Probabilistic Relations Programming Examples

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 lists some examples that use our probabilistic relations, based on Hehner's predicative probabilistic programming [1], for reasoning.

Contents

1 Doctor Who's Tardis Attacker

theory *utp-prob-rel-lattice-dwta*
  imports
    UTP-prob-relations.utp-prob-rel
begin

unbundle *UTP-Syntax*

declare [[*show-types*]]

1.1 Doctor Who's Tardis Attacker

Example 13 from Jim's draft report. Two robots, the Cyberman C and the Dalek D, attack Doctor Whos Tardis once a day between them. C has a probability 1/2 of a successful attack, while D has a probability 3/10 of a successful attack. C attacks more often than D, with a probability of 3/5 on a particular day (and so D attacks with a probability of 2/5 on that day). What is the probability that there is a successful attack today?

1.1.1 State space

datatype *Attacker* = *C* | *D*
find-theorems *name*: *Attacker.induct*

datatype *Status* = *S* | *F*

alphabet *DWTA-state* =
  r:: *Attacker*
  a:: *Status*

find-theorems *name*: *DWTA-state.induct*
find-theorems *name*: *DWTA-state.select-convs*

1.1.2 Finite

lemma *attacker-finite*: *finite* (*UNIV*::*Attacker set*)
  by (*metis Attacker.induct Collect-empty-eq Collect-mem-eq DiffD2 Diff-infinite-finite finite.emptyI finite-insert insertCI*)

lemma *status-finite*: *finite* (*UNIV*::*Status set*)
by (*metis Status.induct Collect-empty-eq Collect-mem-eq DiffD2 Diff-infinite-finite finite.emptyI finite-insert insertCI*)

lemma *dwta-state-univ-rewrite*: $(UNIV::DWTA\text{-}state\ set) = \{(\!|r_v = rr,\ a_v = aa|\!) \mid (rr::Attacker)\ (aa::Status).\ True\ \}$
  by (*metis* (*mono-tags*, *lifting*) *CollectI DWTA-state.cases UNIV-eq-I*)

lemma *dwta-state-subset-finite*: *finite* $\{(\!|r_v = rr,\ a_v = aa|\!) \mid (rr::Attacker)\ (aa::Status).\ True \wedge True\ \}$
  apply (*rule finite-image-set2*[**where** $P{=}\lambda x.\ True$ **and** $Q{=}\lambda x.\ True$ **and** $f = \lambda x\ y.\ (\!|r_v = x,\ a_v = y|\!)$])
  using *attacker-finite status-finite* **by** *force+*

lemma *dwta-state-finite*: *finite* $(UNIV::DWTA\text{-}state\ set)$
  apply (*simp add*: *dwta-state-univ-rewrite*)
  using *dwta-state-subset-finite* **by** *presburger*

lemma *dwta-infsum-sum*: $(\sum_\infty s::DWTA\text{-}state.\ f\ s) = sum\ f\ (UNIV::DWTA\text{-}state\ set)$
  using *dwta-state-finite* **by** (*simp*)

1.1.3 Laws

term $(r := C)::DWTA\text{-}state\ prhfun$

term $(r := C)\ ;\ (if_p\ (1/2)\ then\ (a := S)\ else\ (a := F))$

definition *dwta* :: (*DWTA-state*, *DWTA-state*) *prfun* **where**
dwta =
  $(if_p\ (3/5)$
    $then\ ((r := C)\ ;\ (if_p\ (\ 1/2)\ then\ (a := S)\ else\ (a := F)))$
    $else\ ((r := D)\ ;\ (if_p\ (3/10)\ then\ (a := S)\ else\ (a := F)))$
)

thm *dwta-def*

term C
term $(r^> = C)_e$
term $(\$r^> = C)_e$
term $[\![(r^> = C)_e]\!]_\mathcal{I}$
term $[\![\ r^> = C \wedge a^> = S\ ]\!]_{\mathcal{I}e}$
term $(r := C)::DWTA\text{-}state\ prhfun$

lemma *dwta-scomp-simp*:
  $(((r := C)::DWTA\text{-}state\ prhfun);\ (a := S)) = prfun\text{-}of\text{-}rvfun\ ([\![\ \$r^> = C \wedge \$a^> = S\ ]\!]_{\mathcal{I}e})$
  apply (*simp add*: *prfun-passign-comp*)
  apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
  by (*pred-auto*)

lemma *dwta-infsum-two-instances*: $(\sum_\infty s::DWTA\text{-}state.$
    $p * (if\ (\!|r_v = rr,\ a_v = S|\!) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ +$
    $q * (if\ (\!|r_v = rr,\ a_v = F|\!) = s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))) = (p + q)$
  apply (*simp add*: *dwta-infsum-sum*)
  apply (*subst sum.subset-diff*[**where** $A{=}UNIV$ **and** $B{=}\{(\!|r_v = rr,\ a_v = S|\!),\ (\!|r_v = rr,\ a_v = F|\!)\}$])
  apply (*simp add*: *dwta-state-finite*)+
  apply (*subst sum-nonneg-eq-0-iff*)
  using *dwta-state-finite* **apply** *blast*
  apply *auto*[*1*]
  by *auto*

lemma *dwta-infsum-two-instances'*: $(\sum_\infty s::DWTA\text{-}state.$

$p*$ (*if* r_v $s = rr \wedge a_v$ $s = S$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) +
$q*$ (*if* r_v $s = rr \wedge a_v$ $s = F$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)) = $(p + q)$
apply (*simp add*: *dwta-infsum-sum*)
apply (*subst sum.subset-diff*[**where** *A*=*UNIV* **and** *B*={⦇r_v = *rr*, a_v = *S*⦈, ⦇r_v = *rr*, a_v = *F*⦈}])
apply (*simp add*: *dwta-state-finite*)+
apply (*subst sum-nonneg-eq-0-iff*)
using *dwta-state-finite* **apply** *blast*
apply *auto*[*1*]
by *auto*

lemma *dwta-attack-status*:
shows ((r := «*attacker*»)::(*DWTA-state*, *DWTA-state*) *prfun*) ; (*if*$_p$ («p») *then* (a := *S*) *else* (a := *F*))
= *prfun-of-rvfun* (*ureal2real* «p» $*$ ⟦ $\$r^>$ = «*attacker*» $\wedge$ $\$a^>$ = *S* ⟧$_{\mathcal{I}e}$ +
(*1* − *ureal2real* «p»)$*$ ⟦ $\$r^>$ = «*attacker*» $\wedge$ $\$a^>$ = *F* ⟧$_{\mathcal{I}e}$
)$_e$
proof −
have *f1*: *rvfun-of-prfun* [λs::*DWTA-state* × *DWTA-state*. *p*]$_e$ = (λ*s*. *ureal2real p*)
by (*simp add*: *SEXP-def rvfun-of-prfun-def*)

show *?thesis*
apply (*simp add*: *prfun-seqcomp-left-one-point*)
apply (*simp add*: *pchoice-def*)
apply (*simp add*: *passigns-def pchoice-def*)
apply (*simp add*: *rvfun-assignment-inverse*)
apply (*simp only*: *f1*)
apply (*subst rvfun-pchoice-inverse-c*)
using *rvfun-assignment-is-prob* **apply** *blast*+
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
by (*pred-auto*)
qed

lemma *dwta-simp*: *dwta* = *prfun-of-rvfun* (
3/*10* $*$ ⟦ $\$r^>$ = *C* $\wedge$ $\$a^>$ = *S* ⟧$_{\mathcal{I}e}$ +
3/*10* $*$ ⟦ $\$r^>$ = *C* $\wedge$ $\$a^>$ = *F* ⟧$_{\mathcal{I}e}$ +
6/*50* $*$ ⟦ $\$r^>$ = *D* $\wedge$ $\$a^>$ = *S* ⟧$_{\mathcal{I}e}$ +
14/*50* $*$ ⟦ $\$r^>$ = *D* $\wedge$ $\$a^>$ = *F* ⟧$_{\mathcal{I}e}$
)$_e$
apply (*simp add*: *dwta-def*)
apply (*subst dwta-attack-status*[**where** *p* = *ereal2ureal* ((*1*::*ereal*) / *ereal* (*2*::$\mathbb{R}$)) **and** *attacker* = *C*])
apply (*subst dwta-attack-status*[**where** *p* = *ereal2ureal* (*ereal* ((*3*::$\mathbb{R}$) / (*10*::$\mathbb{R}$))) **and** *attacker* = *D*])
apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *is-prob-def iverson-bracket-def ureal-lower-bound ureal-upper-bound*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *is-prob-def iverson-bracket-def ureal-lower-bound ureal-upper-bound*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *dist-defs expr-defs lens-defs ureal-defs*)
apply (*subst fun-eq-iff*)
apply (*auto*)
apply (*simp add*: *real2uereal-inverse*)
apply (*simp add*: *ereal-1-div*)
apply (*simp add*: *real2uereal-inverse′*)
apply (*simp add*: *real2uereal-inverse′*)+
by (*simp add*: *ereal-1-div real2uereal-inverse′*)+

lemma *dwta-attack-by-C*: *rvfun-of-prfun dwta* $;_f$ ($[\![r^< = C]\!]_{\mathcal{I}e}$) = $(6/10)_e$
apply (*simp add*: *dwta-simp*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*simp add*: *dwta-infsum-sum*)
apply (*subst sum.subset-diff*[**where** A=*UNIV* **and** B={$(\!|r_v = C,\ a_v = S|\!)$, $(\!|r_v = C,\ a_v = F|\!)$, $(\!|r_v = D,\ a_v = S|\!)$, $(\!|r_v = D,\ a_v = F|\!)$}])
apply (*simp add*: *dwta-state-finite*)+
apply (*expr-auto*)
apply (*subst sum-nonneg-eq-0-iff*)
using *dwta-state-finite* **apply** *blast*
apply *auto*[*1*]
by (*smt* (*z3*) *Attacker.exhaust DWTA-state.surjective DiffD2 Status.exhaust insertCI old.unit.exhaust*)

lemma *dwta-successful-attack*: *rvfun-of-prfun dwta* $;_f$ ($[\![a^< = S]\!]_{\mathcal{I}e}$) = $(21/50)_e$
apply (*simp add*: *dwta-simp*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*simp add*: *dwta-infsum-sum*)
apply (*subst sum.subset-diff*[**where** A=*UNIV* **and** B={$(\!|r_v = C,\ a_v = S|\!)$, $(\!|r_v = C,\ a_v = F|\!)$, $(\!|r_v = D,\ a_v = S|\!)$, $(\!|r_v = D,\ a_v = F|\!)$}])
apply (*simp add*: *dwta-state-finite*)+
apply (*expr-auto*)
apply (*subst sum-nonneg-eq-0-iff*)
using *dwta-state-finite* **apply** *blast*
apply *auto*[*1*]
by (*smt* (*z3*) *Attacker.exhaust DWTA-state.surjective DiffD2 Status.exhaust insertCI old.unit.exhaust*)

lemma *dwta-successful-attack-by-D*: *rvfun-of-prfun dwta* $;_f$ ($[\![r^< = D \wedge a^< = S]\!]_{\mathcal{I}e}$) = $(3/25)_e$
apply (*simp add*: *dwta-simp*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs expr-defs*)
apply (*simp add*: *dwta-infsum-sum*)
apply (*subst sum.subset-diff*[**where** A=*UNIV* **and** B={$(\!|r_v = C,\ a_v = S|\!)$, $(\!|r_v = C,\ a_v = F|\!)$, $(\!|r_v = D,\ a_v = S|\!)$, $(\!|r_v = D,\ a_v = F|\!)$}])
apply (*simp add*: *dwta-state-finite*)+
apply (*expr-auto*)
apply (*subst sum-nonneg-eq-0-iff*)
using *dwta-state-finite* **apply** *blast*
apply *auto*[*1*]
by (*smt* (*z3*) *Attacker.exhaust DWTA-state.surjective DiffD2 Status.exhaust insertCI old.unit.exhaust*)

end

2 Monty Hall

theory *utp-prob-rel-lattice-monty-hall*
imports
UTP-prob-relations.utp-prob-rel
begin

unbundle *UTP-Syntax*

declare [[*show-types*]]

named-theorems *dwta-defs*

alphabet *mh-state* =
  p :: *nat*
  c :: *nat*
  m :: *nat*

2.1 Definitions

definition *INIT-p* :: *mh-state prhfun* **where**
[*dwta-defs*]: *INIT-p* = *prfun-of-rvfun* ($p\ \boldsymbol{\mathcal{U}}\ \{0..2\}$)

definition *INIT-c* :: *mh-state prhfun* **where**
[*dwta-defs*]: *INIT-c* = *prfun-of-rvfun* ($c\ \boldsymbol{\mathcal{U}}\ \{0..2\}$)

definition *INIT*:: *mh-state prhfun* **where**
[*dwta-defs*]: *INIT* = *INIT-p* ; *INIT-c*

term $(x)(\!|c_v := Suc\ (0::\mathbb{N})|\!)$
find-theorems *name*:*mh-state*
record $x = i$ :: *nat*

thm *mh-state.select-convs*
thm *mh-state.surjective*
thm *mh-state.update-convs*

abbreviation *MHA-1* :: *mh-state prhfun* **where**
$MHA\text{-}1 \equiv (if_p\ 1/2\ then\ (m := (\$c + 1)\ mod\ 3)\ else\ (m := (\$c + 2)\ mod\ 3))$

definition *MHA*:: *mh-state prhfun* **where**
[*dwta-defs*]: $MHA = (if_c\ c^< = p^<\ then$
    MHA-1
  else
    $(m := 3 - \$c - \$p)$
)

definition *MHA-NC*:: *mh-state prhfun* **where**
[*dwta-defs*]: *MHA-NC* = *MHA* ; *II*

definition *MHA-C*:: *mh-state prhfun* **where**
[*dwta-defs*]: *MHA-C* = *MHA* ; $c := 3 - c - m$

thm *MHA-def*

definition *IMHA-NC* **where**
[*dwta-defs*]: *IMHA-NC* = *INIT* ; *MHA-NC*

definition *IMHA-C* **where**
[*dwta-defs*]: *IMHA-C* = *INIT* ; *MHA-C*

2.2 *INIT*

lemma *zero-to-two*: $\{0..2::\mathbb{N}\} = \{0, 1, 2\}$
  by *force*

lemma *infsum-alt-3*:
$(\sum_\infty v::\mathbb{N}.\ \mathit{if}\ v = (0::\mathbb{N}) \vee v = \mathit{Suc}\ (0::\mathbb{N}) \vee v = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) = (3::\mathbb{R})$
apply (*simp add*: *infsum-constant-finite-states*)
apply (*subgoal-tac* $\{v::\mathbb{N}.\ v = (0::\mathbb{N}) \vee v = \mathit{Suc}\ (0::\mathbb{N}) \vee v = (2::\mathbb{N})\} = \{0,\ \mathit{Suc}\ 0,\ 2\}$)
apply *simp*
by (*simp add*: *set-eq-iff*)

lemma *INIT-p-altdef*:
$\mathit{INIT\text{-}p} = \mathit{prfun\text{-}of\text{-}rvfun}\ ((\llbracket p^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^> = c^<\rrbracket_{\mathcal{I}e} * \llbracket m^> = m^<\rrbracket_{\mathcal{I}e})\ /\ 3)_e$
apply (*simp add*: *zero-to-two INIT-p-def*)
apply (*simp add*: *dist-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*pred-auto*)
by (*simp-all add*: *infsum-alt-3*)

lemma *INIT-p-is-dist*:
is-final-distribution (*rvfun-of-prfun INIT-p*)
apply (*simp add*: *INIT-p-def*)
apply (*subst rvfun-uniform-dist-inverse*)
apply *simp+*
by (*simp add*: *rvfun-uniform-dist-is-dist*)

lemma *INIT-c-altdef*:
$\mathit{INIT\text{-}c} = \mathit{prfun\text{-}of\text{-}rvfun}\ ((\llbracket p^> = p^<\rrbracket_{\mathcal{I}e} * \llbracket c^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket m^> = m^<\rrbracket_{\mathcal{I}e})\ /\ 3)_e$
apply (*simp add*: *zero-to-two INIT-c-def*)
apply (*simp add*: *dist-defs*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*pred-auto*)
by (*simp-all add*: *infsum-alt-3*)

lemma *INIT-c-is-dist*:
is-final-distribution (*rvfun-of-prfun INIT-c*)
apply (*simp add*: *INIT-c-def*)
apply (*subst rvfun-uniform-dist-inverse*)
apply *simp+*
by (*simp add*: *rvfun-uniform-dist-is-dist*)

lemma *record-update-simp*:
assumes $m_v\ (r_1::\mathit{mh\text{-}state}) = m_v\ r_2$
shows $(r_1(\!|p_v := p_v\ (r_2),\ c_v := x|\!) = r_2) \longleftrightarrow c_v\ r_2 = x$
apply (*auto*)
apply (*metis mh-state.select-convs*(*2*) *mh-state.surjective mh-state.update-convs*(*2*))
by (*simp add*: *assms*)

lemma *record-update-simp'*:
assumes $m_v\ r_2 = m_v\ (r_1::\mathit{mh\text{-}state})$
shows $(r_1(\!|p_v := p_v\ (r_2),\ c_v := x|\!) = r_2) \longleftrightarrow c_v\ r_2 = x$
apply (*auto*)
apply (*metis mh-state.select-convs*(*2*) *mh-state.surjective mh-state.update-convs*(*2*))
by (*simp add*: *assms*)

lemma *record-neq-p-c*:
assumes $p_1 \neq p_2 \vee c_1 \neq c_2$
assumes $r_1(\!|p_v := p_1,\ c_v := c_1|\!) = r_1(\!|p_v := p_2,\ c_v := c_2|\!)$
shows *False*

by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*) *assms*(*1*) *assms*(*2*))

lemma *record-neq-p-c'*:
assumes $p_1 \neq p_2 \vee c_1 \neq c_2$
shows $\neg\ r_1(\!|p_v := p_1,\ c_v := c_1|\!) = r_2(\!|p_v := p_2,\ c_v := c_2|\!)$
using *assms record-neq-p-c*
by (*smt* (*verit, ccfv-SIG*) *mh-state.cases-scheme mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*))

lemma *record-neq*:
assumes $p_1 \neq p_2 \vee c_1 \neq c_2 \vee m_1 \neq m_2$
shows $\neg\ (\!|p_v = p_1,\ c_v = c_1,\ m_v = m_1|\!) = (\!|p_v = p_2,\ c_v = c_2,\ m_v = m_2|\!)$
using *assms* **by** *blast*

Below we illustrate the simplification of INIT using two ways:

- *INIT-altdef*: without $[\![finite\ (?A::\mathbb{P}\ ?'a);\ \ vwb\text{-}lens\ (?x::?'a \Longrightarrow ?'b);\ \ \neg\ ?A = \{\}]\!] \Longrightarrow prfun\text{-}of\text{-}rvfun\ (?x\ \boldsymbol{\mathcal{U}}\ ?A)\ ;\ (?P::?'b \times ?'b \Rightarrow ureal) = prfun\text{-}of\text{-}rvfun\ [\lambda s::?'b \times ?'b.\ (\sum v::?'a{\in}?A.\ (subst\text{-}upd\ [\rightsquigarrow]\ (?x^<)\ [\lambda s::?'b \times ?'b.\ v]_e \dagger [rvfun\text{-}of\text{-}prfun\ ?P]_e)\ s)\ /\ real\ (card\ ?A)]_e$. We need to deal with infinite summation and cardinality.
- *INIT-altdef'*: with $[\![finite\ (?A::\mathbb{P}\ ?'a);\ \ vwb\text{-}lens\ (?x::?'a \Longrightarrow ?'b);\ \ \neg\ ?A = \{\}]\!] \Longrightarrow prfun\text{-}of\text{-}rvfun\ (?x\ \boldsymbol{\mathcal{U}}\ ?A)\ ;\ (?P::?'b \times ?'b \Rightarrow ureal) = prfun\text{-}of\text{-}rvfun\ [\lambda s::?'b \times ?'b.\ (\sum v::?'a{\in}?A.\ (subst\text{-}upd\ [\rightsquigarrow]\ (?x^<)\ [\lambda s::?'b \times ?'b.\ v]_e \dagger [rvfun\text{-}of\text{-}prfun\ ?P]_e)\ s)\ /\ real\ (card\ ?A)]_e$. We mainly deal with conditional and propositional logic.

1)

lemma *INIT-altdef*: $INIT = prfun\text{-}of\text{-}rvfun\ (([\![p^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![c^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![m^> = m^<]\!]_{\mathcal{I}e})\ /\ 9)_e$
apply (*simp add*: *INIT-def INIT-p-def INIT-c-def zero-to-two*)
apply (*simp add*: *pfun-defs*)
apply (*simp add*: *prfun-uniform-dist-altdef'*)
apply (*expr-simp-1 add*: *assigns-r-def*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*simp only*: *fun-eq-iff*)
apply (*rule allI*)
proof −
fix $x :: mh\text{-}state \times mh\text{-}state$
let $?rhs = (if\ p_v\ (snd\ x) = (0::\mathbb{N}) \vee p_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee p_v\ (snd\ x) = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ (snd\ x) = (0::\mathbb{N}) \vee c_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee c_v\ (snd\ x) = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ (snd\ x) = m_v\ (fst\ x)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let $?rhs\text{-}1 = (if\ (p_v\ (snd\ x) = (0::\mathbb{N}) \vee p_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee p_v\ (snd\ x) = (2::\mathbb{N})) \wedge$
$(c_v\ (snd\ x) = (0::\mathbb{N}) \vee c_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee c_v\ (snd\ x) = (2::\mathbb{N})) \wedge$
$(m_v\ (snd\ x) = m_v\ (fst\ x))\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$

let $?lhs\text{-}1 = \lambda v_0.\ (if\ v_0 = fst\ x(\!|p_v := 0::\mathbb{N}|\!) \vee v_0 = fst\ x(\!|p_v := Suc\ (0::\mathbb{N})|\!) \vee v_0 = fst\ x(\!|p_v := 2::\mathbb{N}|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R})) *$
$(if\ snd\ x = v_0(\!|c_v := 0::\mathbb{N}|\!) \vee snd\ x = v_0(\!|c_v := Suc\ (0::\mathbb{N})|\!) \vee snd\ x = v_0(\!|c_v := 2::\mathbb{N}|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let $?lhs\text{-}2 = \lambda v_0.\ (if\ (v_0 = fst\ x(\!|p_v := 0::\mathbb{N}|\!) \vee v_0 = fst\ x(\!|p_v := Suc\ (0::\mathbb{N})|\!) \vee v_0 = fst\ x(\!|p_v := 2::\mathbb{N}|\!)) \wedge$
$(snd\ x = v_0(\!|c_v := 0::\mathbb{N}|\!) \vee snd\ x = v_0(\!|c_v := Suc\ (0::\mathbb{N})|\!) \vee snd\ x = v_0(\!|c_v := 2::\mathbb{N}|\!))\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R}))$

have *fr*: *?rhs* / (*9*::ℝ) = *?rhs-1* / (*9*::ℝ)
  by *simp*

have ($\sum_\infty v_0$::*mh-state*. *?lhs-1* v_0 / (*9*::ℝ)) = ($\sum_\infty v_0$::*mh-state*. *?lhs-2* v_0 / (*9*::ℝ))
  by (*simp add*: *infsum-cong*)
also have ... = ($\sum_\infty v_0$::*mh-state*. *?lhs-2* v_0 ∗ (*1* / (*9*::ℝ)))
  by *auto*
also have ... = ($\sum_\infty v_0$::*mh-state*. *?lhs-2* v_0) ∗ (*1* / (*9*::ℝ))
  apply (*subst infsum-cmult-left*[**where** *c* = *1* / (*9*::*real*)])
  apply (*simp add*: *infsum-constant-finite-states-summable*)
  by *simp*

also have *fl*: ... =
  (*1* ∗ *card* {v_0. (v_0 = *fst x*⦇p_v := *0*::ℕ⦈ ∨ v_0 = *fst x*⦇p_v := *Suc* (*0*::ℕ)⦈ ∨ v_0 = *fst x*⦇p_v := *2*::ℕ⦈) ∧
    (*snd x* = v_0⦇c_v := *0*::ℕ⦈ ∨ *snd x* = v_0⦇c_v := *Suc* (*0*::ℕ)⦈ ∨ *snd x* = v_0⦇c_v := *2*::ℕ⦈)}
) ∗ (*1* / (*9*::ℝ))
  by (*simp add*: *infsum-constant-finite-states*)

have *ff1*: *card* {v_0. (v_0 = *fst x*⦇p_v := *0*::ℕ⦈ ∨ v_0 = *fst x*⦇p_v := *Suc* (*0*::ℕ)⦈ ∨ v_0 = *fst x*⦇p_v := *2*::ℕ⦈)
∧
    (*snd x* = v_0⦇c_v := *0*::ℕ⦈ ∨ *snd x* = v_0⦇c_v := *Suc* (*0*::ℕ)⦈ ∨ *snd x* = v_0⦇c_v := *2*::ℕ⦈)}
  = *?rhs-1*
  apply (*simp add*: *if-bool-eq-conj*)
  apply (*rule conjI*)
  apply (*rule impI*)
  apply (*rule card-1-singleton*)
  apply (*rule ex-ex1I*)
  apply (*rule-tac x* = *fst x*⦇p_v := p_v (*snd x*)⦈ **in** *exI*)
  apply (*erule conjE*)+
  apply (*rule conjI*)
  apply *presburger*
  using *record-update-simp* **apply** *metis*
  apply (*erule conjE*)+
  apply (*smt* (*z3*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*))
  apply (*rule conjI*)
  apply (*rule impI*)
  apply (*smt* (*verit*, *ccfv-threshold*) *mh-state.ext-inject mh-state.surjective*
    mh-state.update-convs(*1*) *mh-state.update-convs*(*2*) *less-nat-zero-code*)
  apply (*rule conjI*)
  apply (*rule impI*)
  apply (*smt* (*verit*, *ccfv-threshold*) *mh-state.ext-inject mh-state.surjective*
    mh-state.update-convs(*1*) *mh-state.update-convs*(*2*) *less-nat-zero-code*)
  apply (*rule impI*)
  by (*smt* (*verit*, *ccfv-threshold*) *mh-state.ext-inject mh-state.surjective*
    mh-state.update-convs(*1*) *mh-state.update-convs*(*2*) *less-nat-zero-code*)

show ($\sum_\infty v_0$::*mh-state*. *?lhs-1* v_0 / (*9*::ℝ)) = *?rhs* / (*9*::ℝ)
  apply (*simp only*: *fr fl*)
  using *ff1 calculation fl* **by** *linarith*
qed

lemma *conditionals-combined*:
  assumes $b_1 \wedge b_2$ = *False*
  shows (*if* b_1 *then aa else 0*::ℝ) + (*if* b_2 *then aa else 0*) = (*if* $b_1 \vee b_2$ *then aa else 0*)

by (*simp add*: *assms*)

lemma *INIT-altdef′*: $INIT = prfun\text{-}of\text{-}rvfun\ ((\llbracket p^{>} \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^{>} \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket m^{>} = m^{<}\rrbracket_{\mathcal{I}e})\ /\ 9)_e$

