infty v_0'$::'*a*. *rvfun-of-prfun Q* (v_0, v_0') $*$ *rvfun-of-prfun R* (v_0', *b*)))
let *?lhs'* = ($\sum_\infty v_0$::'*a*.($\sum_\infty v_0'$::'*a*.
rvfun-of-prfun P (*a*, v_0) $*$ *rvfun-of-prfun Q* (v_0, v_0') $*$ *rvfun-of-prfun R* (v_0', *b*)))
let *?rhs* = ($\sum_\infty v_0$::'*a*.
($\sum_\infty v_0'$::'*a*. *rvfun-of-prfun P* (*a*, v_0') $*$ *rvfun-of-prfun Q* (v_0', v_0))
$*$ *rvfun-of-prfun R* (v_0, *b*))
let *?rhs'* = ($\sum_\infty v_0$::'*a*. ($\sum_\infty v_0'$::'*a*.
rvfun-of-prfun P (*a*, v_0') $*$ *rvfun-of-prfun Q* (v_0', v_0) $*$ *rvfun-of-prfun R* (v_0, *b*)))

have *lhs-1*: ($\forall v_0$::'*a*. *rvfun-of-prfun P* (*a*, v_0) $*$
($\sum_\infty v_0'$::'*a*. *rvfun-of-prfun Q* (v_0, v_0') $*$ *rvfun-of-prfun R* (v_0', *b*))
= ($\sum_\infty v_0'$::'*a*.
rvfun-of-prfun P (*a*, v_0) $*$ *rvfun-of-prfun Q* (v_0, v_0') $*$ *rvfun-of-prfun R* (v_0', *b*)))
apply (*rule allI*)
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *infsum-cmult-right' infsum-cong*)
then have *lhs-eq*: *?lhs* = *?lhs'*
by *presburger*

have *rhs-1*: ($\forall v_0$::'*a*. ($\sum_\infty v_0'$::'*a*. *rvfun-of-prfun P* (*a*, v_0') $*$ *rvfun-of-prfun Q* (v_0', v_0))
$*$ *rvfun-of-prfun R* (v_0, *b*)

$= (\sum_{\infty} v_0{}' :: 'a.$
$\quad rvfun\text{-}of\text{-}prfun\ P\ (a,\ v_0{}') * rvfun\text{-}of\text{-}prfun\ Q\ (v_0{}',\ v_0) * rvfun\text{-}of\text{-}prfun\ R\ (v_0,\ b)))$
apply (*rule allI*)
by (*metis* (*mono-tags*, *lifting*) *infsum-cmult-left′ infsum-cong*)
then have *rhs-eq*: *?rhs* = *?rhs′*
by *presburger*

have *lhs-rhs-eq*: *?lhs′* = *?rhs′*
apply (*rule infsum-swap-banach*)
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*subst abs-summable-on-Sigma-iff*)
apply (*rule conjI*)
apply (*auto*)
apply (*subst abs-of-nonneg*)
apply (*simp add*: *is-prob ureal-is-prob*)
apply (*subst mult.assoc*)
apply (*rule summable-on-cmult-right*)
apply (*rule rvfun-product-summable-subdist*)
apply (*simp add*: *assms*)+
apply (*simp add*: *ureal-is-prob*)
apply (*subst abs-of-nonneg*)
apply (*subst abs-of-nonneg*)
apply (*simp add*: *is-prob ureal-is-prob*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *infsum-nonneg rvfun-prob-sum-leq-1-summable*(*1*))
apply (*subst abs-of-nonneg*)
apply (*simp add*: *is-prob ureal-is-prob*)
apply (*subst mult.assoc*)
apply (*subst infsum-cmult-right*)
apply (*rule rvfun-product-summable-subdist*)
apply (*simp add*: *assms*)+
apply (*simp add*: *ureal-is-prob*)
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda s::'a.$ *rvfun-of-prfun P* (*a*, *s*)])
apply (*metis assms*(*1*) *rvfun-prob-sum-leq-1-summable*(*5*) *summable-on-iff-abs-summable-on-real*)
apply (*subgoal-tac* ($\sum_{\infty} y::'a.$ *rvfun-of-prfun Q* (*x*, *y*) $*$ *rvfun-of-prfun R* (*y*, *b*)) $\leq$ *1*)
using *infsum-nonneg mult-right-le-one-le prfun-in-0-1′*
apply (*smt* (*verit*, *ccfv-SIG*) *mult-nonneg-nonneg real-norm-def*)
apply (*subst rvfun-infsum-pcomp-lessthan-1-subdist*)
by (*simp add*: *assms*)+

then show *?lhs* = *?rhs*
using *lhs-eq rhs-eq* **by** *presburger*
qed

term $((P::'a \times 'a \Rightarrow ureal)\ ;\ [\![b^{\Uparrow}]\!]_{\mathcal{I}})$

theorem *prfun-seqcomp-pcond-subdist*:
fixes *Q R* ::*′a prhfun*
assumes *is-final-sub-dist* (*rvfun-of-prfun* (*P*::*′a prhfun*))
shows $P\ ;\ (if_c\ b^{\Uparrow}\ then\ Q\ else\ R) = prfun\text{-}of\text{-}rvfun\ ($
$\quad \bullet(pseqcomp\text{-}f\ (rvfun\text{-}of\text{-}prfun\ P)\ (rvfun\text{-}of\text{-}prfun\ ([\![b^{\Uparrow}]\!]_{\mathcal{I}} * Q)_e)) +$
$\quad \bullet(pseqcomp\text{-}f\ (rvfun\text{-}of\text{-}prfun\ P)\ (rvfun\text{-}of\text{-}prfun\ ([\![\neg((b)^{\Uparrow})]\!]_{\mathcal{I}e} * R)_e)))_e$
apply (*simp add*: *pchoice-def pseqcomp-def pcond-def*)
apply (*subst rvfun-pcond-inverse*)
using *ureal-is-prob* **apply** *blast*+

apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*subst fun-eq-iff*)
apply (*pred-auto*)
proof −
fix *a ba*
let *?lhs* = ($\sum_\infty v_0$::'*a*. *rvfun-of-prfun P* (*a*, v_0) ∗ (*if b* v_0 *then rvfun-of-prfun Q* (v_0, *snd* (*a*, *ba*)) *else rvfun-of-prfun R* (v_0, *snd* (*a*, *ba*))))
let *?rhs-1* = ($\sum_\infty v_0$::'*a*.
rvfun-of-prfun P (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *Q* s) (v_0, *ba*))
let *?rhs-2* = ($\sum_\infty v_0$::'*a*.
rvfun-of-prfun P (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if* ¬ *b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *R* s) (v_0, *ba*))
have *f1*: ∀ v_0. *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *Q* s) (v_0, *ba*)
= (*if b* v_0 *then rvfun-of-prfun Q* (v_0, *ba*) *else 0*)
by (*smt* (*verit*) *SEXP-def fst-conv lambda-one lambda-zero o-def one-ereal-def one-ureal-def real-of-ereal-0 rvfun-of-prfun-def ureal2real-def zero-ereal-def zero-ureal.rep-eq zero-ureal-def*)
have *f2*: ∀ v_0. *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if* ¬ *b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *R* s) (v_0, *ba*)
= (*if b* v_0 *then 0 else rvfun-of-prfun R* (v_0, *ba*))
by (*smt* (*verit*, *best*) *SEXP-def fst-conv lambda-one lambda-zero o-def one-ereal-def one-ureal-def real-of-ereal-0 rvfun-of-prfun-def ureal2real-def zero-ereal-def zero-ureal.rep-eq zero-ureal-def*)
have *f3*: *?lhs* = ($\sum_\infty v_0$::'*a*.
(*rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *Q* s) (v_0, *ba*)) +
(*rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if* ¬ *b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *R* s) (v_0, *ba*)))
apply (*subst infsum-cong*[**where** *g* = λv_0. (*rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *Q* s) (v_0, *ba*)) +
(*rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (λs::'*a* × '*a*. *ereal2ureal* (*ereal* (*if* ¬ *b* (*fst* s) *then 1*::ℝ *else* (*0*::ℝ))) ∗ *R* s) (v_0, *ba*))])
apply (*simp add*: *f1 f2*)
by *simp*
show *?lhs* = *?rhs-1* + *?rhs-2*
apply (*simp add*: *f3*)
apply (*subst infsum-add*)
apply (*subst rvfun-product-summable-subdist*)
using *assms* **apply** *force*
using *ureal-is-prob* **apply** *blast*
apply *simp*
apply (*subst rvfun-product-summable-subdist*)
using *assms* **apply** *force*
using *ureal-is-prob* **apply** *blast*
apply *simp*
by *simp*
qed

find-theorems (*?a* + *?b*) ∗ *?c*
theorem *prfun-pcond-assign-dist*:
assumes *is-final-sub-dist* (*rvfun-of-prfun P*)
assumes *is-final-sub-dist* (*rvfun-of-prfun Q*)
shows (if_p $r^{\Uparrow}$ *then P else Q*) ; *x* := *e* = (if_p $r^{\Uparrow}$ *then* (*P* ; *x* := *e*) *else* (*Q*; *x* := *e*))
apply (*simp add*: *pseqcomp-def*)
apply (*simp add*: *pchoice-def*)

```
apply (subst rvfun-pchoice-inverse)
using ureal-is-prob apply blast+
apply (subst rvfun-seqcomp-inverse-subdist)
apply (simp add: assms(1))
using ureal-is-prob apply blast
apply (subst rvfun-seqcomp-inverse-subdist)
apply (simp add: assms(2))
using ureal-is-prob apply blast
apply (simp)
apply (rule HOL.arg-cong[where f=prfun-of-rvfun])
apply (subst fun-eq-iff)
apply (pred-auto)
apply (simp add: distrib-right)
apply (subst infsum-add)
oops
```

5.6.7 Normalisation

```
theorem rvfun-uniform-dist-empty-zero: (x 𝓤 {}) = rvfun-of-prfun 0
  apply (simp add: dist-defs ureal-defs)
  apply (expr-auto)
  by (simp add: ureal-zero-0)

lemma rvfun-uniform-dist-is-prob:
  assumes finite (A::'a set)
  assumes vwb-lens x
  shows is-prob ((x 𝓤 A))
proof (cases A = {})
  case True
  show ?thesis
    apply (simp add: True)
    apply (simp add: rvfun-uniform-dist-empty-zero)
    by (simp add: ureal-is-prob)
next
  case False
  then show ?thesis
    apply (simp add: dist-defs)
    apply (expr-auto)
    apply (simp add: infsum-nonneg)
    apply (pred-auto)
  proof −
    fix a v xa
    assume a1: v ∈ A
    assume a2: xa ∈ A
    have {va::'a. ∃ vb::'a∈A. put_x (put_x a v) va = put_x a vb} =
        {va::'a. ∃ vb::'a∈A. put_x a va = put_x a vb}
    using assms(2) by auto
    also have ... = {va::'a. ∃ vb::'a∈A. va = vb}
      by (metis assms(2) vwb-lens-wb wb-lens-weak weak-lens.view-determination)
    then have (1::ℝ) ∗ real (card {va::'a. ∃ vb::'a∈A. put_x (put_x a v) va = put_x a vb}) = real (card A)
      by (simp add: calculation)
    then have (∑∞va::'a. if ∃ vb::'a∈A. put_x (put_x a v) va = put_x a vb then 1::ℝ else (0::ℝ)) ≥ 1
      apply (subst infsum-constant-finite-states)
      apply (smt (verit, best) Collect-mem-eq Collect-mono-iff assms(1) assms(2) mem-Collect-eq
            mwb-lens-weak rev-finite-subset vwb-lens.axioms(2) weak-lens.put-get)
      by (smt (verit, best) False assms(1) card-eq-0-iff lambda-one le-square mult.right-neutral
```

mult-cancel-left1 mult-le-mono2 of-nat-1 of-nat-eq-0-iff of-nat-le-iff of-nat-mult rev-finite-subset someI-ex)
then show $(1::\mathbb{R}) / (\sum_\infty va::'a.\ if\ \exists\, vb::'a{\in}A.\ put_x\ (put_x\ a\ v)\ va = put_x\ a\ vb\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) \leq (1::\mathbb{R})$
by *force*
qed
qed

lemma *rvfun-normalisation-is-dist*:
assumes *is-nonneg p*
assumes *final-reachable p*
assumes *summable-on-final p*
shows *is-final-distribution* $(\mathbf{N}_f\ p)$
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*meson assms*(*1*) *divide-nonneg-nonneg infsum-nonneg is-nonneg*)
apply (*smt* (*verit*, *best*) *UNIV-I assms*(*1*) *divide-le-eq-1 infsum-geq-element infsum-not-zero-summable is-nonneg*)
proof −
fix $s_1::'a$
have *f1*: $(\sum_\infty v_0::'b.\ p\ (s_1,\ v_0)) \geq p\ (s_1,\ (SOME\ s'.\ p\ (s_1,\ s') > 0))$
apply (*rule infsum-geq-element*)
using *assms*(*1*) *is-nonneg* **apply** *fastforce*
using *assms*(*3*) **apply** *simp*
by *auto*
have *f2*: ... > *0*
by (*smt* (*verit*, *best*) *assms*(*2*) *f1 someI-ex*)
have *f3*: $(\sum_\infty s::'b.\ p\ (s_1,\ s) / (\sum_\infty v_0::'b.\ p\ (s_1,\ v_0))) =$
$(\sum_\infty s::'b.\ (p\ (s_1,\ s) * (1 / (\sum_\infty v_0::'b.\ p\ (s_1,\ v_0)))))$
by *auto*
also have *f4*: $... = (\sum_\infty s::'b.\ p\ (s_1,\ s)) * (1 / (\sum_\infty v_0::'b.\ p\ (s_1,\ v_0)))$
by (*metis infsum-cmult-left'*)
also have *f5*: ... = *1*
using *f2* **by** *auto*
thus $(\sum_\infty s::'b.\ p\ (s_1,\ s) / (\sum_\infty v_0::'b.\ p\ (s_1,\ v_0))) = (1::\mathbb{R})$
using *calculation* **by** *presburger*
qed

lemma *rvfun-uniform-dist-empty-is-zero*:
assumes *vwb-lens x*
shows $\forall\, v.\ ((x\ \boldsymbol{\mathcal{U}}\ \{\}) \ ;\ ([\![\$x^< = «v»]\!]_{\mathcal{I}e})) = rvfun\text{-}of\text{-}prfun\ 0_p$
apply (*auto*, *simp add*: *rvfun-uniform-dist-empty-zero*)
apply (*simp add*: *pfun-defs ureal-defs*)
apply (*expr-auto*)
by (*simp add*: *ureal-zero-0*)

lemma *rvfun-uniform-dist-is-uniform*:
assumes *finite* (*A*::*'b set*)
assumes *vwb-lens x*
assumes $A \neq \{\}$
shows $\forall\, v \in A.\ ((x\ \boldsymbol{\mathcal{U}}\ A) \ ;\ ([\![\$x^< = «v»]\!]_{\mathcal{I}e}) = (1/card\ «A»)_e)$
apply (*simp add*: *dist-defs pfun-defs*)
apply (*expr-auto*)
apply (*pred-auto*)
proof −

fix v::$'b$ **and** s_1::$'a$
assume *a1*: $v \in A$
let *?f1* $= \lambda v_0.$ (*if* $\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v$ *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))
let *?f2* $= \lambda v_0.$ (*if* $get_x\ v_0 = v$ *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))
let *?f* $= \lambda v_0.$ (*if* $(\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v) \wedge (get_x\ v_0 = v)$ *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))
let *?sum* $= \lambda v_0.$ ($\sum_\infty v$::$'b.$ *if* $\exists\, va$::$'b{\in}A.\ put_x\ v_0\ v = put_x\ s_1\ va$ *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))

have *one-dvd-card-A*: $\forall\, s.\ ((\exists\, v$::$'b{\in}A.\ s = put_x\ s_1\ v) \longrightarrow$
  $(((1$::$\mathbb{R})\ /\ (card\ \{v.\ \exists\, va$::$'b{\in}A.\ put_x\ s\ v = put_x\ s_1\ va\})) = ((1$::$\mathbb{R})\ /\ (card\ A))))$
  apply (*auto*)
  apply (*simp add*: *assms*(*2*))
  apply (*subgoal-tac* $\{v$::$'b.\ \exists\, va$::$'b{\in}A.\ put_x\ s_1\ v = put_x\ s_1\ va\} = A$)
  apply (*simp*)
  apply (*subst set-eq-iff*)
  apply (*auto*)
  proof (*rule ccontr*)
    fix xa::$'b$ **and** xb::$'b$ **and** xaa::$'b$
    assume *a1*: $xa \in A$
    assume *a2*: $xaa \in A$
    assume *a3*: $put_x\ s_1\ xb = put_x\ s_1\ xaa$
    assume *a4*: $\neg\ xb \in A$
    from *a2 a4* **have** $xaa \neq xb$
      by *auto*
    then have $put_x\ s_1\ xaa \neq put_x\ s_1\ xb$
      using *assms*(*2*) **by** (*meson vwb-lens-wb wb-lens-weak weak-lens.view-determination*)
    thus *False*
      using *a3* **by** *presburger*
  qed

have *finite* $\{put_x\ s_1\ xa \mid xa.\ xa \in A\}$
  apply (*rule finite-image-set*)
  using *assms*(*1*) **by** *auto*
then have *finite* $\{v_0.\ (\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v)\}$
  by (*smt* (*verit*, *del-insts*) *Collect-cong*)
then have *finite-states*: *finite* $\{v_0.\ (\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v) \wedge (get_x\ v_0 = v)\}$
  apply (*rule rev-finite-subset*[**where** $B = \{v_0.\ ((\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v))\}$])
  by *auto*

have *card-singleton*: *card* $\{v_0.\ (\exists\, v$::$'b{\in}A.\ v_0 = put_x\ s_1\ v) \wedge (get_x\ v_0 = v)\} = Suc\ (0)$
  apply (*simp add*: *card-1-singleton-iff*)
  apply (*rule-tac* $x = put_x\ s_1\ v$ **in** *exI*)
  using *a1 assms*(*2*) **by** *auto*

have $\forall\, v_0.$ *?f1* v_0 $*$ *?f2* v_0 $=$ *?f* v_0
  by (*auto*)
then have ($\sum_\infty v_0$::$'a.$ *?f1* v_0 $*$ *?f2* v_0 / *?sum* v_0) $=$ ($\sum_\infty v_0$::$'a.$ *?f0* v_0 / *?sum* v_0)
  by *auto*
also have ... $=$ ($\sum_\infty v_0$::$'a.$ *?f0* v_0 / (*card* $\{v.\ \exists\, va$::$'b{\in}A.\ put_x\ v_0\ v = put_x\ s_1\ va\}$))
  apply (*subst infsum-constant-finite-states*)
  apply (*subst finite-Collect-bex*)
  apply (*simp add*: *assms*(*1*))
  apply (*auto*)
  apply (*subgoal-tac* $\forall\, xa.\ (put_x\ s_1\ y = put_x\ v_0\ xa) \longrightarrow y = xa$)
  apply (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *mem-Collect-eq rev-finite-subset subset-iff*)
  using *weak-lens.view-determination vwb-lens-wb wb-lens-weak assms*(*2*) **by** *metis*

also have ... = ($\sum_\infty v_0$::$'a$. (*if* ($\exists\, v$::$'b{\in}A$. v_0 = put_x s_1 v) $\wedge$ (get_x v_0 = v) *then*
((*1*::$\mathbb{R}$) / (*card* {v. $\exists\, va$::$'b{\in}A$. put_x v_0 v = put_x s_1 va}))
else (*0*::$\mathbb{R}$)))
apply (*rule infsum-cong*)
by *simp*
also have ... = ($\sum_\infty v_0$::$'a$. (*if* ($\exists\, v$::$'b{\in}A$. v_0 = put_x s_1 v) $\wedge$ (get_x v_0 = v) *then*
((*1*::$\mathbb{R}$) / (*card A*)) *else* (*0*::$\mathbb{R}$)))
apply (*rule infsum-cong*)
using *one-dvd-card-A* **by** *presburger*
also have ... = ((*1*::$\mathbb{R}$) / (*card A*)) $*$ (*card* {v_0. ($\exists\, v$::$'b{\in}A$. v_0 = put_x s_1 v) $\wedge$ (get_x v_0 = v)})
apply (*rule infsum-constant-finite-states*)
using *finite-states* **by** *blast*
also have ... = ((*1*::$\mathbb{R}$) / (*card A*))
using *card-singleton* **by** *simp*
then show ($\sum_\infty v_0$::$'a$. *?f1* v_0 $*$ *?f2* v_0 / *?sum* v_0) = (*1*::$\mathbb{R}$) / *real* (*card A*)
using *calculation* **by** *presburger*
qed

lemma *rvfun-uniform-dist-inverse*:
assumes *finite* (*A*::$'b$ *set*)
assumes *vwb-lens x*
assumes $A \neq \{\}$
shows *rvfun-of-prfun* (*prfun-of-rvfun* ($x\,\boldsymbol{\mathcal{U}}\;A$)) = ($x\,\boldsymbol{\mathcal{U}}\;A$)
apply (*subst rvfun-inverse*)
apply (*simp add*: *assms*(*1*) *assms*(*2*) *rvfun-uniform-dist-is-prob*)
by *simp*

The possible values of x are chosen from a set A and they are equally likely to be observed in a program constructed by $x\,\boldsymbol{\mathcal{U}}\;A$.

lemma *rvfun-uniform-dist-is-dist*:
assumes *finite* (*A*::$'b$ *set*)
assumes *vwb-lens x*
assumes $A \neq \{\}$
shows *is-final-distribution* (($x\,\boldsymbol{\mathcal{U}}\;A$))
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *infsum-nonneg*)
apply (*smt* (*verit*) *divide-le-eq-1 infsum-0 infsum-geq-element infsum-not-exists*)
apply (*pred-auto*)
proof −
fix s_1::$'a$
let *?f* = λs. (*if* $\exists\, v$::$'b{\in}A$. s = put_x s_1 v *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) /
($\sum_\infty v$::$'b$. *if* $\exists\, va$::$'b{\in}A$. put_x s v = put_x s_1 va *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))
have *one-dvd-card-A*: $\forall\, s$. (($\exists\, xa$::$'b{\in}A$. s = put_x s_1 xa) $\longrightarrow$
(((*1*::$\mathbb{R}$) / (*card* {v. $\exists\, va$::$'b{\in}A$. put_x s v = put_x s_1 va})) = ((*1*::$\mathbb{R}$) / (*card A*))))
apply (*auto*)
apply (*simp add*: *assms*(*2*))
apply (*subgoal-tac* {v::$'b$. $\exists\, va$::$'b{\in}A$. put_x s_1 v = put_x s_1 va} = *A*)
apply (*simp*)
apply (*subst set-eq-iff*)
apply (*auto*)
proof (*rule ccontr*)
fix *xa*::$'b$ **and** *xb*::$'b$ **and** *va*::$'b$
assume *a1*: $xa \in A$
assume *a2*: $va \in A$

```
      assume a3: put_x s_1 xb = put_x s_1 va
      assume a4: ¬ xb ∈ A
      from a2 a4 have va ≠ xb
        by auto
      then have put_x s_1 xb ≠ put_x s_1 va
       using assms(2) by (metis mwb-lens-def vwb-lens-iff-mwb-UNIV-src weak-lens.view-determination)
      thus False
        using a3 by blast
    qed

  have finite {put_x s_1 xa | xa. xa ∈ A}
    apply (rule finite-image-set)
    using assms(1) by auto
  then have finite-states: finite {s. ∃ xa::'b∈A. s = put_x s_1 xa}
    by (smt (verit, del-insts) Collect-cong)

  have inj-on (λxa. put_x s_1 xa) A
    by (meson assms(2) inj-onI vwb-lens-wb wb-lens-def weak-lens.view-determination)
  then have card-A: card ((λxa. put_x s_1 xa) ‘ A ) = card A
    using card-image by blast
  have set-as-f-image: {s. ∃ xa::'b∈A. s= put_x s_1 xa} = ((λxa. put_x s_1 xa) ‘ A )
    by blast
  have (∑∞s::'a. ?f s) = (∑∞s::'a. (if ∃ xa::'b∈A. s= put_x s_1 xa then 1::ℝ else (0::ℝ))
      / (card {v. ∃ va::'b∈A. put_x s v = put_x s_1 va}))
    apply (subst infsum-constant-finite-states)
    apply (subst finite-Collect-bex)
    apply (simp add: assms(1))
    apply (auto)
    apply (subgoal-tac ∀ xa. (put_x s_1 y = put_x s xa) ⟶ y = xa)
    apply (smt (verit, ccfv-SIG) assms(1) mem-Collect-eq rev-finite-subset subset-iff)
    using weak-lens.view-determination vwb-lens-wb wb-lens-weak assms(2) by metis
  also have ... = (∑∞s::'a. (if ∃ xa::'b∈A. s = put_x s_1 xa then
              ((1::ℝ) / (card {v. ∃ va::'b∈A. put_x s v = put_x s_1 va}))
            else (0::ℝ)))
    apply (rule infsum-cong)
    by simp
  also have ... = (∑∞s::'a. (if ∃ xa::'b∈A. s = put_x s_1 xa then
              ((1::ℝ) / (card A)) else (0::ℝ)))
    apply (rule infsum-cong)
    using one-dvd-card-A by presburger
  also have ... = ((1::ℝ) / (card A)) ∗ (card {s. ∃ xa::'b∈A. s = put_x s_1 xa})
    apply (rule infsum-constant-finite-states)
    using finite-states by blast
  also have ... = ((1::ℝ) / (card A)) ∗ (card A)
    using card-A set-as-f-image by presburger
  also have ... = 1
    by (simp add: assms(1) assms(3))
  then show (∑∞s::'a. ?f s) = (1::ℝ)
    using calculation by presburger
qed

lemma rvfun-uniform-dist-is-dist':
  assumes finite (A::'b set)
  assumes vwb-lens x
  assumes A ≠ {}
```

shows *is-final-distribution* (*rvfun-of-prfun* (*prfun-of-rvfun* ($x\,\boldsymbol{\mathcal{U}}\,A$)))
apply (*simp add*: *rvfun-uniform-dist-inverse assms*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *rvfun-uniform-dist-is-dist*)

theorem *rvfun-uniform-dist-altdef*:
assumes *finite* (A::$'a$ *set*)
assumes *vwb-lens* x
assumes $A \neq \{\}$
shows $(x\,\boldsymbol{\mathcal{U}}\,A) = ([\![\bigsqcup\ v \in «A».\ x := «v»]\!]_{\mathcal{I}e}\ /\ card\ «A»)_e$
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*pred-auto*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*verit*, *best*) *Collect-mem-eq Collect-mono-iff assms*(*1*) *assms*(*2*) *mem-Collect-eq mwb-lens-weak rev-finite-subset vwb-lens.axioms*(*2*) *weak-lens.put-get*)
proof −
fix a::$'b$ **and** v::$'a$
assume *a1*: $v \in A$
have $\{s::'a.\ \exists\,va::'a{\in}A.\ put_x\ (put_x\ a\ v)\ s = put_x\ a\ va\} =$
$\{s::'a.\ \exists\,va::'a{\in}A.\ put_x\ a\ s = put_x\ a\ va\}$
using *assms*(*2*) **by** *auto*
also have ... $= \{s::'a.\ \exists\,xb::'a{\in}A.\ xb = s\}$
by (*metis assms*(*2*) *vwb-lens-wb wb-lens-weak weak-lens.view-determination*)
then show $(1::\mathbb{R}) * real\ (card\ \{s::'a.\ \exists\,va::'a{\in}A.\ put_x\ (put_x\ a\ v)\ s = put_x\ a\ va\}) = real\ (card\ A)$
by (*simp add*: *calculation*)
qed

theorem *prfun-uniform-dist-altdef'*:
assumes *finite* (A::$'a$ *set*)
assumes *vwb-lens* x
assumes $A \neq \{\}$
shows *rvfun-of-prfun* (*prfun-of-rvfun* ($x\,\boldsymbol{\mathcal{U}}\,A$)) $= ([\![\bigsqcup\ v \in «A».\ x := «v»]\!]_{\mathcal{I}e}\ /\ card\ «A»)_e$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *rvfun-uniform-dist-inverse rvfun-uniform-dist-altdef*)

theorem *prfun-uniform-dist-left*:
assumes *finite* (A::$'a$ *set*)
assumes *vwb-lens* x
assumes $A \neq \{\}$
shows (*prfun-of-rvfun* ($x\,\boldsymbol{\mathcal{U}}\,A$)) ; P =
prfun-of-rvfun $((\sum v \in «A».\ (([\ x^< \rightsquigarrow «v»\ ]\ \dagger \bullet(rvfun\text{-}of\text{-}prfun\ P))))\ /\ card\ («A»))_e$
apply (*simp add*: *pseqcomp-def*)
apply (*subst prfun-uniform-dist-altdef'*)
apply (*simp-all add*: *assms*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*expr-auto*)
apply (*pred-auto*)
proof −
fix a **and** b :: $'b$
let *?fl-1* $= \lambda v_0.\ (if\ \exists\,v::'a{\in}A.\ v_0 = put_x\ a\ v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?fl-2* $= \lambda v_0.\ rvfun\text{-}of\text{-}prfun\ P\ (v_0,\ b)\ /\ real\ (card\ A)$

have *finite* $\{put_x\ a\ xa \mid xa.\ xa \in A\}$
apply (*rule finite-image-set*)
using *assms*(*1*) **by** *auto*
then have *finite-states*: *finite* $\{v_0.\ \exists\,v::'a{\in}A.\ v_0 = put_x\ a\ v\}$

by (*smt* (*verit*, *del-insts*) *Collect-cong*)

have ($\sum_\infty v_0$::'*b*. *?fl-1* v_0 ∗ *rvfun-of-prfun P* (v_0, *b*) / *real* (*card A*))
= ($\sum_\infty v_0$::'*b*. *?fl-1* v_0 ∗ *?fl-2* v_0)
by *auto*
also have ... = ($\sum_\infty v_0$::'*b* ∈ {v_0. ∃ *v*::'*a*∈*A*. v_0 = put_x *a v*}. *?fl-2* v_0)
apply (*subst infsum-mult-subset-left*)
by *simp*
also have *fl*: ... = ($\sum$ v_0::'*b* ∈ {v_0. ∃ *v*::'*a*∈*A*. v_0 = put_x *a v*}. *rvfun-of-prfun P* (v_0, *b*)) / *real* (*card A*)
by (*smt* (*verit*, *ccfv-SIG*) *finite-states infsum-finite sum.cong sum-divide-distrib*)

have *inj-on-A*: *inj-on* (λ*xa*. put_x *a xa*) *A*
by (*meson assms*(*2*) *inj-onI vwb-lens-wb wb-lens-def weak-lens.view-determination*)

have *frl*: ($\sum$ v_0::'*b* ∈ {v_0. ∃ *v*::'*a*∈*A*. v_0 = put_x *a v*}. *rvfun-of-prfun P* (v_0, *b*))
= ($\sum$ *v*::'*a*∈*A*. *rvfun-of-prfun P* (put_x *a v*, *b*))
apply (*rule sum.reindex-cong*[**where** *l* = (λ*xa*. put_x *a xa*)])
apply (*simp add*: *inj-on-A*)
apply *blast*
by *simp*

show ($\sum_\infty v_0$::'*b*. *?fl-1* v_0 ∗ *rvfun-of-prfun P* (v_0, *b*) / *real* (*card A*)) =
($\sum$ *v*::'*a*∈*A*. *rvfun-of-prfun P* (put_x *a v*, *b*)) / *real* (*card A*)
using *calculation fl frl* **by** *presburger*
qed

5.6.8 Parallel composition

lemma *rvfun-parallel-f-is-prob*:
assumes *is-nonneg* (p ∗ q)$_e$
shows *is-prob* (p $\|_f$ q)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*metis* (*no-types*, *lifting*) *SEXP-def assms divide-nonneg-nonneg infsum-nonneg is-nonneg*)
proof −
fix *a b*
have *nonneg*: ∀ *s*. *p s* ∗ *q s* ≥ *0*
using *assms is-nonneg* **by** (*metis SEXP-def*)
show *p* (*a*, *b*) ∗ *q* (*a*, *b*) / ($\sum_\infty v_0$. *p* (*a*, v_0) ∗ *q* (*a*, v_0)) ≤ (*1*::ℝ)
proof (*cases* (λ*s*′. *p* (*a*, *s*′) ∗ *q* (*a*, *s*′)) *summable-on UNIV*)
assume (λ*s*′. *p* (*a*, *s*′) ∗ *q* (*a*, *s*′)) *summable-on UNIV*
then have ($\sum_\infty v_0$. *p* (*a*, v_0) ∗ *q* (*a*, v_0)) ≥ *p* (*a*, *b*) ∗ *q* (*a*, *b*)
by (*meson UNIV-I infsum-geq-element nonneg*)
then show *p* (*a*, *b*) ∗ *q* (*a*, *b*) / ($\sum_\infty v_0$. *p* (*a*, v_0) ∗ *q* (*a*, v_0)) ≤ (*1*::ℝ)
by (*smt* (*verit*) *nonneg divide-le-eq-1*)
next
assume ¬ ((λ*s*′. *p* (*a*, *s*′) ∗ *q* (*a*, *s*′)) *summable-on UNIV*)

then show *p* (*a*, *b*) ∗ *q* (*a*, *b*) / ($\sum_\infty v_0$. *p* (*a*, v_0) ∗ *q* (*a*, v_0)) ≤ (*1*::ℝ)
by (*simp add*: *infsum-not-exists*)
qed
qed

lemma *divide-eq*: ⟦*p* = *q* ∧ *P* = *Q*⟧ ⟹ (*p*::ℝ) / *P* = *q* / *Q*

by *simp*

theorem *rvfun-parallel-f-assoc*:
assumes
$\forall s.\ (\sum_\infty v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0)) = 0 \longrightarrow$
$((\sum_\infty v_0.\ q\ (s,\ v_0) * r\ (s,\ v_0)) = 0 \vee (\sum_\infty v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0) * r\ (s,\ v_0)) = 0)$
$\forall s.\ (\sum_\infty v_0.\ q\ (s,\ v_0) * r\ (s,\ v_0)) = 0 \longrightarrow$
$((\sum_\infty v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0)) = 0 \vee (\sum_\infty v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0) * r\ (s,\ v_0)) = 0)$
shows $(p \parallel_f q) \parallel_f r = p \parallel_f (q \parallel_f r)$
apply (*simp add*: *dist-defs*)
apply (*simp add*: *fun-eq-iff*)
apply (*rule allI*)+
apply (*rule divide-eq*)
apply (*expr-auto*)
apply (*subst mult.assoc*[*symmetric*])
proof −
fix $a{::}'a$
let $?lhs\text{-}pq = (\sum_\infty v_0{::}'b.\ p\ (a,\ v_0) * q\ (a,\ v_0))$
let $?rhs\text{-}qr = (\sum_\infty v_0{::}'b.\ q\ (a,\ v_0) * r\ (a,\ v_0))$
let $?pqr = (\lambda v_0.\ p\ (a,\ v_0) * q\ (a,\ v_0) * r\ (a,\ v_0))$

let $?lhs = ?lhs\text{-}pq * (\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq)$
let $?rhs = ?rhs\text{-}qr * (\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr)$

show $?lhs = ?rhs$

proof (*cases* $?lhs\text{-}pq = 0$)
case *True*
assume *T-pq*: $?lhs\text{-}pq = 0$
then have *lhs-0*: $?lhs = 0$
using *mult-eq-0-iff* **by** *blast*
then show *?thesis*

proof (*cases* $?rhs\text{-}qr = 0$)
case *True*
assume *T-qr*: $?rhs\text{-}qr = 0$
then have *rhs-0*: $?rhs = 0$
using *mult-eq-0-iff* **by** *blast*
then show *?thesis*
using *lhs-0* **by** *presburger*
next
case *False*
assume *F-qr*: $\neg ?rhs\text{-}qr = 0$
from *T-pq F-qr assms*(*1*) **have** $(\sum_\infty v_0.\ ?pqr\ v_0) = 0$
by *blast*
then have *F-qr-summable*:
((*?pqr summable-on UNIV*) ∧ *has-sum ?pqr UNIV 0*) ∨ ¬ *?pqr summable-on UNIV*
apply (*subst infset-0-not-summable-or-sum-to-zero*)
by *simp*+
then show *?thesis*

proof (*cases* ((*?pqr summable-on UNIV*) ∧ *has-sum ?pqr UNIV 0*))
case *True*
then have *has-sum* $(\lambda v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr)$ *UNIV* $(0\ /\ ?rhs\text{-}qr)$

using *has-sum-cdiv-left* **by** *fastforce*
then have *sum-rhs-pqr-0*: $(\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr) = 0$
by (*simp add*: *infsumI*)
have *sum-lhs-pqr-0*: $(\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq) = 0$
by (*simp add*: *T-pq*)
then show *?thesis*
using *sum-rhs-pqr-0* **by** *simp*
next
case *False*
then have *F-qr-summable-F*: ¬ *?pqr summable-on UNIV*
using *F-qr-summable* **by** *blast*

have $\neg(\lambda v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr)$ *summable-on UNIV*
apply (*subst not-summable-on-cdiv-left′*)
by (*simp add*: *F-qr F-qr-summable-F*)+
then have *sum-rhs-pqr-0*: $(\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr) = 0$
using *infsum-not-zero-summable* **by** *blast*
then show *?thesis*
by (*simp add*: *lhs-0*)
qed
qed
next
case *False*
assume *F-pq*: ¬*?lhs-pq* = *0*
then show *?thesis*

proof (*cases ?rhs-qr* = *0*)
case *True*
assume *T-qr*: *?rhs-qr* = *0*
then have *rhs-0*: *?rhs* = *0*
using *mult-eq-0-iff* **by** *blast*
from *T-qr F-pq assms*(*2*) **have** $(\sum_\infty v_0.\ ?pqr\ v_0) = 0$
by *blast*
then have *F-pq-summable*:
((*?pqr summable-on UNIV*) ∧ *has-sum ?pqr UNIV 0*) ∨ ¬ *?pqr summable-on UNIV*
apply (*subst infset-0-not-summable-or-sum-to-zero*)
by *simp*+
then show *?thesis*

proof (*cases* ((*?pqr summable-on UNIV*) ∧ *has-sum ?pqr UNIV 0*))
case *True*
then have *has-sum* $(\lambda v_0{::}'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq)$ *UNIV* $(0\ /\ ?lhs\text{-}pq)$
using *has-sum-cdiv-left* **by** *fastforce*
then have *sum-lhs-pqr-0*: $(\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq) = 0$
by (*simp add*: *infsumI*)
have *sum-rhs-pqr-0*: $(\sum_\infty v_0{::}'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr) = 0$
by (*simp add*: *T-qr*)
then show *?thesis*
using *sum-lhs-pqr-0* **by** *simp*
next
case *False*
then have *F-pq-summable-F*: ¬ *?pqr summable-on UNIV*
using *F-pq-summable* **by** *blast*
have $\neg(\lambda v_0{::}'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq)$ *summable-on UNIV*
apply (*subst not-summable-on-cdiv-left′*)

by (*simp add*: *F-pq F-pq-summable-F*)+
then have *sum-lhs-pqr-0*: $(\sum_\infty v_0::'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq) = 0$
using *infsum-not-zero-summable* **by** *blast*
then show *?thesis*
by (*simp add*: *rhs-0*)
qed
next
case *False*
assume *F-qr*: $\neg ?rhs\text{-}qr = 0$
show *?thesis*

proof (*cases ?pqr summable-on UNIV*)
case *True*
assume *F-pqr*: *?pqr summable-on UNIV*
have *F-lhs-pqr*: $?lhs = (\sum_\infty v_0::'b.\ ?lhs\text{-}pq * ?pqr\ v_0\ /\ ?lhs\text{-}pq)$
apply (*subst infsum-cmult-right*[*symmetric*])
using *F-pqr summable-on-cdiv-left′* **apply** *fastforce*
by *simp*
have *F-lhs-pqr′*: $... = (\sum_\infty v_0::'b.\ ?pqr\ v_0)$
by (*simp add*: *F-pq*)
have *F-rhs-pqr*: $?rhs = (\sum_\infty v_0::'b.\ ?rhs\text{-}qr * ?pqr\ v_0\ /\ ?rhs\text{-}qr)$
apply (*subst infsum-cmult-right*[*symmetric*])
using *F-pqr summable-on-cdiv-left′* **apply** *fastforce*
by *simp*
have *F-rhs-pqr′*: $... = (\sum_\infty v_0::'b.\ ?pqr\ v_0)$
by (*simp add*: *F-qr*)
show *?thesis*
using *F-lhs-pqr F-lhs-pqr′ F-rhs-pqr F-rhs-pqr′* **by** *presburger*
next
case *False*
assume *F-pqr*: $\neg$*?pqr summable-on UNIV*
have *F-lhs-pqr*: $\neg(\lambda v_0::'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq)$ *summable-on UNIV*
apply (*subst not-summable-on-cdiv-left′*)
by (*simp add*: *F-pq F-pqr*)+
then have *sum-lhs-pqr-0*: $(\sum_\infty v_0::'b.\ ?pqr\ v_0\ /\ ?lhs\text{-}pq) = 0$
using *infsum-not-zero-summable* **by** *blast*
have *F-rhs-pqr*: $\neg(\lambda v_0::'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr)$ *summable-on UNIV*
apply (*subst not-summable-on-cdiv-left′*)
by (*simp add*: *F-qr F-pqr*)+
then have *sum-rhs-pqr-0*: $(\sum_\infty v_0::'b.\ ?pqr\ v_0\ /\ ?rhs\text{-}qr) = 0$
using *infsum-not-zero-summable* **by** *blast*
then show *?thesis*
by (*simp add*: *sum-lhs-pqr-0*)
qed
qed
qed
qed

A specific variant of associativity when p, q, and r all have non-negative real values.

theorem *rvfun-parallel-f-assoc-nonneg*:

assumes *is-nonneg p is-nonneg q is-nonneg r*
$\forall s.\ (\neg\ (\lambda v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0))$ *summable-on UNIV*$) \longrightarrow$
$((\forall v_0.\ q\ (s,\ v_0) * r\ (s,\ v_0) = 0) \vee (\neg\ (\lambda v_0.\ q\ (s,\ v_0) * r\ (s,\ v_0))$ *summable-on UNIV*$))$
$\forall s.\ (\neg\ (\lambda v_0.\ q\ (s,\ v_0) * r\ (s,\ v_0))$ *summable-on UNIV*$) \longrightarrow$

$((\forall v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0) = 0) \vee (\neg\ (\lambda v_0.\ p\ (s,\ v_0) * q\ (s,\ v_0))$ *summable-on UNIV*))
shows $(p \parallel_f q) \parallel_f r = p \parallel_f (q \parallel_f r)$
apply (*rule rvfun-parallel-f-assoc*)
apply (*auto*)
proof −
fix *s*
let $?pq = \lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0)$
let $?qr = \lambda v_0::'b.\ q\ (s,\ v_0) * r\ (s,\ v_0)$
let $?pqr = \lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0) * r\ (s,\ v_0)$

assume *a1*: $(\sum_\infty v_0::'b.\ ?pq\ v_0) = (0::\mathbb{R})$
assume *a2*: $\neg\ (\sum_\infty v_0::'b.\ \ ?pqr\ v_0) = (0::\mathbb{R})$

have *pq-0*: $(\forall s.\ ?pq\ s = 0) \vee \neg\ ?pq$ *summable-on UNIV*
by (*smt* (*verit*, *ccfv-threshold*) *a1 a2 assms*(*1*) *assms*(*2*) *infset-0-not-summable-or-zero infsum-cong is-nonneg mult-cancel-left1 mult-nonneg-nonneg*)
show $(\sum_\infty v_0::'b.\ ?qr\ v_0) = (0::\mathbb{R})$
proof (*cases* $(\forall s.\ ?pq\ s = 0)$)
case *True*
then have $(\forall s.\ ?pqr\ s = 0)$
using *mult-eq-0-iff* **by** *blast*
then have $(\sum_\infty v_0::'b.\ \ ?pqr\ v_0) = (0::\mathbb{R})$
by (*meson infsum-0*)
then show *?thesis*
using *a2* **by** *blast*
next
case *False*
then have ¬ *?pq summable-on UNIV*
using *pq-0* **by** *blast*
then show *?thesis*
using *assms*(*4*) **by** (*meson infsum-0 infsum-not-exists*)
qed
next
fix *s*
let $?pq = \lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0)$
let $?qr = \lambda v_0::'b.\ q\ (s,\ v_0) * r\ (s,\ v_0)$
let $?pqr = \lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0) * r\ (s,\ v_0)$

assume *a1*: $(\sum_\infty v_0::'b.\ ?qr\ v_0) = (0::\mathbb{R})$
assume *a2*: $\neg\ (\sum_\infty v_0::'b.\ \ ?pqr\ v_0) = (0::\mathbb{R})$

have *qr-0*: $(\forall s.\ ?qr\ s = 0) \vee \neg\ ?qr$ *summable-on UNIV*
by (*smt* (*verit*, *ccfv-SIG*) *a1 a2 assms*(*2*) *assms*(*3*) *distrib-left infset-0-not-summable-or-zero infsum-cong is-nonneg mult.assoc mult-nonneg-nonneg*)
show $(\sum_\infty v_0::'b.\ ?pq\ v_0) = (0::\mathbb{R})$
proof (*cases* $(\forall s.\ ?qr\ s = 0)$)
case *True*
then have $(\forall s.\ ?pqr\ s = 0)$
using *mult-eq-0-iff* **by** *auto*
then have $(\sum_\infty v_0.\ \ ?pqr\ v_0) = (0::\mathbb{R})$
by (*meson infsum-0*)
then show *?thesis*
using *a2* **by** *blast*
next
case *False*

then have ¬ *?qr summable-on UNIV*
using *qr-0* **by** *blast*
then show *?thesis*
using *assms*(*5*) **by** (*meson infsum-0 infsum-not-exists*)
qed
qed

theorem *rvfun-parallel-f-assoc-prob*:
assumes ∀ *s*::′*a*. *is-prob* ((*curry p*) *s*)
∀ *s*::′*a*. *is-prob* ((*curry q*) *s*)
∀ *s*::′*a*. *is-prob* ((*curry r*) *s*)
assumes ∀ *s*::′*a*. ((*curry q*) *s*) *summable-on UNIV*
shows $(p \parallel_f q) \parallel_f r = p \parallel_f (q \parallel_f r)$
proof −
fix *a*::′*a*
have *a1*: ∀ *s*. *p s* ≥ *0* ∧ *p s* ≤ *1*
using *assms*(*1*) **by** (*expr-auto add*: *dist-defs*)

have *a2*: ∀ *s*. *q s* ≥ *0* ∧ *q s* ≤ *1*
using *assms*(*2*) **by** (*expr-auto add*: *dist-defs*)

have *a3*: ∀ *s*. *r s* ≥ *0* ∧ *r s* ≤ *1*
using *assms*(*3*) **by** (*expr-auto add*: *dist-defs*)

have *pq-summable*: ∀ *s*. (λv_0::′*b*. *p* (*s*, v_0) ∗ *q* (*s*, v_0)) *summable-on UNIV*
proof (*rule allI*)
fix *s*
show (λv_0::′*b*. *p* (*s*, v_0) ∗ *q* (*s*, v_0)) *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** *g* = λ*x*. *q* (*s*, *x*)])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms*(*4*) **apply** (*metis* (*no-types*, *lifting*) *curry-def summable-on-cong*)
by (*simp add*: *a1 a2 mult-left-le-one-le*)
qed

have *qr-summable*: ∀ *s*. (λv_0::′*b*. *q* (*s*, v_0) ∗ *r* (*s*, v_0)) *summable-on UNIV*
proof (*rule allI*)
fix *s*
show (λv_0::′*b*. *q* (*s*, v_0) ∗ *r* (*s*, v_0)) *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** *g* = λ*x*. *q* (*s*, *x*)])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms*(*4*) **apply** (*metis* (*no-types*, *lifting*) *curry-def summable-on-cong*)
by (*simp add*: *a2 a3 mult-right-le-one-le*)
qed

show *?thesis*
apply (*rule rvfun-parallel-f-assoc-nonneg*)
apply (*simp add*: *a1 a2 a3 is-nonneg*)+
using *pq-summable* **apply** *presburger*
using *qr-summable* **by** *presburger*
qed

theorem *rvfun-parallel-f-assoc-prob*′:
assumes ∀ *s*::′*a*. *is-prob* ((*curry p*) *s*)