apply (*simp add*: *INIT-def INIT-p-def INIT-c-def zero-to-two*)
apply (*simp add*: *prfun-uniform-dist-left*)
apply (*simp add*: *prfun-uniform-dist-altdef′*)
apply (*expr-simp-1 add*: *assigns-r-def*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp only*: *fun-eq-iff*)
apply (*rule allI*)
proof −
fix $x :: mh\text{-}state \times mh\text{-}state$
let $?lhs\text{-}1b = snd\ x = fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}|\!) \lor$
$snd\ x = fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})|\!) \lor$
$snd\ x = fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}|\!)$
let $?lhs\text{-}2b = snd\ x = fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 0{::}\mathbb{N}|\!) \lor$
$snd\ x = fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := Suc\ (0{::}\mathbb{N})|\!) \lor$
$snd\ x = fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 2{::}\mathbb{N}|\!)$
let $?lhs\text{-}3b = snd\ x = fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}|\!) \lor$
$snd\ x = fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})|\!) \lor$
$snd\ x = fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}|\!)$
let $?lhs\text{-}1 = (if\ ?lhs\text{-}1b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}2 = (if\ ?lhs\text{-}2b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}3 = (if\ ?lhs\text{-}3b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs = (?lhs\text{-}1\ /\ (3{::}\mathbb{R}) + (?lhs\text{-}2\ /\ (3{::}\mathbb{R}) + ?lhs\text{-}3\ /\ (3{::}\mathbb{R})))\ /\ (3{::}\mathbb{R})$
let $?rhs\text{-}1b = p_v\ (snd\ x) = (0{::}\mathbb{N}) \lor p_v\ (snd\ x) = Suc\ (0{::}\mathbb{N}) \lor p_v\ (snd\ x) = (2{::}\mathbb{N})$
let $?rhs\text{-}2b = c_v\ (snd\ x) = (0{::}\mathbb{N}) \lor c_v\ (snd\ x) = Suc\ (0{::}\mathbb{N}) \lor c_v\ (snd\ x) = (2{::}\mathbb{N})$
let $?rhs\text{-}3b = m_v\ (snd\ x) = m_v\ (fst\ x)$
let $?rhs = (if\ ?rhs\text{-}1b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ ?rhs\text{-}2b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ ?rhs\text{-}3b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))\ /\ (9{::}\mathbb{R})$
let $?rhs\text{-}1 = (if\ ?rhs\text{-}1b \land ?rhs\text{-}2b \land ?rhs\text{-}3b\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))\ /\ (9{::}\mathbb{R})$
have *rhs-1*: $?rhs = ?rhs\text{-}1$
by *force*
have *lhs-1*: $?lhs = (?lhs\text{-}1 + ?lhs\text{-}2 + ?lhs\text{-}3)\ /\ (9{::}\mathbb{R})$
by *force*

let $?lhs\text{-}1b' = fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}|\!) = snd\ x \lor$
$fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})|\!) = snd\ x \lor$
$fst\ x(\!|p_v := 0{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}|\!) = snd\ x$
let $?lhs\text{-}2b' = fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 0{::}\mathbb{N}|\!) = snd\ x \lor$
$fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := Suc\ (0{::}\mathbb{N})|\!) = snd\ x \lor$
$fst\ x(\!|p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 2{::}\mathbb{N}|\!) = snd\ x$
let $?lhs\text{-}3b' = fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}|\!) = snd\ x \lor$
$fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})|\!) = snd\ x \lor$
$fst\ x(\!|p_v := 2{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}|\!) = snd\ x$
have $((if\ ?lhs\text{-}1b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) +$
$(if\ ?lhs\text{-}2b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) + (if\ ?lhs\text{-}3b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})))$
$=\ (if\ ?lhs\text{-}1b' \lor ?lhs\text{-}2b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) + (if\ ?lhs\text{-}3b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$

apply *auto*
by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*) *One-nat-def one-neq-zero*)+
also have *lhs-2′*: $... =\ (if\ ?lhs\text{-}1b' \lor ?lhs\text{-}2b' \lor ?lhs\text{-}3b'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
apply *auto*

using *record-neq-p-c* **apply** (*metis zero-neq-numeral*)+
using *record-neq-p-c* **by** (*metis n-not-Suc-n numeral-2-eq-2*)+

have *lhs-2*: (*?lhs-1* + *?lhs-2* + *?lhs-3*) = (*if ?lhs-1b* ∨*?lhs-2b* ∨ *?lhs-3b then 1*::ℝ *else* (*0*::ℝ))
using *lhs-2′* **by** (*smt* (*verit, best*) *calculation*)

have *lhs-rhs*: (*if ?lhs-1b* ∨*?lhs-2b* ∨ *?lhs-3b then 1*::ℝ *else* (*0*::ℝ))
= (*if* (p_v (*snd x*) = (*0*::ℕ) ∨ p_v (*snd x*) = *Suc* (*0*::ℕ) ∨ p_v (*snd x*) = (*2*::ℕ)) ∧
(c_v (*snd x*) = (*0*::ℕ) ∨ c_v (*snd x*) = *Suc* (*0*::ℕ) ∨ c_v (*snd x*) = (*2*::ℕ)) ∧
(m_v (*snd x*) = m_v (*fst x*)) *then 1*::ℝ *else* (*0*::ℝ))
apply (*rule if-cong*)
apply (*rule iffI*)
apply (*rule conjI*)+
apply (*smt* (*z3*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*))
apply (*smt* (*z3*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*))
apply (*metis record-update-simp*)
by *simp*+
show *?lhs* = *?rhs*
apply (*simp only*: *lhs-1 rhs-1*)
using *calculation lhs-2 lhs-rhs* **by** *presburger*
qed

lemma *INIT-is-dist*:
is-final-distribution (*rvfun-of-prfun INIT*)
apply (*simp add*: *INIT-def*)
apply (*simp add*: *pseqcomp-def*)
apply (*subst rvfun-seqcomp-inverse*)
apply (*simp add*: *INIT-p-is-dist*)
using *INIT-c-is-dist* **apply** (*simp add*: *ureal-is-prob*)
using *INIT-c-is-dist INIT-p-is-dist rvfun-seqcomp-is-dist* **by** *blast*

2.3 *MHA-NC*

lemma *suc-card-minus*:
assumes $x > 0$
shows (*Suc* (*card A*) = *x*) ⟷ (*card A* = *x* − *1*)
using *assms* **by** *fastforce*

lemma *nine-minus-nine-zero*:
(*9*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) − (*1*::ℕ) = *0*
by *simp*

lemma *card-states-9*:
card {s_1⦇p_v := *0*::ℕ, c_v := *0*::ℕ⦈, s_1⦇p_v := *0*::ℕ, c_v := *Suc* (*0*::ℕ)⦈, s_1⦇p_v := *0*::ℕ, c_v := *2*::ℕ⦈,
s_1⦇p_v := *Suc* (*0*::ℕ), c_v := *0*::ℕ⦈, s_1⦇p_v := *Suc* (*0*::ℕ), c_v := *Suc* (*0*::ℕ)⦈, s_1⦇p_v := *Suc* (*0*::ℕ), c_v
:= *2*::ℕ⦈,
s_1⦇p_v := *2*::ℕ, c_v := *0*::ℕ⦈, s_1⦇p_v := *2*::ℕ, c_v := *Suc* (*0*::ℕ)⦈, s_1⦇p_v := *2*::ℕ, c_v := *2*::ℕ⦈
} = *9*
apply (*subst card-Suc-Diff1* [**where** *x* = s_1⦇p_v := *0*::ℕ, c_v := *0*::ℕ⦈, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** *x* = s_1⦇p_v := *0*::ℕ, c_v := *Suc* (*0*::ℕ)⦈, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def one-neq-zero record-neq-p-c*)

apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := 0{::}\mathbb{N},\ c_v := 2\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def Suc-1 n-not-Suc-n nat.distinct*(*1*) *record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 0{::}\mathbb{N}\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis n-not-Suc-n record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),\ c_v := Suc\ (0{::}\mathbb{N})\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def one-neq-zero record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 2\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def Suc-1 n-not-Suc-n nat.distinct*(*1*) *record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def Suc-1 n-not-Suc-n nat.distinct*(*1*) *record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
using *record-neq-p-c* **apply** *fastforce*
apply (*simp only*: *suc-card-minus*)
apply (*subst card-Suc-Diff1* [**where** $x = s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := 2\rrparenthesis$, *symmetric*])
apply (*meson finite.simps finite-Diff*)
apply (*simp*)
apply (*metis One-nat-def Suc-1 n-not-Suc-n nat.distinct*(*1*) *record-neq-p-c*)
apply (*simp only*: *suc-card-minus*)
apply (*subst nine-minus-nine-zero*)
by (*smt* (*z3*) *Diff-cancel Diff-insert card.empty insert-commute*)

lemma *set-states*: $\forall\, s_1{::}mh\text{-}state.\ \{s{::}mh\text{-}state.\ get_p\ s \le (2{::}\mathbb{N}) \wedge get_c\ s \le (2{::}\mathbb{N}) \wedge get_m\ s = get_m\ s_1\}$
$= \{s_1\llparenthesis p_v := 0{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}\rrparenthesis,\ s_1\llparenthesis p_v := 0{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})\rrparenthesis,\ s_1\llparenthesis p_v := 0{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}\rrparenthesis,$
$s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),\ c_v := 0{::}\mathbb{N}\rrparenthesis,\ s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),\ c_v := Suc\ (0{::}\mathbb{N})\rrparenthesis,\ s_1\llparenthesis p_v := Suc\ (0{::}\mathbb{N}),$
$c_v := 2{::}\mathbb{N}\rrparenthesis,$
$s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := 0{::}\mathbb{N}\rrparenthesis,\ s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := Suc\ (0{::}\mathbb{N})\rrparenthesis,\ s_1\llparenthesis p_v := 2{::}\mathbb{N},\ c_v := 2{::}\mathbb{N}\rrparenthesis\}$
apply (*simp add*: *lens-defs*)
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*z3*) *mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*)
One-nat-def Suc-1 bot-nat-0.extremum-unique c-def le-Suc-eq lens.simps(*1*) *m-def old.unit.exhaust*
p-def)
by (*smt* (*verit*, *best*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*)
mh-state.update-convs(*2*) *One-nat-def bot-nat-0.extremum c-def lens.simps*(*1*) *less-one*
linorder-not-le m-def order-le-less p-def zero-neq-numeral)

lemma *ereal2real-1-2*: *rvfun-of-prfun* $[\lambda x::mh\text{-}state \times mh\text{-}state.$
  ereal2ureal $((1::ereal)\ /\ ereal\ (2::\mathbb{R}))]_e = (1/2)_e$
 apply (*simp add*: *rvfun-of-prfun-simp*)
 apply (*simp add*: *ureal-defs*)
 using *SEXP-def ereal-1-div ereal-less-eq*(*6*) *mult-cancel-left1 real2uereal-min-inverse′ zero-ereal-def* **by** *auto*

lemma *MHA-altdef*: *MHA* =
  prfun-of-rvfun (
   $([\![c^< = p^<]\!]_{\mathcal{I}e} * [\![\ m := (c + 1)\ mod\ 3\ ]\!]_{\mathcal{I}e}\ /\ 2) +$
   $([\![c^< = p^<]\!]_{\mathcal{I}e} * [\![\ m := (c + 2)\ mod\ 3\ ]\!]_{\mathcal{I}e}\ /\ 2) +$
   $([\![c^< \neq p^<]\!]_{\mathcal{I}e} * [\![\ m := 3 - c - p\ ]\!]_{\mathcal{I}e})$
  $)_e$
proof –
 show *?thesis*
 apply (*simp only*: *dwta-defs*)
 apply (*simp add*: *prfun-seqcomp-right-unit*)
 apply (*simp add*: *prfun-pcond-altdef*)
 apply (*simp only*: *pchoice-def passigns-def*)
 apply (*simp only*: *rvfun-assignment-inverse*)
 apply (*simp only*: *ereal2real-1-2*)
 apply (*subst rvfun-pchoice-inverse-c′′*)
 using *rvfun-assignment-is-prob* **apply** *blast+*
 apply (*simp*)
 apply (*simp add*: *expr-defs rel lens-defs prod.case-eq-if alpha-splits*)
 apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
 by *fastforce*
qed

lemma *MHA-is-dist*: *is-final-distribution* (*rvfun-of-prfun MHA*)
proof –
 have *f0*: *is-final-distribution* (*rvfun-of-prfun MHA-1*)
  apply (*simp add*: *pchoice-def*)
  apply (*subst rvfun-pchoice-inverse*)
  apply (*simp add*: *ureal-is-prob*)+
  apply (*simp only*: *ereal2real-1-2*)
  apply (*rule rvfun-pchoice-is-dist-c′*)
  by (*simp add*: *passigns-def rvfun-assignment-inverse rvfun-assignment-is-dist*)+
 show *?thesis*
  apply (*simp only*: *MHA-def*)
  apply (*simp only*: *pcond-def*)
  apply (*subst rvfun-pcond-inverse*)
  using *ureal-is-prob* **apply** *blast+*
  apply (*subst rvfun-pcond-is-dist′*)
  using *f0* **apply** *meson*
  apply (*simp add*: *passigns-def rvfun-assignment-inverse rvfun-assignment-is-dist*)
  apply (*pred-auto*)
  by *simp*
qed

lemma *MHA-NC-MHA-eq*: *MHA-NC* = *MHA*
 apply (*simp only*: *MHA-NC-def*)
 by (*simp add*: *prfun-seqcomp-right-unit*)

2.4 *IMHA-NC*

definition *IMHA-NC-altdef* :: *mh-state* $\times$ *mh-state* $\Rightarrow \mathbb{R}$ **where**
IMHA-NC-altdef = (
$(\llbracket c^{>} = p^{>} \rrbracket_{\mathcal{I}e} * \llbracket p^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket c^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket\ m^{>} = (c^{>} + 1)\ mod\ 3\ \rrbracket_{\mathcal{I}e}\ /\ 18)$ +
$(\llbracket c^{>} = p^{>} \rrbracket_{\mathcal{I}e} * \llbracket p^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket c^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket\ m^{>} = (c^{>} + 2)\ mod\ 3\ \rrbracket_{\mathcal{I}e}\ /\ 18)$ +
$(\llbracket c^{>} \neq p^{>} \rrbracket_{\mathcal{I}e} * \llbracket p^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket c^{>} \in \{0..2\} \rrbracket_{\mathcal{I}e} * \llbracket\ m^{>} = 3 - c^{>} - p^{>}\ \rrbracket_{\mathcal{I}e}\ /\ 9)$
$)_e$

lemma *IMHA-NC-altdef-dist*: *is-final-distribution IMHA-NC-altdef*
apply (*simp add*: *IMHA-NC-altdef-def*)
apply (*simp add*: *dist-defs expr-defs lens-defs*)
proof −
let *?lhs-1* = λs::*mh-state*. (*if* c_v *s* = p_v *s then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$ (*if* p_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* c_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* m_v *s* = *Suc* (c_v *s*) *mod* (3::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$))
let *?lhs-2* = λs::*mh-state*. (*if* c_v *s* = p_v *s then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$ (*if* p_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* c_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* m_v *s* = *Suc* (*Suc* (c_v *s*)) *mod* (3::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$))
let *?lhs-3* = λs::*mh-state*. (*if* $\neg$ c_v *s* = p_v *s then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$ (*if* p_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* c_v *s* $\leq$ (2::$\mathbb{N}$) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$)) $*$
(*if* m_v *s* = (3::$\mathbb{N}$) − (c_v *s* + p_v *s*) *then* 1::$\mathbb{R}$ *else* (0::$\mathbb{R}$))
let *?lhs* = λs::*mh-state*. *?lhs-1 s* / (18::$\mathbb{R}$) + *?lhs-2 s* / (18::$\mathbb{R}$) + *?lhs-3 s* / (9::$\mathbb{R}$)

have *states-1-eq*:{*s*::*mh-state*. ((c_v *s* = p_v *s* $\wedge$ p_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ c_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ m_v *s* = *Suc* (c_v *s*) *mod* (3::$\mathbb{N}$)}
= {⦇p_v = 0::$\mathbb{N}$, c_v = 0::$\mathbb{N}$, m_v = *Suc* (0::$\mathbb{N}$)⦈,⦇p_v = *Suc* (0::$\mathbb{N}$), c_v = *Suc* (0::$\mathbb{N}$), m_v = (2::$\mathbb{N}$)⦈,
⦇p_v = 2::$\mathbb{N}$, c_v = 2::$\mathbb{N}$, m_v = 0::$\mathbb{N}$⦈}
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*z3*) *mh-state.surjective Orderings.order-eq-iff Suc-eq-numeral add.assoc cong-exp-iff-simps*(2) *diff-add-zero diff-is-0-eq le-SucE mod-Suc mod-self numeral-1-eq-Suc-0 numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust one-eq-numeral-iff plus-1-eq-Suc*)
by *force*

have *states-2-eq*:{*s*::*mh-state*. ((c_v *s* = p_v *s* $\wedge$ p_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ c_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ m_v *s* = *Suc* (*Suc* (c_v *s*)) *mod* (3::$\mathbb{N}$)}
= {⦇p_v = 0::$\mathbb{N}$, c_v = 0::$\mathbb{N}$, m_v = (2::$\mathbb{N}$)⦈, ⦇p_v = *Suc* (0::$\mathbb{N}$), c_v = *Suc* (0::$\mathbb{N}$), m_v = (0::$\mathbb{N}$)⦈,
⦇p_v = 2::$\mathbb{N}$, c_v = 2::$\mathbb{N}$, m_v = *Suc* (0::$\mathbb{N}$)⦈}
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective lessI less-2-cases mod-Suc mod-less numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less*)
by *force*

have *states-3-eq*: {*s*::*mh-state*. (($\neg$ c_v *s* = p_v *s* $\wedge$ p_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ c_v *s* $\leq$ (2::$\mathbb{N}$)) $\wedge$ m_v *s* = (3::$\mathbb{N}$) − (c_v *s* + p_v *s*)}
= {⦇p_v = 0::$\mathbb{N}$, c_v = *Suc* (0::$\mathbb{N}$), m_v = (2::$\mathbb{N}$)⦈, ⦇p_v = 0::$\mathbb{N}$, c_v = (2::$\mathbb{N}$), m_v = *Suc* (0::$\mathbb{N}$)⦈,
⦇p_v = *Suc* (0::$\mathbb{N}$), c_v = (0::$\mathbb{N}$), m_v = (2::$\mathbb{N}$)⦈, ⦇p_v = *Suc* (0::$\mathbb{N}$), c_v = (2::$\mathbb{N}$), m_v = (0::$\mathbb{N}$)⦈,
⦇p_v = 2::$\mathbb{N}$, c_v = 0::$\mathbb{N}$, m_v = *Suc* (0::$\mathbb{N}$)⦈, ⦇p_v = 2::$\mathbb{N}$, c_v = *Suc* (0::$\mathbb{N}$), m_v = (0::$\mathbb{N}$)⦈}
apply (*simp add*: *set-eq-iff*)

apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *ccfv-SIG*) *mh-state.surjective One-nat-def diff-add-inverse diff-diff-eq less-2-cases numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

have *lhs-1-summable*: *?lhs-1 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-1-eq* **by** (*simp-all*)

have *lhs-2-summable*: *?lhs-2 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-2-eq* **by** (*simp-all*)

have *lhs-3-summable*: *?lhs-3 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-3-eq* **by** (*simp-all*)

have *lhs-1-infsum*: ($\sum_\infty$*s*::*mh-state*. *?lhs-1 s*) = *3*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-1-eq* **by** (*simp-all*)

have *lhs-2-infsum*: ($\sum_\infty$*s*::*mh-state*. *?lhs-2 s*) = *3*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-2-eq* **by** (*simp-all*)

have *lhs-3-infsum*: ($\sum_\infty$*s*::*mh-state*. *?lhs-3 s*) = *6*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-3-eq* **by** (*simp-all*)

show ($\sum_\infty$*s*::*mh-state*. *?lhs s*) = (*1*::$\mathbb{R}$)
apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)

apply (*simp-all add*: *lhs-3-summable*)
using *lhs-1-infsum lhs-2-infsum lhs-3-infsum* **by** (*simp*)
qed

lemma *IMHA-NC-altdef*: *IMHA-NC* = *prfun-of-rvfun IMHA-NC-altdef*
apply (*simp add*: *IMHA-NC-def zero-to-two IMHA-NC-altdef-def*)
apply (*simp add*: *INIT-altdef MHA-NC-MHA-eq MHA-altdef*)
apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *expr-defs dist-defs*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *expr-defs dist-defs*)
apply (*expr-simp-1 add*: *assigns-r-def*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*subst fun-eq-iff*, *rule allI*)
proof −
fix *x* :: *mh-state* × *mh-state*
let *?lhs-p* = λv_0. (*if* $p_v\ v_0 \le (2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?lhs-c* = λv_0. (*if* $c_v\ v_0 \le (2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?lhs-m* = λv_0. (*if* $m_v\ v_0 = m_v$ (*fst x*) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?lhs-c-p* = λv_0. (*if* $c_v\ v_0 = p_v\ v_0$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?lhs-c-n-p* = λv_0. (*if* $\neg\ c_v\ v_0 = p_v\ v_0$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?m-1-mod* = λv_0. (*if snd x* = $v_0(\!|m_v :=$ *Suc* $(c_v\ v_0)$ *mod* $(3::\mathbb{N})|\!)$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?m-2-mod* = λv_0. (*if snd x* = $v_0(\!|m_v :=$ *Suc* (*Suc* $(c_v\ v_0)$) *mod* $(3::\mathbb{N})|\!)$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?m-3-c-p* = λv_0. (*if snd x* = $v_0(\!|m_v := (3::\mathbb{N}) - (c_v\ v_0 + p_v\ v_0)|\!)$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?lhs* = ($\sum_\infty v_0$::*mh-state*.
?lhs-p v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗
(*?lhs-c-p* v_0 ∗ *?m-1-mod* v_0 / $(2::\mathbb{R})$ +
?lhs-c-p v_0 ∗ *?m-2-mod* v_0 / $(2::\mathbb{R})$ +
?lhs-c-n-p v_0 ∗ *?m-3-c-p* v_0) / $(9::\mathbb{R})$)
let *?rhs-1* = (*if* c_v (*snd x*) = p_v (*snd x*) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* m_v (*snd x*) = *Suc* (c_v (*snd x*)) *mod* $(3::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?rhs-2* = (*if* c_v (*snd x*) = p_v (*snd x*) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* m_v (*snd x*) = *Suc* (*Suc* (c_v (*snd x*))) *mod* $(3::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?rhs-3* = (*if* ¬ c_v (*snd x*) = p_v (*snd x*) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$) ∗
(*if* m_v (*snd x*) = $(3::\mathbb{N})$ − (c_v (*snd x*) + p_v (*snd x*)) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?rhs* = *?rhs-1* / $(18::\mathbb{R})$ + *?rhs-2* / $(18::\mathbb{R})$ + *?rhs-3* / $(9::\mathbb{R})$

let *?rhs-1-1* = (*if* (c_v (*snd x*) = p_v (*snd x*) ∧
(p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$) ∧
(c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$) ∧
(m_v (*snd x*) = *Suc* (c_v (*snd x*)) *mod* $(3::\mathbb{N})$)) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?rhs-1-2* = (*if* (c_v (*snd x*) = p_v (*snd x*) ∧
(p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$) ∧
(c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$) ∧
(m_v (*snd x*) = *Suc* (*Suc* (c_v (*snd x*))) *mod* $(3::\mathbb{N})$)) *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})$)
let *?rhs-1-3* = (*if* (¬c_v (*snd x*) = p_v (*snd x*) ∧
(p_v (*snd x*) = $(0::\mathbb{N})$ ∨ p_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ p_v (*snd x*) = $(2::\mathbb{N})$) ∧
(c_v (*snd x*) = $(0::\mathbb{N})$ ∨ c_v (*snd x*) = *Suc* $(0::\mathbb{N})$ ∨ c_v (*snd x*) = $(2::\mathbb{N})$) ∧

$(m_v\ (snd\ x) = (3::\mathbb{N}) - (c_v\ (snd\ x) + p_v\ (snd\ x))))$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R}))$
have *rhs-1-1*: *?rhs-1* = *?rhs-1-1*
by *simp*
have *rhs-1-2*: *?rhs-2* = *?rhs-1-2*
by *simp*
have *rhs-1-3*: *?rhs-3* = *?rhs-1-3*
by *simp*

have *lhs-1-f0*: $(\lambda v_0.$ *?lhs-p* $v_0 *$ *?lhs-c* $v_0 *$ *?lhs-m* $v_0 *$ *?lhs-c-p* $v_0 *$ *?m-1-mod* $v_0) =$
$(\lambda v_0.$ (*if* $p_v\ v_0 \leq (2::\mathbb{N}) \wedge\ c_v\ v_0 \leq (2::\mathbb{N}) \wedge m_v\ v_0 = m_v\ (fst\ x) \wedge c_v\ v_0 = p_v\ v_0 \wedge$
$v_0(\!|m_v := Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})|\!) = snd\ x$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})))$
by *auto*
have *lhs-1-set-simp*: $\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge$
$c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = m_v\ (fst\ x)|\!),(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = m_v\ (fst\ x)|\!),$