$\forall s::'a.$ *is-prob* ((*curry q*) *s*)
$\forall s::'a.$ *is-prob* ((*curry r*) *s*)
assumes $\forall s::'a.$ ((*curry p*) *s*) *summable-on UNIV* $\wedge$ ((*curry r*) *s*) *summable-on UNIV*
shows $(p \parallel_f q) \parallel_f r = p \parallel_f (q \parallel_f r)$
proof −
fix $a::'a$
have *a1*: $\forall s.\ p\ s \geq 0 \wedge p\ s \leq 1$
using *assms*(*1*) **by** (*expr-auto add*: *dist-defs*)

have *a2*: $\forall s.\ q\ s \geq 0 \wedge q\ s \leq 1$
using *assms*(*2*) **by** (*expr-auto add*: *dist-defs*)

have *a3*: $\forall s.\ r\ s \geq 0 \wedge r\ s \leq 1$
using *assms*(*3*) **by** (*expr-auto add*: *dist-defs*)

have *pq-summable*: $\forall s.\ (\lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0))$ *summable-on UNIV*
proof (*rule allI*)
fix *s*
show $(\lambda v_0::'b.\ p\ (s,\ v_0) * q\ (s,\ v_0))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda x.\ p\ (s,\ x)$])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms*(*4*) **apply** (*metis* (*no-types*, *lifting*) *curry-def summable-on-cong*)
by (*simp add*: *a1 a2 mult-right-le-one-le*)
qed

have *qr-summable*: $\forall s.\ (\lambda v_0::'b.\ q\ (s,\ v_0) * r\ (s,\ v_0))$ *summable-on UNIV*
proof (*rule allI*)
fix *s*
show $(\lambda v_0::'b.\ q\ (s,\ v_0) * r\ (s,\ v_0))$ *summable-on UNIV*
apply (*subst summable-on-iff-abs-summable-on-real*)
apply (*rule abs-summable-on-comparison-test*[**where** $g = \lambda x.\ r\ (s,\ x)$])
apply (*subst summable-on-iff-abs-summable-on-real*[*symmetric*])
using *assms*(*4*) **apply** (*metis* (*no-types*, *lifting*) *curry-def summable-on-cong*)
by (*simp add*: *a2 a3 mult-left-le-one-le*)
qed

show *?thesis*
apply (*rule rvfun-parallel-f-assoc-nonneg*)
apply (*simp add*: *a1 a2 a3 is-nonneg*)+
using *pq-summable* **apply** *presburger*
using *qr-summable* **by** *presburger*
qed

lemma *rvfun-pparallel-is-dist*:
assumes *is-final-prob p*
assumes *is-final-prob q*
assumes *summable-on-final p* $\vee$ *summable-on-final q*
assumes *final-reachable2 p q*
shows *is-final-distribution* (*pparallel-f p q*)
apply (*expr-auto add*: *dist-defs*)
using *infsum-nonneg is-final-prob-altdef assms*(*1*) *assms*(*2*)
apply (*metis* (*mono-tags*, *lifting*) *divide-nonneg-nonneg mult-nonneg-nonneg*)
apply (*subgoal-tac* $p\ (s_1,\ s) * q\ (s_1,\ s) \leq (\sum_\infty v_0.\ p\ (s_1,\ v_0) * q\ (s_1,\ v_0))$)
apply (*smt* (*verit*, *del-insts*) *assms*(*1*) *assms*(*2*) *divide-le-eq-1 is-final-prob-altdef mult-nonneg-nonneg*)

```
  apply (rule infsum-geq-element)
  apply (simp add: assms(1) assms(2) is-final-prob-altdef)
  using assms(1) assms(2) assms(3) rvfun-joint-prob-summable-on-product apply blast
  apply (simp add: assms(1))
proof −
  fix s1
  let ?P = λs'. p (s1, s') > 0 ∧ q (s1, s') > 0

  have f1: ?P (SOME s'. ?P s')
    apply (rule someI-ex[where P=?P])
    using assms(4) by blast
  have f2: (λs. p (s1, s) ∗ q (s1, s)) (SOME s'. ?P s') ≤ (∑∞s'. p (s1, s') ∗ q (s1, s'))
    apply (rule infsum-geq-element)
    apply (simp add: assms(1) assms(2) is-final-prob-altdef)
    apply (simp add: assms(1) assms(2) assms(3) rvfun-joint-prob-summable-on-product)
    by (simp)+
  also have f3: ... > 0
    by (smt (verit, best) f1 f2 mult-le-0-iff)
  have f4: (∑∞s. (p (s1, s) ∗ q (s1, s) / (∑∞s'. p (s1, s') ∗ q (s1, s')))) =
    (∑∞s. (p (s1, s) ∗ q (s1, s) ∗ (1 / (∑∞s'. p (s1, s') ∗ q (s1, s')))))
    by force
  also have f5: ... = (∑∞s. (p (s1, s) ∗ q (s1, s))) ∗ (1 / (∑∞s'. p (s1, s') ∗ q (s1, s')))
    apply (rule infsum-cmult-left)
    by (simp add: infsum-not-zero-summable)
  also have f6: ... = 1
    using f3 by auto
  show (∑∞s. (p (s1, s) ∗ q (s1, s) / (∑∞s'. p (s1, s') ∗ q (s1, s')))) = (1::ℝ)
    using f4 f5 f6 by presburger
qed

lemma rvfun-pparallel-is-conflict-zero:
  assumes is-nonneg p
  assumes is-nonneg q
  assumes conflict: ∀ s1. ¬(∃ s'::'a. p (s1, s') > 0 ∧ q (s1, s') > 0)
  shows (pparallel-f p q) = 0R
  apply (expr-auto add: dist-defs)
  by (smt (verit, best) assms(1) assms(2) conflict is-nonneg)

lemma rvfun-parallel-inverse:
  assumes is-nonneg (p∗q)e
  shows rvfun-of-prfun (prfun-of-rvfun (pparallel-f p q)) = pparallel-f p q
  apply (subst rvfun-inverse)
  apply (simp add: assms(1) is-nonneg2 rvfun-parallel-f-is-prob)
  by simp

theorem prfun-rvfun-parallel-assoc-f:
  fixes P Q R :: ('s1, 's2) rvfun
  assumes is-nonneg P is-nonneg Q is-nonneg R
  assumes summable-on-final2 P Q
  assumes summable-on-final2 Q R
  assumes final-reachable2 P Q
  assumes final-reachable2 Q R
  shows (P ∥ Q) ∥ R = P ∥ (Q ∥ R)
  apply (simp add: pfun-defs)
  apply (rule HOL.arg-cong[where f=prfun-of-rvfun])
```

apply (*subst rvfun-inverse*)
apply (*simp add*: *rvfun-parallel-f-is-prob is-nonneg2 assms*(*1*) *assms*(*2*))
apply (*subst rvfun-parallel-inverse*)
apply (*simp add*: *assms*(*2*) *assms*(*3*) *is-nonneg2*)
apply (*rule rvfun-parallel-f-assoc-nonneg*)
apply (*simp add*: *assms*(*1*−*3*))+
apply (*simp add*: *assms*(*4*))
by (*simp add*: *assms*(*5*))

theorem *prfun-parallel-assoc-p*:
fixes $P\ Q\ R$ :: $('s_1, 's_2)$ *prfun*
assumes *summable-on-final* (*rvfun-of-prfun Q*)
shows $(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*subst rvfun-inverse*)
apply (*simp add*: *prfun-in-0-1′ rvfun-parallel-f-is-prob is-nonneg*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *prfun-in-0-1′ rvfun-parallel-f-is-prob is-nonneg*)
apply (*rule rvfun-parallel-f-assoc-prob*)
apply (*simp add*: *is-prob-final-prob ureal-is-prob*)+
apply (*simp add*: *curry-def*)
using *assms* **by** *blast*

theorem *prfun-parallel-commute-ff*:
fixes $P\ Q$::$('a, 'b)$ *rvfun*
shows $P \parallel Q = Q \parallel P$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
by (*simp add*: *mult.commute*)

theorem *prfun-parallel-commute-pp*:
fixes $P\ Q$::$('a, 'b)$ *prfun*
shows $P \parallel Q = Q \parallel P$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
by (*simp add*: *mult.commute*)

theorem *prfun-parallel-commute-rp*:
fixes P ::$('a, 'b)$ *rvfun* **and** Q :: $('a, 'b)$ *prfun*
shows $P \parallel Q = Q \parallel P$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
by (*simp add*: *mult.commute*)

theorem *prfun-parallel-commute-pf*:
fixes P ::$('a, 'b)$ *prfun* **and** Q :: $('a, 'b)$ *rvfun*
shows $P \parallel Q = Q \parallel P$
apply (*simp add*: *pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
by (*simp add*: *mult.commute*)

Any nonzero constant is a left identity in parallel with a distribution.

theorem *prfun-parallel-left-identity-ff*:
fixes c::$\mathbb{R}$

assumes *is-final-distribution P*
assumes $c \neq 0$
shows $(\lambda s.\ c) \parallel P = $ *prfun-of-rvfun P*
apply (*simp add*: *pfun-defs dist-defs*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*expr-auto*)
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*))
by (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))

theorem *prfun-parallel-left-identity-fp*:
fixes $c::\mathbb{R}$
assumes $c \neq 0$
assumes *is-final-distribution* (*rvfun-of-prfun P*)
shows $(\lambda s.\ c) \parallel P = P$
apply (*simp add*: *pfun-defs dist-defs*)
apply (*expr-auto*)
apply (*subst infsum-cmult-right*)
apply (*simp add*: *assms*(*2*) *pdrfun-prob-sum1-summable'*(*4*))
apply (*simp add*: *ureal-defs*)
apply (*auto*)
using *assms*(*1*) **apply** *presburger*
apply (*subst rvfun-prob-sum1-summable*(*2*))
defer
apply (*metis abs-ereal-ge0 atLeastAtMost-iff div-by-1 ereal-less-eq*(*1*) *ereal-real ereal-times*(*1*) *max.absorb2 min.orderE nle-le ureal2ereal ureal2ereal-inverse*)
proof −
have *is-final-distribution* ((*real-of-ereal* ∘ *ureal2ereal*) $P)_e$
using *assms*(*2*) *ureal-defs*
by (*smt* (*verit*, *best*) *case-prod-curry cond-case-prod-eta curry-def*)
then show *is-final-distribution* ($\lambda a::'a \times 'b.$ *real-of-ereal* (*ureal2ereal* (*P a*)))
by (*simp add*: *comp-def SEXP-def*)
qed

Any nonzero constant is a right identity in parallel with a distribution.

theorem *prfun-parallel-right-identity-ff*:
fixes $c::\mathbb{R}$
assumes *is-final-distribution P*
assumes $c \neq 0$
shows $P \parallel (\lambda s.\ c) = $ *prfun-of-rvfun P*
apply (*simp add*: *pfun-defs dist-defs*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*expr-auto*)
apply (*subst infsum-cmult-left*)
apply (*simp add*: *assms*(*1*) *rvfun-prob-sum1-summable*(*3*))
by (*simp add*: *assms rvfun-prob-sum1-summable*(*2*))

theorem *prel-parallel-right-identity-pf*:
fixes $c::\mathbb{R}$
assumes $c \neq 0$
assumes *is-final-distribution* (*rvfun-of-prfun P*)
shows $P \parallel (\lambda s.\ c) = P$
apply (*simp add*: *pfun-defs dist-defs*)
apply (*expr-auto*)
apply (*subst infsum-cmult-left*)

apply (*simp add*: *assms*(*2*) *pdrfun-prob-sum1-summable′*(*4*))
apply (*simp add*: *ureal-defs*)
apply (*auto*)
using *assms*(*1*) **apply** *presburger*
apply (*subst rvfun-prob-sum1-summable*(*2*))
defer
apply (*metis abs-ereal-ge0 atLeastAtMost-iff div-by-1 ereal-less-eq*(*1*) *ereal-real ereal-times*(*1*) *max.absorb2 min.orderE nle-le ureal2ereal ureal2ereal-inverse*)
proof −
have *is-final-distribution* ((*real-of-ereal* ∘ *ureal2ereal*) $P)_e$
using *assms*(*2*) *ureal-defs*
by (*smt* (*verit*, *best*) *case-prod-curry cond-case-prod-eta curry-def*)
then show *is-final-distribution* (λa::′*a* × ′*b*. *real-of-ereal* (*ureal2ereal* (*P a*)))
by (*simp add*: *comp-def SEXP-def*)
qed

theorem *prfun-parallel-right-zero*:
fixes *P* :: (′*a*, ′*b*) *rvfun*
shows ($P \parallel 0_R) = 0_p$
apply (*simp add*: *pfun-defs dist-defs ureal-defs*)
by (*metis SEXP-apply ureal2ereal-inverse zero-ureal.rep-eq*)

theorem *prfun-parallel-left-zero*:
fixes *Q* :: (′*a*, ′*b*) *rvfun*
shows ($0_R \parallel Q) = 0_p$
apply (*simp add*: *pfun-defs dist-defs ureal-defs*)
by (*metis SEXP-apply ureal2ereal-inverse zero-ureal.rep-eq*)

The parallel composition of a P with a uniform distribution is just a normalised summation of P with x in its final states substituted for each value in A.

theorem *prfun-parallel-uniform-dist*:
fixes *P* ::(′*a*, ′*a*) *rvfun*
assumes *finite A*
assumes *vwb-lens x*
assumes $A \neq \{\}$
shows ($x\,\mathcal{U}\;A) \parallel P$ =
prfun-of-rvfun $((\sum v \in «A».\ ([\![x := «v»]\!]_{\mathcal{I}e} * ([\ x^{>} \leadsto «v»\ ] \dagger P)))$
$/\ (\sum\ v \in «A».\ ([\ x^{>} \leadsto «v»\ ] \dagger P)))_e$
apply (*subst rvfun-uniform-dist-altdef*)
apply (*simp add*: *assms*(*1*−*3*))+
apply (*simp add*: *dist-defs pfun-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*expr-auto add*: *rel*)
apply (*pred-auto*)
proof −
fix *a* **and** *xa*
assume *a1*: $xa \in A$

let *?lhs-1* = (*real* (*card A*) ∗ ($\sum_\infty v_0$::′*a*.
(*if* $\exists v$::′*b*∈*A*. $v_0 = put_x\ a\ v$ *then* *1*::ℝ *else* (*0*::ℝ)) ∗ *P* (*a*, v_0) / *real* (*card A*)))
let *?lhs* = *P* (*a*, put_x *a xa*) / *?lhs-1*

let *?rhs-1* = ($\sum$ *v*::′*b*∈*A*.
(*if* put_x *a xa* = put_x *a v* *then* *1*::ℝ *else* (*0*::ℝ)) ∗ *P* (*a*, put_x (put_x *a xa*) *v*))
let *?rhs-2* = ($\sum$ *v*::′*b*∈*A*. *P* (*a*, put_x (put_x *a xa*) *v*))

let *?rhs* = *?rhs-1* / *?rhs-2*

have *finite* $\{put_x\ a\ xa \mid xa.\ xa \in A\}$
apply (*rule finite-image-set*)
using *assms*(*1*) **by** *auto*
then have *finite-states*: *finite* $\{v_0::'a.\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\}$
by (*smt* (*verit*, *del-insts*) *Collect-cong*)

have *set-eq*: $\{v_0::'a.\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\} = \{put_x\ a\ xa \mid xa.\ xa \in A\}$
by (*smt* (*verit*, *del-insts*) *Collect-cong*)

have *f1*: $(real\ (card\ A) * (\sum_\infty v_0::'a.\ (if\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
$* P\ (a,\ v_0)\ /\ real\ (card\ A))$
$)$
$= (\sum_\infty v_0::'a.\ (if\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * P\ (a,\ v_0))$
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-mult-subset-left-summable*)
apply (*rule summable-on-finite*)
using *finite-states* **apply** *blast*
by (*simp add*: *assms*(*1*))

have *denominator-1*: $(\sum_\infty v_0::'a \in \{v_0::'a.\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\}.\ P\ (a,\ v_0)) =$
$(\sum v_0::'a \in \{v_0::'a.\ \exists\, v::'b{\in}A.\ v_0 = put_x\ a\ v\}.\ P\ (a,\ v_0))$
using *finite-states infsum-finite* **by** *blast*
also have *denominator-2*: ... $= (\sum v::'b{\in}A.\ P\ (a,\ put_x\ (put_x\ a\ xa)\ v))$
apply (*simp add*: *set-eq*)
apply (*subst sum.reindex-cong*[**where** $A=\{uu::'a.\ \exists\, xa::'b.\ uu = put_x\ a\ xa \wedge xa \in A\}$ **and**
$B = A$ **and** $l = \lambda xa.\ put_x\ a\ xa$ **and** $h = \lambda v.\ P\ (a,\ put_x\ (put_x\ a\ xa)\ v)$])
apply (*meson assms*(*2*) *inj-onI vwb-lens.axioms*(*1*) *wb-lens-def weak-lens.view-determination*)
apply (*simp add*: *Setcompr-eq-image*)
apply (*simp add*: *assms*(*2*))
by *blast*

have *numerator-1*: *?rhs-1*
$= (\sum v::'b{\in}A.\ (if\ xa = v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * P\ (a,\ put_x\ (put_x\ a\ xa)\ v))$
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*2*) *mwb-lens.axioms*(*1*) *sum.cong vwb-lens.axioms*(*2*)
weak-lens.view-determination)
have *numerator-2*: ... =
$(\sum v::'b{\in}\{xa\} \cup (A - \{xa\}).\ (if\ xa = v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * P\ (a,\ put_x\ (put_x\ a\ xa)\ v))$
using *a1 insert-Diff* **by** *force*
have *numerator-3*: ... $= (\sum v::'b{\in}\{xa\}.\ P\ (a,\ put_x\ (put_x\ a\ xa)\ v))$
apply (*subst sum-Un*[**where** $A = \{xa\}$ **and** $B = A - \{xa\}$ **and**
$f = \lambda v::'b.\ (if\ xa = v\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * P\ (a,\ put_x\ (put_x\ a\ xa)\ v)$])
apply *simp*
using *assms*(*1*) **apply** *blast*
using *sum.not-neutral-contains-not-neutral* **by** *fastforce*
have *numerator-4*: ... $= P\ (a,\ put_x\ a\ xa)$
by (*simp add*: *assms*(*2*))
show *?lhs* = *?rhs*
apply (*simp add*: *f1*)
apply (*subst infsum-mult-subset-left*)
using *denominator-1 denominator-2 numerator-1 numerator-2 numerator-3 numerator-4* **by** *presburger*
qed

term $([\ x^{>} \rightsquigarrow «v»\ ] \dagger P)_e$
term $(\exists\, v \in A.\ ([\ x^{>} \rightsquigarrow «v»\ ] \dagger P) > 0)_e$
lemma *prfun-parallel-uniform-dist'*:
  fixes P ::('a, 'a) *rvfun*
  assumes *finite* A
  assumes *vwb-lens* x
  assumes $A \neq \{\}$
  assumes $\forall\, s.\ P\ s \geq 0$

  assumes $\forall\, s.\ \exists\, v \in A.\ P\ (s,\ put_x\ s\ v) > 0$
  shows *rvfun-of-prfun* $((x\ \boldsymbol{\mathcal{U}}\ A) \parallel P) =$
    $((\sum v{\in}«A».\ ([\![x := «v»]\!]_{\mathcal{I}e} * ([\ x^{>} \rightsquigarrow «v»\ ] \dagger P)))\ /\ (\sum\ v{\in}«A».\ ([\ x^{>} \rightsquigarrow «v»\ ] \dagger P)))_e$
  apply (*subst prfun-parallel-uniform-dist*)
  apply (*simp add*: *assms*)+
  apply (*subst rvfun-inverse*)
  apply (*expr-auto add*: *dist-defs rel*)
  apply (*simp add*: *assms*(*4*) *sum-nonneg*)
  apply (*smt* (*verit*, *ccfv-SIG*) *assms*(*4*) *divide-le-eq-1 mult-cancel-right1 mult-not-zero sum-mono sum-nonneg*)
  by (*simp*)

5.7 Chains

For the *increasing-chain* and *decreasing-chain*, similar definitions *incseq* and *decseq* exist. Other useful theorems for those definitions include $(\lambda n.\ ?k) \longrightarrow ?l = (?k = ?l)$, $incseq\ ?X \Longrightarrow ?X \longrightarrow \bigsqcup\ range\ ?X$, and more.

5.7.1 Increasing chains

theorem *increasing-chain-mono*:
  assumes *increasing-chain* f
  assumes $m \leq n$
  shows $f\ m \leq f\ n$
  using *assms*(*1*) *assms*(*2*) *increasing-chain-def* **by** *blast*

theorem *increasing-chain-sup-eq-f0-constant*:
  assumes *increasing-chain* f
  assumes $(\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) = f\ 0\ (s,\ s')$
  shows $\forall\, n.\ f\ n\ (s,\ s') = f\ 0\ (s,\ s')$
proof (*rule ccontr*)
  assume $\neg\ (\forall\, n{::}\mathbb{N}.\ f\ n\ (s,\ s') = f\ (0{::}\mathbb{N})\ (s,\ s'))$
  then have $\exists\, n.\ f\ n\ (s,\ s') \neq f\ 0\ (s,\ s')$
    by *blast*
  then have $\exists\, n.\ f\ n\ (s,\ s') > f\ 0\ (s,\ s')$
    using *increasing-chain-mono* **by** (*metis assms*(*1*) *le-funE less-eq-nat.simps*(*1*) *nless-le*)
  then have $(\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) > f\ 0\ (s,\ s')$
    by (*metis SUP-lessD UNIV-I assms*(*2*) *nless-le*)
  then show *False*
    by (*simp add*: *assms*(*2*))
qed

lemma *increasing-chain-sup-subset-eq*:
  assumes *increasing-chain* f
  shows $(\bigsqcup n{::}\mathbb{N}.\ f\ (n + m)) = (\bigsqcup n{::}\mathbb{N}.\ f\ n)$
proof −
  have *f1*: $(\bigsqcup n{::}nat.\ f\ (n + m)) = (\bigsqcup n{\in}\{m..\}.\ f\ n)$

apply (*simp add*: *image-def*)
by (*metis* (*no-types*, *lifting*) *add.commute add.right-neutral atLeast-0 atLeast-iff image-add-atLeast le-add-same-cancel2 rangeE zero-le*)
have *f2*: $\{..m-1\} \cup \{(m::nat)..\} = UNIV$
by (*metis Suc-pred′ Un-UNIV-right atLeast0LessThan atLeast-0 bot-nat-0.not-eq-extremum ivl-disj-un*(*14*) *lessThan-Suc-atMost zero-order*(*1*))
then have *f3*: $(\bigsqcup n::nat.\ f\ n) = (\bigsqcup n::nat \in \{..m-1\} \cup \{(m::nat)..\}.\ f\ n)$
by (*simp add*: *image-def*)
have *f4*: $(\bigsqcup n::nat \in \{..m-1\} \cup \{(m::nat)..\}.\ f\ n) = (\bigsqcup n{\in}\{..m-1\}.\ f\ n) \sqcup (\bigsqcup n{\in}\{m..\}.\ f\ n)$
apply (*subst SUP-union*)
by *blast*
have *f5*: $(\bigsqcup n{\in}\{..m-1\}.\ f\ n) \leq (\bigsqcup n{\in}\{m..\}.\ f\ n)$
apply (*subst SUP-le-iff*)
by (*smt* (*verit*) *SUP-upper2 assms atLeast-iff increasing-chain-mono le-cases3*)
then have *f6*: $(\bigsqcup n{\in}\{..m-1\}.\ f\ n) \sqcup (\bigsqcup n{\in}\{m..\}.\ f\ n) = (\bigsqcup n{\in}\{m..\}.\ f\ n)$
apply (*subst* (*asm*) *le-iff-sup*)
by *blast*
show *?thesis*
using *f1 f3 f4 f6* **by** *presburger*
qed

lemma *increasing-chain-limit-exists-element*:
fixes $f :: nat \Rightarrow ('s_1,\ 's_2)\ prfun$
assumes *increasing-chain f*
assumes $\exists\, n.\ f\ n\ (s,\ s') > 0$
shows $\forall\, e > 0.\ \exists\, m.\ f\ m\ (s,\ s') > (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - e$
apply (*rule ccontr*)
apply (*auto*)
proof −
fix *e*
assume *pos*: $(0::ureal) < e$
assume *a1*: $\forall\, m::\mathbb{N}.\ \neg\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - e < f\ m\ (s,\ s')$

from *a1* **have** $\forall\, m::\mathbb{N}.\ f\ m\ (s,\ s') \leq (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - e$
using *linorder-not-less* **by** *blast*
then have *sup-least*: $(\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) \leq (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - e$
using *SUP-least* **by** *metis*
have $(\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) \geq 0$
using *less-eq-ureal.rep-eq ureal2ereal zero-ureal.rep-eq* **by** *fastforce*
then have $(\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) > 0$
using *assms*(*2*) **by** (*metis Sup-upper linorder-not-le nle-le range-eqI*)
then show *False*
using *pos sup-least* **by** (*meson linorder-not-le ureal-minus-less*)
qed