$(\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = m_v\ (fst\ x)|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*metis* (*mono-tags*, *opaque-lifting*) *mh-state.surjective bot-nat-0.extremum le-SucE le-antisym numeral-2-eq-2 old.unit.exhaust*)
by *fastforce*
have *lhs-1-set-A-finite*: *finite* $\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$
by (*simp add*: *lhs-1-set-simp*)

have *lhs-1-summable*: $(\lambda v_0.$ *?lhs-p* $v_0 *$ *?lhs-c* $v_0 *$ *?lhs-m* $v_0 *$ *?lhs-c-p* $v_0 *$ *?m-1-mod* $v_0)$ *summable-on UNIV*
apply (*simp add*: *lhs-1-f0*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** *B*=
$\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-1-set-A-finite*)
by *blast*

have *lhs-1-infsum*: $(\sum_\infty v_0::mh\text{-}state.$ *?lhs-p* $v_0 *$ *?lhs-c* $v_0 *$ *?lhs-m* $v_0 *$ *?lhs-c-p* $v_0 *$ *?m-1-mod* $v_0)$
= *?rhs-1-1*
apply (*simp only*: *lhs-1-f0*)
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** *B*=
$\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-1-set-A-finite*)
apply (*blast*)
apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!|p_v = Suc\ (Suc\ (m_v\ (snd\ x)))\ mod\ (3::\mathbb{N}),$
$c_v = Suc\ (Suc\ (m_v\ (snd\ x)))\ mod\ (3::\mathbb{N}),\ m_v = m_v\ (fst\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*metis mh-state.select-convs*(*1*) *mod-Suc-le-divisor numeral-2-eq-2 numeral-3-eq-3*)
apply (*rule conjI*)

apply (*metis mh-state.select-convs(2) mod-Suc-le-divisor numeral-2-eq-2 numeral-3-eq-3*)
apply (*rule conjI*)
apply (*metis mh-state.select-convs(3)*)
apply (*rule conjI*)
apply (*metis mh-state.select-convs(1) mh-state.select-convs(2)*)
defer
apply (*metis mh-state.surjective mh-state.update-convs(3)*)
apply (*smt* (*verit*, *best*) *Collect-empty-eq mh-state.select-convs(1) mh-state.select-convs(2) mh-state.select-convs(3) mh-state.surjective mh-state.update-convs(3) card-eq-0-iff less-2-cases less-numeral-extra(3) order-le-less*)
proof −
assume *a1*: m_v (*snd x*) = *Suc* (c_v (*snd x*)) *mod* (*3*::$\mathbb{N}$)
assume *a2*: c_v (*snd x*) = (*0*::$\mathbb{N}$) $\vee$ c_v (*snd x*) = *Suc* (*0*::$\mathbb{N}$) $\vee$ c_v (*snd x*) = (*2*::$\mathbb{N}$)
assume *a3*: p_v (*snd x*) = (*0*::$\mathbb{N}$) $\vee$ p_v (*snd x*) = *Suc* (*0*::$\mathbb{N}$) $\vee$ p_v (*snd x*) = (*2*::$\mathbb{N}$)
assume *a4*: c_v (*snd x*) = p_v (*snd x*)

have ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈
⦇m_v := *Suc* (c_v ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈) *mod* (*3*::$\mathbb{N}$)⦈
= ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈
⦇m_v := *Suc* (*Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$)) *mod* (*3*::$\mathbb{N}$)⦈
by (*metis mh-state.select-convs(2)*)
also have ... = ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈
⦇m_v := m_v (*snd x*)⦈
by (*simp add*: *a1 mod-Suc-eq*)
also have ... = ⦇p_v = *Suc* (*Suc* (*Suc* (c_v (*snd x*)) *mod* (*3*::$\mathbb{N}$))) *mod* (*3*::$\mathbb{N}$),
c_v = *Suc* (*Suc* (*Suc* (c_v (*snd x*)) *mod* (*3*::$\mathbb{N}$))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈⦇m_v := m_v (*snd x*)⦈
by (*simp add*: *a1*)
also have ... = ⦇p_v = c_v (*snd x*), c_v = c_v (*snd x*), m_v = m_v (*fst x*)⦈⦇m_v := m_v (*snd x*)⦈
using *a2* **by** *fastforce*
also have ... = ⦇p_v = c_v (*snd x*), c_v = c_v (*snd x*), m_v = m_v (*snd x*)⦈
by *auto*
also have ... = *snd x*
by (*simp add*: *a4*)
then show ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), m_v = m_v (*fst x*)⦈
⦇m_v := *Suc* (c_v ⦇p_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$), c_v = *Suc* (*Suc* (m_v (*snd x*))) *mod* (*3*::$\mathbb{N}$),
m_v = m_v (*fst x*)⦈) *mod* (*3*::$\mathbb{N}$)⦈ = *snd x*
using *calculation* **by** *presburger*
qed

have *lhs-2-f0*: (λv_0. *?lhs-p* v_0 $*$ *?lhs-c* v_0 $*$ *?lhs-m* v_0 $*$ *?lhs-c-p* v_0 $*$ *?m-2-mod* v_0) =
(λv_0. (*if* p_v v_0 $\leq$ (*2*::$\mathbb{N}$) $\wedge$ c_v v_0 $\leq$ (*2*::$\mathbb{N}$) $\wedge$ m_v v_0 = m_v (*fst x*) $\wedge$ c_v v_0 = p_v v_0 $\wedge$
v_0⦇m_v := *Suc* (*Suc* (c_v v_0)) *mod* (*3*::$\mathbb{N}$)⦈ = *snd x then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)))
by *auto*
have *lhs-2-summable*: (λv_0. *?lhs-p* v_0 $*$ *?lhs-c* v_0 $*$ *?lhs-m* v_0 $*$ *?lhs-c-p* v_0 $*$ *?m-2-mod* v_0) *summable-on UNIV*
apply (*simp add*: *lhs-2-f0*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** *B*=

$\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-1-set-A-finite*)
by *blast*

have *lhs-2-infsum*: $(\sum_\infty v_0::mh\text{-}state.\ ?lhs\text{-}p\ v_0 * ?lhs\text{-}c\ v_0 * ?lhs\text{-}m\ v_0 * ?lhs\text{-}c\text{-}p\ v_0 * ?m\text{-}2\text{-}mod\ v_0)$
$= ?rhs\text{-}1\text{-}2$
apply (*simp only*: *lhs-2-f0*)
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** $B=$
$\{s::mh\text{-}state.\ p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v\ (fst\ x) \wedge c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-1-set-A-finite*)
apply (*blast*)
apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!|p_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),$
$c_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ m_v = m_v\ (fst\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*metis mh-state.select-convs*(*1*) *mod-Suc-le-divisor numeral-2-eq-2 numeral-3-eq-3*)
apply (*rule conjI*)
apply (*metis mh-state.select-convs*(*2*) *mod-Suc-le-divisor numeral-2-eq-2 numeral-3-eq-3*)
apply (*rule conjI*)
apply (*metis mh-state.select-convs*(*3*))
apply (*rule conjI*)
apply (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*))
defer
apply (*metis mh-state.surjective mh-state.update-convs*(*3*))
apply (*smt* (*verit*, *best*) *Collect-empty-eq mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*)
mh-state.select-convs(*3*) *mh-state.surjective mh-state.update-convs*(*3*) *card-eq-0-iff*
less-2-cases less-numeral-extra(*3*) *order-le-less*)
proof −
assume *a1*: $m_v\ (snd\ x) = Suc\ (Suc\ (c_v\ (snd\ x)))\ mod\ (3::\mathbb{N})$
assume *a2*: $c_v\ (snd\ x) = (0::\mathbb{N}) \vee c_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee c_v\ (snd\ x) = (2::\mathbb{N})$
assume *a3*: $p_v\ (snd\ x) = (0::\mathbb{N}) \vee p_v\ (snd\ x) = Suc\ (0::\mathbb{N}) \vee p_v\ (snd\ x) = (2::\mathbb{N})$
assume *a4*: $c_v\ (snd\ x) = p_v\ (snd\ x)$

have $(\!|p_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ c_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := Suc\ (Suc\ (c_v\ (\!|p_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ c_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),$
$m_v = m_v\ (fst\ x)|\!)))\ mod\ (3::\mathbb{N})|\!)$
$= (\!|p_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ c_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := Suc\ (Suc\ (Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}))\ mod\ (3::\mathbb{N}))\ mod\ (3::\mathbb{N})|\!)$
by (*metis mh-state.select-convs*(*2*) *mh-state.unfold-congs*(*3*) *mod-Suc-eq*)
also have ... $= (\!|p_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ c_v = Suc\ (m_v\ (snd\ x))\ mod\ (3::\mathbb{N}),\ m_v = m_v$
$(fst\ x)|\!)$
$(\!|m_v := (m_v\ (snd\ x))|\!)$
by (*simp add*: *a1 mod-Suc-eq*)
also have ... $= (\!|p_v = c_v\ (snd\ x),\ c_v = c_v\ (snd\ x),\ m_v = (m_v\ (snd\ x))|\!)$
by (*smt* (*z3*) *mh-state.update-convs*(*3*) *Suc-mod-eq-add3-mod-numeral a1 a3 a4*
add-cancel-left-left divmod-algorithm-code(*3*) *divmod-algorithm-code*(*4*) *mod-Suc mod-add-self1*
numeral-1-eq-Suc-0 numeral-2-eq-2 one-mod-two-eq-one plus-1-eq-Suc snd-divmod)
also have ... = *snd x*
by (*simp add*: *a4*)

then show $(\!| p_v = \mathit{Suc}\ (m_v\ (\mathit{snd}\ x))\ \mathit{mod}\ (3::\mathbb{N}),\ c_v = \mathit{Suc}\ (m_v\ (\mathit{snd}\ x))\ \mathit{mod}\ (3::\mathbb{N}),\ m_v = m_v\ (\mathit{fst}\ x) |\!)$
$(\!| m_v := \mathit{Suc}\ (\mathit{Suc}\ (c_v\ (\!| p_v = \mathit{Suc}\ (m_v\ (\mathit{snd}\ x))\ \mathit{mod}\ (3::\mathbb{N}),\ c_v = \mathit{Suc}\ (m_v\ (\mathit{snd}\ x))\ \mathit{mod}\ (3::\mathbb{N}),\ m_v = m_v\ (\mathit{fst}\ x) |\!)))\ \mathit{mod}\ (3::\mathbb{N}) |\!) = \mathit{snd}\ x$
using *calculation* **by** *presburger*
qed

have *lhs-3-f0*: $(\lambda v_0.\ ?\mathit{lhs\text{-}p}\ v_0 * ?\mathit{lhs\text{-}c}\ v_0 * ?\mathit{lhs\text{-}m}\ v_0 * ?\mathit{lhs\text{-}c\text{-}n\text{-}p}\ v_0 * ?\mathit{m\text{-}3\text{-}c\text{-}p}\ v_0) =$
$(\lambda v_0.\ (\mathit{if}\ p_v\ v_0 \le (2::\mathbb{N}) \wedge\ c_v\ v_0 \le (2::\mathbb{N}) \wedge m_v\ v_0 = m_v\ (\mathit{fst}\ x) \wedge \neg\ c_v\ v_0 = p_v\ v_0 \wedge$
$v_0(\!| m_v := (3::\mathbb{N}) - (c_v\ v_0 + p_v\ v_0) |\!) = \mathit{snd}\ x\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})))$
by *auto*
have *lhs-3-set-simp*: $\{s::\mathit{mh\text{-}state}.\ p_v\ s \le (2::\mathbb{N}) \wedge$
$c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v\ (\mathit{fst}\ x) \wedge\ \neg c_v\ s = p_v\ s\}$
$= \{(\!| p_v = 0::\mathbb{N},\ c_v = 1::\mathbb{N},\ m_v = m_v\ (\mathit{fst}\ x) |\!),\ (\!| p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = m_v\ (\mathit{fst}\ x) |\!),$
$(\!| p_v = \mathit{Suc}\ (0::\mathbb{N}),\ c_v = (0::\mathbb{N}),\ m_v = m_v\ (\mathit{fst}\ x) |\!),\ (\!| p_v = \mathit{Suc}\ (0::\mathbb{N}),\ c_v = (2::\mathbb{N}),\ m_v = m_v\ (\mathit{fst}\ x) |\!),$
$(\!| p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = m_v\ (\mathit{fst}\ x) |\!),\ (\!| p_v = 2::\mathbb{N},\ c_v = \mathit{Suc}\ (0::\mathbb{N}),\ m_v = m_v\ (\mathit{fst}\ x) |\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*metis* (*mono-tags*, *opaque-lifting*) *mh-state.surjective bot-nat-0.extremum le-SucE*
le-antisym numeral-2-eq-2 old.unit.exhaust)
by *fastforce*
have *lhs-3-set-A-finite*: *finite* $\{s::\mathit{mh\text{-}state}.\ p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v\ (\mathit{fst}\ x) \wedge \neg c_v\ s = p_v\ s\}$
by (*simp add*: *lhs-3-set-simp*)
have *lhs-3-summable*: $(\lambda v_0.\ ?\mathit{lhs\text{-}p}\ v_0 * ?\mathit{lhs\text{-}c}\ v_0 * ?\mathit{lhs\text{-}m}\ v_0 * ?\mathit{lhs\text{-}c\text{-}n\text{-}p}\ v_0 * ?\mathit{m\text{-}3\text{-}c\text{-}p}\ v_0)$ *summable-on UNIV*
apply (*simp add*: *lhs-3-f0*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** *B*=
$\{s::\mathit{mh\text{-}state}.\ p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v\ (\mathit{fst}\ x) \wedge \neg c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-3-set-A-finite*)
by *blast*

have *lhs-3-infsum*: $(\sum_\infty v_0::\mathit{mh\text{-}state}.\ ?\mathit{lhs\text{-}p}\ v_0 * ?\mathit{lhs\text{-}c}\ v_0 * ?\mathit{lhs\text{-}m}\ v_0 * ?\mathit{lhs\text{-}c\text{-}n\text{-}p}\ v_0 * ?\mathit{m\text{-}3\text{-}c\text{-}p}\ v_0)$
$= ?\mathit{rhs\text{-}1\text{-}3}$
apply (*simp only*: *lhs-3-f0*)
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** *B*=
$\{s::\mathit{mh\text{-}state}.\ p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v\ (\mathit{fst}\ x) \wedge \neg c_v\ s = p_v\ s\}$])
apply (*simp add*: *lhs-3-set-A-finite*)
apply (*blast*)
apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!| p_v = (3::\mathbb{N}) - (m_v\ (\mathit{snd}\ x)) - c_v\ (\mathit{snd}\ x),$
$c_v = (3::\mathbb{N}) - (m_v\ (\mathit{snd}\ x)) - p_v\ (\mathit{snd}\ x),\ m_v = m_v\ (\mathit{fst}\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply *fastforce*
apply (*rule conjI*)

apply *fastforce*
apply (*rule conjI*)
apply (*simp*)
apply (*rule conjI*)
apply *fastforce*
defer
apply (*metis mh-state.surjective mh-state.update-convs*(*3*))
apply (*smt* (*verit*, *best*) *Collect-empty-eq mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*3*) *card-eq-0-iff less-2-cases less-numeral-extra*(*3*) *order-le-less*)
proof −
assume *a1*: $m_v\ (snd\ x) = (3{::}\mathbb{N}) - (c_v\ (snd\ x) + p_v\ (snd\ x))$
assume *a2*: $c_v\ (snd\ x) = (0{::}\mathbb{N}) \lor c_v\ (snd\ x) = Suc\ (0{::}\mathbb{N}) \lor c_v\ (snd\ x) = (2{::}\mathbb{N})$
assume *a3*: $p_v\ (snd\ x) = (0{::}\mathbb{N}) \lor p_v\ (snd\ x) = Suc\ (0{::}\mathbb{N}) \lor p_v\ (snd\ x) = (2{::}\mathbb{N})$
assume *a4*: $\neg\ c_v\ (snd\ x) = p_v\ (snd\ x)$

have *f0*: $(3{::}\mathbb{N}) - (((3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x)) + ((3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x)))$
$= (2 * m_v\ (snd\ x)) + p_v\ (snd\ x) + c_v\ (snd\ x) - 3$
using *a1 a2 a3 diff-zero* **by** *fastforce*
also have *f1*: $\ldots = 3 - p_v\ (snd\ x) - c_v\ (snd\ x)$
using *a1 a2 a3 a4* **by** *auto*
have *lhs-0*: $(\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := (3{::}\mathbb{N}) - (c_v\ (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!) +$
$p_v\ (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!))|\!)$
$= (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := (3{::}\mathbb{N}) - (((3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x)) + ((3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x)))|\!)$
by *force*
have *lhs-1*: $\ldots = (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := 3 - p_v\ (snd\ x) - c_v\ (snd\ x)|\!)$
using *f0 f1* **by** *presburger*
have *lhs-2*: $\ldots = (\!|p_v = p_v\ (snd\ x),\ c_v = c_v\ (snd\ x),\ m_v = m_v\ (snd\ x)|\!)$
using *mh-state.update-convs*(*3*) *a1 a2 a3 a4 add.commute add.right-neutral* **by** *fastforce*
have *lhs-3*: $\ldots = snd\ x$
by (*simp add*: *a4*)
show $(\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!)$
$(\!|m_v := (3{::}\mathbb{N}) - (c_v\ (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!) +$
$p_v\ (\!|p_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - c_v\ (snd\ x),\ c_v = (3{::}\mathbb{N}) - m_v\ (snd\ x) - p_v\ (snd\ x),\ m_v = m_v\ (fst\ x)|\!))|\!) = snd\ x$
using *lhs-0 lhs-1 lhs-2 lhs-3* **by** *presburger*
qed

have *lhs-1*: $?lhs = (\sum_\infty v_0{::}mh\text{-}state.$
$?lhs\text{-}p\ v_0 * ?lhs\text{-}c\ v_0 * ?lhs\text{-}m\ v_0 *$
$(?lhs\text{-}c\text{-}p\ v_0 * ?m\text{-}1\text{-}mod\ v_0\ /\ (18{::}\mathbb{R}) +$
$?lhs\text{-}c\text{-}p\ v_0 * ?m\text{-}2\text{-}mod\ v_0\ /\ (18{::}\mathbb{R}) +$
$?lhs\text{-}c\text{-}n\text{-}p\ v_0 * ?m\text{-}3\text{-}c\text{-}p\ v_0\ /\ (9{::}\mathbb{R})))$
apply (*rule infsum-cong*)
by (*simp add*: *add-divide-distrib*)

also have *lhs-2*: ... = $(\sum_\infty v_0$::*mh-state*.
?lhs-p v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-1-mod* v_0 / (*18*::ℝ) +
?lhs-p v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-2-mod* v_0 / (*18*::ℝ) +
?lhs-p v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-n-p* v_0 ∗ *?m-3-c-p* v_0 / (*9*::ℝ))
apply (*rule infsum-cong*)
by *simp*
also have *lhs-3*: ... =
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-1-mod* v_0 / (*18*::ℝ)) +
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-2-mod* v_0 / (*18*::ℝ)) +
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-n-p* v_0 ∗ *?m-3-c-p* v_0 / (*9*::ℝ))
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cdiv-left*)
using *lhs-1-summable* **apply** *blast*
apply (*rule summable-on-cdiv-left*)
using *lhs-2-summable* **apply** *blast*
apply (*rule summable-on-cdiv-left*)
using *lhs-3-summable* **apply** *blast*
apply (*subst infsum-add*)
apply (*rule summable-on-cdiv-left*)
using *lhs-1-summable* **apply** *blast*
apply (*rule summable-on-cdiv-left*)
using *lhs-2-summable* **apply** *blast*
by *meson*
also have *lhs-4*: ... =
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-1-mod* v_0) / (*18*::ℝ) +
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-p* v_0 ∗ *?m-2-mod* v_0) / (*18*::ℝ) +
$(\sum_\infty v_0$::*mh-state*. *?lhs-p* v_0 ∗ *?lhs-c* v_0 ∗ *?lhs-m* v_0 ∗ *?lhs-c-n-p* v_0 ∗ *?m-3-c-p* v_0) / (*9*::ℝ)
apply (*subst infsum-cdiv-left*)
using *lhs-1-summable* **apply** *blast*
apply (*subst infsum-cdiv-left*)
using *lhs-2-summable* **apply** *blast*
apply (*subst infsum-cdiv-left*)
using *lhs-3-summable* **apply** *blast*
by *simp*
then show *?lhs* = *?rhs*
using *calculation lhs-1-infsum lhs-2-infsum lhs-3-infsum rhs-1-1 rhs-1-2 rhs-1-3* **by** *presburger*
qed

lemma *IMHA-NC-altdef-states-1-eq*:
{*s*::*mh-state*. ((c_v *s* = p_v *s* ∧ p_v *s* ≤ (*2*::ℕ)) ∧ c_v *s* ≤ (*2*::ℕ)) ∧ m_v *s* = *Suc* (c_v *s*) *mod* (*3*::ℕ)}
= {⦇p_v = *0*::ℕ, c_v = *0*::ℕ, m_v = *Suc* (*0*::ℕ)⦈,⦇p_v = *Suc* (*0*::ℕ), c_v = *Suc* (*0*::ℕ), m_v = (*2*::ℕ)⦈,
⦇p_v = *2*::ℕ, c_v = *2*::ℕ, m_v = *0*::ℕ⦈}
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*z3*) *mh-state.surjective Orderings.order-eq-iff Suc-eq-numeral add.assoc cong-exp-iff-simps*(*2*) *diff-add-zero diff-is-0-eq le-SucE mod-Suc mod-self numeral-1-eq-Suc-0 numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust one-eq-numeral-iff plus-1-eq-Suc*)
by *force*

lemma *IMHA-NC-altdef-states-2-eq*:
{*s*::*mh-state*. ((c_v *s* = p_v *s* ∧ p_v *s* ≤ (*2*::ℕ)) ∧ c_v *s* ≤ (*2*::ℕ)) ∧ m_v *s* = *Suc* (*Suc* (c_v *s*)) *mod* (*3*::ℕ)}
= {⦇p_v = *0*::ℕ, c_v = *0*::ℕ, m_v = (*2*::ℕ)⦈, ⦇p_v = *Suc* (*0*::ℕ), c_v = *Suc* (*0*::ℕ), m_v = (*0*::ℕ)⦈,
⦇p_v = *2*::ℕ, c_v = *2*::ℕ, m_v = *Suc* (*0*::ℕ)⦈}

apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective lessI less-2-cases mod-Suc mod-less numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less*)
by *force*

lemma *IMHA-NC-altdef-states-3-eq*:
$\{s{::}mh\text{-}state.\ ((\neg\ c_v\ s = p_v\ s \wedge p_v\ s \le (2{::}\mathbb{N})) \wedge c_v\ s \le (2{::}\mathbb{N})) \wedge m_v\ s = (3{::}\mathbb{N}) - (c_v\ s + p_v\ s)\}$
$= \{(\!|p_v = 0{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = (2{::}\mathbb{N})|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = (2{::}\mathbb{N}),\ m_v = Suc\ (0{::}\mathbb{N})|\!),$
$(\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = (0{::}\mathbb{N}),\ m_v = (2{::}\mathbb{N})|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = (2{::}\mathbb{N}),\ m_v = (0{::}\mathbb{N})|\!),$
$(\!|p_v = 2{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N})|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = (0{::}\mathbb{N})|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *ccfv-SIG*) *mh-state.surjective One-nat-def diff-add-inverse diff-diff-eq less-2-cases numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

lemma *IMHA--NC-win*: $rvfun\text{-}of\text{-}prfun\ (IMHA\text{-}NC)\ ;\ [\![c^{<} = p^{<}]\!]_{\mathcal{I}e} = (1/3)_e$
apply (*simp add*: *IMHA-NC-altdef*)
apply (*subst rvfun-inverse*)
using *IMHA-NC-altdef-dist* **apply** (*simp add*: *is-dist-def is-final-prob-prob*)
apply (*simp add*: *IMHA-NC-altdef-def*)
apply (*expr-auto*)
apply (*simp add*: *ring-distribs*(*2*))
proof −
let $?lhs\text{-}1 = \lambda s{::}mh\text{-}state.\ (if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}2 = \lambda s{::}mh\text{-}state.\ (if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}3 = \lambda s{::}mh\text{-}state.\ (if\ \neg\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ s = (3{::}\mathbb{N}) - (c_v\ s + p_v\ s)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs = \lambda s{::}mh\text{-}state.\ ?lhs\text{-}1\ s\ /\ (18{::}\mathbb{R}) + ?lhs\text{-}2\ s\ /\ (18{::}\mathbb{R}) + ?lhs\text{-}3\ s\ /\ (9{::}\mathbb{R})$

let $?lhs\text{-}1' = \lambda s{::}mh\text{-}state.\ (if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$

let $?lhs\text{-}2' = \lambda s{::}mh\text{-}state.\ (if\ c_v\ s = p_v\ s\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ c_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$

have *lhs-1-eq*: $?lhs\text{-}1 = ?lhs\text{-}1'$

by *auto*
have *lhs-2-eq*: *?lhs-2* = *?lhs-2'*
by *auto*

have *lhs-3-zero*: *?lhs-3* = (*λs::mh-state. 0*)
by *auto*

have *lhs-1-summable*: *?lhs-1 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-NC-altdef-states-1-eq* **apply** (*metis* (*mono-tags*, *lifting*) *Collect-mono finite.emptyI finite.insertI finite-subset*)
by *simp*

have *lhs-2-summable*: *?lhs-2 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-NC-altdef-states-2-eq* **apply** (*metis* (*mono-tags*, *lifting*) *Collect-mono finite.emptyI finite.insertI finite-subset*)
by *simp*

have *lhs-3-summable*: *?lhs-3 summable-on UNIV*
by (*meson lhs-3-zero summable-on-0*)

have *lhs-1-infsum*: ($\sum_\infty$*s::mh-state. ?lhs-1 s*) = *3*
apply (*simp add*: *lhs-1-eq*)
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *IMHA-NC-altdef-states-1-eq* **apply** (*metis* (*no-types*, *lifting*) *finite.emptyI finite.insertI*)
apply (*subst IMHA-NC-altdef-states-1-eq*)
by *auto*

have *lhs-2-infsum*: ($\sum_\infty$*s::mh-state. ?lhs-2 s*) = *3*
apply (*simp add*: *lhs-2-eq*)
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *IMHA-NC-altdef-states-2-eq* **apply** (*metis* (*no-types*, *lifting*) *finite.emptyI finite.insertI*)
apply (*subst IMHA-NC-altdef-states-2-eq*)
by *auto*

have *lhs-3-infsum*: ($\sum_\infty$*s::mh-state. ?lhs-3 s*) = *0*
by (*simp add*: *lhs-3-zero*)

show ($\sum_\infty$*s::mh-state. ?lhs s*) ∗ *3* = *1*
apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)

apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
using *lhs-1-infsum lhs-2-infsum lhs-3-infsum* **by** (*simp*)
qed

2.4.1 Average values

Average value of p after the execution of *IMHA-C*, a No Change Strategy.

lemma *IMHA-NC-average-p*: *rvfun-of-prfun IMHA-NC* ; $(\$p^{<})_e = (1)_e$
apply (*simp add*: *IMHA-NC-altdef*)
apply (*subst rvfun-inverse*)
using *IMHA-NC-altdef-dist*
apply (*simp add*: *is-final-distribution-prob is-final-prob-prob*)
apply (*simp add*: *IMHA-NC-altdef-def*)
apply (*expr-auto*)
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst conditional-conds-conj*)+
apply (*subst times-divide-eq-right*[*symmetric*])+
apply (*subst conditional-cmult-1*)+
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-NC-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-NC-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-NC-altdef-states-3-eq*)
apply *blast*+
apply (*subst infsum-add*)
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-NC-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-NC-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-NC-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-NC-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-NC-altdef-states-3-eq*)
apply *blast*+
apply (*subst IMHA-NC-altdef-states-1-eq*)
apply (*subst IMHA-NC-altdef-states-2-eq*)
apply (*subst IMHA-NC-altdef-states-3-eq*)

apply (*subst sum-divide-distrib*[*symmetric*])+
by (*simp*)

2.5 *IMHA-C*

definition *IMHA-C-altdef* :: *mh-state* × *mh-state* ⇒ ℝ **where**
IMHA-C-altdef = (
$(\llbracket p^{>} \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^{>} = 3 - p^{>} - m^{>}\rrbracket_{\mathcal{I}e} * \llbracket\ m^{>} = (p^{>} + 1)\ mod\ 3\ \rrbracket_{\mathcal{I}e}\ /\ 18) +$
$(\llbracket p^{>} \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^{>} = 3 - p^{>} - m^{>}\rrbracket_{\mathcal{I}e} * \llbracket\ m^{>} = (p^{>} + 2)\ mod\ 3\ \rrbracket_{\mathcal{I}e}\ /\ 18) +$
$(\llbracket 3 - m^{>} - p^{>} \neq p^{>}\rrbracket_{\mathcal{I}e} * \llbracket p^{>} \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket 3 - m^{>} - p^{>} \leq 2\rrbracket_{\mathcal{I}e} * \llbracket 3 - m^{>} \geq p^{>}\rrbracket_{\mathcal{I}e} * \llbracket\ c^{>}$
$= p^{>}\ \rrbracket_{\mathcal{I}e}\ /\ 9)$
$)_e$

lemma *IMHA-C-altdef-dist*: *is-final-distribution IMHA-C-altdef*
proof −
let *?lhs-1* = $\lambda(s_1$::*mh-state*) *s*::*mh-state*.
(*if* get_p (get_{snd_L} (s_1, s)) ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_c (get_{snd_L} (s_1, s)) =
(*3*::ℕ) − (get_p (get_{snd_L} (s_1, s)) + get_m (get_{snd_L} (s_1, s)))
then *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_m (get_{snd_L} (s_1, s)) = *Suc* (get_p (get_{snd_L} (s_1, s))) *mod* (*3*::ℕ) *then* *1*::ℝ
else (*0*::ℝ))
let *?lhs-2* = $\lambda(s_1$::*mh-state*) *s*::*mh-state*.
(*if* get_p (get_{snd_L} (s_1, s)) ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_c (get_{snd_L} (s_1, s)) =
(*3*::ℕ) − (get_p (get_{snd_L} (s_1, s)) + get_m (get_{snd_L} (s_1, s)))
then *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_m (get_{snd_L} (s_1, s)) = *Suc* (*Suc* (get_p (get_{snd_L} (s_1, s)))) *mod* (*3*::ℕ) *then* *1*::ℝ
else (*0*::ℝ))
let *?lhs-3* = $\lambda(s_1$::*mh-state*) *s*::*mh-state*.
(*if* ¬ (*3*::ℕ) − (get_m (get_{snd_L} (s_1, s)) + get_p (get_{snd_L} (s_1, s))) =
get_p (get_{snd_L} (s_1, s)) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_p (get_{snd_L} (s_1, s)) ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* (*3*::ℕ) − (get_m (get_{snd_L} (s_1, s)) + get_p (get_{snd_L} (s_1, s))) ≤ (*2*::ℕ) *then* *1*::ℝ
else (*0*::ℝ)) ∗
(*if* get_p (get_{snd_L} (s_1, s)) ≤ (*3*::ℕ) − get_m (get_{snd_L} (s_1, s)) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* get_c (get_{snd_L} (s_1, s)) = get_p (get_{snd_L} (s_1, s)) *then* *1*::ℝ *else* (*0*::ℝ))
let *?lhs* = $\lambda(s_1$::*mh-state*) *s*::*mh-state*. *?lhs-1* s_1 *s* / *18* + *?lhs-2* s_1 *s* / *18* + *?lhs-3* s_1 *s* / *9*

let *?lhs-1'* = λ*s*::*mh-state*.
(*if* p_v *s* ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* c_v *s* = (*3*::ℕ) − (p_v *s* + m_v *s*) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* m_v *s* = *Suc* (p_v *s*) *mod* (*3*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ))
let *?lhs-2'* = λ*s*::*mh-state*. (*if* p_v *s* ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* c_v *s* = (*3*::ℕ) − (p_v *s* + m_v *s*) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* m_v *s* = *Suc* (*Suc* (p_v *s*)) *mod* (*3*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ))
let *?lhs-3'* = λ*s*::*mh-state*. (*if* ¬ (*3*::ℕ) − (m_v *s* + p_v *s*) = p_v *s then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* p_v *s* ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* (*3*::ℕ) − (m_v *s* + p_v *s*) ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* p_v *s* ≤ (*3*::ℕ) − m_v *s then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* c_v *s* = p_v *s then* *1*::ℝ *else* (*0*::ℝ))
let *?lhs'* = λ*s*::*mh-state*. *?lhs-1'* *s* / *18* + *?lhs-2'* *s* / *18* + *?lhs-3'* *s* / *9*

have *lhs-1-eq*: ∀ (s_1::*mh-state*) *s*::*mh-state*. *?lhs-1* s_1 *s* = *?lhs-1'* *s*

by (*expr-simp*)

have *lhs-2-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\text{-}2\ s_1\ s = ?lhs\text{-}2'\ s$
by (*expr-simp*)

have *lhs-3-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\text{-}3\ s_1\ s = ?lhs\text{-}3'\ s$
by (*pred-simp*)

have *lhs-lhs′-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\ s_1\ s = ?lhs'\ s$
by (*simp add*: *c-def m-def p-def*)

have *states-1-eq*:
$\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (p_v\ s)\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = (0::\mathbb{N}),\ m_v = (2::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 1::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective Nat.add-0-right Nat.add-diff-assoc One-nat-def Suc-1 Suc-le-mono add.commute add-2-eq-Suc′ add-cancel-left-left bot-nat-0.extremum diff-Suc-Suc diff-Suc-diff-eq2 diff-diff-left diff-is-0-eq diff-self-eq-0 eval-nat-numeral*(*3*) *le0 le-SucE le-antisym lessI less-2-cases mod-Suc mod-Suc-eq-mod-add3 mod-by-Suc-0 mod-less mod-mod-trivial mod-self nat.inject not-mod2-eq-Suc-0-eq-0 numeral-1-eq-Suc-0 numeral-3-eq-3 numeral-plus-numeral old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

have *infsum-lhs-1*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}1'\ s) = 3$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-1-eq* **apply** *auto*[*1*]
using *states-1-eq* **by** *force*

have *states-2-eq*:
$\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (Suc\ (p_v\ s))\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = (2::\mathbb{N}),\ m_v = (0::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = 1::\mathbb{N}|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective Nat.add-0-right Nat.add-diff-assoc One-nat-def Suc-1 Suc-le-mono add.commute add-2-eq-Suc′ add-cancel-left-left bot-nat-0.extremum diff-Suc-Suc diff-Suc-diff-eq2 diff-diff-left diff-is-0-eq diff-self-eq-0 eval-nat-numeral*(*3*) *le0 le-SucE le-antisym lessI less-2-cases mod-Suc mod-Suc-eq-mod-add3 mod-by-Suc-0 mod-less mod-mod-trivial mod-self nat.inject not-mod2-eq-Suc-0-eq-0 numeral-1-eq-Suc-0 numeral-3-eq-3 numeral-plus-numeral old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

have *infsum-lhs-2*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}2'\ s) = 3$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-2-eq* **apply** *auto*[*1*]
using *states-2-eq* **by** *force*

have *states-3-eq*:

$\{s::mh\text{-}state.\ (((\neg\ (3::\mathbb{N}) - (m_v\ s + p_v\ s) = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge$
$(3::\mathbb{N}) - (m_v\ s + p_v\ s) \leq (2::\mathbb{N})) \wedge p_v\ s \leq (3::\mathbb{N}) - m_v\ s) \wedge c_v\ s = p_v\ s\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),$
$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (0::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v =$
$(2::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *ccfv-SIG*) *mh-state.ext-inject mh-state.select-convs*(*1*)
mh-state.select-convs(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective*
Nat.add-0-right One-nat-def Suc-1 Suc-eq-numeral bot-nat-0.extremum diff-add-inverse
diff-commute diff-diff-cancel diff-diff-left diff-is-0-eq diff-is-0-eq′ diff-le-self
diff-self-eq-0 eval-nat-numeral(*3*) *le-Suc-eq le-antisym less-2-cases nat.distinct*(*1*)
nle-le not-less-eq-eq old.nat.exhaust old.unit.exhaust order-le-less plus-1-eq-Suc)
by *force*

have *infsum-lhs-3*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}3'\ s) = 6$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *states-3-eq* **apply** *auto*[*1*]
using *states-3-eq* **by** *force*

have *lhs-1-summable*: *?lhs-1′ summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-1-eq* **by** (*simp-all*)

have *lhs-2-summable*: *?lhs-2′ summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-2-eq* **by** (*simp-all*)

have *lhs-3-summable*: *?lhs-3′ summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *states-3-eq* **by** (*simp-all*)

have *infsum-lhs-lhs′-eq*: $\forall\, s_1::mh\text{-}state.\ (\sum_\infty s::mh\text{-}state.\ ?lhs\ s_1\ s) = (\sum_\infty s::mh\text{-}state.\ ?lhs'\ s)$
apply (*rule allI*)
by (*metis* (*full-types*) *lhs-lhs′-eq*)

have *infsum-lhs′-1*: $(\sum_\infty s::mh\text{-}state.\ ?lhs'\ s) = 1$
apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)

```
    apply (simp-all add: lhs-2-summable)
    apply (subst infsum-cdiv-left)
    apply (simp-all add: lhs-1-summable)
    apply (subst infsum-cdiv-left)
    apply (simp-all add: lhs-2-summable)
    apply (subst infsum-cdiv-left)
    apply (simp-all add: lhs-3-summable)
    using infsum-lhs-1 infsum-lhs-2 infsum-lhs-3 by (simp)

  have infsum-lhs-1: ∀ s₁::mh-state. (∑∞s::mh-state. ?lhs s₁ s) = 1
    using infsum-lhs'-1 infsum-lhs-lhs'-eq by presburger

  have lhs'-leq-1: ∀ s::mh-state. ?lhs' s ≤ infsum ?lhs' UNIV
    apply (rule allI)
    apply (rule infsum-geq-element)
    apply fastforce
    apply (subst summable-on-add)
    apply (subst summable-on-add)
    apply (subst summable-on-cdiv-left)
    apply (simp-all add: lhs-1-summable)
    apply (subst summable-on-cdiv-left)
    apply (simp-all add: lhs-2-summable)
    apply (subst summable-on-cdiv-left)
    by (simp-all add: lhs-3-summable)
  have lhs'-leq-1': ∀ s::mh-state. ?lhs' s ≤ 1
    using infsum-lhs'-1 lhs'-leq-1 by presburger
  have lhs-leq-1: ∀ s₁::mh-state. (∀ s::mh-state. ?lhs s₁ s ≤ 1)
    by (simp add: c-def lhs'-leq-1' m-def p-def )

  show ?thesis
  apply (simp add: IMHA-C-altdef-def)
  apply (simp add: dist-defs)
  apply (simp only: expr-defs)
  apply (rule allI)
  apply (rule conjI)
  apply (rule allI)
  apply (rule conjI)
  using add-divide-distrib div-by-1 divide-divide-eq-right divide-le-0-1-iff mult-not-zero apply auto[1]
  using lhs-leq-1 apply blast
  using infsum-lhs-1 by blast
qed

lemma IMHA-C-altdef: IMHA-C = prfun-of-rvfun IMHA-C-altdef
  apply (simp only: IMHA-C-def MHA-C-def)
  apply (subst prfun-seqcomp-assoc)
  apply (rule INIT-is-dist)
    apply (rule MHA-is-dist)
  apply (simp add: passigns-def rvfun-assignment-inverse rvfun-assignment-is-dist)
  apply (simp add: MHA-NC-MHA-eq[symmetric])
  apply (simp add: IMHA-NC-def[symmetric])
  apply (simp add: IMHA-NC-altdef)
  apply (simp add: pfun-defs)
  apply (subst rvfun-inverse)
  using IMHA-NC-altdef-dist apply (simp add: is-final-distribution-prob is-final-prob-prob)
  apply (simp add: rvfun-assignment-inverse)
```

apply (*simp add*: *IMHA-NC-altdef-def IMHA-C-altdef-def*)
apply (*expr-simp-1 add*: *assigns-r-def*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp only*: *fun-eq-iff*)
apply (*rule allI*)
apply (*subst ring-distribs*(*2*))
apply (*subst ring-distribs*(*2*))
apply (*subst times-divide-eq-left*)+
proof −
fix x::*mh-state* × *mh-state*
let *?lhs-1* = λv_0::*mh-state*. ($if\ c_v\ v_0 = p_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ p_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ snd\ x = v_0(\!|c_v := (3::\mathbb{N}) - (c_v\ v_0 + m_v\ v_0)|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?lhs-2* = λv_0::*mh-state*. $(if\ c_v\ v_0 = p_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ p_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (Suc\ (c_v\ v_0))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ snd\ x = v_0(\!|c_v := (3::\mathbb{N}) - (c_v\ v_0 + m_v\ v_0)|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?lhs-3* = λv_0::*mh-state*. $(if\ \neg\ c_v\ v_0 = p_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ p_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = (3::\mathbb{N}) - (c_v\ v_0 + p_v\ v_0)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ snd\ x = v_0(\!|c_v := (3::\mathbb{N}) - (c_v\ v_0 + m_v\ v_0)|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?lhs* = λs::*mh-state*. *?lhs-1 s* / $(18::\mathbb{R})$ + *?lhs-2 s* / $(18::\mathbb{R})$ + *?lhs-3 s* / $(9::\mathbb{R})$

let *?rhs-1* = $(if\ p_v\ (snd\ x) \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ (snd\ x) = (3::\mathbb{N}) - (p_v\ (snd\ x) + m_v\ (snd\ x))\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ (snd\ x) = Suc\ (p_v\ (snd\ x))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?rhs-2* = $(if\ p_v\ (snd\ x) \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ (snd\ x) = (3::\mathbb{N}) - (p_v\ (snd\ x) + m_v\ (snd\ x))\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ (snd\ x) = Suc\ (Suc\ (p_v\ (snd\ x)))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?rhs-3* = $(if\ \neg\ (3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) = p_v\ (snd\ x)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ p_v\ (snd\ x) \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ (3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ p_v\ (snd\ x) \le (3::\mathbb{N}) - m_v\ (snd\ x)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ (snd\ x) = p_v\ (snd\ x)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let *?rhs* = *?rhs-1* / $(18::\mathbb{R})$ + *?rhs-2* / $(18::\mathbb{R})$ + *?rhs-3* / $(9::\mathbb{R})$

have *states-1-eq*: $\{s::mh\text{-}state.\ ((c_v\ s = p_v\ s \wedge p_v\ s \le (2::\mathbb{N})) \wedge c_v\ s \le (2::\mathbb{N})) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (2::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*z3*) *mh-state.surjective Orderings.order-eq-iff Suc-eq-numeral add.assoc cong-exp-iff-simps*(*2*) *diff-add-zero diff-is-0-eq le-SucE mod-Suc mod-self numeral-1-eq-Suc-0 numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust one-eq-numeral-iff plus-1-eq-Suc*)
by *force*

have *states-2-eq*: $\{s::mh\text{-}state.\ ((c_v\ s = p_v\ s \wedge p_v\ s \le (2::\mathbb{N})) \wedge c_v\ s \le (2::\mathbb{N})) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (0::\mathbb{N})|\!),$

$(\!| p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective lessI less-2-cases mod-Suc mod-less numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less*)
by *force*

have *states-3-eq*: $\{s::mh\text{-}state.\ ((\neg\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s)\}$
$= \{(\!| p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (2::\mathbb{N}) |\!),\ (\!| p_v = 0::\mathbb{N},\ c_v = (2::\mathbb{N}),\ m_v = Suc\ (0::\mathbb{N}) |\!),$
$(\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = (0::\mathbb{N}),\ m_v = (2::\mathbb{N}) |\!),\ (\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = (2::\mathbb{N}),\ m_v = (0::\mathbb{N}) |\!),$
$(\!| p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!),\ (\!| p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (0::\mathbb{N}) |\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *ccfv-SIG*) *mh-state.surjective One-nat-def diff-add-inverse diff-diff-eq less-2-cases numeral-2-eq-2 numeral-3-eq-3 old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

have *lhs-1-summable*: *?lhs-1 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B=\{s::mh\text{-}state.$
$((c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\}$])
using *states-1-eq* **apply** *simp*
apply *blast*
by *simp*

have *lhs-2-summable*: *?lhs-2 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B= \{s::mh\text{-}state.$
$((c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\}$])
using *states-2-eq* **apply** *simp*
apply *blast*
by *simp*

have *lhs-3-summable*: *?lhs-3 summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B= \{s::mh\text{-}state.\ ((\neg\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge$
$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s)\}$])
using *states-3-eq* **apply** *simp*
apply *blast*
by *simp*

have *lhs-1-infsum*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}1\ s) = ?rhs\text{-}1$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** $B=\{s::mh\text{-}state.$
$((c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\}$])
using *states-1-eq* **apply** *simp*
apply *fastforce*

apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!| p_v = p_v\ (snd\ x),\ c_v = p_v\ (snd\ x),\ m_v = m_v\ (snd\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*simp*)
apply (*simp*)
apply (*metis* (*no-types*, *lifting*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*2*))
apply (*auto*)
proof −
assume *a1*: $\neg\ c_v\ (snd\ x) = (3::\mathbb{N}) - (p_v\ (snd\ x) + m_v\ (snd\ x))$
have $\neg(\exists\, s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*)
mh-state.surjective mh-state.update-convs(*2*))
then show *card* $\{s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0::\mathbb{N})$
using *card-0-singleton* **by** *blast*
next
assume *a1*: $\neg\ p_v\ (snd\ x) \leq (2::\mathbb{N})$
have $\neg(\exists\, s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*)
mh-state.surjective mh-state.update-convs(*2*))
then show *card* $\{s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0::\mathbb{N})$
using *card-0-singleton* **by** *blast*
next
assume *a1*: $\neg\ m_v\ (snd\ x) = Suc\ (p_v\ (snd\ x))\ mod\ (3::\mathbb{N})$
have $\neg(\exists\, s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*)
mh-state.surjective mh-state.update-convs(*2*))
then show *card* $\{s::mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0::\mathbb{N})$
using *card-0-singleton* **by** *blast*
qed

have *lhs-2-infsum*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}2\ s) = ?rhs\text{-}2$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** $B=\{s::mh\text{-}state.$
$((c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N})) \wedge c_v\ s \leq (2::\mathbb{N})) \wedge m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\}$])
using *states-2-eq* **apply** *simp*
apply *fastforce*
apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!| p_v = p_v\ (snd\ x),\ c_v = p_v\ (snd\ x),\ m_v = m_v\ (snd\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+

apply (*rule conjI*)
apply (*simp*)
apply (*simp*)
apply (*metis mh-state.select-convs*(*1*) *mh-state.surjective mh-state.update-convs*(*2*))
apply (*auto*)
proof −
assume *a1*: $\neg\ c_v\ (snd\ x) = (3{::}\mathbb{N}) - (p_v\ (snd\ x) + m_v\ (snd\ x))$
have $\neg(\exists\, s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*2*))
then show $card\ \{s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0{::}\mathbb{N})$
using *card-0-singleton* **by** *blast*
next
assume *a1*: $\neg\ p_v\ (snd\ x) \leq (2{::}\mathbb{N})$
have $\neg(\exists\, s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*2*))
then show $card\ \{s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0{::}\mathbb{N})$
using *card-0-singleton* **by** *blast*
next
assume *a1*: $\neg\ m_v\ (snd\ x) = Suc\ (Suc\ (p_v\ (snd\ x)))\ mod\ (3{::}\mathbb{N})$
have $\neg(\exists\, s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!))$
using *a1* **by** (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*2*))
then show $card\ \{s{::}mh\text{-}state.\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) \wedge snd\ x = s(\!|c_v := (3{::}\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = (0{::}\mathbb{N})$
using *card-0-singleton* **by** *blast*
qed

have *lhs-3-infsum*: $(\sum_\infty s{::}mh\text{-}state.\ ?lhs\text{-}3\ s) = ?rhs\text{-}3$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ ((\neg\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2{::}\mathbb{N})) \wedge$
$c_v\ s \leq (2{::}\mathbb{N})) \wedge\ m_v\ s = (3{::}\mathbb{N}) - (c_v\ s + p_v\ s)\}$])
using *states-3-eq* **apply** *simp*
apply *fastforce*
apply (*simp add*: *if-bool-eq-conj*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule card-1-singleton*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = (\!|p_v = p_v\ (snd\ x),\ c_v = 3 - (p_v\ (snd\ x) + m_v\ (snd\ x)),\ m_v = m_v\ (snd\ x)\ |\!)$ **in** *exI*)
apply (*erule conjE*)+
apply (*rule conjI*, *simp*)
apply (*rule conjI*, *simp*)
apply (*rule conjI*, *simp*)
apply (*rule conjI*, *simp*)
apply *simp*
apply (*erule conjE*)+

proof −
fix s::*mh-state* **and** y::*mh-state*
assume *s1*: $snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)$
assume *y1*: $snd\ x = \ y(\!|c_v := (3::\mathbb{N}) - (c_v\ y + m_v\ y)|\!)$
assume *s2*: $m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s)$
assume *y2*: $m_v\ y = (3::\mathbb{N}) - (c_v\ y + p_v\ y)$
assume *s3*: $p_v\ s \le (2::\mathbb{N})$
assume *y3*: $p_v\ y \le (2::\mathbb{N})$
assume *s4*: $p_v\ (snd\ x) \le (2::\mathbb{N})$
assume $(3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) \le (2::\mathbb{N})$
assume $p_v\ (snd\ x) \le (3::\mathbb{N}) - m_v\ (snd\ x)$
assume $c_v\ (snd\ x) = p_v\ (snd\ x)$
assume $\neg\ (3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) = p_v\ (snd\ x)$
assume *s4*: $\neg\ c_v\ s = p_v\ s$
assume *y4*: $\neg\ c_v\ y = p_v\ y$
assume *s5*: $c_v\ s \le (2::\mathbb{N})$
assume *y5*: $c_v\ y \le (2::\mathbb{N})$

have *psy*: $p_v\ s = p_v\ y$
using *s1 y1* **by** (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*2*))
have *msy*: $m_v\ s = m_v\ y$
using *s1 y1* **by** (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*2*))
have *csy*: $c_v\ s = c_v\ y$
using *psy msy s2 y2*
by (*metis One-nat-def s4 y4 s5 y5 add.commute add-le-mono add-right-cancel diff-diff-cancel le-Suc-eq numeral-2-eq-2 numeral-3-eq-3 plus-1-eq-Suc s3*)
show $s = y$
using *psy msy csy* **by** *simp*
next
have *pm-equal-snd-x*:
$\forall s$::*mh-state*. $snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!) \longrightarrow p_v\ s = p_v\ (snd\ x) \land m_v\ s = m_v\ (snd\ x)$
by (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*2*))
show $(p_v\ (snd\ x) \le (3::\mathbb{N}) - m_v\ (snd\ x) \longrightarrow (3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) \le (2::\mathbb{N}) \longrightarrow$
$p_v\ (snd\ x) \le (2::\mathbb{N}) \longrightarrow (3::\mathbb{N}) - (m_v\ (snd\ x) + p_v\ (snd\ x)) = p_v\ (snd\ x) \lor \neg\ c_v\ (snd\ x) = p_v\ (snd\ x)) \longrightarrow$
card $\{s$::*mh-state*. $\neg\ c_v\ s = p_v\ s \land p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}) \land m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \land$
$snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)\} = \ (0::\mathbb{N})$
apply (*auto*)
apply (*subgoal-tac* $\neg(\exists\, s$::*mh-state*. $\neg\ c_v\ s = p_v\ s \land p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}) \land$
$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \land snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)))$
using *card-0-singleton* **apply** *blast*
apply (*metis mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective mh-state.update-convs*(*2*) *Nat.le-diff-conv2 One-nat-def Suc-1 add.commute diff-le-mono2 diff-le-self le-SucI le-add2 numeral-3-eq-3*)
apply (*subgoal-tac* $\neg(\exists\, s$::*mh-state*. $\neg\ c_v\ s = p_v\ s \land p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}) \land$
$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \land snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)))$
using *card-0-singleton* **apply** *blast*
apply (*smt* (*verit*, *ccfv-SIG*) *add.assoc add.commute le-cases3 le-diff-conv le-trans pm-equal-snd-x*)
apply (*subgoal-tac* $\neg(\exists\, s$::*mh-state*. $\neg\ c_v\ s = p_v\ s \land p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}) \land$
$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \land snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)))$
using *card-0-singleton* **apply** *blast*
apply (*metis pm-equal-snd-x*)
apply (*subgoal-tac* $\neg(\exists\, s$::*mh-state*. $\neg\ c_v\ s = p_v\ s \land p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}) \land$