This lemma represents limit in a complete lattice ereal. So (0 - e) is not equal to 0 as in ureal

theorem *increasing-chain-limit-is-lub*:
fixes $f :: nat \Rightarrow ('s_1,\ 's_2)\ prfun$
assumes *increasing-chain f*

shows $(\lambda n.\ ureal2real\ (f\ n\ (s,\ s'))) \longrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')))$
proof (*cases* $\exists\, n.\ f\ n\ (s,\ s') > 0$)
case *True*
show *?thesis*
apply (*subst LIMSEQ-iff*)

```
  apply (auto)
  proof -
    fix r
    assume a1: (0::ℝ) < r
    have sup-upper: ∀ n. ureal2real (f n (s, s')) - ureal2real (⨆ n::ℕ. f n (s, s')) ≤ 0
      apply (auto)
      apply (rule ureal2real-mono)
      by (meson SUP-upper UNIV-I)
    then have dist-equal: ∀ n. |ureal2real (f n (s, s')) - ureal2real (⨆ n::ℕ. f n (s, s'))| =
        ureal2real (⨆ n::ℕ. f n (s, s')) - ureal2real (f n (s, s'))
      by auto
    from a1 have r-gt-0: real2ureal r > 0
      by (rule ureal-gt-zero)
    obtain m where P-m: f m (s, s') > (⨆ n::ℕ. f n (s, s')) - real2ureal r
      using r-gt-0 by (metis assms(1) True increasing-chain-limit-exists-element)
    have ∃ no::ℕ. ∀ n≥no. ureal2real (⨆ n::ℕ. f n (s, s')) - ureal2real (f n (s, s')) < r
      apply (rule-tac x = m in exI)
      apply (auto)
    proof -
      fix n
      assume a2: m ≤ n
      then have f m (s, s') ≤ f n (s, s')
        by (metis assms(1) increasing-chain-mono le-fun-def)
      then have (⨆ n::ℕ. f n (s, s')) - real2ureal r < f n (s, s')
        using P-m by force
      then have (⨆ n::ℕ. f n (s, s')) - (f n (s, s')) <
          (⨆ n::ℕ. f n (s, s')) - ((⨆ n::ℕ. f n (s, s')) - real2ureal r)
        apply (rule ureal-minus-larger-less)
        by (meson SUP-upper UNIV-I)
      also have ... ≤ real2ureal r
        by (metis nle-le ureal-minus-larger-zero-unit ureal-minus-less-diff)
      then have (⨆ n::ℕ. f n (s, s')) - (f n (s, s')) < real2ureal r
        using calculation by auto
      then have ureal2real ((⨆ n::ℕ. f n (s, s')) - (f n (s, s'))) < ureal2real (real2ureal r)
        using ureal2real-mono-strict by blast
      then have ureal2real (⨆ n::ℕ. f n (s, s')) - ureal2real (f n (s, s')) < ureal2real (real2ureal r)
        by (smt (verit, ccfv-threshold) ureal-minus-larger-than-real-minus)
      then show ureal2real (⨆ n::ℕ. f n (s, s')) - ureal2real (f n (s, s')) < r
        by (meson a1 less-eq-real-def order-less-le-trans ureal-real2ureal-smaller)
    qed
    then show ∃ no::ℕ. ∀ n≥no. |ureal2real (f n (s, s')) - ureal2real (⨆ n::ℕ. f n (s, s'))| < r
        using dist-equal by presburger
  qed
next
  case False
  then show ?thesis
    by (smt (verit, best) SUP-least bot.extremum bot-ureal.rep-eq eventually-sequentially
        linorder-not-le nle-le tendsto-def ureal2ereal-inverse zero-ureal.rep-eq)
qed

theorem increasing-chain-limit-is-lub':
  fixes f :: nat ⇒ ('s₁, 's₂) prfun
  assumes increasing-chain f
  shows ∀ s s'. (λn. ureal2real (f n (s, s'))) ⟶ (ureal2real (⨆ n::ℕ. f n (s, s')))
  apply (auto)
```

by (*simp add*: *assms increasing-chain-limit-is-lub*)

lemma *Inter-atLeast-not-empty-finite*:
assumes $A \neq \{\}$
assumes *finite A*
shows $\exists n. \forall m \in A. n \in (\lambda m. \{n::nat. n \geq m\})\ m$
using *assms*(*2*) *finite-nat-set-iff-bounded-le* **by** *auto*

lemma *Inter-atLeast-not-empty-finite*′:
assumes $A \neq \{\}$
assumes *finite A*
shows $\exists n. \forall m \in A. n \in \{(m::nat)..\}$
using *assms*(*2*) *finite-nat-set-iff-bounded-le* **by** *auto*

lemma *max-bounded-e*:
assumes $m \in A\ A \neq \{\}$ *finite A Max A* $\leq n$
shows $m \leq n$
by (*meson Max.boundedE assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*))

theorem *increasing-chain-limit-is-lub-all*:
fixes $f :: nat \Rightarrow ('s_1, 's_2)\ prfun$
assumes *increasing-chain f*

assumes $\mathcal{FS}$ *f*
shows $\forall r > 0::real. \exists no::nat. \forall n \geq no.$
$\forall s\ s'.\ ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) - ureal2real\ (f\ n\ (s, s')) < r$
apply (*auto*)
proof −
fix *r*::*real*
assume *a1*: $0 < r$
have *sup-upper*: $\forall s\ s'. \forall n.\ ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) \leq 0$
apply (*auto*)
apply (*rule ureal2real-mono*)
by (*meson SUP-upper UNIV-I*)
then have *dist-equal*: $\forall s\ s'. \forall n.\ |ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s'))| =$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) - ureal2real\ (f\ n\ (s, s'))$
by *auto*
have *limit-is-lub*: $\forall s\ s'.\ (\lambda n.\ ureal2real\ (f\ n\ (s, s'))) \longrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')))$
by (*simp add*: *assms*(*1*) *increasing-chain-limit-is-lub*)
then have *limit-is-lub-def*: $\forall s\ s'.\ (\exists no::\mathbb{N}. \forall n{\geq}no.\ norm\ (ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s'))) < r)$
using *LIMSEQ-iff* **by** (*metis a1*)
then have *limit-is-lub-def*′: $\forall s\ s'. \exists no::nat. \forall n \geq no.\ ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) - ureal2real\ (f\ n\ (s, s')) < r$
by (*simp add*: *dist-equal*)

— The supreme of *f* is larger than its initial value *f 0* and the difference is at least *r*. Therefore, a unique number *no+1* must exist such that *f* (*no+1*) inside the supreme minus *r* and *f no* outside the supreme minus *r*.
let $?P\text{-}larger\text{-}sup = \lambda s\ s'.\ ((ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) > ureal2real\ (f\ 0\ (s, s'))) \wedge$
$(ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s, s')) - ureal2real\ (f\ 0\ (s, s'))) \geq r)$

let $?P\text{-}mu\text{-}no = \lambda s\ s'.\ \lambda no.\ (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (no{+}1)\ (s,\ s')) < r \wedge$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s')) \geq r)$

— The uniqueness is proved.

have *f-larger-supreme-unique-no*:
$\forall s\ s'.\ ?P\text{-}larger\text{-}sup\ s\ s' \longrightarrow (\exists!no::nat.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)$
apply (*auto*)
defer
apply (*smt* (*verit*, *best*) *assms*(*1*) *increasing-chain-mono le-fun-def nle-le not-less-eq-eq ureal2real-mono*)
proof −
fix $s\ s'$
assume *a11*: $ureal2real\ (f\ (0::\mathbb{N})\ (s,\ s')) < ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s'))$
assume *a12*: $r \leq ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (0::\mathbb{N})\ (s,\ s'))$
show $\exists\ no::\mathbb{N}.$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (Suc\ no)\ (s,\ s')) < r \wedge$
$r \leq ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s'))$
apply (*rule ccontr*, *auto*)
proof −
assume *a110*: $\forall\ no::\mathbb{N}.$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (Suc\ no)\ (s,\ s')) < r \longrightarrow$
$\neg\ r \leq ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s'))$
then have *f110*: $\forall\ no::\mathbb{N}.$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (Suc\ no)\ (s,\ s')) < r \longrightarrow$
$ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s')) < r$
by *auto*
have *f111*: $\exists\ no::nat.\ ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s')) < r$
using *limit-is-lub-def'* **by** *blast*
obtain *no* **where** *P-no*: $ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s')) < r$
using *f111* **by** *blast*
have $\forall\ m::nat.\ ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (no - m)\ (s,\ s')) < r$
apply (*auto*)
apply (*induct-tac m*)
using *P-no minus-nat.diff-0* **apply** *presburger*
by (*smt* (*verit*, *best*) *Suc-diff-Suc a12 bot-nat-0.extremum f110 linorder-not-less nless-le zero-less-diff*)
then have $ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (no - no)\ (s,\ s')) < r$
by *blast*
then show *False*
using *a12* **by** *force*
qed
qed

— If $f\ n$ is constant or $f\ 0$ is inside the supreme minus r, then for any number, the distance between $f\ n$ and the supreme is less than r.

have *f-const-or-larger-dist-universal*: $\forall\ s\ s'.$
$((ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) = ureal2real\ (f\ 0\ (s,\ s'))) \vee$
$(ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ 0\ (s,\ s'))) < r)$
$\longrightarrow$
$(\forall\ no.\ ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ no\ (s,\ s')) < r)$
apply (*auto*)
apply (*smt* (*verit*) *SUP-cong a1 assms*(*1*) *increasing-chain-sup-eq-f0-constant ureal2real-eq*)
by (*smt* (*verit*, *best*) *assms*(*1*) *bot-nat-0.extremum increasing-chain-mono le-fun-def ureal2real-mono*)

let $?mu\text{-}no\text{-}set = \{THE\ no.\ ?P\text{-}mu\text{-}no\ s\ s'\ no \mid s\ s'.\ ?P\text{-}larger\text{-}sup\ s\ s'\}$

— We use another form *?mu-no-set1* in order to prove it is finite more conveniently using $finite\ \{y.\ ?P\ y\} \Longrightarrow finite\ \{x.\ \exists\, y.\ ?P\ y \wedge\ ?Q\ x\ y\} = (\forall\, y.\ ?P\ y \longrightarrow finite\ \{x.\ ?Q\ x\ y\})$

let $?mu\text{-}no\text{-}set1 = \{THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no \mid s.\ ?P\text{-}larger\text{-}sup\ (fst\ s)\ (snd\ s)\}$

have *mu-no-set-eq*: $?mu\text{-}no\text{-}set = ?mu\text{-}no\text{-}set1$
by *auto*

— A *no* is obtained as the maximum number of unique numbers for all states, and so for any number $n \geq no$, the distance between $f\ n$ and the supreme is less than r for any state.

obtain *no* **where** *P-no*:
$no = (if\ ?mu\text{-}no\text{-}set = \{\}\ then\ 0\ else\ (Max\ ?mu\text{-}no\text{-}set + 1))$
by *blast*

have *mu-no-set-rewrite*: $?mu\text{-}no\text{-}set = (\bigcup (s,\ s') \in \{(s,\ s').\ ?P\text{-}larger\text{-}sup\ s\ s'\}.$
$\{uu.\ uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)\})$
by *auto*

have $(\forall\, s\ s'.\ ?P\text{-}larger\text{-}sup\ s\ s' \longrightarrow finite\ \{uu.\ uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)\})$
by *simp*

have *mu-no-set1-finite-iff*: $(finite\ ?mu\text{-}no\text{-}set1) \longleftrightarrow (\forall\, s.\ ?P\text{-}larger\text{-}sup\ (fst\ s)\ (snd\ s) \longrightarrow$
$finite\ \{uu.\ uu = (THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no)\})$
proof −
have $?mu\text{-}no\text{-}set1 = (\bigcup s{\in}\{s.\ ?P\text{-}larger\text{-}sup\ (fst\ s)\ (snd\ s)\}.$
$\{uu.\ uu = (THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no)\})$
by *auto*
with *assms*(*2*) **show** *?thesis*
by *simp*
qed

then have *mu-no-set1-finite*: *finite ?mu-no-set1*
by *auto*

show $\exists\, no{::}\mathbb{N}.\ \forall\, n{\geq}no.\ \forall\,(s{::}'s_1)\ s'{::}'s_2.\ ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ n\ (s,\ s')) < r$
apply (*rule-tac x = no* **in** *exI*)
apply (*auto*)
apply (*simp add*: *P-no*)
proof −
fix $n\ s\ s'$
assume *a11*: $(if\ \forall\,(s{::}'s_1)\ s'{::}'s_2.$
$ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s')) < ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) \longrightarrow$
$\neg\ r \leq ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s'))$
$then\ 0{::}\mathbb{N}$
$else\ Max\ \{uu{::}\mathbb{N}.\ \exists\,(s{::}'s_1)\ s'{::}'s_2.$
$uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)\ \wedge$
$?P\text{-}larger\text{-}sup\ s\ s'\} + 1)$
$\leq n$

show $ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ n\ (s,\ s')) < r$
proof $(cases\ ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) = ureal2real\ (f\ 0\ (s,\ s')) \vee$
$\neg\ r \leq ureal2real\ (\bigsqcup n{::}\mathbb{N}.\ f\ n\ (s,\ s')) - ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s')))$
case *True*
then have $n \geq 0$
by *blast*
then show *?thesis*

using *True f-const-or-larger-dist-universal* **by** *fastforce*
next
case *False*
then have *max-leq-n*: (*Max* {*uu*::$\mathbb{N}$. $\exists$(*s*::$'s_1$) *s'*::$'s_2$.
uu = (*THE no*::$\mathbb{N}$. *?P-mu-no s s' no*) $\wedge$ *?P-larger-sup s s'*} + *1*) $\leq$ *n*
by (*smt* (*verit*, *ccfv-SIG*) *SUP-cong a1 a11*)
then have *mu-no-in*: (*THE no*::$\mathbb{N}$. *?P-mu-no s s' no*) $\in$ *?mu-no-set*
apply (*subst mem-Collect-eq*)
using *False a1* **by** *fastforce*
have *mu-no-le-n*: (*THE no*::$\mathbb{N}$. *?P-mu-no s s' no*) $\leq$ *n* − *1*
apply (*rule max-bounded-e*[**where** *A* = *?mu-no-set*])
using *mu-no-in* **apply** *blast*
using *mu-no-in* **apply** *blast*
using *mu-no-set1-finite mu-no-set-eq* **apply** *presburger*
using *max-leq-n* **by** (*meson Nat.le-diff-conv2 add-leE*)
have *P-mu-no*: *?P-mu-no s s'* (*THE no*::$\mathbb{N}$. *?P-mu-no s s' no*)
apply (*rule theI'*)
by (*smt* (*verit*, *best*) *False Sup.SUP-cong f-larger-supreme-unique-no sup-upper*)
have *ureal2real* (*f* ((*THE no*::$\mathbb{N}$. *?P-mu-no s s' no*) + (*1*::$\mathbb{N}$)) (*s*, *s'*)) $\leq$ *ureal2real* (*f n* (*s*,*s'*))
using *mu-no-le-n* **by** (*metis* (*mono-tags*, *lifting*) *Nat.le-diff-conv2 add-leE assms*(*1*) *increasing-chain-mono le-fun-def max-leq-n ureal2real-mono*)
then show *?thesis*
using *P-mu-no* **by** *linarith*
qed
qed
qed

lemma *increasing-chain-fun*:
assumes *increasing-chain f*
shows *increasing-chain* (λ*n*. *f n s*)
by (*metis* (*mono-tags*, *lifting*) *assms increasing-chain-def le-funE*)

5.7.2 Decreasing chains

theorem *decreasing-chain-antitone*:
assumes *decreasing-chain f*
assumes *m* $\leq$ *n*
shows *f m* $\geq$ *f n*
using *assms*(*1*) *assms*(*2*) *decreasing-chain-def* **by** *blast*

theorem *decreasing-chain-inf-eq-f0-constant*:
assumes *decreasing-chain f*
assumes ($\prod$*n*::$\mathbb{N}$. *f n* (*s*, *s'*)) = *f 0* (*s*, *s'*)
shows $\forall$*n*. *f n* (*s*, *s'*) = *f 0* (*s*, *s'*)
proof (*rule ccontr*)
assume $\neg$ ($\forall$*n*::$\mathbb{N}$. *f n* (*s*, *s'*) = *f* (*0*::$\mathbb{N}$) (*s*, *s'*))
then have $\exists$*n*. *f n* (*s*, *s'*) $\neq$ *f 0* (*s*, *s'*)
by *blast*
then have $\exists$*n*. *f n* (*s*, *s'*) < *f 0* (*s*, *s'*)
using *decreasing-chain-antitone*
by (*metis assms*(*1*) *le-funE less-eq-nat.simps*(*1*) *order-neq-le-trans*)
then have ($\prod$*n*::$\mathbb{N}$. *f n* (*s*, *s'*)) < *f 0* (*s*, *s'*)
by (*metis INF-lower assms*(*2*) *iso-tuple-UNIV-I less-le-not-le*)
then show *False*
by (*simp add*: *assms*(*2*))
qed

lemma *decreasing-chain-inf-subset-eq*:
  assumes *decreasing-chain f*
  shows $(\bigsqcap n::\mathbb{N}.\ f\ (n + m)) = (\bigsqcap n::\mathbb{N}.\ f\ n)$
proof −
  have *f1*: $(\bigsqcap n::nat.\ f\ (n + m)) = (\bigsqcap n\in\{m..\}.\ f\ n)$
    apply (*simp add*: *image-def*)
    by (*metis* (*no-types*, *lifting*) *add.commute add.right-neutral atLeast-0 atLeast-iff image-add-atLeast le-add-same-cancel2 rangeE zero-le*)
  have *f2*: $\{..m-1\} \cup \{(m::nat)..\} = UNIV$
    by (*metis Suc-pred′ atLeast0LessThan atLeast-0 bot-nat-0.extremum bot-nat-0.not-eq-extremum ivl-disj-un*(*14*) *lessThan-Suc-atMost sup-commute sup-top-left*)
  then have *f3*: $(\bigsqcap n::nat.\ f\ n) = (\bigsqcap n::nat \in \{..m-1\} \cup \{(m::nat)..\}.\ f\ n)$
    by (*simp add*: *image-def*)
  have *f4*: $(\bigsqcap n::nat \in \{..m-1\} \cup \{(m::nat)..\}.\ f\ n) = (\bigsqcap n\in\{..m-1\}.\ f\ n) \sqcap (\bigsqcap n\in\{m..\}.\ f\ n)$
    apply (*subst INF-union*)
    by *blast*
  have *f5*: $(\bigsqcap n\in\{m..\}.\ f\ n) \le (\bigsqcap n\in\{..m-1\}.\ f\ n)$
    apply (*rule INF-greatest*)
    by (*metis INF-lower add.commute assms atLeast-iff bot-nat-0.extremum decreasing-chain-antitone le-add-same-cancel2 order-trans*)
  then have *f6*: $(\bigsqcap n\in\{..m-1\}.\ f\ n) \sqcap (\bigsqcap n\in\{m..\}.\ f\ n) = (\bigsqcap n\in\{m..\}.\ f\ n)$
    apply (*subst* (*asm*) *le-iff-inf*)
    by (*simp add*: *inf-commute*)
  show *?thesis*
    using *f1 f3 f4 f6* **by** *presburger*
qed

lemma *decreasing-chain-limit-exists-element*:
  fixes $f :: nat \Rightarrow ('s_1,\ 's_2)\ prfun$
  assumes *decreasing-chain f*
  assumes $\exists\, n.\ f\ n\ (s,\ s') < 1$
  shows $\forall\ e > 0.\ \exists\, m.\ f\ m\ (s,\ s') < (\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) + e$
  apply (*rule ccontr*)
  apply (*auto*)
proof −
  fix *e*
  assume *pos*: $(0::ureal) < e$
  assume *a1*: $\forall\, m::\mathbb{N}.\ \neg\ f\ m\ (s,\ s') < (\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) + e$

  from *a1* **have** $\forall\, m::\mathbb{N}.\ f\ m\ (s,\ s') \ge (\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) + e$
    by (*meson linorder-not-le*)
  then have *inf-greatest*: $(\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) + e \le (\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s'))$
    using *INF-greatest* **by** *metis*
  have $(\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) \le 1$
    by (*metis one-ureal.rep-eq top-greatest top-ureal.rep-eq ureal2ereal-inject*)
  then have $(\bigsqcap n::\mathbb{N}.\ f\ n\ (s,\ s')) < 1$
    using *assms*(*2*) **by** (*metis INF-lower UNIV-I linorder-not-less order-le-less*)
  then show *False*
    using *pos inf-greatest* **by** (*meson linorder-not-le ureal-plus-greater*)
qed

theorem *decreasing-chain-limit-is-glb*:
  fixes $f :: nat \Rightarrow ('s_1,\ 's_2)\ prfun$
  assumes *decreasing-chain f*

shows $(\lambda n.\ \mathit{ureal2real}\ (f\ n\ (s,\ s'))) \longlonglongrightarrow (\mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')))$
proof $(\mathit{cases}\ \exists n.\ f\ n\ (s,\ s') < \mathit{1})$
case *True*
show *?thesis*
apply (*subst LIMSEQ-iff*)
apply (*auto*)
proof −
fix r
assume *a1*: $(\mathit{0}{::}\mathbb{R}) < r$
have *sup-upper*: $\forall n.\ \mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) \geq \mathit{0}$
apply (*auto*)
apply (*rule ureal2real-mono*)
by (*meson INF-lower UNIV-I*)
then have *dist-equal*: $\forall n.\ |\mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))| =$
$\mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))$
by *auto*
from *a1* **have** *r-gt-0*: $\mathit{real2ureal}\ r > \mathit{0}$
by (*rule ureal-gt-zero*)
obtain m **where** *P-m*: $f\ m\ (s,\ s') < (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) + \mathit{real2ureal}\ r$
using *r-gt-0* **by** (*metis assms*(*1*) *True decreasing-chain-limit-exists-element*)
have $\exists no{::}\mathbb{N}.\ \forall n{\geq}no.\ \ \mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
apply (*rule-tac* $x = m$ **in** *exI*)
apply (*auto*)
proof −
fix n
assume *a2*: $m \leq n$
then have $f\ m\ (s,\ s') \geq f\ n\ (s,\ s')$
by (*metis assms*(*1*) *decreasing-chain-antitone le-fun-def*)
then have $f\ n\ (s,\ s') < (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) + \mathit{real2ureal}\ r$
using *P-m* **by** *force*
then have $(f\ n\ (s,\ s')) - (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) <$
$((\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) + \mathit{real2ureal}\ r) - (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))$
apply (*subst ureal-larger-minus-greater*)
apply (*meson INF-lower UNIV-I*)
apply *meson*
by *simp*
also have ... $\leq \mathit{real2ureal}\ r$
by (*metis linorder-not-le nle-le ureal-plus-eq-1-minus-less ureal-plus-less-1-unit*)
then have $(f\ n\ (s,\ s')) - (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < \mathit{real2ureal}\ r$
using *calculation* **by** *auto*
then have $\mathit{ureal2real}\ ((f\ n\ (s,\ s')) - (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))) < \mathit{ureal2real}\ (\mathit{real2ureal}\ r)$
by (*rule ureal2real-mono-strict*)
then have $\mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < \mathit{ureal2real}\ (\mathit{real2ureal}\ r)$
by (*smt* (*verit*, *ccfv-threshold*) *ureal-minus-larger-than-real-minus*)
then show $\mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
by (*meson a1 less-eq-real-def order-less-le-trans ureal-real2ureal-smaller*)
qed
then show $\exists no{::}\mathbb{N}.\ \forall n{\geq}no.\ |\mathit{ureal2real}\ (f\ n\ (s,\ s')) - \mathit{ureal2real}\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))| < r$
using *dist-equal* **by** *presburger*
qed
next
case *False*
then have $\forall n{::}\mathbb{N}.\ f\ n\ (s{::}'s_1,\ s'{::}'s_2) = (\mathit{1}{::}\mathit{ureal})$
by (*metis antisym-conv2 one-ureal.rep-eq top-greatest top-ureal.rep-eq ureal2ereal-inject*)
then show *?thesis*

by *force*
qed

theorem *decreasing-chain-limit-is-glb-all*:
fixes $f :: nat \Rightarrow ('s_1, 's_2)\ prfun$
assumes *decreasing-chain f*
assumes $\mathcal{FS}\ f$
shows $\forall r > 0::real.\ \exists no::nat.\ \forall n \geq no.$
$\forall s\ s'.\ ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r$
apply (*auto*)
proof −
fix $r::real$
assume *a1*: $0 < r$
have *sup-upper*: $\forall s\ s'.\ \forall n.\ ureal2real\ (f\ n\ (s, s')) \geq ureal2real\ (\bigsqcap v::\mathbb{N}.\ f\ n\ (s, s'))$
by (*auto*)
then have *dist-equal*: $\forall s\ s'.\ \forall n.\ |ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))| =$
$ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))$
by (*simp add*: *Inf-lower ureal2real-mono*)
have *limit-is-glb*: $\forall s\ s'.\ (\lambda n.\ ureal2real\ (f\ n\ (s, s'))) \longrightarrow (ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')))$
by (*simp add*: *assms decreasing-chain-limit-is-glb*)
then have *limit-is-glb-def*: $\forall s\ s'.\ (\exists no::\mathbb{N}.\ \forall n \geq no.\ norm\ (ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))) < r)$
using *LIMSEQ-iff* **by** (*metis a1*)
then have *limit-is-glb-def'*: $\forall s\ s'.\ \exists no::nat.\ \forall n \geq no.\ ureal2real\ (f\ n\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r$
by (*simp add*: *dist-equal*)