$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)))$
using *card-0-singleton* **apply** *blast*
apply (*smt* (*z3*) *ab-semigroup-add-class.add-ac*(*1*) *add.right-neutral diff-add-inverse2 diff-is-0-eq′ le-SucE le-add-diff nle-le numeral-3-eq-3 one-neq-zero plus-1-eq-Suc pm-equal-snd-x*)
apply (*subgoal-tac* $\neg(\exists\, s::mh\text{-}state.\ \neg\ c_v\ s = p_v\ s \wedge p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge$
$m_v\ s = (3::\mathbb{N}) - (c_v\ s + p_v\ s) \wedge snd\ x = s(\!|c_v := (3::\mathbb{N}) - (c_v\ s + m_v\ s)|\!)))$
using *card-0-singleton* **apply** *blast*
by (*auto*)
qed

show $(\sum_\infty s::mh\text{-}state.\ ?lhs\ s) = ?rhs$
apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
using *lhs-1-summable* **apply** *blast+*
apply (*subst summable-on-cdiv-left*)
using *lhs-2-summable* **apply** *blast+*
apply (*subst summable-on-cdiv-left*)
using *lhs-3-summable* **apply** *blast+*
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
using *lhs-1-summable* **apply** *blast+*
apply (*subst summable-on-cdiv-left*)
using *lhs-2-summable* **apply** *blast+*
apply (*subst infsum-cdiv-left*)
using *lhs-1-summable* **apply** *blast+*
apply (*subst infsum-cdiv-left*)
using *lhs-2-summable* **apply** *blast+*
apply (*subst infsum-cdiv-left*)
using *lhs-3-summable* **apply** *blast+*
using *lhs-1-infsum lhs-2-infsum lhs-3-infsum* **by** *presburger*
qed

lemma *IMHA-C-altdef-states-1-eq*:
$\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (p_v\ s)\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = (0::\mathbb{N}),\ m_v = (2::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 1::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective Nat.add-0-right Nat.add-diff-assoc One-nat-def Suc-1 Suc-le-mono add.commute add-2-eq-Suc′ add-cancel-left-left bot-nat-0.extremum diff-Suc-Suc diff-Suc-diff-eq2 diff-diff-left diff-is-0-eq diff-self-eq-0 eval-nat-numeral*(*3*) *le0 le-SucE le-antisym lessI less-2-cases mod-Suc mod-Suc-eq-mod-add3 mod-by-Suc-0 mod-less mod-mod-trivial mod-self nat.inject not-mod2-eq-Suc-0-eq-0 numeral-1-eq-Suc-0 numeral-3-eq-3 numeral-plus-numeral old.unit.exhaust order-le-less plus-1-eq-Suc*)
by (*auto*)

lemma *IMHA-C-altdef-states-2-eq*:
$\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (Suc\ (p_v\ s))\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = (2::\mathbb{N}),\ m_v = (0::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = 1::\mathbb{N}|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)

apply (*rule iffI*)
apply (*smt* (*verit*, *best*) *mh-state.surjective Nat.add-0-right Nat.add-diff-assoc One-nat-def Suc-1 Suc-le-mono add.commute add-2-eq-Suc′ add-cancel-left-left bot-nat-0.extremum diff-Suc-Suc diff-Suc-diff-eq2 diff-diff-left diff-is-0-eq diff-self-eq-0 eval-nat-numeral*(*3*) *le0 le-SucE le-antisym lessI less-2-cases mod-Suc mod-Suc-eq-mod-add3 mod-by-Suc-0 mod-less mod-mod-trivial mod-self nat.inject not-mod2-eq-Suc-0-eq-0 numeral-1-eq-Suc-0 numeral-3-eq-3 numeral-plus-numeral old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

lemma *IMHA-C-altdef-states-3-eq*:
$\{s{::}mh\text{-}state.\ (((\neg\ (3{::}\mathbb{N}) - (m_v\ s + p_v\ s) = p_v\ s \wedge p_v\ s \le (2{::}\mathbb{N})) \wedge (3{::}\mathbb{N}) - (m_v\ s + p_v\ s) \le (2{::}\mathbb{N})) \wedge p_v\ s \le (3{::}\mathbb{N}) - m_v\ s) \wedge c_v\ s = p_v\ s\}$
$= \{(\!|p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N})|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = (2{::}\mathbb{N})|\!),$
$(\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = (0{::}\mathbb{N})|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = (2{::}\mathbb{N})|\!),$
$(\!|p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = 0{::}\mathbb{N}|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N})|\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*smt* (*verit*, *ccfv-SIG*) *mh-state.ext-inject mh-state.select-convs*(*1*) *mh-state.select-convs*(*2*) *mh-state.select-convs*(*3*) *mh-state.surjective Nat.add-0-right One-nat-def Suc-1 Suc-eq-numeral bot-nat-0.extremum diff-add-inverse diff-commute diff-diff-cancel diff-diff-left diff-is-0-eq diff-is-0-eq′ diff-le-self diff-self-eq-0 eval-nat-numeral*(*3*) *le-Suc-eq le-antisym less-2-cases nat.distinct*(*1*) *nle-le not-less-eq-eq old.nat.exhaust old.unit.exhaust order-le-less plus-1-eq-Suc*)
by *force*

lemma *IMHA-C-win*: $rvfun\text{-}of\text{-}prfun\ (IMHA\text{-}C)\ ;\ [\![c^< = p^<]\!]_{\mathcal{I}e} = (2/3)_e$
proof −
let $?lhs\text{-}1 = \lambda(s_1{::}mh\text{-}state)\ s{::}mh\text{-}state.$
$(if\ get_p\ (get_{snd_L}\ (s_1,\ s)) \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_c\ (get_{snd_L}\ (s_1,\ s)) = (3{::}\mathbb{N}) - (get_p\ (get_{snd_L}\ (s_1,\ s)) + get_m\ (get_{snd_L}\ (s_1,\ s)))\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_m\ (get_{snd_L}\ (s_1,\ s)) = Suc\ (get_p\ (get_{snd_L}\ (s_1,\ s)))\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}2 = \lambda(s_1{::}mh\text{-}state)\ s{::}mh\text{-}state.$
$(if\ get_p\ (get_{snd_L}\ (s_1,\ s)) \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_c\ (get_{snd_L}\ (s_1,\ s)) = (3{::}\mathbb{N}) - (get_p\ (get_{snd_L}\ (s_1,\ s)) + get_m\ (get_{snd_L}\ (s_1,\ s)))\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_m\ (get_{snd_L}\ (s_1,\ s)) = Suc\ (Suc\ (get_p\ (get_{snd_L}\ (s_1,\ s))))\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}3 = \lambda(s_1{::}mh\text{-}state)\ s{::}mh\text{-}state.$
$(if\ \neg\ (3{::}\mathbb{N}) - (get_m\ (get_{snd_L}\ (s_1,\ s)) + get_p\ (get_{snd_L}\ (s_1,\ s))) = get_p\ (get_{snd_L}\ (s_1,\ s))\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_p\ (get_{snd_L}\ (s_1,\ s)) \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ (3{::}\mathbb{N}) - (get_m\ (get_{snd_L}\ (s_1,\ s)) + get_p\ (get_{snd_L}\ (s_1,\ s))) \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_p\ (get_{snd_L}\ (s_1,\ s)) \le (3{::}\mathbb{N}) - get_m\ (get_{snd_L}\ (s_1,\ s))\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ get_c\ (get_{snd_L}\ (s_1,\ s)) = get_p\ (get_{snd_L}\ (s_1,\ s))\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs = \lambda(s_1{::}mh\text{-}state)\ s{::}mh\text{-}state.\ ?lhs\text{-}1\ s_1\ s\ /\ 18 + ?lhs\text{-}2\ s_1\ s\ /\ 18 + ?lhs\text{-}3\ s_1\ s\ /\ 9$

let $?lhs\text{-}1' = \lambda s{::}mh\text{-}state.$
$(if\ p_v\ s \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$

$(if\ c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (p_v\ s)\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let $?lhs\text{-}2' = \lambda s::mh\text{-}state.\ (if\ p_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (Suc\ (p_v\ s))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let $?lhs\text{-}3' = \lambda s::mh\text{-}state.\ (if\ \neg\ (3::\mathbb{N}) - (m_v\ s + p_v\ s) = p_v\ s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ p_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ (3::\mathbb{N}) - (m_v\ s + p_v\ s) \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ p_v\ s \le (3::\mathbb{N}) - m_v\ s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ c_v\ s = p_v\ s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
let $?lhs' = \lambda s::mh\text{-}state.\ ?lhs\text{-}1'\ s\ /\ 18 + ?lhs\text{-}2'\ s\ /\ 18 + ?lhs\text{-}3'\ s\ /\ 9$

have *lhs-1-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\text{-}1\ s_1\ s = ?lhs\text{-}1'\ s$
by (*expr-simp*)

have *lhs-2-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\text{-}2\ s_1\ s = ?lhs\text{-}2'\ s$
by (*expr-simp*)

have *lhs-3-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\text{-}3\ s_1\ s = ?lhs\text{-}3'\ s$
by (*expr-simp-1*)

have *lhs-lhs'-eq*: $\forall (s_1::mh\text{-}state)\ s::mh\text{-}state.\ ?lhs\ s_1\ s = ?lhs'\ s$
by (*simp add*: *c-def m-def p-def*)

have *infsum-lhs-1*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}1'\ s) = 3$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *IMHA-C-altdef-states-1-eq* **apply** *auto*[*1*]
using *IMHA-C-altdef-states-1-eq* **by** *force*

have *infsum-lhs-2*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}2'\ s) = 3$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *IMHA-C-altdef-states-2-eq* **apply** *auto*[*1*]
using *IMHA-C-altdef-states-2-eq* **by** *force*

have *infsum-lhs-3*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}3'\ s) = 6$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
using *IMHA-C-altdef-states-3-eq* **apply** *auto*[*1*]
using *IMHA-C-altdef-states-3-eq* **by** *force*

have *lhs-1-summable*: $?lhs\text{-}1'$ *summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-C-altdef-states-1-eq* **by** (*simp-all*)

let $?lhs\text{-}cp = \lambda s.\ (if\ c_v\ s = p_v\ s\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$

have *lhs-1'-summable*: $(\lambda s.\ ?lhs\text{-}1'\ s * ?lhs\text{-}cp\ s)$ *summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule finite-subset*[**where** $B = \{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge$
$c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (p_v\ s)\ mod\ (3::\mathbb{N}))\}$])
apply *force*

using *IMHA-C-altdef-states-1-eq* **by** (*simp-all*)

have *lhs-1′-infsum*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}1'\ s * ?lhs\text{-}cp\ s) = 0$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *Collect-mono finite.emptyI finite-insert finite-subset IMHA-C-altdef-states-1-eq*)
apply (*subgoal-tac* $\neg(\exists\, s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (p_v\ s)\ mod\ (3::\mathbb{N})) \wedge c_v\ s = p_v\ s)$)
apply (*simp add*: *card-0-singleton*)
by (*metis* (*no-types*, *lifting*) *add-cancel-left-right add-diff-cancel-left add-diff-cancel-left′ diff-is-0-eq le-SucE lessI mod-less mod-less-eq-dividend mod-self nat.distinct*(*1*) *numeral-2-eq-2 numeral-3-eq-3 plus-1-eq-Suc*)

have *lhs-2-summable*: *?lhs-2′ summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-C-altdef-states-2-eq* **by** (*simp-all*)

have *lhs-2′-summable*: $(\lambda s.\ ?lhs\text{-}2'\ s * ?lhs\text{-}cp\ s)$ *summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule finite-subset*[**where** $B=\{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (Suc\ (p_v\ s))\ mod\ (3::\mathbb{N}))\}$])
apply *force*
using *IMHA-C-altdef-states-2-eq* **by** (*simp-all*)

have *lhs-2′-infsum*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}2'\ s * ?lhs\text{-}cp\ s) = 0$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *Collect-mono finite.emptyI finite-insert finite-subset IMHA-C-altdef-states-2-eq*)
apply (*subgoal-tac* $\neg(\exists\, s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s = (3::\mathbb{N}) - (p_v\ s + m_v\ s)) \wedge m_v\ s = Suc\ (Suc\ (p_v\ s))\ mod\ (3::\mathbb{N})) \wedge c_v\ s = p_v\ s)$)
apply (*simp add*: *card-0-singleton*)
by (*smt* (*z3*) *Suc-diff-le Suc-n-not-le-n diff-add-zero diff-le-self le-SucE le-add-diff-inverse2 mod-less mod-self numeral-2-eq-2 numeral-3-eq-3 order-le-less zero-less-diff*)

have *lhs-3-summable*: *?lhs-3′ summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-C-altdef-states-3-eq* **by** (*simp-all*)

have *lhs-3′-summable*: $(\lambda s.\ ?lhs\text{-}3'\ s * ?lhs\text{-}cp\ s)$ *summable-on UNIV*
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-constant-finite-states-summable*)
using *IMHA-C-altdef-states-3-eq* **by** (*simp-all*)

have *lhs-3′-infsum*: $(\sum_\infty s::mh\text{-}state.\ ?lhs\text{-}3'\ s * ?lhs\text{-}cp\ s) = 6$
apply (*subst infsum-lhs-3*[*symmetric*])
by (*smt* (*verit*) *infsum-cong mult-cancel-left2 mult-cancel-right*)

have *infsum-lhs-lhs′-eq*: $\forall\, s_1::mh\text{-}state.\ (\sum_\infty s::mh\text{-}state.\ ?lhs\ s_1\ s) = (\sum_\infty s::mh\text{-}state.\ ?lhs'\ s)$
apply (*rule allI*)
by (*metis* (*full-types*) *lhs-lhs′-eq*)

have *infsum-lhs′-1*: $(\sum_\infty s::mh\text{-}state.\ ?lhs'\ s) = 1$

apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-3-summable*)
using *infsum-lhs-1 infsum-lhs-2 infsum-lhs-3* **by** (*simp*)

have *infsum-lhs-1*: $\forall\, s_1$::*mh-state*. ($\sum_\infty s$::*mh-state*. *?lhs* s_1 s) = *1*
using *infsum-lhs′-1 infsum-lhs-lhs′-eq* **by** *presburger*

have *lhs′-leq-1*: $\forall\, s$::*mh-state*. *?lhs′* $s \leq$ *infsum ?lhs′ UNIV*
apply (*rule allI*)
apply (*rule infsum-geq-element*)
apply *fastforce*
apply (*subst summable-on-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2-summable*)
apply (*subst summable-on-cdiv-left*)
by (*simp-all add*: *lhs-3-summable*)
have *lhs′-leq-1′*: $\forall\, s$::*mh-state*. *?lhs′* $s \leq$ *1*
using *infsum-lhs′-1 lhs′-leq-1* **by** *presburger*
have *lhs-leq-1*: $\forall\, s_1$::*mh-state*. ($\forall\, s$::*mh-state*. *?lhs* s_1 $s \leq$ *1*)
by (*simp add*: *c-def lhs′-leq-1′ m-def p-def*)

have *IMHA-C-altdef-dist*: *is-final-distribution IMHA-C-altdef*
apply (*simp add*: *IMHA-C-altdef-def*)
apply (*simp add*: *dist-defs*)
apply (*simp only*: *expr-defs*)
apply (*rule allI*)
apply (*rule conjI*)
apply (*rule allI*)
apply (*rule conjI*)
using *add-divide-distrib div-by-1 divide-divide-eq-right divide-le-0-1-iff mult-not-zero* **apply** *auto*[*1*]
using *lhs-leq-1* **apply** *blast*
using *infsum-lhs-1* **by** *blast*

show *?thesis*
apply (*simp add*: *IMHA-C-altdef*)

apply (*subst rvfun-inverse*)
using *IMHA-C-altdef-dist* **apply** (*simp add*: *is-dist-def is-final-prob-prob*)
apply (*simp add*: *IMHA-C-altdef-def*)
apply (*expr-auto*)
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst infsum-add*)
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1′-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2′-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-3′-summable*)
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-1′-summable*)
apply (*subst summable-on-cdiv-left*)
apply (*simp-all add*: *lhs-2′-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-1′-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-2′-summable*)
apply (*subst infsum-cdiv-left*)
apply (*simp-all add*: *lhs-3′-summable*)
using *lhs-1′-infsum lhs-2′-infsum lhs-3′-infsum* **by** *linarith*
qed

2.5.1 Average values

Average value of p after the execution of *IMHA-C*, a Change Strategy.

term $(p^<)_e$
term $(\$p^<)_e$
term *rvfun-of-prfun IMHA-C* ; $(\$p^<)_e$
lemma *IMHA-C-average-p*: *rvfun-of-prfun IMHA-C* ; $(\$p^<)_e = (1)_e$
apply (*simp add*: *IMHA-C-altdef*)
apply (*subst rvfun-inverse*)
using *IMHA-C-altdef-dist* **apply** (*simp add*: *is-final-distribution-prob is-final-prob-prob*)
apply (*simp add*: *IMHA-C-altdef-def*)
apply (*expr-auto*)
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst conditional-conds-conj*)+
apply (*subst times-divide-eq-right*[*symmetric*])+
apply (*subst conditional-cmult-1*)+
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply *blast*+
apply (*subst infsum-add*)

apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply *blast*+
apply (*subst IMHA-C-altdef-states-1-eq*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply (*subst sum-divide-distrib*[*symmetric*])+
by (*simp*)

lemma *IMHA-C-average-c*: *rvfun-of-prfun IMHA-C* ; $(\$c^{<})_e = (\mathit{1})_e$
apply (*simp add*: *IMHA-C-altdef*)
apply (*subst rvfun-inverse*)
using *IMHA-C-altdef-dist* **apply** (*simp add*: *is-final-distribution-prob is-final-prob-prob*)
apply (*simp add*: *IMHA-C-altdef-def*)
apply (*expr-auto*)
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst conditional-conds-conj*)+
apply (*subst times-divide-eq-right*[*symmetric*])+
apply (*subst conditional-cmult-1*)+
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply *blast*+
apply (*subst infsum-add*)
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states-summable*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-1-eq*)
apply *blast*+
apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply *blast*+

apply (*subst infsum-cond-finite-states*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply *blast+*
apply (*subst IMHA-C-altdef-states-1-eq*)
apply (*subst IMHA-C-altdef-states-2-eq*)
apply (*subst IMHA-C-altdef-states-3-eq*)
apply (*subst sum-divide-distrib*[*symmetric*])+
by (*simp*)

2.6 Learn the fact (forgetful Monty)

Suppose now that Monty forgets which door has the prize behind it. He just opens either of the doors not chosen by the contestant. If the prize is revealed ($m' = p'$), then obviously the contestant switches their choice to that door. So the contestant will surely win.
However, if the prize is not revealed ($m' \neq p'$), should the contestant switch?

definition *Forgetful-Monty* **where**
$Forgetful\text{-}Monty = INIT\ ;\ (if_p\ 1/2\ then\ (m := (\$c + 1)\ mod\ 3)\ else\ (m := (\$c + 2)\ mod\ 3))$

definition *Learn-fact* :: (*mh-state*, *mh-state*) *prfun*
where $Learn\text{-}fact = prfun\text{-}of\text{-}rvfun\ ((rvfun\text{-}of\text{-}prfun\ Forgetful\text{-}Monty)\ \|_f\ [\![m^> \neq p^>]\!]_{\mathcal{I}e})$

definition *Forgetful-Monty'* :: (*mh-state*, *mh-state*) *rvfun* **where**
$Forgetful\text{-}Monty' = (([\![p^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![c^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![m^> = ((c^> + 1)\ mod\ 3)]\!]_{\mathcal{I}e})\ /\ 18\ +$
$([\![p^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![c^> \in \{0..2\}]\!]_{\mathcal{I}e} * [\![m^> = ((c^> + 2)\ mod\ 3)]\!]_{\mathcal{I}e})\ /\ 18)_e$

lemma *Forgetful-Monty-altdef*: $Forgetful\text{-}Monty = prfun\text{-}of\text{-}rvfun\ Forgetful\text{-}Monty'$
proof −

have *set-states*: $\forall m.\ \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N})) \wedge m_v\ s = m\}$
$= \{(\!|p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = m|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = m|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = m|\!),$
$(\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = 0{::}\mathbb{N},\ m_v = m|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = m|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = 2{::}\mathbb{N},\ m_v = m|\!),$
$(\!|p_v = 2{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = m|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = m|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = m|\!)$
$\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*z3*) *mh-state.surjective mh-state.update-convs*(*1*) *mh-state.update-convs*(*2*) *One-nat-def Suc-1 bot-nat-0.extremum-unique c-def le-Suc-eq lens.simps*(*1*) *m-def old.unit.exhaust p-def*)
by (*smt* (*verit*, *best*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*1*)
mh-state.update-convs(*2*) *One-nat-def bot-nat-0.extremum c-def lens.simps*(*1*) *less-one*
linorder-not-le m-def order-le-less p-def zero-neq-numeral)

have *card-states*: $\forall mm.\ card\ \{(\!|p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = mm|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = mm|\!),\ (\!|p_v = 0{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = mm|\!),$
$(\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = 0{::}\mathbb{N},\ m_v = mm|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = mm|\!),\ (\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = 2{::}\mathbb{N},\ m_v = mm|\!),$
$(\!|p_v = 2{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = mm|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = mm|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = mm|\!)$

} = *9*
apply (*rule allI*)
using *record-neq-p-c* **by** *fastforce*

have *finite-states*: $\forall m.$ *finite* {*s*::*mh-state*. (p_v *s* ≤ (*2*::ℕ) ∧ c_v *s* ≤ (*2*::ℕ)) ∧ m_v *s* = *m*}
using *local.set-states* **by** *auto*

have *summable-on*: ∀ (m_v′::ℕ) (p_v′::ℕ) c_v′::ℕ. (λv_0::*mh-state*.
(*if* p_v v_0 ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗ (*if* c_v v_0 ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* m_v v_0 = m_v′ *then* *1*::ℝ *else* (*0*::ℝ)) ∗
((*if* ⦇p_v = p_v′, c_v = c_v′, m_v = *Suc* c_v′ *mod* (*3*::ℕ)⦈ = v_0⦇m_v := *Suc* (c_v v_0) *mod* (*3*::ℕ)⦈ *then* *1*::ℝ
else (*0*::ℝ)) / (*2*::ℝ) +
(*if* ⦇p_v = p_v′, c_v = c_v′, m_v = *Suc* c_v′ *mod* (*3*::ℕ)⦈ = v_0⦇m_v := *Suc* (*Suc* (c_v v_0)) *mod* (*3*::ℕ)⦈ *then* *1*::ℝ
else (*0*::ℝ)) / (*2*::ℝ))) *summable-on UNIV*
proof (*rule allI*)+
fix m_v′::ℕ **and** p_v′::ℕ **and** c_v′::ℕ
show (λv_0::*mh-state*.
(*if* p_v v_0 ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗ (*if* c_v v_0 ≤ (*2*::ℕ) *then* *1*::ℝ *else* (*0*::ℝ)) ∗
(*if* m_v v_0 = m_v′ *then* *1*::ℝ *else* (*0*::ℝ)) ∗
((*if* ⦇p_v = p_v′, c_v = c_v′, m_v = *Suc* c_v′ *mod* (*3*::ℕ)⦈ = v_0⦇m_v := *Suc* (c_v v_0) *mod* (*3*::ℕ)⦈ *then* *1*::ℝ *else* (*0*::ℝ)) / (*2*::ℝ) +
(*if* ⦇p_v = p_v′, c_v = c_v′, m_v = *Suc* c_v′ *mod* (*3*::ℕ)⦈ = v_0⦇m_v := *Suc* (*Suc* (c_v v_0)) *mod* (*3*::ℕ)⦈ *then* *1*::ℝ *else* (*0*::ℝ)) /
(*2*::ℝ))) *summable-on*
UNIV
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst conditional-conds-conj*)+
apply (*subst summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** *B* = {*s*::*mh-state*. (p_v *s* ≤ (*2*::ℕ) ∧ c_v *s* ≤ (*2*::ℕ) ∧ m_v *s* = m_v′)}])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** *B* = {*s*::*mh-state*. (p_v *s* ≤ (*2*::ℕ) ∧ c_v *s* ≤ (*2*::ℕ) ∧ m_v *s* = m_v′)}])
using *finite-states* **apply** *presburger*
by *fastforce*+
qed

show *?thesis*
apply (*simp add*: *Forgetful-Monty-def Forgetful-Monty′-def*)
apply (*simp add*: *INIT-altdef*)
apply (*simp only*: *pseqcomp-def passigns-def pchoice-def*)
apply (*simp only*: *rvfun-assignment-inverse*)
apply (*simp only*: *ereal2real-1-2*)
apply (*subst rvfun-pchoice-inverse-c″*)
apply (*simp*)
using *rvfun-assignment-is-prob* **apply** *blast*
using *rvfun-assignment-is-prob* **apply** *blast*

apply (*simp*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *is-prob-def iverson-bracket-def*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*pred-auto*)
apply (*subst infsum-cdiv-left*)
using *summable-on* **apply** *blast*
using *mod-Suc* **apply** *force*
using *mod-Suc* **apply** *force*
using *mod-Suc* **apply** *force*
proof −
fix $m_v{}'$::ℕ **and** $p_v{}'$::ℕ **and** $c_v{}'$::ℕ
assume *a1*: $p_v{}' \leq (2::\mathbb{N})$
assume *a2*: $c_v{}' \leq (2::\mathbb{N})$
have *set-1-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
$(\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = Suc\ c_v{}'\ mod\ (3::\mathbb{N})|\!) = s(\!|m_v := Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})|\!)\}$
$= \{(\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}'|\!)\}$
apply (*auto*)
apply (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*))
by (*simp add*: *a1 a2*)+

have *set-2-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
$(\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = Suc\ c_v{}'\ mod\ (3::\mathbb{N})|\!) = s(\!|m_v := Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})|\!)\} =$
$\{\}$
apply (*auto*)
by (*smt* (*verit*, *best*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*)
lessI less-2-cases mod-Suc-eq mod-less mod-self nat.simps(*3*) *numeral-2-eq-2 numeral-3-eq-3*
order-le-less)

show $(\sum_\infty v_0::mh\text{-}state.$
$(if\ p_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = m_v{}'\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((if\ (\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = Suc\ c_v{}'\ mod\ (3::\mathbb{N})|\!) = v_0(\!|m_v := Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})|\!)$
$then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (2::\mathbb{R})\ +$
$(if\ (\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = Suc\ c_v{}'\ mod\ (3::\mathbb{N})|\!) = v_0(\!|m_v := Suc\ (Suc\ (c_v\ v_0))\ mod$
$(3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /$
$(2::\mathbb{R}))\ /\ (9::\mathbb{R})) * (18::\mathbb{R}) = (1::\mathbb{R})$
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s$
$= m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s$
$= m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-add*)

apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst set-1-eq*, *subst set-2-eq*)
by *simp*
next
fix $m_v'::\mathbb{N}$ **and** $p_v'::\mathbb{N}$ **and** $c_v'::\mathbb{N}$
assume *a1*: $p_v' \leq (2::\mathbb{N})$
assume *a2*: $c_v' \leq (2::\mathbb{N})$
have *set-1-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v') \wedge$
$(\!|p_v = p_v',\ c_v = c_v',\ m_v = Suc\ (Suc\ c_v')\ mod\ (3::\mathbb{N})|\!) = s(\!|m_v := Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})|\!)\}$
$= \{\}$
apply (*auto*)
by (*smt* (*verit*, *best*) *mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*)
lessI less-2-cases mod-Suc-eq mod-less mod-self nat.simps(*3*) *numeral-2-eq-2 numeral-3-eq-3*
order-le-less)

have *set-2-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v') \wedge$
$(\!|p_v = p_v',\ c_v = c_v',\ m_v = Suc\ (Suc\ c_v')\ mod\ (3::\mathbb{N})|\!) = s(\!|m_v := Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})|\!)\}$

$= \{(\!|p_v = p_v',\ c_v = c_v',\ m_v = m_v'|\!)\}$
apply (*auto*)
apply (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*))
by (*simp add*: *a1 a2*)+

show $(\sum_\infty v_0::mh\text{-}state.$
$(if\ p_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = m_v'\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((if\ (\!|p_v = p_v',\ c_v = c_v',\ m_v = Suc\ (Suc\ c_v')\ mod\ (3::\mathbb{N})|\!) = v_0(\!|m_v := Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /$

$(2::\mathbb{R})$ +
(*if* $(\!| p_v = p_v{}', c_v = c_v{}', m_v = Suc\ (Suc\ c_v{}')\ mod\ (3::\mathbb{N}) |\!) = v_0(\!| m_v := Suc\ (Suc\ (c_v\ v_0))\ mod\ (3::\mathbb{N}) |\!)$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R}))$ /
$(2::\mathbb{R}))$ / $(9::\mathbb{R})) * (18::\mathbb{R}) = (1::\mathbb{R})$
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst set-1-eq*, *subst set-2-eq*)
by *simp*
next
fix $m_v{}'::\mathbb{N}$ **and** $p_v{}'::\mathbb{N}$ **and** $c_v{}'::\mathbb{N}$ **and** $m_v{}''::\mathbb{N}$
assume *a1*: $p_v{}' \le (2::\mathbb{N})$

assume $a2$: $c_v{}' \leq (2{::}\mathbb{N})$
assume $a3$: $\neg\ m_v{}'' = Suc\ c_v{}'\ mod\ (3{::}\mathbb{N})$
assume $a4$: $\neg\ m_v{}'' = Suc\ (Suc\ c_v{}')\ mod\ (3{::}\mathbb{N})$
have *set-1-eq*: $\{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
$(\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}''|\!) = s(\!|m_v := Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N})|\!)\} = \{\}$
apply (*auto*)
by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a3*)

have *set-2-eq*: $\{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
$(\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}''|\!) = s(\!|m_v := Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N})|\!)\} = \{\}$
apply (*auto*)
by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a4*)

show $(\sum_\infty v_0{::}mh\text{-}state.$
$(if\ p_v\ v_0 \leq (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ c_v\ v_0 \leq (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ v_0 = m_v{}'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$((if\ (\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}''|\!) = v_0(\!|m_v := Suc\ (c_v\ v_0)\ mod\ (3{::}\mathbb{N})|\!)\ then\ 1{::}\mathbb{R}\ else$
$(0{::}\mathbb{R})) / (2{::}\mathbb{R}) +$
$(if\ (\!|p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}''|\!) = v_0(\!|m_v := Suc\ (Suc\ (c_v\ v_0))\ mod\ (3{::}\mathbb{N})|\!)\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R})) / (2{::}\mathbb{R})) /$
$(9{::}\mathbb{R})) =$
$(0{::}\mathbb{R})$
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s =$
$m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s =$
$m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s =$
$m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s =$
$m_v{}')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \leq (2{::}\mathbb{N}) \wedge c_v\ s \leq (2{::}\mathbb{N}) \wedge m_v\ s =$

$m_v{}'$)}])
  using *finite-states* **apply** *presburger*
  apply *fastforce*+
  apply (*subst infsum-cdiv-left*)
  apply (*subst infsum-constant-finite-states-summable*)
  apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
  using *finite-states* **apply** *presburger*
  apply *fastforce*+
  apply (*subst infsum-constant-finite-states*)
  apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
  apply (*subst infsum-constant-finite-states*)
  apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
  apply (*subst set-1-eq*, *subst set-2-eq*)
  by *simp*
next
 fix $m_v{}'{::}\mathbb{N}$ **and** $p_v{}'{::}\mathbb{N}$ **and** $c_v{}'{::}\mathbb{N}$ **and** $m_v{}''{::}\mathbb{N}$
 assume *a1*: $p_v{}' \le (2{::}\mathbb{N})$
 assume *a2*: $\neg\ c_v{}' \le (2{::}\mathbb{N})$
 have *set-1-eq*: $\{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
   $(\!| p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}'' |\!) = s(\!| m_v := Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N}) |\!)\} = \{\}$
  apply (*auto*)
  by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a2*)

 have *set-2-eq*: $\{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}') \wedge$
   $(\!| p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}'' |\!) = s(\!| m_v := Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}) |\!)\} = \{\}$
  apply (*auto*)
  by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a2*)

 show $(\sum_\infty v_0{::}mh\text{-}state.$
   $(if\ p_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ c_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
   $(if\ m_v\ v_0 = m_v{}'\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
   $((if\ (\!| p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}'' |\!) = v_0(\!| m_v := Suc\ (c_v\ v_0)\ mod\ (3{::}\mathbb{N}) |\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) / (2{::}\mathbb{R}) +$
   $(if\ (\!| p_v = p_v{}',\ c_v = c_v{}',\ m_v = m_v{}'' |\!) = v_0(\!| m_v := Suc\ (Suc\ (c_v\ v_0))\ mod\ (3{::}\mathbb{N}) |\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) / (2{::}\mathbb{R})) /$
   $(9{::}\mathbb{R})) = (0{::}\mathbb{R})$
  apply (*subst conditional-conds-conj*)+
  apply (*simp add*: *ring-distribs*(*1*))
  apply (*subst conditional-conds-conj*)+
  apply (*subst infsum-cdiv-left*)
  apply (*rule summable-on-add*)
  apply (*subst summable-on-cdiv-left*)
  apply (*subst infsum-constant-finite-states-summable*)
  apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
  using *finite-states* **apply** *presburger*
  apply *fastforce*+
  apply (*subst summable-on-cdiv-left*)
  apply (*subst infsum-constant-finite-states-summable*)
  apply (*rule rev-finite-subset*[**where** $B = \{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = m_v{}')\}$])
  using *finite-states* **apply** *presburger*
  apply *fastforce*+
  apply (*subst infsum-add*)

apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst set-1-eq*, *subst set-2-eq*)
by *simp*
next
fix $m_v'::\mathbb{N}$ **and** $p_v'::\mathbb{N}$ **and** $c_v'::\mathbb{N}$ **and** $m_v''::\mathbb{N}$
assume *a1*: $\neg\ p_v' \leq (2::\mathbb{N})$
have *set-1-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v') \wedge$
$(\!|p_v = p_v',\ c_v = c_v',\ m_v = m_v''|\!) = s(\!|m_v := Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})|\!)\} = \{\}$
apply (*auto*)
by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a1*)

have *set-2-eq*: $\{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v') \wedge$
$(\!|p_v = p_v',\ c_v = c_v',\ m_v = m_v''|\!) = s(\!|m_v := Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})|\!)\} = \{\}$
apply (*auto*)
by (*metis mh-state.ext-inject mh-state.surjective mh-state.update-convs*(*3*) *a1*)

show $(\sum_\infty v_0::mh\text{-}state.$
$(if\ p_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ v_0 \leq (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = m_v'\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((if\ (\!|p_v = p_v',\ c_v = c_v',\ m_v = m_v''|\!) = v_0(\!|m_v := Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (2::\mathbb{R}) +$
$(if\ (\!|p_v = p_v',\ c_v = c_v',\ m_v = m_v''|\!) = v_0(\!|m_v := Suc\ (Suc\ (c_v\ v_0))\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (2::\mathbb{R}))\ /$
$(9::\mathbb{R})) =$
$(0::\mathbb{R})$
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst conditional-conds-conj*)+

apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
apply (*rule rev-finite-subset*[**where** $B = \{s::mh\text{-}state.\ (p_v\ s \leq (2::\mathbb{N}) \wedge c_v\ s \leq (2::\mathbb{N}) \wedge m_v\ s = m_v')\}$])
using *finite-states* **apply** *presburger*
apply *fastforce+*
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*no-types*, *lifting*) *Collect-mono finite-states finite-subset*)
apply (*subst set-1-eq*, *subst set-2-eq*)
by *simp*
qed
qed

definition *Forgetful-Monty'-learned* :: (*mh-state*, *mh-state*) *rvfun* **where**
$Forgetful\text{-}Monty'\text{-}learned = ((\llbracket p^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket m^> = ((c^> + 1)\ mod\ 3)\rrbracket_{\mathcal{I}e} * \llbracket m^> \neq p^>\rrbracket_{\mathcal{I}e})\ /\ 12\ +$
$(\llbracket p^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket c^> \in \{0..2\}\rrbracket_{\mathcal{I}e} * \llbracket m^> = ((c^> + 2)\ mod\ 3)\rrbracket_{\mathcal{I}e} * \llbracket m^> \neq p^>\rrbracket_{\mathcal{I}e})\ /\ 12)_e$

lemma *Forgetful-Monty-win*: $rvfun\text{-}of\text{-}prfun\ Learn\text{-}fact\ ;\ \llbracket c^< = p^<\rrbracket_{\mathcal{I}e} = (1/2)_e$
proof −

— Forgetful Monty

have *set-states-1*: $\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!),$
$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)$
$\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective*
One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral(*3*)
le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less
mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct(*1*) *nle-le not-less-eq-eq*
numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral
pred-numeral-simps(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-1*: *card* $\{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!),$
$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)$
$\} = 9$
using *record-neq-p-c* **by** *fastforce*

have *finite-states-1*: *finite* $\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\}$
using *local.set-states-1* **by** *auto*

have *set-states-2*: $\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),$
$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!),\ (\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),$
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!),\ (\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!)$
$\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective*
One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral(*3*)
le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less
mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct(*1*) *nle-le not-less-eq-eq*
numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral
pred-numeral-simps(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-2*: *card* $\{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!),\ (\!|p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!),$

$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!)$, $(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!)$, $(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!)$,
$(\!|p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N})|\!)$, $(\!|p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N}|\!)$, $(\!|p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!)$
$\} = 9$
using *record-neq-p-c* **by** *fastforce*

have *finite-states-2*: *finite* $\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N})) \land m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\}$
using *local.set-states-2* **by** *auto*

have *infsum-1*: $(\sum_\infty s::mh\text{-}state.$
$(if\ p_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (18::\mathbb{R}) +$
$(if\ p_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ s \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (18::\mathbb{R})) = (1::\mathbb{R})$
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-states-1* **apply** *blast*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-states-2* **apply** *blast*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-states-1* **apply** *blast*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-states-2* **apply** *blast*+
apply (*subst infsum-constant-finite-states*)
using *finite-states-1* **apply** *blast*+
apply (*subst infsum-constant-finite-states*)
using *finite-states-2* **apply** *blast*+
apply (*subst set-states-1*, *subst card-states-1*)
apply (*subst set-states-2*, *subst card-states-2*)
by (*simp*)

— The final statesuf of Forgetful Monty is a distribution
have *Forgetful-Monty′-dist*: *is-final-distribution* (*Forgetful-Monty′*)
apply (*simp add*: *dist-defs Forgetful-Monty′-def*)
apply (*expr-auto*)
using *infsum-1* **by** *blast*

— And so conversion is still itself.
have *Forgetful-Monty′′*: *rvfun-of-prfun* (*prfun-of-rvfun Forgetful-Monty′*) = *Forgetful-Monty′*
apply (*subst rvfun-inverse*)
apply (*simp add*: *Forgetful-Monty′-dist is-final-distribution-prob is-final-prob-prob*)
by (*simp add*: *Forgetful-Monty′-dist*)+

— Learn a new fact
have *set-states-1′*: $\{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \land c_v\ s \le (2::\mathbb{N}))$
$\land m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})) \land \neg\ m_v\ s = p_v\ s\}$
$= \{(\!|p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N})|\!)$, $(\!|p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!)$,
$(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N}|\!)$, $(\!|p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N}|\!)$,

$(\!| p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)$, $(\!| p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N} |\!)$
$\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective*
One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral(*3*)
le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less
mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct(*1*) *nle-le not-less-eq-eq*
numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral
pred-numeral-simps(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-1'*: *card* $\{(\!| p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!), (\!| p_v = 0::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),$
$m_v = 2::\mathbb{N} |\!)$,
$(\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 2::\mathbb{N} |\!), (\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N} |\!)$,
$(\!| p_v = 2::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)$, $(\!| p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = 0::\mathbb{N} |\!)$
$\} = 6$
using *record-neq-p-c* **by** *fastforce*

have *finite-state-1'*: *finite* $\{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s\}$
apply (*rule rev-finite-subset*[**where** $B =$
$\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3::\mathbb{N}))\}$])
using *finite-states-1* **apply** *presburger*
by *fastforce*

have *set-states-2'*: $\{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s\}$
$= \{(\!| p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N}) |\!), (\!| p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)$,
$(\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N}) |\!), (\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N} |\!)$,
$(\!| p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N} |\!), (\!| p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)$
$\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective*
One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral(*3*)
le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less
mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct(*1*) *nle-le not-less-eq-eq*
numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral
pred-numeral-simps(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-2'*: *card* $\{(\!| p_v = 0::\mathbb{N},\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N}) |\!), (\!| p_v = 0::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc$
$(0::\mathbb{N}) |\!)$,
$(\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = 0::\mathbb{N},\ m_v = (2::\mathbb{N}) |\!), (\!| p_v = Suc\ (0::\mathbb{N}),\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N} |\!)$,
$(\!| p_v = 2::\mathbb{N},\ c_v = Suc\ (0::\mathbb{N}),\ m_v = 0::\mathbb{N} |\!), (\!| p_v = 2::\mathbb{N},\ c_v = 2::\mathbb{N},\ m_v = Suc\ (0::\mathbb{N}) |\!)$
$\} = 6$
using *record-neq-p-c* **by** *fastforce*

have *finite-state-2'*: *finite* $\{s::mh\text{-}state.\ ((p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N})) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s\}$
apply (*rule rev-finite-subset*[**where** $B =$
$\{s::mh\text{-}state.\ (p_v\ s \le (2::\mathbb{N}) \wedge c_v\ s \le (2::\mathbb{N}) \wedge m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3::\mathbb{N}))\}$])

using *finite-states-2* **apply** *presburger*
by *fastforce*

— After a new fact is learned, 1/3 states are excluded because these states have its m_v v_0 equal to p_v v_0.
let *?infsum* = $(\sum_\infty v_0::mh\text{-}state.$
$((if\ p_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (c_v\ v_0)\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /$
$(18::\mathbb{R}) +$
$(if\ p_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ c_v\ v_0 \le (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (Suc\ (c_v\ v_0))\ mod\ (3::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /$
$(18::\mathbb{R})) * (if\ \neg\ m_v\ v_0 = p_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$

have *infsum-2-3*: *?infsum* = *2/3*
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1′* **apply** *blast*
apply *fastforce*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2′* **apply** *blast*
apply *fastforce*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1′* **apply** *blast*
apply *fastforce*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2′* **apply** *blast*
apply *fastforce*+
apply (*subst infsum-constant-finite-states*)
using *finite-state-1′* **apply** *blast*
apply (*subst infsum-constant-finite-states*)
using *finite-state-2′* **apply** *blast*
apply (*subst set-states-1′*, *subst card-states-1′*)
apply (*subst set-states-2′*, *subst card-states-2′*)
by (*simp*)

have *Forgetful-Monty‴*: (*Forgetful-Monty′* $\|_f$ $[\![m^> \neq p^>]\!]_{\mathcal{I}e}$) = *Forgetful-Monty′-learned*
apply (*simp add*: *dist-defs Forgetful-Monty′-def Forgetful-Monty′-learned-def*)
apply (*expr-auto*)
apply (*metis One-nat-def Suc-n-not-n mod-Suc one-eq-numeral-iff semiring-norm*(*86*))
using *mod-Suc* **apply** *auto*[*1*]
using *infsum-2-3* **by** *linarith*+

— The final states of the learned program is also a distribution.
have *Forgetful-Monty′-learned-dist*: *is-final-distribution Forgetful-Monty′-learned*
apply (*simp add*: *dist-defs Forgetful-Monty′-learned-def*)
apply (*expr-auto*)
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)

apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1′* **apply** *blast*
apply *fastforce+*
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2′* **apply** *blast*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1′* **apply** *blast*
apply *fastforce+*
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2′* **apply** *blast*
apply *fastforce+*
apply (*subst infsum-constant-finite-states*)
using *finite-state-1′* **apply** *blast*
apply (*subst infsum-constant-finite-states*)
using *finite-state-2′* **apply** *blast*
apply (*subst set-states-1′*, *subst card-states-1′*)
apply (*subst set-states-2′*, *subst card-states-2′*)
by (*simp*)

— Win when $c_v\ s = p_v\ s$
have *set-states-1″*: $\{s{::}mh\text{-}state.\ (((p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}))$
$\wedge\ m_v\ s = Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s) \wedge c_v\ s = p_v\ s\}$
$= \{(\!| p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N}) |\!),\ (\!| p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = 2{::}\mathbb{N} |\!),$
$(\!| p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = 0{::}\mathbb{N} |\!)\}$
apply (*simp add*: *set-eq-iff*)
apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective*
One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral(*3*)
le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less
mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct(*1*) *nle-le not-less-eq-eq*
numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral
pred-numeral-simps(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-1″*: $card\ \{(\!| p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N}) |\!),$
$(\!| p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = 2{::}\mathbb{N} |\!),\ (\!| p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = 0{::}\mathbb{N} |\!)\} = 3$
using *record-neq-p-c* **by** *fastforce*

have *finite-state-1″*: $finite\ \{s{::}mh\text{-}state.\ (((p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N})) \wedge$
$m_v\ s = Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s) \wedge c_v\ s = p_v\ s\}$
apply (*rule rev-finite-subset*[**where** $B =$
$\{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = Suc\ (c_v\ s)\ mod\ (3{::}\mathbb{N}))\}$])
using *finite-states-1* **apply** *presburger*
by *fastforce*

have *set-states-2″*: $\{s{::}mh\text{-}state.\ (((p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N})) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s) \wedge c_v\ s = p_v\ s\}$
$= \{(\!| p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = (2{::}\mathbb{N}) |\!),\ (\!| p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = 0{::}\mathbb{N} |\!),$
$(\!| p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N}) |\!)\ \}$
apply (*simp add*: *set-eq-iff*)

apply (*rule allI*)+
apply (*rule iffI*)
apply (*smt* (*verit*) *mh-state.select-convs*(*1*) *mh-state.select-convs*(*3*) *mh-state.surjective* *One-nat-def Suc-1 Suc-eq-numeral Suc-eq-plus1 Suc-le-mono add-Suc-right eval-nat-numeral*(*3*) *le-0-eq le-Suc-eq le-add2 lessI less-Suc-eq mod-Suc mod-Suc-le-divisor mod-less* *mod-less-eq-dividend mod-self n-not-Suc-n nat.distinct*(*1*) *nle-le not-less-eq-eq* *numeral-One numeral-eq-one-iff old.unit.exhaust one-add-one one-le-numeral* *pred-numeral-simps*(*2*) *trans-le-add2*)
by *fastforce*

have *card-states-2''*: $card\ \{(\!|p_v = 0{::}\mathbb{N},\ c_v = 0{::}\mathbb{N},\ m_v = (2{::}\mathbb{N})|\!),$
$(\!|p_v = Suc\ (0{::}\mathbb{N}),\ c_v = Suc\ (0{::}\mathbb{N}),\ m_v = 0{::}\mathbb{N}|\!),\ (\!|p_v = 2{::}\mathbb{N},\ c_v = 2{::}\mathbb{N},\ m_v = Suc\ (0{::}\mathbb{N})|\!)\ \} = 3$
using *record-neq-p-c* **by** *fastforce*

have *finite-state-2''*: $finite\ \{s{::}mh\text{-}state.\ (((p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N})) \wedge$
$m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N})) \wedge \neg\ m_v\ s = p_v\ s) \wedge c_v\ s = p_v\ s\}$
apply (*rule rev-finite-subset*[**where** $B =$
$\{s{::}mh\text{-}state.\ (p_v\ s \le (2{::}\mathbb{N}) \wedge c_v\ s \le (2{::}\mathbb{N}) \wedge m_v\ s = Suc\ (Suc\ (c_v\ s))\ mod\ (3{::}\mathbb{N}))\}$])
using *finite-states-2* **apply** *presburger*
by *fastforce*

— After learning a new fact, the probability to win is 1/2, and so it doesn't matter if the contestant chooses to switch or not.
have *infsum-1-2*: $(\sum_\infty v_0{::}mh\text{-}state.$
$((if\ p_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ c_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (c_v\ v_0)\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ \neg\ m_v\ v_0 = p_v\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))\ /$
$(12{::}\mathbb{R})\ +$
$(if\ p_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * (if\ c_v\ v_0 \le (2{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ m_v\ v_0 = Suc\ (Suc\ (c_v\ v_0))\ mod\ (3{::}\mathbb{N})\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$(if\ \neg\ m_v\ v_0 = p_v\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))\ /$
$(12{::}\mathbb{R})) *$
$(if\ c_v\ v_0 = p_v\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))) * (2{::}\mathbb{R}) = (1{::}\mathbb{R})$
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst conditional-conds-conj*)+
apply (*subst infsum-add*)
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1''* **apply** *blast*+
apply (*subst summable-on-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2''* **apply** *blast*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-1''* **apply** *blast*+
apply (*subst infsum-cdiv-left*)
apply (*subst infsum-constant-finite-states-summable*)
using *finite-state-2''* **apply** *blast*+
apply (*subst infsum-constant-finite-states*)
using *finite-state-1''* **apply** *blast*+
apply (*subst infsum-constant-finite-states*)
using *finite-state-2''* **apply** *blast*+
apply (*subst set-states-1''*, *subst card-states-1''*)
apply (*subst set-states-2''*, *subst card-states-2''*)
by (*simp*)

show *?thesis*
apply (*simp add*: *Learn-fact-def Forgetful-Monty-altdef*)
apply (*subst Forgetful-Monty''*)
apply (*subst Forgetful-Monty'''*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *Forgetful-Monty'-learned-dist is-final-distribution-prob is-final-prob-prob*)
apply (*simp add*: *Forgetful-Monty'-learned-def dist-defs*)
apply (*expr-auto*)
by (*simp add*: *infsum-1-2*)
qed

end

3 Robot localisation

theory *utp-prob-rel-lattice-robot-localisation*
imports
UTP-prob-relations.utp-prob-rel
begin

unbundle *UTP-Syntax*

declare [[*show-types*]]

named-theorems *robot-local-defs*

3.1 Definitions

alphabet *robot-local-state* =
bel :: *nat*

definition $door\ p = ((p = (0{::}\mathbb{N})) \vee (p = 2))$

definition *init* :: *robot-local-state rvhfun* **where**
$init = bel\,\boldsymbol{\mathcal{U}}\ \{(0{::}\mathbb{N}),\ 1,\ 2\}$

A noisy sensor is more likely to get a right reading than a wrong reading: 4 vs. 1.

definition *scale-door* :: *robot-local-state rvhfun* **where**
$scale\text{-}door = (3 * [\![«door»\ (bel^{>})]\!]_{\mathcal{I}e} + 1)_e$

definition *scale-wall* :: *robot-local-state rvhfun* **where**
$scale\text{-}wall = (3 * [\![\neg«door»\ (bel^{>})]\!]_{\mathcal{I}e} + 1)_e$

definition *move-right* :: *robot-local-state prhfun* **where**
$move\text{-}right = (bel := (bel + 1)\ mod\ 3)$

definition *robot-localisation* **where**
$robot\text{-}localisation = ((((init \parallel scale\text{-}door)\ ;\ move\text{-}right) \parallel scale\text{-}door)\ ;\ move\text{-}right) \parallel scale\text{-}wall$

definition *believe-1*::*robot-local-state rvhfun* **where**
$believe\text{-}1 \equiv (4/9 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 1/9 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 4/9 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$

definition *move-right-1*::*robot-local-state rvhfun* **where**
$move\text{-}right\text{-}1 \equiv (4/9 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 4/9 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 1/9 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$

definition *believe-2*::*robot-local-state rvhfun* **where**
$believe\text{-}2 \equiv (2/3 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 1/6 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 1/6 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$

definition *move-right-2*::*robot-local-state rvhfun* **where**
$move\text{-}right\text{-}2 \equiv (1/6 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 2/3 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 1/6 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$

definition *believe-3*::*robot-local-state rvhfun* **where**
$believe\text{-}3 \equiv (1/18 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 8/9 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 1/18 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$

3.2 First sensor reading

lemma *init-knowledge-sum*: $(\sum_{\infty} v_0::robot\text{-}local\text{-}state.$
$(if\ v_0 = (\!|bel_v = 0::\mathbb{N}|\!) \vee v_0 = (\!|bel_v = Suc\ (0::\mathbb{N})|\!) \vee v_0 = (\!|bel_v = 2::\mathbb{N}|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \vee bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) + (1::\mathbb{R})) /$
$(3::\mathbb{R})) = 3$
proof −
let $?bel\text{-}set = \{(\!|bel_v = 0::\mathbb{N}|\!), (\!|bel_v = Suc\ (0::\mathbb{N})|\!), (\!|bel_v = 2::\mathbb{N}|\!)\}$
let $?sum = (\sum_{\infty} v_0::robot\text{-}local\text{-}state.$
$(if\ v_0 = (\!|bel_v = 0::\mathbb{N}|\!) \vee v_0 = (\!|bel_v = Suc\ (0::\mathbb{N})|\!) \vee v_0 = (\!|bel_v = 2::\mathbb{N}|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \vee bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) + (1::\mathbb{R})) /$
$(3::\mathbb{R}))$
let $?fun = \lambda v_0.\ (if\ v_0 = (\!|bel_v = 0::\mathbb{N}|\!) \vee v_0 = (\!|bel_v = 2|\!)\ then\ 4::\mathbb{R}\ else$
$(if\ (\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))) / 3$
have $?sum = (\sum_{\infty} v_0::robot\text{-}local\text{-}state.\ ?fun\ v_0)$
apply (*subst infsum-cong*[**where** $g{=}\lambda v_0.\ (if\ v_0 = (\!|bel_v = 0::\mathbb{N}|\!) \vee v_0 = (\!|bel_v = 2|\!)\ then\ 4::\mathbb{R}\ else$
$(if\ v_0 = (\!|bel_v = Suc\ (0::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))) / 3$])
apply *simp*
by (*simp add*: *infsum-cong*)
also have ... $= (\sum_{\infty} v_0::robot\text{-}local\text{-}state \in ?bel\text{-}set \cup (UNIV - ?bel\text{-}set).\ ?fun\ v_0)$
by *auto*
also have ... $= (\sum_{\infty} v_0::robot\text{-}local\text{-}state \in ?bel\text{-}set.\ ?fun\ v_0)$
apply (*rule infsum-cong-neutral*)
apply *fastforce*
apply *fastforce*
by *blast*
also have ... $= (\sum v_0::robot\text{-}local\text{-}state \in \{(\!|bel_v = 0::\mathbb{N}|\!)\}.\ ?fun\ v_0) +$
$(\sum v_0::robot\text{-}local\text{-}state \in \{(\!|bel_v = Suc\ (0::\mathbb{N})|\!), (\!|bel_v = (2::\mathbb{N})|\!)\}.\ ?fun\ v_0)$
apply (*subst infsum-finite*)
apply (*simp*)
by *force*
also have ... $= (\sum v_0::robot\text{-}local\text{-}state \in \{(\!|bel_v = 0::\mathbb{N}|\!)\}.\ ?fun\ v_0) +$
$(\sum v_0::robot\text{-}local\text{-}state \in \{(\!|bel_v = Suc\ (0::\mathbb{N})|\!)\}.\ ?fun\ v_0) +$
$(\sum v_0::robot\text{-}local\text{-}state \in \{(\!|bel_v = (2::\mathbb{N})|\!)\}.\ ?fun\ v_0)$
by *force*
also have ... $= 3$
by *simp*
then show *?thesis*
using *calculation* **by** *presburger*
qed

lemma *believe-1-simp*: (*init* ∥ *scale-door*) = *prfun-of-rvfun believe-1*
apply (*simp add*: *pparallel-def init-def scale-door-def believe-1-def*)
apply (*simp add*: *dist-norm-final-def*)
apply (*simp add*: *rvfun-uniform-dist-altdef*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])

apply (*simp add*: *door-def*)
apply (*simp add*: *expr-defs assigns-r-def*)
apply (*pred-auto*)
using *init-knowledge-sum* **apply** *auto*[*1*]
using *init-knowledge-sum* **apply** *linarith*
apply (*simp add*: *init-knowledge-sum*)
using *init-knowledge-sum* **by** *auto*[*1*]

lemma *believe-1-simp'*: (*init* $\|$ *scale-door*) = *prfun-of-rvfun believe-1*
apply (*simp add*: *init-def believe-1-def*)
apply (*subst prfun-parallel-uniform-dist*)
apply (*simp*)+
apply (*simp add*: *scale-door-def*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *door-def*)
apply (*simp add*: *expr-defs*)
by (*pred-auto*)