— The infimum of f is less than its initial value $f\ 0$ and the difference is at least r. Therefore, a unique number $no+1$ must exist such that $f\ (no+1)$ inside the supreme minus r and $f\ no$ outside the supreme minus r.
let $?P\text{-}less\text{-}inf = \lambda s\ s'.\ ((ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < ureal2real\ (f\ 0\ (s, s'))) \wedge$
$(ureal2real\ (f\ 0\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))) \geq r)$
let $?P\text{-}mu\text{-}no = \lambda s\ s'.\ \lambda no.\ (ureal2real\ (f\ (no+1)\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r \wedge$
$ureal2real\ (f\ no\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) \geq r)$
— The uniqueness is proved.
have *f-larger-supreme-unique-no*:
$\forall s\ s'.\ ?P\text{-}less\text{-}inf\ s\ s' \longrightarrow (\exists! no::nat.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)$
apply (*auto*)
defer
apply (*smt* (*verit*, *best*) *assms*(*1*) *decreasing-chain-antitone le-fun-def nle-le not-less-eq-eq ureal2real-mono*)
proof −
fix $s\ s'$
assume *a11*: $ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < ureal2real\ (f\ (0::\mathbb{N})\ (s, s'))$
assume *a12*: $r \leq ureal2real\ (f\ (0::\mathbb{N})\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))$
show $\exists no::\mathbb{N}.$
$ureal2real\ (f\ (Suc\ no)\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r \wedge$
$r \leq ureal2real\ (f\ no\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))$
apply (*rule ccontr*, *auto*)
proof −
assume *a110*: $\forall no::\mathbb{N}.$
$ureal2real\ (f\ (Suc\ no)\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r \longrightarrow$
$\neg\ r \leq ureal2real\ (f\ no\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s'))$
then have *f110*: $\forall no::\mathbb{N}.$
$ureal2real\ (f\ (Suc\ no)\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r \longrightarrow$
$ureal2real\ (f\ no\ (s, s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ f\ n\ (s, s')) < r$

by *auto*
have *f111*: $\exists\, no{::}nat.\ ureal2real\ (f\ no\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
using *limit-is-glb-def′* **by** *blast*
obtain *no* **where** *P-no*: $ureal2real\ (f\ no\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
using *f111* **by** *blast*
have $\forall\, m{::}nat.\ ureal2real\ (f\ (no - m)\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
apply (*auto*)
apply (*induct-tac m*)
using *P-no* **apply** *simp*
by (*metis Suc-diff-Suc a12 diff-is-0-eq f110 linorder-not-less*)
then have $ureal2real\ (f\ (no - no)\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
by *blast*
then show *False*
using *a12* **by** *simp*
qed
qed

— If $f\ n$ is constant or $f\ 0$ is inside the infimum minus r, then for any number, the distance between $f\ n$ and the infimum is less than r.
have *f-const-or-larger-dist-universal*: $\forall\, s\ s'.$
$((ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) = ureal2real\ (f\ 0\ (s,\ s'))) \lor$
$(ureal2real\ (f\ 0\ (s,\ s'))) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r)$
$\longrightarrow$
$(\forall\, no.\ (ureal2real\ (f\ no\ (s,\ s'))) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r)$
apply (*auto*)
apply (*smt* (*verit*, *ccfv-threshold*) *Sup.SUP-cong a1 assms*(*1*) *decreasing-chain-inf-eq-f0-constant ureal2real-eq*)
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *decreasing-chain-antitone le-fun-def less-eq-nat.simps*(*1*) *ureal2real-mono*)

let $?mu\text{-}no\text{-}set = \{THE\ no.\ ?P\text{-}mu\text{-}no\ s\ s'\ no \mid s\ s'.\ ?P\text{-}less\text{-}inf\ s\ s'\}$
— We use another form *?mu-no-set1* in order to prove it is finite more conveniently using $finite\ \{y.\ ?P\ y\} \Longrightarrow finite\ \{x.\ \exists\, y.\ ?P\ y \land ?Q\ x\ y\} = (\forall\, y.\ ?P\ y \longrightarrow finite\ \{x.\ ?Q\ x\ y\})$
let $?mu\text{-}no\text{-}set1 = \{THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no \mid s.\ ?P\text{-}less\text{-}inf\ (fst\ s)\ (snd\ s)\}$

have *mu-no-set-eq*: $?mu\text{-}no\text{-}set = ?mu\text{-}no\text{-}set1$
by *auto*

— A *no* is obtained as the maximum number of unique numbers for all states, and so for any number $n \geq no$, the distance between $f\ n$ and the supreme is less than r for any state.
obtain *no* **where** *P-no*:
$no = (if\ ?mu\text{-}no\text{-}set = \{\}\ then\ 0\ else\ (Max\ ?mu\text{-}no\text{-}set + 1))$
by *blast*

have *mu-no-set-rewrite*: $?mu\text{-}no\text{-}set = (\bigcup (s,\ s') \in \{(s,\ s').\ ?P\text{-}less\text{-}inf\ s\ s'\}.$
$\{uu.\ uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)\})$
by *auto*

have *f-less-inf-finite*: $finite\ \{(s,\ s').\ ?P\text{-}less\text{-}inf\ s\ s'\}$
proof −
have $\{(s,\ s').\ ?P\text{-}less\text{-}inf\ s\ s'\} \subseteq \{s.\ ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ s) < ureal2real\ (f\ 0\ s)\}$
by *blast*
then show *?thesis*
using *assms*(*2*) *rev-finite-subset* **by** *blast*
qed

have $(\forall s\ s'.\ ?P\text{-}less\text{-}inf\ s\ s' \longrightarrow finite\ \{uu.\ uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)\})$
  by *simp*

have *mu-no-set1-finite-iff*: $(finite\ ?mu\text{-}no\text{-}set1) \longleftrightarrow (\forall s.\ ?P\text{-}less\text{-}inf\ (fst\ s)\ (snd\ s) \longrightarrow$
      $finite\ \{uu.\ uu = (THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no)\})$
proof −
  have $?mu\text{-}no\text{-}set1 = (\bigcup s{\in}\{s.\ ?P\text{-}less\text{-}inf\ (fst\ s)\ (snd\ s)\}.$
          $\{uu.\ uu = (THE\ no.\ ?P\text{-}mu\text{-}no\ (fst\ s)\ (snd\ s)\ no)\})$
    by *auto*
  with *assms* **show** *?thesis*
    by *simp*
qed

then have *mu-no-set1-finite*: *finite ?mu-no-set1*
  by *auto*

show $\exists no{::}\mathbb{N}.\ \forall n{\geq}no.\ \forall (s{::}'s_1)\ s'{::}'s_2.\ ureal2real\ (f\ n\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
  apply (*rule-tac* $x = no$ **in** *exI*)
  apply (*auto*)
  apply (*simp add*: *P-no*)
proof −
  fix $n\ s\ s'$
  assume *a11*: $(if\ \forall (s{::}'s_1)\ s'{::}'s_2.$
          $ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s')) \longrightarrow$
          $\neg\ r \leq ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s'))$
      $then\ 0{::}\mathbb{N}$
      $else\ Max\ \{uu{::}\mathbb{N}.\ \exists (s{::}'s_1)\ s'{::}'s_2.$
          $uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no) \wedge$
          $?P\text{-}less\text{-}inf\ s\ s'\} + 1)$
        $\leq n$

  show $ureal2real\ (f\ n\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) < r$
  proof $(cases\ ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')) = ureal2real\ (f\ 0\ (s,\ s')) \vee$
    $\neg\ r \leq ureal2real\ (f\ (0{::}\mathbb{N})\ (s,\ s')) - ureal2real\ (\bigsqcap n{::}\mathbb{N}.\ f\ n\ (s,\ s')))$
    case *True*
    then have $n \geq 0$
      by *blast*
    then show *?thesis*
      using *True f-const-or-larger-dist-universal* **by** *fastforce*
  next
    case *False*
    then have *max-leq-n*: $(Max\ \{uu{::}\mathbb{N}.\ \exists (s{::}'s_1)\ s'{::}'s_2.$
          $uu = (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no) \wedge ?P\text{-}less\text{-}inf\ s\ s'\} + 1) \leq n$
      by (*smt* (*verit*) *Sup.SUP-cong a1 a11*)
    then have *mu-no-in*: $(THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no) \in ?mu\text{-}no\text{-}set$
      apply (*subst mem-Collect-eq*)
      using *False a1* **by** *fastforce*
    have *mu-no-le-n*: $(THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no) \leq n - 1$
      apply (*rule max-bounded-e*[**where** $A = ?mu\text{-}no\text{-}set$])
      using *mu-no-in* **apply** *blast*
      using *mu-no-in* **apply** *blast*
      using *mu-no-set1-finite mu-no-set-eq* **apply** *presburger*
      using *max-leq-n* **by** (*meson Nat.le-diff-conv2 add-leE*)
    have *P-mu-no*: $?P\text{-}mu\text{-}no\ s\ s'\ (THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no)$
      apply (*rule theI'*)

using *False a1 f-larger-supreme-unique-no* **by** *auto*
have *ureal2real* $(f\ ((THE\ no{::}\mathbb{N}.\ ?P\text{-}mu\text{-}no\ s\ s'\ no) + (1{::}\mathbb{N}))\ (s,\ s')) \geq$ *ureal2real* $(f\ n\ (s,s'))$
using *mu-no-le-n*
by (*smt* (*verit*, *best*) *Nat.le-diff-conv2 add-leD2 assms*(*1*) *decreasing-chain-antitone le-fun-def max-leq-n ureal2real-mono*)
then show *?thesis*
using *P-mu-no* **by** *linarith*
qed
qed
qed

5.8 While loop

term $\lambda X.$ (*if*$_c$ *b then* $(P\ ;\ X)$ *else II*)
term *Inf*

print-locale *ord*
print-locale *order*
print-locale *lattice*
print-locale *bot*
print-locale *complete-lattice*

Existence of a fixed point for a mono function F in ureal: See `Knaster_Tarski` under `HOL/Examples`

lemma *mu-id*: $(\mu_p\ (X{::}'a \Rightarrow ureal) \cdot X) = \mathbf{0}$
apply (*simp add*: *lfp-def*)
by (*metis bot.extremum-uniqueI bot-fun-def bot-ureal.rep-eq dual-order.refl less-eq-ureal.rep-eq zero-ureal.rep-eq*)

lemma *mu-const*: $(\mu_p\ X \cdot P) = P$
by (*simp add*: *lfp-const*)

lemma *nu-id*: $(\nu_p\ (X{::}'a \Rightarrow ureal) \cdot X) = \mathbf{1}$
apply (*simp add*: *gfp-def*)
using *one-ureal-def top-ureal-def* **by** *auto*

lemma *nu-const*: $(\nu_p\ X \cdot P) = P$
by (*simp add*: *gfp-const*)

term *Complete-Partial-Order.chain* $(\leq)$ *x*
term *monotone*
thm *Complete-Partial-Order.iterates.induct*

theorem *loopfunc-mono*:
assumes *is-final-distribution* (*rvfun-of-prfun* $(P{::}('s,\ 's)$ *prfun*))
shows *mono* $(\mathcal{F}\ b\ P)$
apply (*simp add*: *mono-def loopfunc-def*)
apply (*auto*)
apply (*subst prfun-pcond-mono*)
apply (*subst prfun-pseqcomp-mono*)
apply (*auto*)
by (*simp add*: *assms pdrfun-product-summable''*)+

theorem *loopfunc-monoE*:

assumes *is-final-distribution* (*rvfun-of-prfun* (P::($'s$, $'s$) *prfun*))
assumes $X \leq Y$
shows $\mathcal{F}\ b\ P\ X \leq \mathcal{F}\ b\ P\ Y$
by (*simp add*: *loopfunc-mono assms*(*1*) *assms*(*2*) *monoD*)

theorem *mono-func-increasing-chain-is-increasing*:
assumes *increasing-chain c*
assumes *mono F*
shows *increasing-chain* ($\lambda n.\ F\ (c\ n)$)
apply (*simp add*: *increasing-chain-def*)
using *assms* **by** (*simp add*: *increasing-chain-mono monoD*)

theorem *mono-func-decreasing-chain-is-decreasing*:
assumes *decreasing-chain c*
assumes *mono F*
shows *decreasing-chain* ($\lambda n.\ F\ (c\ n)$)
apply (*simp add*: *decreasing-chain-def*)
using *assms* **by** (*simp add*: *decreasing-chain-antitone monoD*)

lemma *loopfunc-minus-distr*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::($'s$, $'s$) *prfun*))
assumes *is-final-prob* (*rvfun-of-prfun* (X::($'s$, $'s$) *prfun*))
assumes *is-final-prob* (*rvfun-of-prfun* (Y::($'s$, $'s$) *prfun*))
assumes $X \geq Y$
shows (*rvfun-of-prfun* ($\mathcal{F}\ b\ P\ X$) $-$ *rvfun-of-prfun* ($\mathcal{F}\ b\ P\ Y$)) =
(($[\![b]\!]_{\mathcal{I}}$) $*$ $\bullet$(*rvfun-of-prfun* ((P ; ($X - Y$)))))$_e$ (**is** *?lhs* = *?rhs*)
apply (*subst fun-eq-iff*, *auto*)
proof $-$
fix $s\ s'$
let *?lhs* = *rvfun-of-prfun* ($\mathcal{F}\ b\ P\ X$) (s, s') $-$ *rvfun-of-prfun* ($\mathcal{F}\ b\ P\ Y$) (s, s')
have *f1*: *rvfun-of-prfun* (*prfun-of-rvfun* [λs::$'s \times 's$.
($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; X) s + ($[\![[\lambda s::'s \times 's.\ \neg\ b\ s]_e]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun II* s]$_e$) (s, s')
= *rvfun-of-prfun* (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; X) s]$_e$) (s, s') +
rvfun-of-prfun (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![[\lambda s::'s \times 's.\ \neg\ b\ s]_e]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun II* s]$_e$)(s, s')
by (*smt* (*verit*) *SEXP-def iverson-bracket-def mult-cancel-left2 prfun-in-0-1′ prfun-of-rvfun-def rvfun-of-prfun-def ureal-real2ureal-smaller*)

have *f2*: *rvfun-of-prfun* (*prfun-of-rvfun* [λs::$'s \times 's$.
($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; Y) s + ($[\![[\lambda s::'s \times 's.\ \neg\ b\ s]_e]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun II* s]$_e$) (s, s')
= *rvfun-of-prfun* (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; Y) s]$_e$) (s, s') +
rvfun-of-prfun (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![[\lambda s::'s \times 's.\ \neg\ b\ s]_e]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun II* s]$_e$)(s, s')
apply (*simp add*: *prfun-of-rvfun-def*)
by (*smt* (*verit*) *SEXP-def iverson-bracket-def mult-cancel-left2 prfun-in-0-1′ prfun-of-rvfun-def rvfun-of-prfun-def ureal-real2ureal-smaller*)

have *f3*: *?lhs* = *rvfun-of-prfun* (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; X) s]$_e$) (s, s') $-$
rvfun-of-prfun (*prfun-of-rvfun* [λs::$'s \times 's$. ($[\![b]\!]_{\mathcal{I}}$) s $*$ *rvfun-of-prfun* (P ; Y) s]$_e$) (s, s')
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *prfun-pcond-altdef*)
using *f1 f2* **by** *simp*

have *f4*: ($\sum_\infty v_0$::$'s$. *rvfun-of-prfun* P (s, v_0) $*$ *rvfun-of-prfun* X (v_0, s')) $-$
($\sum_\infty v_0$::$'s$. *rvfun-of-prfun* P (s, v_0) $*$ *rvfun-of-prfun* Y (v_0, s')) =
($\sum_\infty v_0$::$'s$. *rvfun-of-prfun* P (s, v_0) $*$ *rvfun-of-prfun* X (v_0, s')) +

$(\sum_\infty v_0::'s. - (\textit{rvfun-of-prfun}\ P\ (s,\ v_0) * \textit{rvfun-of-prfun}\ Y\ (v_0,\ s')))$
apply (*subst infsum-uminus*)
by *auto*
also have *f5*: ... = $(\sum_\infty v_0::'s.\ \textit{rvfun-of-prfun}\ P\ (s,\ v_0) * \textit{rvfun-of-prfun}\ X\ (v_0,\ s') +$
$(- (\textit{rvfun-of-prfun}\ P\ (s,\ v_0) * \textit{rvfun-of-prfun}\ Y\ (v_0,\ s'))))$
apply (*subst infsum-add*)
apply (*simp add*: *assms*(*1*) *is-final-dist-subdist rvfun-product-summable-subdist ureal-is-prob*)
apply (*subst summable-on-uminus*)
apply (*simp add*: *assms*(*1*) *is-final-dist-subdist rvfun-product-summable-subdist ureal-is-prob*)
by *auto*
also have *f6*: ... = $(\sum_\infty v_0::'s.\ \textit{rvfun-of-prfun}\ P\ (s,\ v_0) * (\textit{rvfun-of-prfun}\ X\ (v_0,\ s') - \textit{rvfun-of-prfun}$
$Y\ (v_0,\ s')))$
by (*metis* (*mono-tags*, *opaque-lifting*) *ab-group-add-class.ab-diff-conv-add-uminus right-diff-distrib'*)
also have *f7*: ... = $(\sum_\infty v_0::'s.\ \textit{rvfun-of-prfun}\ P\ (s,\ v_0) * (\textit{rvfun-of-prfun}\ (X-Y)\ (v_0,\ s')))$
using *prfun-minus-distribution* **by** (*metis* (*mono-tags*, *opaque-lifting*) *assms*(*4*) *minus-apply*)

show $\textit{rvfun-of-prfun}\ (\mathcal{F}\ b\ P\ X)\ (s,\ s') - \textit{rvfun-of-prfun}\ (\mathcal{F}\ b\ P\ Y)\ (s,\ s') =$
$(\llbracket b \rrbracket_{\mathcal{I}})\ (s,\ s') * \textit{rvfun-of-prfun}\ (P\ ;\ (X - Y))\ (s,\ s')$
apply (*simp add*: *f3*)

apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-seqcomp-inverse*)
apply (*simp add*: *assms*(*1*))
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-seqcomp-inverse*)
apply (*simp add*: *assms*(*1*))
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-seqcomp-inverse*)
apply (*simp add*: *assms*(*1*))
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)
using *rvfun-product-prob-dist-leq-1 assms*(*1*) *ureal-is-prob* **apply** *fastforce*
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)
using *rvfun-product-prob-dist-leq-1 assms*(*1*) *ureal-is-prob* **apply** *fastforce*
apply (*expr-auto*)
using *calculation f7* **by** *presburger*
qed

lemma *loopfunc-minus-distr'*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes *is-final-prob* (*rvfun-of-prfun* (X::('s, 's) *prfun*))
assumes *is-final-prob* (*rvfun-of-prfun* (Y::('s, 's) *prfun*))
assumes $X \geq Y$
shows $(\textit{ureal2real}\ (\mathcal{F}\ b\ P\ X\ (s,s')) - \textit{ureal2real}\ (\mathcal{F}\ b\ P\ Y\ (s,s'))) =$
$(\llbracket b \rrbracket_{\mathcal{I}})\ (s,s') * \textit{ureal2real}\ ((P\ ;\ (X - Y))\ (s,s'))$ (**is** *?lhs = ?rhs*)
proof −
have $(\textit{rvfun-of-prfun}\ (\mathcal{F}\ b\ P\ X) - \textit{rvfun-of-prfun}\ (\mathcal{F}\ b\ P\ Y)) =$
$((\llbracket b \rrbracket_{\mathcal{I}}) * \bullet(\textit{rvfun-of-prfun}\ ((P\ ;\ (X - Y)))))_e$
using *loopfunc-minus-distr assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) **by** *blast*

then have (*ureal2real* ($\mathcal{F}$ *b P X* (*s*,*s'*)) − *ureal2real* ($\mathcal{F}$ *b P Y* (*s*,*s'*))) =
([[*b*]]$_\mathcal{I}$) (*s*,*s'*) ∗ *ureal2real* ((*P* ; (*X* − *Y*)) (*s*,*s'*))
using *rvfun-of-prfun-def* **by** (*smt* (*verit*, *del-insts*) *SEXP-def fun-diff-def*)
then show *?thesis*
by *simp*
qed

theorem *pwhile-unfold*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(*'s*, *'s*) *prfun*))
shows *while$_p$ b do P od* = (*if$_c$ b then* (*P* ; (*while$_p$ b do P od*)) *else II*)
proof −
have *m*:*mono* (λ*X*. (*if$_c$ b then* (*P* ; *X*) *else II*))
apply (*simp add*: *mono-def*, *auto*)
apply (*subst prfun-pcond-mono*)
apply (*subst prfun-pseqcomp-mono*)
apply (*auto*)
by (*simp add*: *assms pdrfun-product-summable''*)+
have (*while$_p$ b do P od*) = (μ_p *X* · (*if$_c$ b then* (*P* ; *X*) *else II*))
by (*simp add*: *pwhile-def loopfunc-def*)
also have ... = ((*if$_c$ b then* (*P* ; (μ_p *X* · (*if$_c$ b then* (*P* ; *X*) *else II*))) *else II*))
apply (*subst lfp-unfold*)
apply (*simp add*: *m*)
by (*simp add*: *lfp-const*)
also have ... = (*if$_c$ b then* (*P* ; (*while$_p$ b do P od*)) *else II*)
by (*simp add*: *pwhile-def loopfunc-def*)
finally show *?thesis* **.**
qed

theorem *pwhile-false*: *while$_p$ false do P od* = *II*
apply (*simp add*: *pwhile-def loopfunc-def pcond-def*)
apply (*subst rvfun-pcond-altdef*)
apply (*pred-auto*)
by (*simp add*: *prfun-inverse utp-prob-rel-lattice-laws.mu-const*)

theorem *pwhile-true*: *while$_p$ true do P od* = 0_p
apply (*simp add*: *pwhile-def pcond-def pzero-def*)
apply (*rule antisym*)
apply (*rule lfp-lowerbound*)
apply (*simp add*: *loopfunc-def true-pred-def*)
apply (*simp add*: *prfun-zero-right*)
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *ureal-zero ureal-zero'*)
by (*rule ureal-bottom-least*)

theorem *pwhile-top-unfold*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(*'s*, *'s*) *prfun*))
shows *while$_p{}^\top$ b do P od* = (*if$_c$ b then* (*P* ; (*while$_p{}^\top$ b do P od*)) *else II*)
proof −
have *m*:*mono* (λ*X*. (*if$_c$ b then* (*P* ; *X*) *else II*))
apply (*simp add*: *mono-def*, *auto*)
apply (*subst prfun-pcond-mono*)
apply (*subst prfun-pseqcomp-mono*)
apply (*auto*)

by (*simp add*: *assms pdrfun-product-summable''*)+
have ($while_p{}^\top$ *b do P od*) = (ν_p *X* · (if_c *b then* (*P* ; *X*) *else II*))
by (*simp add*: *pwhile-top-def loopfunc-def*)
also have ... = ((if_c *b then* (*P* ; (ν_p *X* · (if_c *b then* (*P* ; *X*) *else II*))) *else II*))
apply (*subst gfp-unfold*)
apply (*simp add*: *m*)
by (*simp add*: *lfp-const*)
also have ... = (if_c *b then* (*P* ; ($while_p{}^\top$ *b do P od*)) *else II*)
by (*simp add*: *pwhile-top-def loopfunc-def*)
finally show *?thesis* **.**
qed

theorem *pwhile-top-false*: $while_p{}^\top$ *false do P od* = *II*
apply (*simp add*: *pwhile-top-def loopfunc-def pcond-def*)
apply (*subst rvfun-pcond-altdef*)
apply (*pred-auto*)
by (*simp add*: *prfun-inverse utp-prob-rel-lattice-laws.nu-const*)

theorem *pwhile-top-true*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::('*s*, '*s*) *prfun*))
shows $while_p{}^\top$ *true do P od* = 1_p
apply (*simp add*: *pwhile-top-def pcond-def pzero-def*)
apply (*rule antisym*)
apply (*simp add*: *ureal-top-greatest'*)
apply (*rule gfp-upperbound*)
apply (*simp add*: *loopfunc-def true-pred-def*)
apply (*simp add*: *prfun-seqcomp-one assms*)
apply (*simp add*: *pfun-defs*)
by (*simp add*: *SEXP-def prfun-inverse*)