3.3 First move

lemma *move-right-1-simp*: (*init* $\|$ *scale-door*) ; *move-right* = *prfun-of-rvfun move-right-1*
apply (*simp add*: *pseqcomp-def move-right-1-def*)

apply (*simp add*: *init-def*)
apply (*subst prfun-parallel-uniform-dist'*)
apply (*simp*)+
apply (*simp add*: *scale-door-def door-def*)
apply (*expr-auto*)
apply (*simp add*: *scale-door-def door-def*)
apply (*expr-auto*)
apply (*simp add*: *pfun-defs dist-norm-final-def move-right-def scale-door-def door-def*)
apply (*subst rvfun-assignment-inverse*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*expr-auto add*: *rel assigns-r-def*)
proof −
let *?lhs-f* = λv_0::*robot-local-state*. ((*if* v_0 = (|bel_v = *0*::ℕ|) *then* *1*::ℝ *else* (*0*::ℝ)) ∗ (*4*::ℝ) +
((*if* v_0 = (|bel_v = *Suc* (*0*::ℕ)|) *then* *1*::ℝ *else* (*0*::ℝ)) +
(*if* v_0 = (|bel_v = *2*::ℕ|) *then* *1*::ℝ *else* (*0*::ℝ)) ∗ (*4*::ℝ))) ∗
(*if* (|bel_v = *0*::ℕ|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::ℕ)|) *then* *1*::ℝ *else* (*0*::ℝ)) / *9*
let *?lhs* = ($\sum_\infty v_0$::*robot-local-state*. *?lhs-f* v_0)

have *f1*: ∀ v_0. ¬(v_0 = (|bel_v = *0*::ℕ|) ∧ (|bel_v = *0*::ℕ|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::ℕ)|))
by (*auto*)
have *f2*: ∀ v_0. ¬(v_0 = (|bel_v = *Suc* (*0*::ℕ)|) ∧ (|bel_v = *0*::ℕ|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::ℕ)|))
by (*auto*)
have *f3*: ∀ v_0. (v_0 = (|bel_v = *2*::ℕ|) ∧ (|bel_v = *0*::ℕ|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::ℕ)|)) =
(v_0 = (|bel_v = *2*::ℕ|))
by (*auto*)
have *?lhs* = (*4* / *9*)
apply (*subst ring-distribs*(*2*))+
apply (*simp add*: *mult.commute*[**where** *b* = (*4*::ℝ)])+
apply (*simp add*: *mult.assoc*)+
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *f1 f2 f3*)
apply (*subst infsum-cdiv-left*)

apply (*rule summable-on-cmult-right*)
apply (*smt* (*verit*, *best*) *infsum-singleton-summable summable-on-cong zero-neq-one*)
apply (*subst infsum-cmult-right*)
apply (*smt* (*verit*, *best*) *infsum-singleton-summable summable-on-cong zero-neq-one*)
apply (*subst infsum-constant-finite-states*)
by (*simp*)+

then show *?lhs* $*$ (*9*::$\mathbb{R}$) = (*4*::$\mathbb{R}$)
by *linarith*
next
let *?lhs-f* = λv_0::*robot-local-state*. ((*if* v_0 = (|bel_v = *0*::$\mathbb{N}$|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$ (*4*::$\mathbb{R}$) +
((*if* v_0 = (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) +
(*if* v_0 = (|bel_v = *2*::$\mathbb{N}$|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$ (*4*::$\mathbb{R}$))) $*$
(*if* (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / *9*
let *?lhs* = ($\sum_\infty v_0$::*robot-local-state*. *?lhs-f* v_0)

have *f1*: $\forall v_0$. (v_0 = (|bel_v = *0*::$\mathbb{N}$|) $\wedge$ (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
=
(v_0 = (|bel_v = *0*::$\mathbb{N}$|))
by (*auto*)
have *f2*: $\forall v_0$. $\neg$(v_0 = (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) $\wedge$ (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
by (*auto*)
have *f3*: $\forall v_0$. $\neg$(v_0 = (|bel_v = *2*::$\mathbb{N}$|) $\wedge$ (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
by (*auto*)
have *?lhs* = (*4* / *9*)
apply (*subst ring-distribs*(*2*))+
apply (*simp add*: *mult.commute*[**where** *b* = (*4*::$\mathbb{R}$)])+
apply (*simp add*: *mult.assoc*)+
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *f1 f2 f3*)
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*smt* (*verit*, *best*) *infsum-singleton-summable summable-on-cong zero-neq-one*)
apply (*subst infsum-cmult-right*)
apply (*smt* (*verit*, *best*) *infsum-singleton-summable summable-on-cong zero-neq-one*)
apply (*subst infsum-constant-finite-states*)
by (*simp*)+

then show *?lhs* $*$ (*9*::$\mathbb{R}$) = (*4*::$\mathbb{R}$)
by *linarith*
next
let *?lhs-f* = λv_0::*robot-local-state*. ((*if* v_0 = (|bel_v = *0*::$\mathbb{N}$|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$ (*4*::$\mathbb{R}$) +
((*if* v_0 = (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) +
(*if* v_0 = (|bel_v = *2*::$\mathbb{N}$|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$ (*4*::$\mathbb{R}$))) $*$
(*if* (|bel_v = (*2*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / *9*
let *?lhs* = ($\sum_\infty v_0$::*robot-local-state*. *?lhs-f* v_0)

have *f1*: $\forall v_0$. $\neg$(v_0 = (|bel_v = *0*::$\mathbb{N}$|) $\wedge$ (|bel_v = (*2*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
by (*auto*)
have *f2*: $\forall v_0$. (v_0 = (|bel_v = *Suc* (*0*::$\mathbb{N}$)|) $\wedge$ (|bel_v = (*2*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
= (v_0 = (|bel_v = *Suc* (*0*::$\mathbb{N}$)|))
by (*auto*)
have *f3*: $\forall v_0$. $\neg$(v_0 = (|bel_v = *2*::$\mathbb{N}$|) $\wedge$ (|bel_v = (*2*::$\mathbb{N}$)|) = v_0(|bel_v := *Suc* (bel_v v_0) *mod* (*3*::$\mathbb{N}$)|))
by (*auto*)

have *?lhs* = (*1* / *9*)
apply (*subst ring-distribs*(*2*))+
apply (*simp add*: *mult.commute*[**where** $b = (4::\mathbb{R})$])+
apply (*simp add*: *mult.assoc*)+
apply (*subst conditional-conds-conj*)+
apply (*simp add*: *f1 f2 f3*)
apply (*subst infsum-cdiv-left*)
apply (*smt* (*verit*, *best*) *infsum-singleton-summable summable-on-cong zero-neq-one*)
apply (*subst infsum-constant-finite-states*)
by (*simp*)+

then show *?lhs* $* (9::\mathbb{R}) = (1::\mathbb{R})$
by *linarith*
next
fix *bel*
assume *a1*: $(0::\mathbb{N}) < bel$
assume *a2*: $\neg\ bel = Suc\ (0::\mathbb{N})$
assume *a3*: $\neg\ bel = (2::\mathbb{N})$
let *?lhs-f* = λv_0::*robot-local-state*. $((if\ v_0 = (\!|bel_v = 0::\mathbb{N}|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (4::\mathbb{R}) +$
$((if\ v_0 = (\!|bel_v = Suc\ (0::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ v_0 = (\!|bel_v = 2::\mathbb{N}|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (4::\mathbb{R}))) *$
$(if\ (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ 9$
let *?lhs* = $(\sum_\infty v_0$::*robot-local-state*. *?lhs-f* $v_0)$

have *f1*: $\forall v_0.\ \neg(v_0 = (\!|bel_v = 0::\mathbb{N}|\!) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
using *a2* **by** *force*
have *f2*: $\forall v_0.\ \neg(v_0 = (\!|bel_v = Suc\ (0::\mathbb{N})|\!) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
using *a3* **by** *force*
have *f3*: $\forall v_0.\ \neg(v_0 = (\!|bel_v = 2::\mathbb{N}|\!) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
using *a1* **by** *force*
have *?lhs* = *0*
apply (*subst ring-distribs*(*2*))+
apply (*simp add*: *mult.commute*[**where** $b = (4::\mathbb{R})$])+
apply (*simp add*: *mult.assoc*)+
apply (*subst conditional-conds-conj*)+
by (*simp add*: *f1 f2 f3*)

then show *?lhs* = *0*
by *linarith*
qed

lemma *move-right-1-dist*: *rvfun-of-prfun* (*prfun-of-rvfun move-right-1*) = *move-right-1*
proof −
have *summable-1*: (λs::*robot-local-state*. $(4::\mathbb{R}) * (if\ bel_v\ s = (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (9::\mathbb{R}))$
summable-on UNIV
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono card-0-eq finite.insertI infinite-arbitrarily-large rev-finite-subset robot-local-state.surjective singleton-conv unit.exhaust*)

have *summable-2*: (λs::*robot-local-state*. $(4::\mathbb{R}) * (if\ bel_v\ s = Suc\ (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /$
$(9::\mathbb{R}))$
summable-on UNIV
apply (*rule summable-on-cdiv-left*)

apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *summable-3*: (λs::*robot-local-state*. (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*9*::$\mathbb{R}$)) *summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *sum-1*: ($\sum_\infty$*s*::*robot-local-state*. (*4*::$\mathbb{R}$) * (*if* bel_v *s* = (*0*::$\mathbb{N}$) *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*9*::$\mathbb{R}$)) = *4/9*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-2*: ($\sum_\infty$*s*::*robot-local-state*. (*4*::$\mathbb{R}$) * (*if* bel_v *s* = *Suc* (*0*::$\mathbb{N}$) *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*9*::$\mathbb{R}$)) = *4/9*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = *Suc* (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-3*: ($\sum_\infty$*s*::*robot-local-state*. (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*9*::$\mathbb{R}$)) = *1/9*
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac* $x = (\!| bel_v = (2::\mathbb{N}) |\!)$ **in** *exI*)
by *force*

show *?thesis*
apply (*simp add*: *move-right-1-def*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
by *simp*
qed

3.4 Second sensor reading

lemma *believe-2-sum*: $(\sum_\infty v_0::robot\text{-}local\text{-}state.$
$(4::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /$
$(9::\mathbb{R}) +$
$(4::\mathbb{R}) * (if\ bel_v\ v_0 = Suc\ (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /$
$(9::\mathbb{R}) +$
$(if\ bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /$
$(9::\mathbb{R})) = 8\ /3$
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst mult.assoc*[*symmetric*,**where** $b = 3$])
apply (*subst mult.commute*[**where** $b = 3$])
apply (*subst mult.assoc*)
apply (*subst conditional-conds-conj*)+
proof −
let $?f1 = (\lambda v_0::robot\text{-}local\text{-}state.\ ((12::\mathbb{R}) *$
$(if\ bel_v\ v_0 = (0::\mathbb{N}) \land (bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N}))\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(4::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))\ /$
$(9::\mathbb{R}))$
let $?f2 = (\lambda v_0::robot\text{-}local\text{-}state.\ ((12::\mathbb{R}) *$
$(if\ bel_v\ v_0 = Suc\ (0::\mathbb{N}) \land (bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N}))\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(4::\mathbb{R}) * (if\ bel_v\ v_0 = Suc\ (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))\ /$
$(9::\mathbb{R}))$
let $?f3 = (\lambda v_0::robot\text{-}local\text{-}state.\ ((3::\mathbb{R}) * (if\ bel_v\ v_0 = (2::\mathbb{N}) \land (bel_v\ v_0 = (0::\mathbb{N}) \lor bel_v\ v_0 = (2::\mathbb{N}))$
$then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ bel_v\ v_0 = (2::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))\ /$
$(9::\mathbb{R}))$
have *summable-1*: *?f1 summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)

have *summable-2*: *?f2 summable-on UNIV*
  apply (*rule summable-on-cdiv-left*)
  apply (*rule summable-on-add*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
have *summable-3*: *?f3 summable-on UNIV*
  apply (*rule summable-on-cdiv-left*)
  apply (*rule summable-on-add*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*rule infsum-constant-finite-states-summable*)
  by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)

have *card-1*: *card* $\{s::robot\text{-}local\text{-}state.\ bel_v\ s = 0\} = Suc\ (0)$
  apply (*subst card-1-singleton-iff*)
  by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)
have *card-2*: *card* $\{s::robot\text{-}local\text{-}state.\ bel_v\ s = Suc\ (0)\} = Suc\ (0)$
  apply (*subst card-1-singleton-iff*)
  by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)
have *card-3*: *card* $\{s::robot\text{-}local\text{-}state.\ bel_v\ s = 2\} = Suc\ (0)$
  apply (*subst card-1-singleton-iff*)
  by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)

have *sum-1*: $(\sum_\infty v_0::robot\text{-}local\text{-}state.\ ?f1\ v_0) = 16\ /\ 9$
  apply (*subst infsum-cdiv-left*)
  apply (*rule summable-on-add*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
  apply (*subst infsum-add*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*rule summable-on-cmult-right*)
  apply (*rule infsum-constant-finite-states-summable*)
  apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)

apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
using *card-1* **by** (*smt* (*verit*, *ccfv-SIG*) *Collect-cong One-nat-def of-nat-1*)

have *sum-2*: ($\sum_\infty v_0$::*robot-local-state*. *?f2* v_0) = *4* / *9*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
using *card-2* **by** (*simp add*: *card-0-singleton*)

have *sum-3*: ($\sum_\infty v_0$::*robot-local-state*. *?f3* v_0) = *4* / *9*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)

apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
using *card-3* **by** (*smt* (*verit*, *ccfv-SIG*) *Collect-cong One-nat-def of-nat-1*)

show $(\sum_\infty v_0$::*robot-local-state*. *?f1* v_0 + *?f2* v_0 + *?f3* $v_0) * 3 = 8$
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
using *summable-1* **apply** *blast*
using *summable-2* **apply** *blast*
using *summable-3* **apply** *blast*
apply (*subst infsum-add*)
using *summable-1* **apply** *blast*
using *summable-2* **apply** *blast*
by (*simp add*: *sum-1 sum-2 sum-3*)
qed

lemma *believe-2-simp*: (((*init* ∥ *scale-door*) ; *move-right*) ∥ *scale-door*) =
prfun-of-rvfun believe-2
apply (*simp add*: *move-right-1-simp believe-2-def*)
apply (*simp add*: *scale-door-def door-def pfun-defs*)
apply (*simp add*: *move-right-1-dist*)
apply (*simp add*: *move-right-1-def dist-defs*)
apply (*expr-simp-1*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst fun-eq-iff*, *rule allI*)
apply (*auto*)
by (*simp add*: *believe-2-sum*)+

lemma *believe-2-dist*: *rvfun-of-prfun* (*prfun-of-rvfun believe-2*) = *believe-2*
proof −
have *summable-1*: (λ*s*::*robot-local-state*. $(2::\mathbb{R}) * ($*if* bel_v *s* $= (0::\mathbb{N})$ *then* $1::\mathbb{R}$ *else* $(0::\mathbb{R})) / (3::\mathbb{R}))$

summable-on UNIV
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono card-0-eq finite.insertI infinite-arbitrarily-large rev-finite-subset robot-local-state.surjective singleton-conv unit.exhaust*)

have *summable-2*: (λs::*robot-local-state*. (*if* bel_v *s* = *Suc* (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) *summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *summable-3*: (λs::*robot-local-state*. (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) *summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *sum-1*: ($\sum_\infty$*s*::*robot-local-state*. (*2*::$\mathbb{R}$) $*$ (*if* bel_v *s* = (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*3*::$\mathbb{R}$)) = *2/3*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-2*: ($\sum_\infty$*s*::*robot-local-state*. (*if* bel_v *s* = *Suc* (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) = *1/6*
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = *Suc* (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-3*: ($\sum_\infty$*s*::*robot-local-state*. (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) = *1/6*
apply (*subst infsum-cdiv-left*)

  apply (*rule infsum-constant-finite-states-summable*)
  apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*subst infsum-constant-finite-states*)
  apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
  apply (*simp*)
  apply (*subst card-1-singleton-iff*)
  apply (*rule-tac* $x = (\!|bel_v = (2::\mathbb{N})|\!)$ **in** *exI*)
 by *force*

 show *?thesis*
  apply (*simp add*: *believe-2-def*)
  apply (*subst rvfun-inverse*)
  apply (*expr-auto add*: *dist-defs*)
  by (*simp*)
qed

3.5 Second move

lemma *move-right-2-simp*:
 $((((init \parallel scale\text{-}door) \ ; \ move\text{-}right) \parallel scale\text{-}door) \ ; \ move\text{-}right) = prfun\text{-}of\text{-}rvfun\ move\text{-}right\text{-}2$
 apply (*simp add*: *believe-2-simp*)
 apply (*simp add*: *move-right-2-def move-right-def*)
 apply (*simp add*: *pfun-defs*)
 apply (*simp add*: *believe-2-dist*)
 apply (*subst rvfun-assignment-inverse*)
 apply (*simp add*: *believe-2-def*)
 apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
 apply (*expr-auto add*: *rel assigns-r-def*)
 apply (*simp-all add*: *ring-distribs*(*2*))
 apply (*simp add*: *mult.assoc*)+
 apply (*subst conditional-conds-conj*)+
 defer
 apply (*simp add*: *mult.assoc*)+
 apply (*subst conditional-conds-conj*)+
 defer
 apply (*simp add*: *mult.assoc*)+
 apply (*subst conditional-conds-conj*)+
 defer
 apply (*simp add*: *mult.assoc*)+
 apply (*subst conditional-conds-conj*)+
 defer
proof −
 let $?lhs\text{-}f = \lambda v_0::robot\text{-}local\text{-}state.\ (2::\mathbb{R}) *$
   $(if\ bel_v\ v_0 = (0::\mathbb{N}) \wedge (\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
   $else\ (0::\mathbb{R}))\ /\ (3::\mathbb{R})\ +$
   $(if\ bel_v\ v_0 = Suc\ (0::\mathbb{N}) \wedge (\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
   $else\ (0::\mathbb{R}))\ /\ (6::\mathbb{R})\ +$
   $(if\ bel_v\ v_0 = (2::\mathbb{N}) \wedge (\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
   $else\ (0::\mathbb{R}))\ /\ (6::\mathbb{R})$
 let $?lhs = (\sum_\infty v_0::robot\text{-}local\text{-}state.\ ?lhs\text{-}f\ v_0)$

 have *f1*: $\forall v_0.\ (bel_v\ v_0 = (0::\mathbb{N}) \wedge ((\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))) =$
   $((\!|bel_v = 0::\mathbb{N}|\!) = v_0)$
  by *auto*

have *f2*: $\forall v_0.\ \neg(bel_v\ v_0 = Suc\ (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = Suc\ (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis)$
  apply (*auto*)
  by (*metis n-not-Suc-n robot-local-state.select-convs(1) robot-local-state.surjective robot-local-state.update-convs(1)*)
have *f3*: $\forall v_0.\ \neg(bel_v\ v_0 = (2{::}\mathbb{N}) \wedge \llparenthesis bel_v = Suc\ (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis)$
  apply (*auto*)
  by (*metis n-not-Suc-n robot-local-state.select-convs(1) robot-local-state.surjective robot-local-state.update-convs(1)*)
show $?lhs * (3{::}\mathbb{R}) = (2{::}\mathbb{R})$
  apply (*simp add*: *f1 f2 f3*)
  apply (*subst infsum-cdiv-left*)
  apply (*rule summable-on-cmult-right*)
  apply (*simp add*: *infsum-singleton-summable*)
  apply (*subst infsum-cmult-right*)
  apply (*simp add*: *infsum-singleton-summable*)
  apply (*subst infsum-constant-finite-states*)
  by (*simp*)+
next
let $?lhs\text{-}f = \lambda v_0{::}robot\text{-}local\text{-}state.\ (2{::}\mathbb{R}) *$
$(if\ bel_v\ v_0 = (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (3{::}\mathbb{R})\ +$
$(if\ bel_v\ v_0 = Suc\ (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (6{::}\mathbb{R})\ +$
$(if\ bel_v\ v_0 = (2{::}\mathbb{N}) \wedge \llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (6{::}\mathbb{R})$
let $?lhs = (\sum_\infty v_0{::}robot\text{-}local\text{-}state.\ ?lhs\text{-}f\ v_0)$

have *f1*: $\forall v_0.\ \neg(bel_v\ v_0 = (0{::}\mathbb{N}) \wedge (\llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis))$
  apply (*auto*)
  by (*metis n-not-Suc-n robot-local-state.select-convs(1) robot-local-state.surjective robot-local-state.update-convs(1)*)
have *f2*: $\forall v_0.\ \neg(bel_v\ v_0 = Suc\ (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis)$
  apply (*auto*)
  by (*metis nat.distinct(1) robot-local-state.select-convs(1) robot-local-state.surjective robot-local-state.update-convs(1)*)
have *f3*: $\forall v_0.\ (bel_v\ v_0 = (2{::}\mathbb{N}) \wedge \llparenthesis bel_v = (0{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis) =$
$(\llparenthesis bel_v = 2{::}\mathbb{N}\rrparenthesis = v_0)$
  by (*auto*)
show $?lhs * (6{::}\mathbb{R}) = (1{::}\mathbb{R})$
  apply (*simp add*: *f1 f2 f3*)
  apply (*subst infsum-cdiv-left*)
  apply (*simp add*: *infsum-singleton-summable*)
  apply (*subst infsum-constant-finite-states*)
  by (*simp*)+
next
let $?lhs\text{-}f = \lambda v_0{::}robot\text{-}local\text{-}state.\ (2{::}\mathbb{R}) *$
$(if\ bel_v\ v_0 = (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = (2{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (3{::}\mathbb{R})\ +$
$(if\ bel_v\ v_0 = Suc\ (0{::}\mathbb{N}) \wedge \llparenthesis bel_v = (2{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (6{::}\mathbb{R})\ +$
$(if\ bel_v\ v_0 = (2{::}\mathbb{N}) \wedge \llparenthesis bel_v = (2{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis\ then\ 1{::}\mathbb{R}$
$else\ (0{::}\mathbb{R}))\ /\ (6{::}\mathbb{R})$
let $?lhs = (\sum_\infty v_0{::}robot\text{-}local\text{-}state.\ ?lhs\text{-}f\ v_0)$

have *f1*: $\forall v_0.\ \neg(bel_v\ v_0 = (0{::}\mathbb{N}) \wedge (\llparenthesis bel_v = (2{::}\mathbb{N})\rrparenthesis = v_0\llparenthesis bel_v := Suc\ (bel_v\ v_0)\ mod\ (3{::}\mathbb{N})\rrparenthesis))$

apply (*auto*)
by (*metis n-not-Suc-n numeral-2-eq-2 robot-local-state.select-convs*(*1*)
robot-local-state.surjective robot-local-state.update-convs(*1*))
have *f2*: $\forall v_0.\ (bel_v\ v_0 = Suc\ (0::\mathbb{N}) \wedge (\!|bel_v = (2::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)) =$
$((\!|bel_v = Suc\ (0::\mathbb{N})|\!) = v_0)$
by (*auto*)
have *f3*: $\forall v_0.\ \neg(bel_v\ v_0 = (2::\mathbb{N}) \wedge (\!|bel_v = (2::\mathbb{N})|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
apply (*auto*)
by (*metis robot-local-state.select-convs*(*1*) *robot-local-state.surjective*
robot-local-state.update-convs(*1*) *zero-neq-numeral*)
show $?lhs * (6::\mathbb{R}) = (1::\mathbb{R})$
apply (*simp add*: *f1 f2 f3*)
apply (*subst infsum-cdiv-left*)
apply (*simp add*: *infsum-singleton-summable*)
apply (*subst infsum-constant-finite-states*)
by (*simp*)+
next
fix *bel*
assume *a1*: $\neg\ bel = Suc\ (0::\mathbb{N})$
assume *a2*: $(0::\mathbb{N}) < bel$
assume *a3*: $\neg\ bel = (2::\mathbb{N})$

have *f1*: $\forall v_0.\ \neg(bel_v\ v_0 = (0::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
apply (*auto*)
by (*metis a1 robot-local-state.select-convs*(*1*) *robot-local-state.surjective*
robot-local-state.update-convs(*1*))
have *f2*: $\forall v_0.\ \neg(bel_v\ v_0 = Suc\ (0::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
apply (*auto*)
by (*metis a3 numeral-2-eq-2 robot-local-state.select-convs*(*1*) *robot-local-state.surjective*
robot-local-state.update-convs(*1*))
have *f3*: $\forall v_0.\ \neg(bel_v\ v_0 = (2::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!))$
apply (*auto*)
by (*metis a2 nat-neq-iff robot-local-state.select-convs*(*1*) *robot-local-state.surjective*
robot-local-state.update-convs(*1*))

show $(\sum_\infty v_0::robot\text{-}local\text{-}state.$
$(2::\mathbb{R}) * (if\ bel_v\ v_0 = (0::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R})) /$
$(3::\mathbb{R}) +$
$(if\ bel_v\ v_0 = Suc\ (0::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R})) /$
$(6::\mathbb{R}) +$
$(if\ bel_v\ v_0 = (2::\mathbb{N}) \wedge (\!|bel_v = bel|\!) = v_0(\!|bel_v := Suc\ (bel_v\ v_0)\ mod\ (3::\mathbb{N})|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R})) /$
$(6::\mathbb{R})) =$
$(0::\mathbb{R})$
by (*simp add*: *f1 f2 f3*)
qed

lemma *move-right-2-dist*: *rvfun-of-prfun* (*prfun-of-rvfun move-right-2*) = *move-right-2*
proof −
have *summable-1*: $(\lambda s::robot\text{-}local\text{-}state.\ (if\ bel_v\ s = (0::\mathbb{N})\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (6::\mathbb{R}))$
summable-on UNIV
apply (*rule summable-on-cdiv-left*)

apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono card-0-eq finite.insertI infinite-arbitrarily-large rev-finite-subset robot-local-state.surjective singleton-conv unit.exhaust*)

have *summable-2*: (λs::*robot-local-state.* (*2*::$\mathbb{R}$) $*$ (*if* bel_v *s* = *Suc* (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*3*::$\mathbb{R}$))
summable-on UNIV
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *summable-3*: (λs::*robot-local-state.* (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$))
summable-on UNIV
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
by (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)

have *sum-1*: ($\sum_\infty s$::*robot-local-state.* (*if* bel_v *s* = (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) = *1/6*
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-2*: ($\sum_\infty s$::*robot-local-state.* (*2*::$\mathbb{R}$) $*$ (*if* bel_v *s* = *Suc* (*0*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*3*::$\mathbb{R}$)) = *2/3*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac x* = ⦇bel_v = *Suc* (*0*::$\mathbb{N}$)⦈ **in** *exI*)
by *force*

have *sum-3*: ($\sum_\infty s$::*robot-local-state.* (*if* bel_v *s* = (*2*::$\mathbb{N}$) *then* *1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / (*6*::$\mathbb{R}$)) = *1/6*
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)

apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*smt* (*z3*) *Collect-mono finite.emptyI finite.insertI rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*simp*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac* $x = (\!|bel_v = (2::\mathbb{N})|\!)$ **in** *exI*)
by *force*

show *?thesis*
apply (*simp add*: *move-right-2-def*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
by (*simp*)
qed