5.8.1 Iteration

lemma *iterate 0 b P* 0_p = 0_p
by *simp*

lemma *iterate 0 b P* 1_p = 1_p
by *simp*

lemma *iterate-mono*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::('*s*, '*s*) *prfun*))
shows *monotone* (≤) (≤) (*iterate n b P*)
unfolding *monotone-def* **apply** (*auto*)
apply (*induction n*)
apply (*auto*)
by (*metis loopfunc-mono assms monoE*)

lemma *iterate-monoE*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::('*s*, '*s*) *prfun*))
assumes *X* ≤ *Y*
shows (*iterate n b P X*) ≤ (*iterate n b P Y*)
by (*metis assms*(*1*) *assms*(*2*) *iterate-mono monotone-def*)

lemma *iterate-increasing*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::('*s*, '*s*) *prfun*))
shows (*iterate n b P* 0_p) ≤ (*iterate* (*Suc n*) *b P* 0_p)
apply (*induction n*)

apply (*simp*)
using *ureal-bottom-least′* **apply** *blast*
apply (*simp*)
apply (*subst loopfunc-monoE*)
by (*simp add*: *assms*)+

lemma *iterate-increasing1*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows (*iterate n b P* 0_p) ≤ (*iterate* (*n*+*m*) *b P* 0_p)
apply (*induction m*)
apply (*simp*)
by (*metis* (*full-types*) *assms add-Suc-right dual-order.trans iterate-increasing*)

lemma *iterate-increasing2*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
assumes *n* ≤ *m*
shows (*iterate n b P* 0_p) ≤ (*iterate m b P* 0_p)
using *iterate-increasing1 assms nat-le-iff-add* **by** *auto*

lemma *iterate-increasing-chain-bot*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows *Complete-Partial-Order.chain* (≤) {(*iterate n b P* 0_p) | *n*::*nat*. *True*}
(**is** *Complete-Partial-Order.chain - ?C*)
proof (*rule Complete-Partial-Order.chainI*)
fix *x y*
assume *x* ∈ *?C y* ∈ *?C*
then show *x* ≤ *y* ∨ *y* ≤ *x*
by (*smt* (*verit*) *assms iterate-increasing2 mem-Collect-eq nle-le*)
qed

lemma *iterate-increasing-chain*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows *increasing-chain* (λ*n*. (*iterate n b P* 0_p))
(**is** *increasing-chain ?C*)
apply (*simp add*: *increasing-chain-def*)
by (*simp add*: *assms iterate-increasing2*)

lemma *iterate-decreasing*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows (*iterate n b P* 1_p) ≥ (*iterate* (*Suc n*) *b P* 1_p)
apply (*induction n*)
apply (*metis le-fun-def linorder-not-le o-def one-ureal.rep-eq pone-def real-ereal-1 ureal2real-def ureal2real-mono-strict ureal-upper-bound utp-prob-rel-lattice.iterate.simps*(*1*))
by (*simp add*: *loopfunc-monoE assms*)

lemma *iterate-decreasing1*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows (*iterate n b P* 1_p) ≥ (*iterate* (*n*+*m*) *b P* 1_p)
apply (*induction m*)
apply (*simp*)
by (*metis* (*no-types*, *opaque-lifting*) *assms gfp.leq-trans iterate-decreasing nat-arith.suc1*)

lemma *iterate-decreasing2*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))

assumes $n \leq m$
shows $(\mathit{iterate}\ n\ b\ P\ 1_p) \geq (\mathit{iterate}\ m\ b\ P\ 1_p)$
using *iterate-decreasing1 assms* **using** *nat-le-iff-add* **by** *auto*

lemma *iterate-decreasing-chain-top*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
shows *Complete-Partial-Order.chain* $(\geq)$ $\{(\mathit{iterate}\ n\ b\ P\ 1_p) \mid n{::}\mathit{nat}.\ \mathit{True}\}$
(**is** *Complete-Partial-Order.chain - ?C*)
proof (*rule Complete-Partial-Order.chainI*)
fix $x\ y$
assume $x \in\ ?C\ y \in\ ?C$
then show $x \leq y \vee y \leq x$
by (*smt* (*verit*) *assms iterate-decreasing2 mem-Collect-eq nle-le*)
qed

lemma *iterate-decreasing-chain*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
shows *decreasing-chain* $(\lambda n.\ (\mathit{iterate}\ n\ b\ P\ 1_p))$
(**is** *decreasing-chain ?C*)
apply (*simp add*: *decreasing-chain-def*)
by (*simp add*: *assms iterate-decreasing2*)

5.8.2 Supreme

lemma *sup-iterate-not-zero-strict-increasing*:
shows $(\exists n.\ \mathit{iterate}\ n\ b\ P\ 0_p\ s \neq 0) \longleftrightarrow$
$(\mathit{ureal2real}\ (\mathit{iter}_p\ (0{::}\mathbb{N})\ b\ P\ 0_p\ s) < \mathit{ureal2real}\ (\bigsqcup n{::}\mathbb{N}.\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s))$
apply (*rule iffI*)
proof (*rule ccontr*)
assume $a1$: $\exists n{::}\mathbb{N}.\ \neg\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s = (0{::}\mathit{ureal})$
assume $a2$: $\neg\ \mathit{ureal2real}\ (\mathit{iter}_p\ (0{::}\mathbb{N})\ b\ P\ 0_p\ s) < \mathit{ureal2real}\ (\bigsqcup n{::}\mathbb{N}.\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s)$
then have $(\bigsqcup n{::}\mathbb{N}.\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s) = (\mathit{iter}_p\ (0{::}\mathbb{N})\ b\ P\ 0_p\ s)$
by (*metis not-le-imp-less pzero-def ureal2real-mono-strict ureal-minus-larger-zero ureal-minus-larger-zero-unit utp-prob-rel-lattice.iterate.simps*(*1*))
then have $\forall n.\ \mathit{iterate}\ n\ b\ P\ 0_p\ s = (\mathit{iter}_p\ (0{::}\mathbb{N})\ b\ P\ 0_p\ s)$
by (*metis SUP-upper bot.extremum bot-ureal.rep-eq iso-tuple-UNIV-I nle-le pzero-def ureal2ereal-inverse utp-prob-rel-lattice.iterate.simps*(*1*) *zero-ureal.rep-eq*)
then show *False*
by (*metis a1 pzero-def utp-prob-rel-lattice.iterate.simps*(*1*))
next
assume $\mathit{ureal2real}\ (\mathit{iter}_p\ (0{::}\mathbb{N})\ b\ P\ 0_p\ s) < \mathit{ureal2real}\ (\bigsqcup n{::}\mathbb{N}.\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s)$
then show $\exists n{::}\mathbb{N}.\ \neg\ \mathit{iter}_p\ n\ b\ P\ 0_p\ s = (0{::}\mathit{ureal})$
by (*smt* (*verit*, *best*) *SUP-bot-conv*(*2*) *bot-ureal.rep-eq ureal2ereal-inverse zero-ureal.rep-eq*)
qed

lemma *sup-iterate-continuous-limit*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes $\mathcal{FS}\ (\lambda n.\ \mathit{iterate}\ n\ b\ P\ 0_p)$
shows $(\lambda n.\ \mathit{ureal2real}\ (\mathcal{F}\ b\ P\ (\mathit{iterate}\ n\ b\ P\ 0_p)\ (s, s'))) \longrightarrow$
$\mathit{ureal2real}\ ((\mathcal{F}\ b\ P\ (\bigsqcup n{::}\mathit{nat}.\ \mathit{iterate}\ n\ b\ P\ 0_p))\ (s, s'))$
apply (*subst LIMSEQ-iff*)
apply (*auto*)
proof −
fix r
assume $a1$: $(0{::}\mathbb{R}) < r$
have $f1$: $\forall n.\ \mathit{ureal2real}\ (\mathcal{F}\ b\ P\ (\mathit{iter}_p\ n\ b\ P\ 0_p)\ (s, s')) \leq$

```
          ureal2real (ℱ b P (⨆n::ℕ. iter_p n b P 0_p) (s, s'))
    apply (auto)
    apply (rule ureal2real-mono)
    by (smt (verit) loopfunc-monoE SUP-upper UNIV-I assms le-fun-def)
  have f2: ∀n. |ureal2real (ℱ b P (iter_p n b P 0_p) (s, s')) −
          ureal2real (ℱ b P (⨆n::ℕ. iter_p n b P 0_p) (s, s'))| =
   (ureal2real (ℱ b P (⨆n::ℕ. iter_p n b P 0_p) (s, s')) −
    ureal2real (ℱ b P (iter_p n b P 0_p) (s, s')))
    using f1 by force


  let ?f = (λn. (iter_p n b P 0_p))

  have f3: ∀n. ∀s s'. ureal2real (?f n (s, s')) ≤ ureal2real (⨆n::ℕ. ?f n (s, s'))
    apply (auto)
    apply (rule ureal2real-mono)
    by (smt (verit) loopfunc-monoE SUP-upper UNIV-I assms le-fun-def)


  have Sn-limit-sup: (λn. ureal2real (?f n (s, s'))) ⟶ (ureal2real (⨆n::ℕ. ?f n (s, s')))
    apply (subst increasing-chain-limit-is-lub)
    apply (simp add: assms(1) increasing-chain-def iterate-increasing2)
    by simp
  then have Sn-limit: ∀r>0. ∃no::ℕ. ∀n≥no.
          |ureal2real (?f n (s, s')) − ureal2real (⨆n::ℕ. ?f n (s, s'))| < r
    using Sn-limit-sup LIMSEQ-iff by (smt (verit, del-insts) real-norm-def)
  have exist-N: ∃no::ℕ. ∀n≥no. |ureal2real (?f n (s, s')) − ureal2real (⨆n::ℕ. ?f n (s, s'))| < r
    using Sn-limit a1 by blast


  have exist-NN: ∃no::nat. ∀n ≥ no.
         ∀s s'. ureal2real (⨆n::ℕ. ?f n (s, s')) − ureal2real (?f n (s, s')) < r
    apply (subst increasing-chain-limit-is-lub-all)
    apply (simp add: assms(1) iterate-increasing-chain)
   using assms(2) sup-iterate-not-zero-strict-increasing apply (smt (verit) Collect-cong Sup.SUP-cong)
    by (simp add: a1)+

  obtain NN where P-NN: ∀n≥NN. ∀s s'. |ureal2real (?f n (s, s')) − ureal2real (⨆n::ℕ. ?f n (s, s'))|
< r
    using exist-NN f3 by auto

  have P-NN': ∀n≥NN. ∀s s'. ureal2real (⨆n::ℕ. ?f n (s, s')) − ureal2real (?f n (s, s')) < r
    by (smt (verit, del-insts) P-NN)

  have ∀n≥NN. (ureal2real (ℱ b P (⨆n::ℕ. iter_p n b P 0_p) (s, s')) −
    ureal2real (ℱ b P (iter_p n b P 0_p) (s, s'))) < r
    apply (auto)
    apply (subst loopfunc-minus-distr')
    apply (simp add: assms)
    apply (simp add: is-prob-final-prob ureal-is-prob)+
    apply (meson SUP-upper UNIV-I)
    apply (simp add: pseqcomp-def)
    apply (expr-auto)
  proof −
    fix n::nat
```

assume *a10*: $NN \leq n$
assume *a11*: $b\ (s,\ s')$
let *?lhs* = *ureal2real*
(*prfun-of-rvfun*
(λs::$'s \times\ 's$.
$\sum_{\infty} v_0$::$'s$.
rvfun-of-prfun P (*fst* s, v_0) $*$
rvfun-of-prfun (($\bigsqcup n$::$\mathbb{N}$. $iter_p\ n\ b\ P\ 0_p$) $-$ $iter_p\ n\ b\ P\ 0_p$) (v_0, *snd* s))
(s, s'))
have *f10*: $\forall\, s\ s'$. (*ureal2real* ($\bigsqcup n$::$\mathbb{N}$. *?f* n (s, s')) $-$ *ureal2real* (*?f* n (s, s'))) =
(*ureal2real* (($\bigsqcup n$::$\mathbb{N}$. *?f* n (s, s')) $-$ (*?f* n (s, s'))))
by (*metis f3 linorder-not-le ureal2real-distr ureal2real-mono-strict*)
have *f11*: (($\sum_{\infty} v_0$::$'s$.
ureal2real (P (s, v_0)) $*$
ureal2real (($\bigsqcup f$::$'s \times\ 's \Rightarrow$ *ureal*$\in$*range* (λn::$\mathbb{N}$. $iter_p\ n\ b\ P\ 0_p$). f (v_0, s')) $-$ $iter_p\ n\ b\ P\ 0_p$
(v_0, s'))))
= (($\sum_{\infty} v_0$::$'s$.
ureal2real (P (s, v_0)) $*$ (*ureal2real* ($\bigsqcup n$::$\mathbb{N}$. *?f* n (v_0, s')) $-$ *ureal2real* (*?f* n (v_0, s')))))
apply (*rule infsum-cong*)
by (*smt* (*verit*, *best*) *Sup.SUP-cong f10 image-image*)
have *f12*: ... $<$ ($\sum_{\infty} v_0$::$'s$. *ureal2real* (P (s, v_0)) $*$ r)
proof $-$
let *?lhs* = λv_0. *ureal2real* (P (s, v_0)) $*$
(*ureal2real* ($\bigsqcup n$::$\mathbb{N}$. $iter_p\ n\ b\ P\ 0_p$ (v_0, s')) $-$ *ureal2real* ($iter_p\ n\ b\ P\ 0_p$ (v_0, s')))
let *?rhs* = λv_0. *ureal2real* (P (s, v_0)) $*$ r
obtain v_0 **where** *P-v_0*: P (s, v_0) $>$ *0*
using *assms rvfun-prob-sum1-summable*(*4*)
by (*smt* (*verit*, *best*) *SEXP-def bot.extremum bot-ureal.rep-eq nless-le rvfun-of-prfun-def*
ureal2ereal-inverse ureal2real-mono-strict ureal-lower-bound ureal-real2ureal-smaller zero-ureal.rep-eq)
have *lhs-0*: ($\sum_{\infty} v_0$::$'s$. *?lhs* v_0) = ($\sum_{\infty} v_0$::$'s \in (\{v_0\} \cup (-\{v_0\}))$. *?lhs* v_0)
by *auto*
have *lhs-1*: ... = ($\sum_{\infty} v_0$::$'s \in \{v_0\}$. *?lhs* v_0) + ($\sum_{\infty} v_0$::$'s \in -\{v_0\}$. *?lhs* v_0)
apply (*rule infsum-Un-disjoint*)
apply *auto*[*1*]
apply (*simp add*: *f10*)
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst pdrfun-product-summable'*)
by (*simp add*: *assms*)+
have *rhs-0*: ($\sum_{\infty} v_0$::$'s$. *?rhs* v_0) = ($\sum_{\infty} v_0$::$'s \in (\{v_0\} \cup (-\{v_0\}))$. *?rhs* v_0)
by *auto*
have *rhs-1*: ... = ($\sum_{\infty} v_0$::$'s\in (\{v_0\})$. *?rhs* v_0) + ($\sum_{\infty} v_0$::$'s\in ((-\{v_0\}))$. *?rhs* v_0)
apply (*rule infsum-Un-disjoint*)
apply *auto*[*1*]
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst summable-on-cmult-left*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable*(*4*))
by (*simp*)+
have *lhs-0-rhs-0*: ($\sum_{\infty} v_0$::$'s \in -\{v_0\}$. *?lhs* v_0) $\leq$ ($\sum_{\infty} v_0$::$'s\in ((-\{v_0\}))$. *?rhs* v_0)
apply (*rule infsum-mono*)
apply (*simp add*: *f10*)
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst pdrfun-product-summable'*)
apply (*simp add*: *assms*)+
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst summable-on-cmult-left*)

```
        apply (simp add: assms pdrfun-prob-sum1-summable(4))
        apply (simp)+
        by (smt (verit, ccfv-SIG) P-NN' Sup.SUP-cong a10 left-diff-distrib
            linordered-comm-semiring-strict-class.comm-mult-strict-left-mono ureal-lower-bound)
      have lhs-2: (∑∞v0::'s ∈ {v0}. ?lhs v0) = ?lhs v0
        by (rule infsum-on-singleton)
      have rhs-2: (∑∞v0::'s∈ ({v0}). ?rhs v0) = ?rhs v0
        by (rule infsum-on-singleton)
      have lhs-1-rhs-1: ?lhs v0 < ?rhs v0
      by (smt (verit, best) P-NN' P-v0 Sup.SUP-cong a10 linordered-comm-semiring-strict-class.comm-mult-strict-left-mon
ureal2real-mono-strict ureal-lower-bound)
      show ?thesis
        apply (simp only: lhs-0 rhs-0 lhs-1 rhs-1)
        using lhs-0-rhs-0 lhs-1-rhs-1 lhs-2 rhs-2 by linarith
    qed
    also have ... = (∑∞v0::'s. ureal2real (P (s, v0))) * r
      apply (rule infsum-cmult-left)
      by (simp add: assms pdrfun-prob-sum1-summable(4))
    also have ... = r
      by (simp add: assms pdrfun-prob-sum1-summable(3))
    then have f13: ( (∑∞v0::'s.
          ureal2real (P (s, v0)) * (ureal2real (⨆n::ℕ. ?f n (v0, s')) − ureal2real (?f n (v0, s')))))) < r
      using calculation by linarith

    have f14: ?lhs = ureal2real
        (real2ureal ( (∑∞v0::'s.
          ureal2real (P (s, v0)) * (ureal2real (⨆n::ℕ. ?f n (v0, s')) − ureal2real (?f n (v0, s'))))))
      apply (simp add: prfun-of-rvfun-def)
      apply (simp add: rvfun-of-prfun-def)
      by (simp add: f11)
    show ?lhs  < r
      apply (simp add: f14)
        using f13 by (smt (verit, del-insts) f11 infsum-nonneg mult-nonneg-nonneg ureal-lower-bound
ureal-real2ureal-smaller)
  next
    show (0::ℝ) < r
      by (simp add: a1)
  qed

  then show ∃no::ℕ. ∀n≥no.
            |ureal2real (ℱ b P (iter_p n b P 0_p) (s, s')) −
             ureal2real (ℱ b P (⨆n::ℕ. iter_p n b P 0_p) (s, s'))| < r
    apply (simp add: loopfunc-def)
    by (metis loopfunc-def f2)
qed

lemma sup-iterate-suc: (⨆x ∈ {(iterate n b P 0_p) | n::nat. True}. (ℱ b P x)) =
      (⨆n::nat. (iterate (Suc n) b P 0_p))
  apply (simp add: image-def)
  by metis

lemma sup-iterate-subset-eq:
  (⨆n::nat. (iterate (Suc n) b P 0_p)) = (⨆n::nat. (iterate n b P 0_p))
proof −
  have f1: (⨆n::nat. (iterate (Suc n) b P 0_p)) = (⨆n∈{1..}. (iterate n b P 0_p))
```

apply (*simp add*: *image-def*)
by (*metis atLeast-iff bot-nat-0.extremum not0-implies-Suc not-less-eq-eq utp-prob-rel-lattice.iterate.simps*(*2*))
have $insert\ (0::nat)\ \{1..\} = UNIV$
using *UNIV-nat-eq atLeast-Suc-greaterThan* **by** *auto*
then have $f2$: $(\bigsqcup n::nat.\ (iterate\ n\ b\ P\ 0_p)) = (\bigsqcup n::nat \in insert\ 0\ \{1..\}.\ (iterate\ n\ b\ P\ 0_p))$
by (*simp add*: *image-def*)
have $f3$: $(\bigsqcup n::nat \in insert\ 0\ \{1..\}.\ (iterate\ n\ b\ P\ 0_p)) = (iterate\ 0\ b\ P\ 0_p) \sqcup (\bigsqcup n\in\{1..\}.\ (iterate\ n$
$b\ P\ 0_p))$
apply (*subst SUP-insert*)
using *sup-commute* **by** *blast*
have $f4$: $... = (\bigsqcup n\in\{1..\}.\ (iterate\ n\ b\ P\ 0_p))$
using *le-iff-sup ureal-bottom-least'* **by** *auto*
show *?thesis*
using *f1 f2 f3 f4* **by** *presburger*
qed

lemma *sup-iterate-continuous'*:
assumes $is\text{-}final\text{-}distribution\ (rvfun\text{-}of\text{-}prfun\ (P::('s,\ 's)\ prfun))$
assumes $\mathcal{FS}\ (\lambda n.\ iterate\ n\ b\ P\ 0_p)$

shows $\mathcal{F}\ b\ P\ (\bigsqcup n::nat.\ iterate\ n\ b\ P\ 0_p) = (\bigsqcup x \in \{(iterate\ n\ b\ P\ 0_p) \mid n::nat.\ True\}.\ (\mathcal{F}\ b\ P\ x))$
apply (*subst fun-eq-iff*)
apply (*auto*)
proof −
fix $s\ s'$
let $?f = \lambda n.\ \mathcal{F}\ b\ P\ (iterate\ n\ b\ P\ 0_p)$
have *increasing-chain ?f*
by (*simp add*: *loopfunc-monoE assms increasing-chain-def iterate-increasing2*)
then have $(\lambda n.\ ureal2real\ (?f\ n\ (s,\ s'))) \longrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ ?f\ n\ (s,\ s')))$
by (*rule increasing-chain-limit-is-lub*)
then have $ureal2real\ (\bigsqcup n::\mathbb{N}.\ ?f\ n\ (s,\ s')) = ureal2real\ ((\mathcal{F}\ b\ P\ (\bigsqcup n::nat.\ iterate\ n\ b\ P\ 0_p))\ (s,\ s'))$
apply (*subst LIMSEQ-unique*[**where** $X=(\lambda n.\ ureal2real\ (?f\ n\ (s,\ s')))$ **and** $a = ureal2real\ (\bigsqcup n::\mathbb{N}.$
$?f\ n\ (s,\ s'))$ **and**
$b = ureal2real\ ((\mathcal{F}\ b\ P\ (\bigsqcup n::nat.\ iterate\ n\ b\ P\ 0_p))\ (s,\ s'))$])
apply *meson*
apply (*subst sup-iterate-continuous-limit*)
using *assms*(*1*) **apply** *blast*
using *assms*(*2*) **apply** *blast*
by (*simp*)+

then have $f1$: $(\bigsqcup n::\mathbb{N}.\ ?f\ n\ (s,\ s')) = ((\mathcal{F}\ b\ P\ (\bigsqcup n::nat.\ iterate\ n\ b\ P\ 0_p))\ (s,\ s'))$
using *ureal2real-eq* **by** *blast*

have $f2$: $(\bigsqcup x::'s \times 's \Rightarrow ureal\in \mathcal{F}\ b\ P\ `\ \{uu::'s \times 's \Rightarrow ureal.\ \exists n::\mathbb{N}.\ uu = iter_p\ n\ b\ P\ 0_p\}.\ x\ (s,\ s'))$
$= Sup\ ((\lambda x.\ x\ (s,\ s'))\ `\ (\mathcal{F}\ b\ P\ `\ \{uu::'s \times 's \Rightarrow ureal.\ \exists n::\mathbb{N}.\ uu = iter_p\ n\ b\ P\ 0_p\}))$
by *auto*
have $f3$: $(\bigsqcup n::\mathbb{N}.\ \mathcal{F}\ b\ P\ (iter_p\ n\ b\ P\ 0_p)\ (s,\ s')) = (Sup\ (range\ (\lambda n.\ \mathcal{F}\ b\ P\ (iter_p\ n\ b\ P\ 0_p)\ (s,\ s'))))$
by *simp*
have $f4$: $((\lambda x.\ x\ (s,\ s'))\ `\ (\mathcal{F}\ b\ P\ `\ \{uu::'s \times 's \Rightarrow ureal.\ \exists n::\mathbb{N}.\ uu = iter_p\ n\ b\ P\ 0_p\})) =$
$(range\ (\lambda n.\ \mathcal{F}\ b\ P\ (iter_p\ n\ b\ P\ 0_p)\ (s,\ s')))$
apply (*simp add*: *image-def*)
by (*auto*)
show $\mathcal{F}\ b\ P\ (\bigsqcup n::\mathbb{N}.\ iter_p\ n\ b\ P\ 0_p)\ (s,\ s') =$
$(\bigsqcup x::'s \times 's \Rightarrow ureal\in\mathcal{F}\ b\ P\ `\ \{uu::'s \times 's \Rightarrow ureal.\ \exists n::\mathbb{N}.\ uu = iter_p\ n\ b\ P\ 0_p\}.\ x\ (s,\ s'))$

apply (*simp add*: *f1*[*symmetric*])
using *f4* **by** *presburger*
qed

theorem *sup-iterate-continuous*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::($'s$, $'s$) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate* n b P 0_p)
shows $\mathcal{F}$ b P ($\bigsqcup n$::*nat*. *iterate* n b P 0_p) = ($\bigsqcup n$::*nat*. (*iterate* n b P 0_p))
apply (*subst sup-iterate-continuous$'$*)
apply (*simp add*: *assms*(*1*))
using *assms*(*2*) **apply** *auto*[*1*]
using *sup-iterate-suc sup-iterate-subset-eq* **by** *metis*

5.8.3 Infimum

lemma *inf-iterate-not-zero-strict-decreasing*:
shows ($\exists n.$ *iterate* n b P 1_p $s \neq 1$) $\longleftrightarrow$
(*ureal2real* (*iter*$_p$ (*0*::$\mathbb{N}$) b P 1_p s) > *ureal2real* ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p s))
apply (*rule iffI*)
proof (*rule ccontr*)
assume *a1*: $\exists n$::$\mathbb{N}$. $\neg$ *iter*$_p$ n b P 1_p s = (*1*::*ureal*)
assume *a2*: $\neg$ *ureal2real* ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p s) < *ureal2real* (*iter*$_p$ (*0*::$\mathbb{N}$) b P 1_p s)
then have ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p s) = (*iter*$_p$ (*0*::$\mathbb{N}$) b P 1_p s)
by (*metis linorder-not-less not-less-iff-gr-or-eq o-apply one-ureal.rep-eq pone-def real-ereal-1 ureal2real-def ureal2real-mono-strict ureal-upper-bound utp-prob-rel-lattice.iterate.simps*(*1*))
then have $\forall n.$ *iterate* n b P 1_p s = (*iter*$_p$ (*0*::$\mathbb{N}$) b P 1_p s)
by (*smt* (*verit*, *best*) *INF-top-conv*(*2*) *UNIV-I linorder-not-less not-less-iff-gr-or-eq o-apply one-ureal.rep-eq pone-def real-ereal-1 top-greatest ureal2real-def ureal2real-mono-strict ureal-upper-bound utp-prob-rel-lattice.iterate.simps*(*1*))
then show *False*
by (*metis a1 pone-def utp-prob-rel-lattice.iterate.simps*(*1*))
next
assume *ureal2real* ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p s) < *ureal2real* (*iter*$_p$ (*0*::$\mathbb{N}$) b P 1_p s)
then show $\exists n$::$\mathbb{N}$. $\neg$ *iter*$_p$ n b P 1_p s = (*1*::*ureal*)
by (*smt* (*verit*, *ccfv-threshold*) *INF-top-conv*(*2*) *one-ureal.rep-eq top-ureal.rep-eq ureal2ereal-inject*)
qed

lemma *inf-iterate-continuous-limit*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::($'s$, $'s$) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate* n b P 1_p)
shows ($\lambda n.$ *ureal2real* ($\mathcal{F}$ b P (*iterate* n b P 1_p) (s, s'))) $\longrightarrow$
ureal2real (($\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p)) (s, s'))
apply (*subst LIMSEQ-iff*)
apply (*auto*)
proof −
fix r
assume *a1*: (*0*::$\mathbb{R}$) < r
have *f1*: $\forall n.$ *ureal2real* ($\mathcal{F}$ b P (*iter*$_p$ n b P 1_p) (s, s')) $\geq$
ureal2real ($\mathcal{F}$ b P ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p) (s, s'))
apply (*auto*)
apply (*rule ureal2real-mono*)
by (*smt* (*verit*) *loopfunc-monoE INF-lower UNIV-I assms*(*1*) *le-fun-def*)
have *f2*: $\forall n.$ |*ureal2real* ($\mathcal{F}$ b P (*iter*$_p$ n b P 1_p) (s, s')) −
ureal2real ($\mathcal{F}$ b P ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p) (s, s'))| =
(*ureal2real* ($\mathcal{F}$ b P (*iter*$_p$ n b P 1_p) (s, s')) −
ureal2real ($\mathcal{F}$ b P ($\bigsqcap n$::$\mathbb{N}$. *iter*$_p$ n b P 1_p) (s, s')))

```
    using f1 by force

  let ?f = (λn. (iter_p n b P 1_p))

  have f3: ∀ n. ∀ s s'. ureal2real (?f n (s, s')) ≥ ureal2real (⨅ n::ℕ. ?f n (s, s'))
    apply (auto)
    apply (rule ureal2real-mono)
    by (meson INF-lower UNIV-I)

  have Sn-limit-inf: (λn. ureal2real (?f n (s, s'))) ⟶ (ureal2real (⨅ n::ℕ. ?f n (s, s')))
    apply (subst decreasing-chain-limit-is-glb)
    apply (simp add: assms decreasing-chain-def iterate-decreasing2)
    by simp
  then have Sn-limit: ∀ r>0. ∃ no::ℕ. ∀ n≥no.
          |ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s'))| < r
    using Sn-limit-inf LIMSEQ-iff by (smt (verit, del-insts) real-norm-def)
  have exist-N: ∃ no::ℕ. ∀ n≥no. |ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s'))| < r
    using Sn-limit a1 by blast

  have exist-NN: ∃ no::nat. ∀ n ≥ no.
          ∀ s s'. ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s')) < r
    apply (subst decreasing-chain-limit-is-glb-all)
      apply (simp add: assms iterate-decreasing-chain)
    using assms(2) inf-iterate-not-zero-strict-decreasing apply (smt (verit) Collect-cong Sup.SUP-cong)
    by (simp add: a1)+

  obtain NN where P-NN: ∀ n≥NN. ∀ s s'. |ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s'))|
< r
    using exist-NN f3 by auto

  have P-NN': ∀ n≥NN. ∀ s s'. ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s')) < r
    by (smt (verit, del-insts) P-NN)

  have ∀ n≥NN. (ureal2real (ℱ b P (iter_p n b P 1_p) (s, s')) −
          ureal2real (ℱ b P (⨅ n::ℕ. iter_p n b P 1_p) (s, s'))) < r
    apply (auto)
    apply (subst loopfunc-minus-distr')
    apply (simp add: assms)
    apply (simp add: is-prob-final-prob ureal-is-prob)+
    apply (meson INF-lower UNIV-I)
    apply (simp add: pseqcomp-def)
    apply (expr-auto)
  proof −
    fix n::nat
    assume a10: NN ≤ n
    assume a11: b (s, s')
    let ?lhs = ureal2real
        (prfun-of-rvfun
          (λs::'s × 's.
              ∑∞ v₀::'s.
                rvfun-of-prfun P (fst s, v₀) ∗
                rvfun-of-prfun (iter_p n b P 1_p − (⨅ n::ℕ. iter_p n b P 1_p)) (v₀, snd s))
          (s, s'))
    have f10: ∀ s s'. (ureal2real (?f n (s, s')) − ureal2real (⨅ n::ℕ. ?f n (s, s'))) =
          (ureal2real ((?f n (s, s')) − (⨅ n::ℕ. ?f n (s, s'))))
```

by (*metis f3 linorder-not-le ureal2real-distr ureal2real-mono-strict*)
have *f11*: (($\sum_\infty v_0$::′s.
ureal2real (*P* (*s*, v_0)) ∗
ureal2real (*iter*$_p$ *n b P 1*$_p$ (v_0, *s*′) − (⨅*f*::′s × ′s ⇒ *ureal*∈*range* (λ*n*::ℕ. *iter*$_p$ *n b P 1*$_p$). *f* (v_0, *s*′)))))
= (($\sum_\infty v_0$::′s.
ureal2real (*P* (*s*, v_0)) ∗ (*ureal2real* (*?f n* (v_0, *s*′)) − *ureal2real* (⨅*n*::ℕ. *?f n* (v_0, *s*′)))))
apply (*rule infsum-cong*)
by (*smt* (*verit*, *best*) *Sup.SUP-cong f10 image-image*)
have *f12*: ... < ($\sum_\infty v_0$::′s. *ureal2real* (*P* (*s*, v_0)) ∗ *r*)
proof −
let *?lhs* = λv_0. *ureal2real* (*P* (*s*, v_0)) ∗
(*ureal2real* (*iter*$_p$ *n b P 1*$_p$ (v_0, *s*′)) − *ureal2real* (⨅*n*::ℕ. *iter*$_p$ *n b P 1*$_p$ (v_0, *s*′)))
let *?rhs* = λv_0. *ureal2real* (*P* (*s*, v_0)) ∗ *r*
obtain v_0 **where** *P-*v_0: *P* (*s*, v_0) > *0*
using *assms rvfun-prob-sum1-summable*(*4*)
by (*smt* (*verit*, *ccfv-threshold*) *SEXP-def bot.extremum bot-ureal.rep-eq linorder-not-le nle-le rvfun-of-prfun-def ureal2ereal-inverse ureal2real-mono-strict ureal-real2ureal-smaller zero-ureal.rep-eq*)
have *lhs-0*: ($\sum_\infty v_0$::′s. *?lhs* v_0) = ($\sum_\infty v_0$::′s ∈ ({v_0} ∪ (−{v_0})). *?lhs* v_0)
by *auto*
have *lhs-1*: ... = ($\sum_\infty v_0$::′s ∈ {v_0}. *?lhs* v_0) + ($\sum_\infty v_0$::′s ∈ −{v_0}. *?lhs* v_0)
apply (*rule infsum-Un-disjoint*)
apply *auto*[*1*]
apply (*simp add*: *f10*)
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst pdrfun-product-summable*′)
by (*simp add*: *assms*)+
have *rhs-0*: ($\sum_\infty v_0$::′s. *?rhs* v_0) = ($\sum_\infty v_0$::′s ∈ ({v_0} ∪ (−{v_0})). *?rhs* v_0)
by *auto*
have *rhs-1*: ... = ($\sum_\infty v_0$::′s∈ ({v_0}). *?rhs* v_0) + ($\sum_\infty v_0$::′s∈ ((−{v_0})). *?rhs* v_0)
apply (*rule infsum-Un-disjoint*)
apply *auto*[*1*]
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst summable-on-cmult-left*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable*(*4*))
by (*simp*)+
have *lhs-0-rhs-0*: ($\sum_\infty v_0$::′s ∈ −{v_0}. *?lhs* v_0) ≤ ($\sum_\infty v_0$::′s∈ ((−{v_0})). *?rhs* v_0)
apply (*rule infsum-mono*)
apply (*simp add*: *f10*)
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst pdrfun-product-summable*′)
apply (*simp add*: *assms*)+
apply (*rule summable-on-subset-banach*[**where** *A*=*UNIV*])
apply (*subst summable-on-cmult-left*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable*(*4*))
apply (*simp*)+
by (*smt* (*verit*, *ccfv-SIG*) *P-NN*′ *Sup.SUP-cong a10 left-diff-distrib linordered-comm-semiring-strict-class.comm-mult-strict-left-mono ureal-lower-bound*)
have *lhs-2*: ($\sum_\infty v_0$::′s ∈ {v_0}. *?lhs* v_0) = *?lhs* v_0
by (*rule infsum-on-singleton*)
have *rhs-2*: ($\sum_\infty v_0$::′s∈ ({v_0}). *?rhs* v_0) = *?rhs* v_0
by (*rule infsum-on-singleton*)
have *lhs-1-rhs-1*: *?lhs* v_0 < *?rhs* v_0
by (*smt* (*verit*, *best*) *P-NN*′ *P-*v_0 *Sup.SUP-cong a10 linordered-comm-semiring-strict-class.comm-mult-strict-left-mon*
ureal2real-mono-strict ureal-lower-bound)

show *?thesis*
apply (*simp only*: *lhs-0 rhs-0 lhs-1 rhs-1*)
using *lhs-0-rhs-0 lhs-1-rhs-1 lhs-2 rhs-2* **by** *linarith*
qed
also have ... = $(\sum_\infty v_0::'s.\ ureal2real\ (P\ (s,\ v_0))) * r$
apply (*rule infsum-cmult-left*)
by (*simp add*: *assms pdrfun-prob-sum1-summable*(*4*))
also have ... = r
by (*simp add*: *assms pdrfun-prob-sum1-summable*(*3*))
then have *f13*: $(\ (\sum_\infty v_0::'s.$
$ureal2real\ (P\ (s,\ v_0)) * (ureal2real\ (?f\ n\ (v_0,\ s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ ?f\ n\ (v_0,\ s'))))) < r$
using *calculation* **by** *linarith*

have *f14*: *?lhs* = *ureal2real*
$(real2ureal\ (\ (\sum_\infty v_0::'s.$
$ureal2real\ (P\ (s,\ v_0)) * (ureal2real\ (?f\ n\ (v_0,\ s')) - ureal2real\ (\bigsqcap n::\mathbb{N}.\ ?f\ n\ (v_0,\ s'))))))$
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*simp add*: *rvfun-of-prfun-def*)
by (*simp add*: *f11*)
show *?lhs* $< r$
apply (*simp add*: *f14*)
using *f13* **by** (*smt* (*verit*, *del-insts*) *f11 infsum-nonneg mult-nonneg-nonneg ureal-lower-bound ureal-real2ureal-smaller*)
next
show $(0::\mathbb{R}) < r$
by (*simp add*: *a1*)
qed

then show $\exists\, no::\mathbb{N}.\ \forall\, n \geq no.$
$|ureal2real\ (\mathcal{F}\ b\ P\ (iter_p\ n\ b\ P\ 1_p)\ (s,\ s')) -$
$ureal2real\ (\mathcal{F}\ b\ P\ (\bigsqcap n::\mathbb{N}.\ iter_p\ n\ b\ P\ 1_p)\ (s,\ s'))|$
$< r$
apply (*simp add*: *loopfunc-def*)
by (*metis loopfunc-def f2*)
qed

lemma *inf-iterate-suc*: $(\bigsqcap x \in \{(iterate\ n\ b\ P\ 1_p) \mid n::nat.\ True\}.\ (\mathcal{F}\ b\ P\ x)) =$
$(\bigsqcap n::nat.\ (iterate\ (Suc\ n)\ b\ P\ 1_p))$
apply (*simp add*: *image-def*)
by *metis*

lemma *inf-iterate-subset-eq*:
$(\bigsqcap n::nat.\ (iterate\ (Suc\ n)\ b\ P\ 1_p)) = (\bigsqcap n::nat.\ (iterate\ n\ b\ P\ 1_p))$
proof −
have *f1*: $(\bigsqcap n::nat.\ (iterate\ (Suc\ n)\ b\ P\ 1_p)) = (\bigsqcap n \in \{1..\}.\ (iterate\ n\ b\ P\ 1_p))$
apply (*simp add*: *image-def*)
by (*metis atLeast-iff bot-nat-0.extremum not0-implies-Suc not-less-eq-eq utp-prob-rel-lattice.iterate.simps*(*2*))
have *insert* $(0::nat)\ \{1..\} = UNIV$
using *UNIV-nat-eq atLeast-Suc-greaterThan* **by** *auto*
then have *f2*: $(\bigsqcap n::nat.\ (iterate\ n\ b\ P\ 1_p)) = (\bigsqcap n::nat \in insert\ 0\ \{1..\}.\ (iterate\ n\ b\ P\ 1_p))$
by (*simp add*: *image-def*)
have *f3*: $(\bigsqcap n::nat \in insert\ 0\ \{1..\}.\ (iterate\ n\ b\ P\ 1_p)) = (iterate\ 0\ b\ P\ 1_p) \sqcap (\bigsqcap n \in \{1..\}.\ (iterate\ n\ b\ P\ 1_p))$
apply (*subst INF-insert*)
using *sup-commute* **by** *blast*

have *f4*: ... = ($\bigsqcap n \in \{1..\}$. (*iterate* n b P 1_p))
by (*smt* (*verit*, *del-insts*) *inf-top-left le-fun-def le-iff-inf pone-def ureal-top-greatest utp-prob-rel-lattice.iterate.simps*(*1*))
show *?thesis*
using *f1 f2 f3 f4* **by** *presburger*
qed

lemma *inf-iterate-continuous'*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate* n b P 1_p)
shows $\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p) = ($\bigsqcap x \in \{$(*iterate* n b P 1_p) | n::*nat*. *True*$\}$. ($\mathcal{F}$ b P x))
apply (*subst fun-eq-iff*)
apply (*auto*)
proof −
fix s s'
let $?f = \lambda n.$ $\mathcal{F}$ b P (*iterate* n b P 1_p)
have *decreasing-chain ?f*
by (*simp add*: *loopfunc-monoE assms decreasing-chain-def iterate-decreasing2*)
then have ($\lambda n.$ *ureal2real* ($?f$ n (s, s'))) $\longrightarrow$ (*ureal2real* ($\bigsqcap n$::$\mathbb{N}$. $?f$ n (s, s')))
by (*rule decreasing-chain-limit-is-glb*)
then have *ureal2real* ($\bigsqcap n$::$\mathbb{N}$. $?f$ n (s, s')) = *ureal2real* (($\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p)) (s, s'))
apply (*subst LIMSEQ-unique*[**where** X=($\lambda n.$ *ureal2real* ($?f$ n (s, s'))) **and** a = *ureal2real* ($\bigsqcap n$::$\mathbb{N}$. $?f$ n (s, s')) **and**
b = *ureal2real* (($\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p)) (s, s'))])
apply *meson*
apply (*subst inf-iterate-continuous-limit*)
using *assms*(*1*) **apply** *blast*
using *assms*(*2*) **apply** *blast*
by (*simp*)+

then have *f1*: ($\bigsqcap n$::$\mathbb{N}$. $?f$ n (s, s')) = (($\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p)) (s, s'))
using *ureal2real-eq* **by** *blast*

have *f2*: ($\bigsqcap x$::'s × 's ⇒ *ureal*∈ $\mathcal{F}$ b P ` $\{uu$::'s × 's ⇒ *ureal*. $\exists n$::$\mathbb{N}$. uu = $iter_p$ n b P $1_p\}$. x (s, s'))
= *Inf* (($\lambda x.$ x (s, s')) ` ($\mathcal{F}$ $b$ $P$ ` $\{uu$::'s × 's ⇒ *ureal*. $\exists n$::$\mathbb{N}$. uu = $iter_p$ n b P $1_p\}$))
by *auto*
have *f3*: ($\bigsqcap n$::$\mathbb{N}$. $\mathcal{F}$ b P ($iter_p$ n b P 1_p) (s, s')) = (*Inf* (*range* ($\lambda n.$ $\mathcal{F}$ b P ($iter_p$ n b P 1_p) (s, s'))))
by *simp*
have *f4*: (($\lambda x.$ x (s, s')) ` ($\mathcal{F}$ $b$ $P$ ` $\{uu$::'s × 's ⇒ *ureal*. $\exists n$::$\mathbb{N}$. uu = $iter_p$ n b P $1_p\}$)) =
(*range* ($\lambda n.$ $\mathcal{F}$ b P ($iter_p$ n b P 1_p) (s, s')))
apply (*simp add*: *image-def*)
by (*auto*)
show $\mathcal{F}$ b P ($\bigsqcap n$::$\mathbb{N}$. $iter_p$ n b P 1_p) (s, s') =
($\bigsqcap x$::'s × 's ⇒ *ureal*∈$\mathcal{F}$ b P ` $\{uu$::'s × 's ⇒ *ureal*. $\exists n$::$\mathbb{N}$. uu = $iter_p$ n b P $1_p\}$. x (s, s'))
apply (*simp add*: *f1*[*symmetric*])
using *f4* **by** *presburger*
qed

theorem *inf-iterate-continuous*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate* n b P 1_p)
shows $\mathcal{F}$ b P ($\bigsqcap n$::*nat*. *iterate* n b P 1_p) = ($\bigsqcap n$::*nat*. (*iterate* n b P 1_p))
apply (*subst inf-iterate-continuous'*)
apply (*simp add*: *assms*(*1*))
using *assms*(*2*) **apply** *auto*[*1*]
using *inf-iterate-suc inf-iterate-subset-eq* **by** *metis*

5.8.4 Kleene fixed-point theorem

lemma *fp-between-lfp-gfp*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
  assumes $\mathcal{F}$ b P $fp = fp$
  shows ($\bigsqcup n::\mathbb{N}.\ iter_p\ n\ b\ P\ 0_p) \le fp$
    $fp \le (\bigsqcap n::nat.\ (iterate\ n\ b\ P\ 1_p))$
proof −
  show ($\bigsqcup n::\mathbb{N}.\ iter_p\ n\ b\ P\ 0_p) \le fp$
    apply (*rule Sup-least*)
    apply (*simp add*: *image-def*)
    proof −
      fix x
      assume *a11*: $\exists xa::\mathbb{N}.\ x = iter_p\ xa\ b\ P\ 0_p$
      have $\forall n.\ iter_p\ n\ b\ P\ 0_p \le fp$
        apply (*rule allI*)
        apply (*induct-tac n*)
        apply (*simp add*: *ureal-bottom-least'*)
        by (*metis loopfunc-monoE assms*(*2*) *assms*(*1*) *utp-prob-rel-lattice.iterate.simps*(*2*))
      then show $x \le fp$
        using *a11* **by** *blast*
    qed

  show $fp \le (\bigsqcap n::\mathbb{N}.\ iter_p\ n\ b\ P\ 1_p)$
    apply (*rule Inf-greatest*)
    apply (*simp add*: *image-def*)
    proof −
      fix x
      assume *a11*: $\exists xa::\mathbb{N}.\ x = iter_p\ xa\ b\ P\ 1_p$
      have $\forall n.\ iter_p\ n\ b\ P\ 1_p \ge fp$
        apply (*rule allI*)
        apply (*induct-tac n*)
        apply (*simp add*: *ureal-top-greatest'*)
        by (*metis loopfunc-monoE assms*(*2*) *assms*(*1*) *utp-prob-rel-lattice.iterate.simps*(*2*))
      then show $fp \le x$
        using *a11* **by** *blast*
    qed
qed

theorem *sup-continuous-lfp-iteration*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
  assumes $\mathcal{FS}$ ($\lambda n.\ iterate\ n\ b\ P\ 0_p$)
  shows $while_p\ b\ do\ P\ od = (\bigsqcup n::nat.\ (iterate\ n\ b\ P\ 0_p))$
  apply (*simp add*: *pwhile-def*)
  apply (*rule lfp-eqI*)
  apply (*simp add*: *loopfunc-mono assms*)
  using *assms sup-iterate-continuous* **apply** *blast*
  by (*simp add*: *assms*(*1*) *fp-between-lfp-gfp*(*1*))

theorem *inf-continuous-gfp-iteration*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
  assumes $\mathcal{FS}$ ($\lambda n.\ iterate\ n\ b\ P\ 1_p$)
  shows ${while_p}^{\top}\ b\ do\ P\ od = (\bigsqcap n::nat.\ (iterate\ n\ b\ P\ 1_p))$
  apply (*simp add*: *pwhile-top-def*)
  apply (*rule gfp-eqI*)
  apply (*simp add*: *loopfunc-mono assms*)

using *assms inf-iterate-continuous* **apply** *blast*
by (*simp add*: *assms*(*1*) *fp-between-lfp-gfp*(*2*))

5.8.5 Unique fixed point

lemma *unique-fixed-point*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate n b P* 0_p)
assumes ($\bigsqcap n$::*nat*. (*iterate n b P* 1_p)) = ($\bigsqcup n$::*nat*. (*iterate n b P* 0_p))

shows $\exists!$ *fp*. $\mathcal{F}$ *b P fp* = *fp*
apply (*simp add*: *Ex1-def*)
apply (*rule-tac x* = ($\bigsqcup n$::*nat*. (*iterate n b P* 0_p)) **in** *exI*)
apply (*rule conjI*)
using *assms sup-iterate-continuous* **apply** *blast*
proof (*auto*)
fix *y* :: 's × 's ⇒ *ureal*
assume *a1*: $\mathcal{F}$ *b P y* = *y*
from *a1* **have** *f1*: ($\bigsqcup n$::$\mathbb{N}$. $iter_p$ *n b P* 0_p) ≤ *y*
by (*metis assms*(*1*) *fp-between-lfp-gfp*(*1*))
from *a1* **have** *f2*: *y* ≤ ($\bigsqcap n$::*nat*. (*iterate n b P* 1_p))
by (*metis assms*(*1*) *fp-between-lfp-gfp*(*2*))
then show *y* = ($\bigsqcup n$::$\mathbb{N}$. $iter_p$ *n b P* 0_p)
by (*simp add*: *assms*(*3*) *f1 order-antisym*)
qed

theorem *unique-fixed-point-lfp-gfp*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
assumes $\mathcal{FS}$ ($\lambda n.$ *iterate n b P* 0_p)
assumes ($\bigsqcap n$::*nat*. (*iterate n b P* 1_p)) = ($\bigsqcup n$::*nat*. (*iterate n b P* 0_p))
assumes $\mathcal{F}$ *b P fp* = *fp*
shows $while_p$ *b do P od* = *fp*
$while_p{}^\top$ *b do P od* = *fp*
apply (*smt* (*verit*) *Collect-cong Sup.SUP-cong assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)
sup-continuous-lfp-iteration sup-iterate-continuous unique-fixed-point)
by (*smt* (*z3*) *Collect-cong loopfunc-mono Sup.SUP-cong assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *gfp-unfold*

pwhile-top-def unique-fixed-point)

lemma *iterate-bot-leq-top*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
shows *iterate n b P* 0_p ≤ *iterate n b P* 1_p
apply (*induction n*)
apply (*simp*)
apply (*simp add*: *ureal-top-greatest'*)
apply (*simp*)
by (*simp add*: *loopfunc-monoE assms*)

lemma *iterate-top-is-prob*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
shows *is-prob* ((•(*rvfun-of-prfun* (*iterate n b P* 0_p)) + •(*rvfun-of-prfun* (*iterdiff n b P* 1_p)))$_e$)
apply (*induction n*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *prfun-in-0-1'*)
apply (*simp add*: *one-ureal.rep-eq pone-def pzero-def rvfun-of-prfun-def ureal2real-def zero-ureal.rep-eq*)

apply (*simp*)
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *pcond-def*)
apply (*simp only*: *prfun-skip'*)
apply (*simp only*: *pfun-defs*)
apply (*subst rvfun-seqcomp-inverse*)
using *assms* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-seqcomp-inverse*)
using *assms* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-pcond-inverse*)
using *assms rvfun-seqcomp-dist-is-prob ureal-is-prob* **apply** *blast*
using *rvfun-skip-f-is-prob* **apply** *blast*
apply (*subst rvfun-pcond-inverse*)
using *assms rvfun-seqcomp-dist-is-prob ureal-is-prob* **apply** *blast*
using *ureal-is-prob* **apply** *blast*
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)
defer
apply (*simp add*: *prfun-in-0-1'*)
apply (*simp add*: *rvfun-of-prfun-def ureal2real-def zero-ureal.rep-eq*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)
defer
apply (*simp add*: *prfun-in-0-1'*)
apply (*simp add*: *rvfun-of-prfun-def ureal2real-def zero-ureal.rep-eq*)
apply (*pred-auto*)
proof −
fix *n ba*
assume *a1*: $\forall (a::'s)\ ba::'s.$
$(0::\mathbb{R}) \le$ *rvfun-of-prfun* ($iter_p$ *n b P* **0**) (*a*, *ba*) + *rvfun-of-prfun* (*iterdiff n b P* **1**) (*a*, *ba*) ∧
rvfun-of-prfun ($iter_p$ *n b P* **0**) (*a*, *ba*) + *rvfun-of-prfun* (*iterdiff n b P* **1**) (*a*, *ba*) $\le (1::\mathbb{R})$
have ($\sum_\infty v_0::'s.$ *rvfun-of-prfun P* (*ba*, v_0) ∗ *rvfun-of-prfun* ($iter_p$ *n b P* **0**) (v_0, *ba*)) +
($\sum_\infty v_0::'s.$ *rvfun-of-prfun P* (*ba*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*)) =
($\sum_\infty v_0::'s.$ *rvfun-of-prfun P* (*ba*, v_0) ∗ *rvfun-of-prfun* ($iter_p$ *n b P* **0**) (v_0, *ba*) +
rvfun-of-prfun P (*ba*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*))
apply (*rule infsum-add*[*symmetric*])
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule pdrfun-product-summable'*)
apply (*simp add*: *assms*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule pdrfun-product-summable'*)
by (*simp add*: *assms*)
also have ... = ($\sum_\infty v_0::'s.$ *rvfun-of-prfun P* (*ba*, v_0) ∗ (*rvfun-of-prfun* ($iter_p$ *n b P* **0**) (v_0, *ba*) +
rvfun-of-prfun (*iterdiff n b P* **1**) (v_0, *ba*)))
by (*simp add*: *distrib-left*)
also have ... ≤ ($\sum_\infty v_0::'s.$ *rvfun-of-prfun P* (*ba*, v_0))
apply (*rule infsum-mono*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule pdrfun-product-summable'-1*)
using *assms*(*1*) **apply** *blast*
apply (*smt* (*verit*, *ccfv-SIG*) *SEXP-def a1 is-prob-def rvfun-of-prfun-def taut-def*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable'*(*4*))
by (*simp add*: *a1 mult-left-le prfun-in-0-1'*)

also have ... = *1*
  by (*simp add*: *assms pdrfun-prob-sum1-summable*′(*3*))
then show ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*ba*, v_0) ∗ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*)) +
  ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*ba*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*)) ≤ (*1*::ℝ)
  using *calculation* **by** *presburger*
next
fix *n a ba*
assume *a1*: ∀ (*a*::′*s*) *ba*::′*s*.
  (*0*::ℝ) ≤ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (*a*, *ba*) + *rvfun-of-prfun* (*iterdiff n b P* **1**) (*a*, *ba*) ∧
  rvfun-of-prfun (*iter*$_p$ *n b P* **0**) (*a*, *ba*) + *rvfun-of-prfun* (*iterdiff n b P* **1**) (*a*, *ba*) ≤ (*1*::ℝ)
have ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*)) +
  ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*)) =
 ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*) +
  rvfun-of-prfun P (*a*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*))
  apply (*rule infsum-add*[*symmetric*])
  apply (*simp add*: *rvfun-of-prfun-def*)
  apply (*rule pdrfun-product-summable*′)
  apply (*simp add*: *assms*)
  apply (*simp add*: *rvfun-of-prfun-def*)
  apply (*rule pdrfun-product-summable*′)
  by (*simp add*: *assms*)
also have ... = ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ (*rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*) +
  rvfun-of-prfun (*iterdiff n b P* **1**) (v_0, *ba*)))
  by (*simp add*: *distrib-left*)
also have ... ≤ ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0))
  apply (*rule infsum-mono*)
  apply (*simp add*: *rvfun-of-prfun-def*)
  apply (*rule pdrfun-product-summable*′*-1*)
  using *assms*(*1*) **apply** *blast*
  apply (*smt* (*verit*, *ccfv-SIG*) *SEXP-def a1 is-prob-def rvfun-of-prfun-def taut-def*)
  apply (*simp add*: *assms pdrfun-prob-sum1-summable*′(*4*))
  by (*simp add*: *a1 mult-left-le prfun-in-0-1*′)
also have ... = *1*
  by (*simp add*: *assms pdrfun-prob-sum1-summable*′(*3*))
then show ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*)) +
  ($\sum_\infty v_0$::′*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*))
  ≤ (*1*::ℝ)
  using *calculation* **by** *presburger*
qed

lemma *iterate-top-is-prob*′:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
shows ∀ *s*. *ureal2real* (*iter*$_p$ *n b P* **0** *s*) + *ureal2real* (*iterdiff n b P* **1** *s*) ≤ (*1*::ℝ)
proof −
have *is-prob* ((•(*rvfun-of-prfun* (*iterate n b P* 0_p)) + •(*rvfun-of-prfun* (*iterdiff n b P* 1_p)))$_e$)
  using *iterate-top-is-prob assms* **by** *blast*
then have ∀ *s*. *rvfun-of-prfun* (*iter*$_p$ *n b P* 0_p) *s* + *rvfun-of-prfun* (*iterdiff n b P* 1_p) *s* ≤ *1*
  apply (*subst* (*asm*) *dist-defs taut-def*)
  by (*simp add*: *taut-def*)
then show *?thesis*
  apply (*subst* (*asm*) *rvfun-of-prfun-def*)
  apply (*subst* (*asm*) *rvfun-of-prfun-def*)
  by (*metis SEXP-def order-antisym ureal-bottom-least ureal-bottom-least*′ *ureal-top-greatest ureal-top-greatest*′)
qed

lemma *iterate-top-eq-bot-plus*:
assumes *is-final-distribution* (*rvfun-of-prfun* (P::('s, 's) *prfun*))
shows *iterate n b P* 1_p = ($\bullet$(*iterate n b P* 0_p) + $\bullet$(*iterdiff n b P* 1_p))$_e$
apply (*induction n*)
apply (*simp add*: *pzero-def*)
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *pcond-def*)
apply (*simp only*: *prfun-skip'*)
apply (*simp only*: *pfun-defs*)
apply (*subst rvfun-seqcomp-inverse*)
using *assms* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-seqcomp-inverse*)
using *assms* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*subst rvfun-seqcomp-inverse*)
using *assms* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*subst fun-eq-iff*)
apply (*expr-auto*)
defer
apply (*simp add*: *ureal-defs*)
apply (*metis add.right-neutral ereal2ureal-def ureal-zero-0 zero-ereal-def zero-ureal-def*)
defer
apply (*metis SEXP-def add-0 nle-le real2ureal-def rvfun-of-prfun-def ureal-lower-bound ureal-real2ureal-smaller zero-ereal-def zero-ureal-def*)
apply (*pred-auto*)
proof −
fix *n ba*
assume *a1*: $\forall$ (a::'s) ba::'s. $iter_p$ *n b P* $\mathbf{1}$ (a, ba) = $iter_p$ *n b P* $\mathbf{0}$ (a, ba) + *iterdiff n b P* $\mathbf{1}$ (a, ba)
let *?lhs* = ($\sum_\infty v_0$::'s.
rvfun-of-prfun P (ba, v_0) $*$
rvfun-of-prfun (λa::'s $\times$'s. $iter_p$ *n b P* $\mathbf{0}$ a + *iterdiff n b P* $\mathbf{1}$ a) (v_0, ba))
let *?rhs-1* = ($\sum_\infty v_0$::'s. *rvfun-of-prfun P* (ba, v_0) $*$ *rvfun-of-prfun* ($iter_p$ *n b P* $\mathbf{0}$) (v_0, ba))
let *?rhs-2* = ($\sum_\infty v_0$::'s. *rvfun-of-prfun P* (ba, v_0) $*$ *rvfun-of-prfun* (*iterdiff n b P* $\mathbf{1}$) (v_0, ba))

have *f0*: $\forall$ v_0. *rvfun-of-prfun* (λa::'s $\times$'s. $iter_p$ *n b P* $\mathbf{0}$ a + *iterdiff n b P* $\mathbf{1}$ a) (v_0, ba)
= (*rvfun-of-prfun* (λa::'s $\times$'s. $iter_p$ *n b P* $\mathbf{0}$ a) (v_0, ba) +
rvfun-of-prfun (λa::'s $\times$'s. *iterdiff n b P* $\mathbf{1}$ a) (v_0, ba))
apply (*simp add*: *ureal-defs*)
apply (*subst ureal2ereal-add-dist*)
apply (*rule ureal2real-add-leq-1-ureal2ereal*)
using *iterate-top-is-prob'* *assms* **apply** *blast*
by (*metis abs-ereal-ge0 atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-times*(*1*) *nle-le real-of-ereal-add ureal2ereal*)
have *f1*: *?lhs* = ($\sum_\infty v_0$::'s.
rvfun-of-prfun P (ba, v_0) $*$
(*rvfun-of-prfun* (λa::'s $\times$'s. $iter_p$ *n b P* $\mathbf{0}$ a) (v_0, ba) +
rvfun-of-prfun (λa::'s $\times$'s. *iterdiff n b P* $\mathbf{1}$ a) (v_0, ba)))
apply (*rule infsum-cong*)
using *f0* **by** *presburger*
have *f2*: ... = *?rhs-1* + *?rhs-2*
apply (*simp add*: *distrib-left*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule infsum-add*)

by (*simp add*: *assms pdrfun-product-summable'*)+
have *f3*: *?lhs* $\leq$ ($\sum_\infty v_0$::'*s*. *rvfun-of-prfun P* (*ba*, v_0))
apply (*rule infsum-mono*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule pdrfun-product-summable'-1*)+
using *assms* **apply** *force*
apply (*simp add*: *is-prob-def ureal-lower-bound ureal-upper-bound*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable'*(*4*))
by (*meson mult-right-le-one-le prfun-in-0-1'*)
have *f4*: ... = *1*
by (*simp add*: *assms pdrfun-prob-sum1-summable'*(*3*))
show *real2ureal ?lhs* = *real2ureal ?rhs-1* + *real2ureal ?rhs-2*
apply (*simp add*: *f1*)
apply (*simp add*: *f2*)
apply (*subst real2ureal-add-dist*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)+
apply (*simp add*: *f2*[*symmetric*])
apply (*simp add*: *f1*[*symmetric*])
using *f3 f4* **apply** *auto*[*1*]
by *simp*
next
fix *n a ba*
assume *a1*: $\forall$(*a*::'*s*) *ba*::'*s*. *iter*$_p$ *n b P* **1** (*a*, *ba*) = *iter*$_p$ *n b P* **0** (*a*, *ba*) + *iterdiff n b P* **1** (*a*, *ba*)
let *?lhs* = ($\sum_\infty v_0$::'*s*.
rvfun-of-prfun P (*a*, v_0) ∗
rvfun-of-prfun (λ*a*::'*s* × '*s*. *iter*$_p$ *n b P* **0** *a* + *iterdiff n b P* **1** *a*) (v_0, *ba*))
let *?rhs-1* = ($\sum_\infty v_0$::'*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iter*$_p$ *n b P* **0**) (v_0, *ba*))
let *?rhs-2* = ($\sum_\infty v_0$::'*s*. *rvfun-of-prfun P* (*a*, v_0) ∗ *rvfun-of-prfun* (*iterdiff n b P* **1**) (v_0, *ba*))

have *f0*: $\forall v_0$. *rvfun-of-prfun* (λ*a*::'*s* × '*s*. *iter*$_p$ *n b P* **0** *a* + *iterdiff n b P* **1** *a*) (v_0, *ba*)
= (*rvfun-of-prfun* (λ*a*::'*s* × '*s*. *iter*$_p$ *n b P* **0** *a*) (v_0, *ba*) +
rvfun-of-prfun (λ*a*::'*s* × '*s*. *iterdiff n b P* **1** *a*) (v_0, *ba*))
apply (*simp add*: *ureal-defs*)
apply (*subst ureal2ereal-add-dist*)
apply (*rule ureal2real-add-leq-1-ureal2ereal*)
using *iterate-top-is-prob' assms* **apply** *blast*
by (*metis abs-ereal-ge0 atLeastAtMost-iff ereal-less-eq*(*1*) *ereal-times*(*1*) *nle-le real-of-ereal-add ureal2ereal*)
have *f1*: *?lhs* = ($\sum_\infty v_0$::'*s*.
rvfun-of-prfun P (*a*, v_0) ∗
(*rvfun-of-prfun* (λ*a*::'*s* × '*s*. *iter*$_p$ *n b P* **0** *a*) (v_0, *ba*) +
rvfun-of-prfun (λ*a*::'*s* × '*s*. *iterdiff n b P* **1** *a*) (v_0, *ba*)))
apply (*rule infsum-cong*)
using *f0* **by** *presburger*
have *f2*: ... = *?rhs-1* + *?rhs-2*
apply (*simp add*: *distrib-left*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule infsum-add*)
by (*simp add*: *assms pdrfun-product-summable'*)+
have *f3*: *?lhs* $\leq$ ($\sum_\infty v_0$::'*s*. *rvfun-of-prfun P* (*a*, v_0))
apply (*rule infsum-mono*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*rule pdrfun-product-summable'-1*)+
using *assms* **apply** *force*
apply (*simp add*: *is-prob-def ureal-lower-bound ureal-upper-bound*)
apply (*simp add*: *assms pdrfun-prob-sum1-summable'*(*4*))

by (*meson mult-right-le-one-le prfun-in-0-1'*)
have *f4*: ... = *1*
by (*simp add*: *assms pdrfun-prob-sum1-summable'*(*3*))
show *real2ureal ?lhs* = *real2ureal ?rhs-1* + *real2ureal ?rhs-2*
apply (*simp add*: *f1*)
apply (*simp add*: *f2*)
apply (*subst real2ureal-add-dist*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)+
apply (*simp add*: *f2*[*symmetric*])
apply (*simp add*: *f1*[*symmetric*])
using *f3 f4* **apply** *auto*[*1*]
by *simp*
qed

lemma *iterdiff-decreasing*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(*'s*, *'s*) *prfun*))
shows *decseq* (λn. ((*iterdiff n b P* 1_p) *s*))
apply (*simp add*: *decseq-def*)
proof (*auto*)
fix *m n* :: $\mathbb{N}$
assume *a1*: $m \leq n$
obtain *nn* **where** *P-nn*: $m + nn = n$
using *nat-le-iff-add a1* **by** *auto*
have *f1*: $\forall$ *nn*. (*iterdiff nn b P* 1_p) $\geq$ (*iterdiff* (*nn* + *1*) *b P* 1_p)
proof
fix *nn*
show *iterdiff* (*nn* + (*1*::$\mathbb{N}$)) *b P* 1_p $\leq$ *iterdiff nn b P* 1_p
apply (*induction nn*)
apply (*simp add*: *ureal-top-greatest'*)
apply (*simp*)
apply (*subst prfun-pcond-mono*, *auto*)
apply (*subst prfun-pseqcomp-mono'*, *auto*)
apply (*subst pdrfun-product-summable'-1*, *auto*)
apply (*simp add*: *assms*)
apply (*simp add*: *is-prob-def ureal-lower-bound ureal-upper-bound*)
apply (*subst pdrfun-product-summable'-1*, *auto*)
apply (*simp add*: *assms*)
by (*simp add*: *is-prob-def ureal-lower-bound ureal-upper-bound*)
qed
have *f2*: (*iterdiff m b P* 1_p) $\geq$ (*iterdiff* (*m* + *nn*) *b P* 1_p)
apply (*induction nn*)
apply *force*
using *f1 order.trans* **by** *auto*
show *iterdiff n b P* 1_p *s* $\leq$ *iterdiff m b P* 1_p *s*
using *P-nn f2 le-fun-def* **by** *fastforce*
qed

lemma *iterate-sup-inf-eq*:
assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(*'s*, *'s*) *prfun*))
assumes $\mathcal{FS}$ (λn. *iterate n b P* 0_p)
assumes $\forall$ *s*. (λn. *ureal2real* ((*iterdiff n b P* 1_p) *s*)) $\longrightarrow$ *0*
shows ($\bigsqcap$ *n*::*nat*. (*iterate n b P* 1_p)) = ($\bigsqcup$ *n*::*nat*. (*iterate n b P* 0_p))
proof −
let *?f1* = λn. (*iterate n b P* 0_p)
let *?f2* = λn. (*iterate n b P* 1_p)

have *f1*: $\forall s.\ (\lambda n.\ ureal2real\ (?f1\ n\ s)) \longlonglongrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ ?f1\ n\ s))$
  apply (*auto*, *rule increasing-chain-limit-is-lub*)
  using *assms*(*1*) *iterate-increasing-chain* **by** *blast*

have *f2*: $\forall s.\ (\lambda n.\ ureal2real\ (?f2\ n\ s)) \longlonglongrightarrow (ureal2real\ (\bigsqcap n::\mathbb{N}.\ ?f2\ n\ s))$
  apply (*auto*, *rule decreasing-chain-limit-is-glb*)
  using *assms*(*1*) *iterate-decreasing-chain* **by** *blast*

have *f3*: $\forall n.\ ?f2\ n = (\bullet(?f1\ n) + \bullet(iterdiff\ n\ b\ P\ 1_p))_e$
  using *assms*(*1*) *iterate-top-eq-bot-plus* **by** *blast*

have *f4*: $\forall s.\ (\lambda n.\ ureal2real\ (?f2\ n\ s)) = (\lambda n.\ ureal2real\ (?f1\ n\ s + (iterdiff\ n\ b\ P\ 1_p)\ s))$
  using *f3* **by** *simp*

have *f5*: $\forall s.\ (\lambda n.\ ureal2real\ (?f1\ n\ s + (iterdiff\ n\ b\ P\ 1_p)\ s)) = (\lambda n.\ ureal2real\ (?f1\ n\ s) + ureal2real\ ((iterdiff\ n\ b\ P\ 1_p)\ s))$
  apply (*subst fun-eq-iff*)
  apply (*auto*)
  apply (*rule ureal2real-add-dist*)
  using *iterate-top-is-prob′* **by** (*metis assms*(*1*) *order-antisym ureal-bottom-least*
    ureal-bottom-least′ ureal-top-greatest ureal-top-greatest′)

have *f6*: $\forall s.\ (\lambda n.\ ureal2real\ (?f2\ n\ s)) \longlonglongrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ ?f1\ n\ s)) + 0$
  apply (*rule allI*)
  apply (*simp only*: *f4 f5*)
  apply (*rule tendsto-add*)
  using *f1* **apply** *blast*
  by (*simp add*: *assms*(*3*))

have $\forall s.\ (ureal2real\ (\bigsqcup n::\mathbb{N}.\ ?f1\ n\ s)) = (ureal2real\ (\bigsqcap n::\mathbb{N}.\ ?f2\ n\ s))$
proof
  fix *s*
  show $ureal2real\ (\bigsqcup n::\mathbb{N}.\ iter_p\ n\ b\ P\ 0_p\ s) = ureal2real\ (\bigsqcap n::\mathbb{N}.\ iter_p\ n\ b\ P\ 1_p\ s)$
  apply (*rule LIMSEQ-unique*[**where** $X = (\lambda n.\ ureal2real\ (?f2\ n\ s))$])
  using *f6* **apply** *fastforce*
  using *f2* **by** *blast*
qed
then have $\forall s.\ (\bigsqcup n::\mathbb{N}.\ ?f1\ n\ s) = (\bigsqcap n::\mathbb{N}.\ ?f2\ n\ s)$
  using *ureal2real-eq* **by** *blast*
then show $(\bigsqcap n::\mathbb{N}.\ iter_p\ n\ b\ P\ 1_p) = (\bigsqcup n::\mathbb{N}.\ iter_p\ n\ b\ P\ 0_p)$
  apply (*subst fun-eq-iff*)
  apply (*rule allI*)
  by (*metis INF-apply SUP-apply*)
qed

theorem *unique-fixed-point-lfp-gfp′*:
  assumes *is-final-distribution* (*rvfun-of-prfun* (*P*::(′*s*, ′*s*) *prfun*))
  assumes $\mathcal{FS}\ (\lambda n.\ iterate\ n\ b\ P\ 0_p)$
  assumes $\forall s.\ (\lambda n.\ ureal2real\ ((iterdiff\ n\ b\ P\ 1_p)\ s)) \longlonglongrightarrow 0$
  assumes $\mathcal{F}\ b\ P\ fp = fp$
  shows $while_p\ b\ do\ P\ od = fp$
    $while_p{}^{\top}\ b\ do\ P\ od = fp$
  using *assms iterate-sup-inf-eq unique-fixed-point-lfp-gfp*(*1*) **apply** *blast*
  using *assms iterate-sup-inf-eq unique-fixed-point-lfp-gfp*(*2*) **by** *blast*

end

6 The Hehner's predicative probabilistic programming in UTP

theory *utp-prob-rel*
  imports
    utp-iverson-bracket
    utp-distribution
    utp-prob-rel-lattice
    utp-prob-rel-lattice-laws
begin end

Acknowledgements.

References

[1] E. C. R. Hehner, “A probability perspective,” vol. 23, no. 4, pp. 391–419. [Online]. Available: https://doi.org/10.1007/s00165-010-0157-0