3.6 Third sensor reading

lemma *believe-3-sum*: $(\sum_\infty v_0::\mathit{robot\text{-}local\text{-}state}.$
$(\mathit{if}\ bel_v\ v_0 = (0::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (\mathit{if}\ (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /\ (6::\mathbb{R})$
$+\ (2::\mathbb{R}) * (\mathit{if}\ bel_v\ v_0 = \mathit{Suc}\ (0::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (\mathit{if}\ (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /$
$(3::\mathbb{R}) +\ (\mathit{if}\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) *$
$((3::\mathbb{R}) * (\mathit{if}\ (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) + (1::\mathbb{R}))\ /$
$(6::\mathbb{R})) = 3$
apply (*simp add*: *ring-distribs*(*1*))
apply (*subst mult.assoc*[*symmetric*,**where** $b = 3$])
apply (*subst mult.commute*[**where** $b = 3$])
apply (*subst mult.assoc*)
apply (*subst mult.assoc*[*symmetric*,**where** $b = 3$])
apply (*subst mult.commute*[**where** $b = 3$])
apply (*subst mult.assoc*)
apply (*subst conditional-conds-conj*)+
proof −
let *?f1* $= (\lambda v_0::\mathit{robot\text{-}local\text{-}state}.$
$((3::\mathbb{R}) * (\mathit{if}\ bel_v\ v_0 = (0::\mathbb{N}) \wedge (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) +$
$(\mathit{if}\ bel_v\ v_0 = (0::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})))\ /\ (6::\mathbb{R}))$
let *?f2* $= (\lambda v_0::\mathit{robot\text{-}local\text{-}state}.$
$((6::\mathbb{R}) * (\mathit{if}\ bel_v\ v_0 = \mathit{Suc}\ (0::\mathbb{N}) \wedge (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) +$
$(2::\mathbb{R}) * (\mathit{if}\ bel_v\ v_0 = \mathit{Suc}\ (0::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})))\ /$
$(3::\mathbb{R}))$
let *?f3* $= (\lambda v_0::\mathit{robot\text{-}local\text{-}state}.$
$((3::\mathbb{R}) * (\mathit{if}\ bel_v\ v_0 = (2::\mathbb{N}) \wedge (0::\mathbb{N}) < bel_v\ v_0 \wedge \neg\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})) +$
$(\mathit{if}\ bel_v\ v_0 = (2::\mathbb{N})\ \mathit{then}\ 1::\mathbb{R}\ \mathit{else}\ (0::\mathbb{R})))\ /$
$(6::\mathbb{R}))$
have *summable-1*: *?f1 summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD*

robot-local-state.equality unit.exhaust)
have *summable-2*: *?f2 summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
have *summable-3*: *?f3 summable-on UNIV*
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
by (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)

have *card-1*: $card\ \{s{::}robot\text{-}local\text{-}state.\ bel_v\ s = 0\} = Suc\ (0)$
apply (*subst card-1-singleton-iff*)
by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)
have *card-2*: $card\ \{s{::}robot\text{-}local\text{-}state.\ bel_v\ s = Suc\ (0)\} = Suc\ (0)$
apply (*subst card-1-singleton-iff*)
by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)
have *card-2'*: $card\ \{s{::}robot\text{-}local\text{-}state.\ bel_v\ s = Suc\ (0{::}\mathbb{N}) \land (0{::}\mathbb{N}) < bel_v\ s \land \neg\ bel_v\ s = (2{::}\mathbb{N})\} = Suc\ 0$
apply (*subst card-1-singleton-iff*)
by (*metis* (*mono-tags*, *lifting*) *Collect-cong card-1-singleton-iff card-2 less-Suc0 n-not-Suc-n numeral-2-eq-2*)
have *card-3*: $card\ \{s{::}robot\text{-}local\text{-}state.\ bel_v\ s = 2\} = Suc\ (0)$
apply (*subst card-1-singleton-iff*)
by (*smt* (*verit*, *del-insts*) *Collect-cong robot-local-state.equality robot-local-state.select-convs*(*1*) *singleton-conv unit.exhaust*)
have *card-3'*: $card\ \{s{::}robot\text{-}local\text{-}state.\ bel_v\ s = (2{::}\mathbb{N}) \land (0{::}\mathbb{N}) < bel_v\ s \land \neg\ bel_v\ s = (2{::}\mathbb{N})\} = 0$
by (*simp add*: *card-0-singleton*)

have *f1*: $\forall v_0.\ \neg(bel_v\ v_0 = (0{::}\mathbb{N}) \land (0{::}\mathbb{N}) < bel_v\ v_0 \land \neg\ bel_v\ v_0 = (2{::}\mathbb{N}))$
by *auto*
have *sum-1*: $(\sum_\infty v_0{::}robot\text{-}local\text{-}state.\ ?f1\ v_0) = 1\ /\ 6$
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)

apply (*simp add*: *f1*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
using *card-1* **by** (*smt* (*verit*, *ccfv-SIG*) *Collect-cong One-nat-def of-nat-1*)

have *sum-2*: ($\sum_\infty v_0$::*robot-local-state*. *?f2* v_0) = *8/3*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
by (*simp add*: *card-2 card-2'*)

have *sum-3*: ($\sum_\infty v_0$::*robot-local-state*. *?f3* v_0) = *1 / 6*
apply (*subst infsum-cdiv-left*)
apply (*rule summable-on-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-add*)
apply (*rule summable-on-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)

apply (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite.emptyI finite.insertI not-finite-existsD rev-finite-subset robot-local-state.equality singleton-conv unit.exhaust*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-cmult-right*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
apply (*subst infsum-constant-finite-states*)
apply (*metis* (*mono-tags*, *lifting*) *card.infinite card-1-singleton nat.simps*(*3*) *not-finite-existsD robot-local-state.equality unit.exhaust*)
by (*simp add*: *card-3 card-3'*)

show $(\sum_\infty v_0::robot\text{-}local\text{-}state.\ ?f1\ v_0 + ?f2\ v_0 + ?f3\ v_0) = 3$
apply (*subst infsum-add*)
apply (*rule summable-on-add*)
using *summable-1* **apply** *blast*
using *summable-2* **apply** *blast*
using *summable-3* **apply** *blast*
apply (*subst infsum-add*)
using *summable-1* **apply** *blast*
using *summable-2* **apply** *blast*
by (*simp add*: *sum-1 sum-2 sum-3*)
qed

lemma *believe-3-simp*: *robot-localisation* = *prfun-of-rvfun believe-3*
apply (*simp add*: *robot-localisation-def*)
apply (*simp add*: *move-right-2-simp believe-3-def*)
apply (*simp add*: *scale-wall-def door-def pfun-defs*)
apply (*simp add*: *move-right-2-dist*)
apply (*simp add*: *move-right-2-def dist-defs*)
apply (*expr-simp-1*)
apply (*rule HOL.arg-cong*[**where** *f*=*prfun-of-rvfun*])
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst fun-eq-iff*, *rule allI*)
apply (*auto*)
by (*simp add*: *believe-3-sum*)+

lemma *robot-localisation*:
(((*init* ∥ *scale-door*) ;
move-right ∥ *scale-door*) ;
move-right ∥ *scale-wall*)
=
prfun-of-rvfun (
$1/18 * [\![bel^> = 0]\!]_{\mathcal{I}e}$ +
$8/9 * [\![bel^> = 1]\!]_{\mathcal{I}e}$ +
$1/18 * [\![bel^> = 2]\!]_{\mathcal{I}e}$
$)_e$
apply (*simp add*: *robot-localisation-def*)
apply (*simp add*: *move-right-2-simp believe-3-def*)
apply (*simp add*: *scale-wall-def door-def pfun-defs*)

apply (*simp add*: *move-right-2-dist*)
apply (*simp add*: *move-right-2-def dist-defs*)
apply (*expr-simp-1*)
apply (*rule HOL.arg-cong*[**where** *f=prfun-of-rvfun*])
apply (*simp add*: *ring-distribs*(*2*))
apply (*subst fun-eq-iff*, *rule allI*)
apply (*auto*)
by (*simp add*: *believe-3-sum*)+

lemma *robot-localisation'*:
$((((init \parallel scale\text{-}door) \; ; \; move\text{-}right) \parallel scale\text{-}door) \; ; \; move\text{-}right) \parallel scale\text{-}wall$
$= prfun\text{-}of\text{-}rvfun \; (1/18 * [\![bel^{>} = 0]\!]_{\mathcal{I}e} + 8/9 * [\![bel^{>} = 1]\!]_{\mathcal{I}e} + 1/18 * [\![bel^{>} = 2]\!]_{\mathcal{I}e})_e$
using *believe-3-def believe-3-simp robot-localisation-def* **by** *presburger*
end

4 (Parametric) Coin flip

theory *utp-prob-rel-lattice-coin*
imports
UTP-prob-relations.utp-prob-rel
begin

unbundle *UTP-Syntax*

declare [[*show-types*]]

4.1 Single coin flip without time

datatype *Tcoin* = *chead* | *ctail*
thm *Tcoin.exhaust*

alphabet *cstate* =
c :: *Tcoin*

definition *cflip*:: *cstate prhfun* **where**
$cflip = if_p \; 0.5 \; then \; (c := chead) \; else \; (c := ctail)$

definition *cflip-loop* **where**
$cflip\text{-}loop = while_p \; (c^{<} = ctail)_e \; do \; cflip \; od$

definition *cH* :: *cstate rvhfun* **where**
$cH = ([\![c^{>} = chead]\!]_{\mathcal{I}e})_e$

definition *cH'*:: *cstate rvhfun* **where**
$cH' = ([\![c^{<} = chead]\!]_{\mathcal{I}e} * ([\![c^{>} = chead]\!]_{\mathcal{I}e}) + [\![\neg c^{<} = chead]\!]_{\mathcal{I}e} * [\![c^{>} = chead]\!]_{\mathcal{I}e})_e$

lemma $cH = cH'$
apply (*simp add*: *cH-def cH'-def*)
by (*expr-auto*)

lemma *r-simp*: $rvfun\text{-}of\text{-}prfun \; [\lambda s{::}cstate \times cstate. \; p]_e = (\lambda s. \; ureal2real \; p)$
by (*simp add*: *SEXP-def rvfun-of-prfun-def*)

lemma *cflip-is-dist*: *is-final-distribution* (*rvfun-of-prfun cflip*)
apply (*simp add*: *cflip-def pfun-defs*)

apply (*subst rvfun-assignment-inverse*)+
apply (*simp add*: *r-simp*)
apply (*subst rvfun-pchoice-inverse-c*)
apply (*simp add*: *rvfun-assignment-is-prob*)+
using *rvfun-pchoice-is-dist'*
using *rvfun-assignment-is-dist* **by** *fastforce*

lemma *cflip-altdef*: *rvfun-of-prfun cflip* = $([\![\bigsqcup\ v \in \{ctail,\ chead\}.\ c := «v»]\!]_{\mathcal{I}e}\ /\ 2)_e$
apply (*simp add*: *cflip-def pfun-defs*)
apply (*subst rvfun-assignment-inverse*)+
apply (*simp add*: *r-simp*)
apply (*subst rvfun-pchoice-inverse-c*)
apply (*simp add*: *rvfun-assignment-is-prob*)+
apply (*pred-auto*)
by (*simp add*: *ereal2ureal-def real2uereal-inverse' ureal2real-def*)+

lemma *cstate-UNIV-set*: $(UNIV::\mathbb{P}\ cstate) = \{(\!|c_v = chead|\!),\ (\!|c_v = ctail|\!)\}$
apply (*auto*)
by (*metis Tcoin.exhaust cstate.cases*)

lemma *cstate-head*: $\{s::cstate.\ c_v\ s = chead\} = \{(\!|c_v = chead|\!)\}$
apply (*subst set-eq-iff*)
by (*auto*)

lemma *cstate-tail*: $\{s::cstate.\ c_v\ s = ctail\} = \{(\!|c_v = ctail|\!)\}$
apply (*subst set-eq-iff*)
by (*auto*)

lemma *cstate-rel-UNIV-set*:
$\{s::cstate \times cstate.\ True\} = \{((\!|c_v = chead|\!),\ (\!|c_v = chead|\!)),$
$((\!|c_v = chead|\!),\ (\!|c_v = ctail|\!)),\ ((\!|c_v = ctail|\!),\ (\!|c_v = ctail|\!)),\ ((\!|c_v = ctail|\!),\ (\!|c_v = chead|\!))\}$
apply (*simp*)
apply (*subst set-eq-iff*)
apply (*rule allI*)
apply (*rule iffI*)
apply (*clarify*)
using *cstate-UNIV-set* **apply** *blast*
apply (*clarify*)
by *blast*

lemma *ureal2real-1-2*: $ureal2real\ (ereal2ureal\ (ereal\ (1::\mathbb{R})))\ /\ (2::\mathbb{R}) = (1::\mathbb{R})\ /\ (2::\mathbb{R})$
apply (*simp add*: *ureal-defs*)
using *real-1* **by** *presburger*

lemma *sum-1-2*: $(sum\ ((\hat{}\,)\ ((1::\mathbb{R})\ /\ (2::\mathbb{R})))\ \{Suc\ (0::\mathbb{N})..n\} +$
$((1::\mathbb{R})\ /\ (2::\mathbb{R}))\ \hat{}\ n\ /\ (2::\mathbb{R})) =$
$(sum\ ((\hat{}\,)\ ((1::\mathbb{R})\ /\ (2::\mathbb{R})))\ \{Suc\ (0::\mathbb{N})..n+1\})$
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1 add-is-0 less-Suc0 one-neq-zero one-power2 power-Suc power-add power-one-over sum.cl-ivl-Suc times-divide-eq-left times-divide-eq-right*)

lemma *sum-geometric-series*:

$(sum\ ((\hat{}\,)\ ((1::\mathbb{R})\ /\ (2::\mathbb{R})))\ \{Suc\ (0::\mathbb{N})..n + (1::\mathbb{N})\}) = 1 - 1\ /\ 2\ \hat{}\,(n+1)$
apply (*induction n*)
apply (*simp*)

by (*simp add*: *power-one-over sum-gp*)

lemma *sum-geometric-series-1*:
(sum ((^) ((*1*::ℝ) / (*2*::ℝ))) {*1*..*n* + (*1*::ℕ)}) = *1* − *1* / *2* ^(*n*+*1*)
apply (*induction n*)
apply (*simp*)
using *One-nat-def sum-geometric-series* **by** *presburger*

lemma *sum-geometric-series'*:
(sum ((^) ((*1*::ℝ) / (*2*::ℝ))) {*Suc* (*0*::ℕ)..*n*}) = *1* − *1* / *2* ^(*n*)
apply (*induction n*)
apply (*simp*)
by (*simp add*: *power-one-over sum-gp*)

lemma *sum-geometric-series-ureal*:
ureal2real (*ereal2ureal* (*ereal* (*sum* ((^) ((*1*::ℝ) / (*2*::ℝ))) {*Suc* (*0*::ℕ)..*n* + (*1*::ℕ)}))) / (*2*::ℝ)
= (*1* − *1* / *2* ^(*n*+*1*))/*2*
apply (*subst sum-geometric-series*)
apply (*simp add*: *ureal-defs*)
apply (*subst real2uereal-inverse*)
using *max.cobounded1* **apply** *blast*
apply *simp*
apply (*simp add*: *max-def*)
by (*smt* (*z3*) *one-le-power*)

lemma *iterate-cflip-bottom-simp*:
shows $iter_p$ *0* ($c^<$ = *ctail*)$_e$ *cflip* 0_p = 0_p
$iter_p$ (*Suc 0*) ($c^<$ = *ctail*)$_e$ *cflip* 0_p = (⟦\$$c^<$ = *chead* ∧ \$$c^>$ = *chead*⟧$_{\mathcal{I}e}$)
$iter_p$ (*n*+*2*) ($c^<$ = *ctail*)$_e$ *cflip* 0_p =
(⟦\$$c^<$ = *chead* ∧ \$$c^>$ = *chead*⟧$_{\mathcal{I}e}$ +
⟦\$$c^<$ = *ctail* ∧ \$$c^>$ = *chead*⟧$_{\mathcal{I}e}$ ∗ ($\sum$ *i*∈{*1*..«*n*+*1*»}. (*1*/*2*)^*i*))$_e$
apply (*auto*)
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *prfun-zero-right'*)
apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-skip-inverse*)
apply (*subst ureal-zero*)
apply (*simp add*: *ureal-defs*)
apply (*subst fun-eq-iff*)
apply (*pred-auto*)
apply (*meson Tcoin.exhaust*)
apply (*induct-tac n*)
apply (*simp*)
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *prfun-zero-right'*)
apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-skip-inverse*)+
apply (*subst ureal-zero*)
apply (*subst rvfun-pcond-inverse*)
apply (*metis ureal-is-prob ureal-zero*)
apply (*simp add*: *rvfun-skip-f-is-prob*)
apply (*subst cflip-altdef*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)

apply (*simp add*: *infsum-nonneg iverson-bracket-def*)
apply (*pred-auto*)
apply (*simp add*: *cstate-UNIV-set*)
apply (*smt* (*verit*, *ccfv-SIG*) *prfun-in-0-1′ rvfun-skip-inverse*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*simp only*: *skip-def*)
apply (*expr-auto add*: *assigns-r-def*)
apply (*simp add*: *real2ureal-def*)
apply (*smt* (*verit*, *best*) *SEXP-def case-prod-conv cstate.select-convs*(*1*) *cstate.surjective div-0 infsum-0 mult-cancel-right1 real2ureal-def rvfun-skip_f-simp skip-def snd-conv*)
apply (*meson Tcoin.exhaust*)
apply (*simp add*: *cstate-UNIV-set*)
apply (*pred-auto*)
apply (*simp add*: *real2ureal-def*)
using *real2ureal-def* **apply** *blast+*
apply (*simp add*: *cstate-UNIV-set*)
apply (*pred-auto*)
using *real2ureal-def* **apply** *blast+*
apply (*simp add*: *cstate-UNIV-set*)
apply (*pred-auto*)
using *real2ureal-def* **apply** *blast+*

apply (*simp*)
apply (*subst loopfunc-def*)
apply (*subst pseqcomp-def*)
apply (*subst pcond-def*)
apply (*subst cflip-altdef*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*expr-auto*)
apply (*simp add*: *infsum-nonneg prfun-in-0-1′*)
apply (*pred-auto*)
apply (*simp add*: *cstate-UNIV-set*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*auto*)
apply (*smt* (*verit*, *best*) *field-sum-of-halves ureal-upper-bound*)
using *ureal-upper-bound* **apply** *blast*
apply (*subst prfun-of-rvfun-def*)
apply (*subst rvfun-of-prfun-def*)+
apply (*expr-auto*)
apply (*simp add*: *cstate-UNIV-set*)
apply (*pred-auto*)
defer
apply (*subst prfun-skip-id*)
apply (*simp add*: *one-ureal.rep-eq real2ureal-def ureal2real-def*)
using *Tcoin.exhaust* **apply** *blast*
apply (*metis* (*full-types*) *Tcoin.exhaust cstate.select-convs*(*1*) *ereal-real o-def prfun-skip-not-id real2ureal-def ureal2real-def zero-ereal-def zero-ureal.rep-eq*)
apply (*subst infsum-0*)
apply (*subst ureal-defs*)
apply (*smt* (*verit*, *best*) *divide-eq-0-iff ereal-max min.absorb2 min.commute mult-eq-0-iff o-apply real-of-ereal-0 ureal2ereal-inverse ureal2real-def zero-ereal-def zero-less-one-ereal zero-ureal.rep-eq*)
using *real2ureal-def* **apply** *presburger*
using *Tcoin.exhaust* **apply** *blast*
apply (*subst infsum-0*)

apply (*subst ureal-defs*)
apply (*smt* (*verit*, *best*) *divide-eq-0-iff ereal-max min.absorb2 min.commute mult-eq-0-iff o-apply real-of-ereal-0 ureal2ereal-inverse ureal2real-def zero-ereal-def zero-less-one-ereal zero-ureal.rep-eq*)
using *real2ureal-def* **apply** *blast*
apply (*metis* (*full-types*) *Tcoin.exhaust cstate.ext-inject o-def prfun-skip-not-id real2ureal-def real-of-ereal-0 ureal2real-def zero-ureal.rep-eq*)
apply (*subst ureal2real-1-2*)
apply (*subst sum-1-2*)
apply (*subst sum-geometric-series-ureal*)
apply (*subst sum-geometric-series'*)
apply (*subst ureal-defs*)+
proof −
fix n
have *f1*: $((1::\mathbb{R}) / (2::\mathbb{R}) + ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} (n + (1::\mathbb{N}))) / (2::\mathbb{R})) = ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} (n + 2))$
by (*simp add*: *add.assoc diff-divide-distrib*)
have *f2*: $((3::\mathbb{R}) * ((1::\mathbb{R}) / (2::\mathbb{R})) \hat{} n / (4::\mathbb{R}) + ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} n)) = ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} (n+2))$
apply (*auto*)
by (*simp add*: *power-one-over*)
show *ereal2ureal'* $(min (max (0::ereal) (ereal ((1::\mathbb{R}) / (2::\mathbb{R}) + ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} (n + (1::\mathbb{N}))) / (2::\mathbb{R})))) (1::ereal)) =$
ereal2ureal' $(min (max (0::ereal) (ereal ((3::\mathbb{R}) * ((1::\mathbb{R}) / (2::\mathbb{R})) \hat{} n / (4::\mathbb{R}) + ((1::\mathbb{R}) - (1::\mathbb{R}) / (2::\mathbb{R}) \hat{} n)))) (1::ereal))$
using *f1 f2* **by** *presburger*
qed

lemma *cflip-drop-initial-segments-eq*:
$(\bigsqcup n::\mathbb{N}.\ iter_p\ (n+2)\ (c^< = ctail)_e\ cflip\ 0_p) = (\bigsqcup n::\mathbb{N}.\ iter_p\ (n)\ (c^< = ctail)_e\ cflip\ 0_p)$
apply (*rule increasing-chain-sup-subset-eq*)
apply (*rule iterate-increasing-chain*)
by (*simp add*: *cflip-is-dist*)

lemma *cflip-iterate-limit-sup*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (c^< = ctail)_e\ cflip\ 0_p))$
shows $(\lambda n.\ ureal2real\ (f\ n\ s)) \longrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$
apply (*simp only*: *assms*)
apply (*subst LIMSEQ-ignore-initial-segment*[**where** $k = 2$])
apply (*subst increasing-chain-sup-subset-eq*[**where** $m = 2$])
apply (*rule increasing-chain-fun*)
apply (*rule iterate-increasing-chain*)
apply (*simp add*: *cflip-is-dist*)
apply (*subst increasing-chain-limit-is-lub'*)
apply (*simp add*: *increasing-chain-def*)
apply (*auto*)
apply (*rule le-funI*)
by (*smt* (*verit*, *ccfv-threshold*) *cflip-is-dist iterate-increasing2 le-fun-def*)

lemma *fa*: $(\lambda n::\mathbb{N}.\ ureal2real\ (ereal2ureal\ (ereal\ ((1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{} n)))))$
$= (\lambda n::\mathbb{N}.\ ((1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{} n)))$
apply (*subst fun-eq-iff*)
apply (*auto*)
apply (*simp add*: *ureal-defs*)
apply (*subst real2uereal-inverse*)
apply (*meson max.cobounded1*)

apply *simp*
proof −
fix x
have *f1*: $(max\ (0::ereal)\ (ereal\ ((1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{}\ x)))) =$
$(ereal\ ((1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{}\ x)))$
apply (*simp add*: *max-def*)
by (*smt* (*z3*) *one-le-power*)
show *real-of-ereal* $(max\ (0::ereal)\ (ereal\ ((1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{}\ x)))) =$
$(1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{}\ x)$
by (*simp add*: *f1*)
qed

lemma *fb*:
$(\lambda n::\mathbb{N}.\ (1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R}) \hat{}\ n)) \longrightarrow (1::\mathbb{R})$
proof −
have *f0*: $(\lambda n::\mathbb{N}.\ ((1::\mathbb{R}) - ((1::\mathbb{R}) / (2::\mathbb{R})) \hat{}\ (n+1))) = (\lambda n::\mathbb{N}.\ (1::\mathbb{R}) - (1::\mathbb{R}) / ((2::\mathbb{R}) * (2::\mathbb{R})$
$\hat{}\ n))$
apply (*subst fun-eq-iff*)
apply (*auto*)
using *power-one-over* **by** *blast*
have *f1*: $(\lambda n::\mathbb{N}.\ ((1::\mathbb{R}) - ((1::\mathbb{R}) / (2::\mathbb{R})) \hat{}\ (n+1))) \longrightarrow (1 - 0)$
apply (*rule tendsto-diff*)
apply (*auto*)
apply (*rule LIMSEQ-power-zero*)
by *simp*
show *?thesis*
using *f0 f1* **by** *auto*
qed

lemma *cflip-iterate-limit-cH*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (c^< = ctail)_e\ cflip\ 0_p))$
shows $(\lambda n.\ ureal2real\ (f\ n\ s)) \longrightarrow ((\llbracket c^> = chead \rrbracket_{\mathcal{I}e})_e\ s)$
apply (*simp only*: *assms*)
apply (*subst iterate-cflip-bottom-simp*(*3*))
apply (*subst sum-geometric-series-1*)
apply (*pred-auto*)
apply (*simp add*: *fa*)
apply (*simp add*: *fb*)
apply (*metis LIMSEQ-const-iff nle-le real2ureal-def ureal-lower-bound ureal-real2ureal-smaller*)
apply (*metis comp-def one-ereal-def one-ureal.rep-eq one-ureal-def real-ereal-1 tendsto-const ureal2real-def*)
apply (*metis LIMSEQ-const-iff nle-le real2ureal-def ureal-lower-bound ureal-real2ureal-smaller*)
by (*meson Tcoin.exhaust*)+

lemma *fh*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (c^< = ctail)_e\ cflip\ 0_p))$
shows $((\llbracket c^> = chead \rrbracket_{\mathcal{I}e})_e\ s) = (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$
apply (*subst LIMSEQ-unique*[**where** $X = (\lambda n.\ ureal2real\ (f\ n\ s))$ **and** $a = ((\llbracket c^> = chead \rrbracket_{\mathcal{I}e})_e\ s)$
and
$b = (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$])
using *cflip-iterate-limit-cH* **apply** (*simp add*: *assms*)
using *cflip-iterate-limit-sup* **apply** (*simp add*: *assms*)
by *auto*

lemma *fi*: $(\bigsqcup n::\mathbb{N}.\ iter_p\ (n+2)\ (c^< = ctail)_e\ cflip\ 0_p) =$
$(\lambda x::cstate \times cstate.\ ereal2ureal\ (ereal\ ((\llbracket c^> = chead \rrbracket_{\mathcal{I}e})_e\ x)))$

apply (*simp only*: *fh*)
apply (*simp add*: *ureal2rereal-inverse*)
using *SUP-apply* **by** *fastforce*

lemma *coin-flip-loop*: *cflip-loop* = *prfun-of-rvfun cH*
apply (*simp add*: *cflip-loop-def cH-def prfun-of-rvfun-def real2ureal-def*)
apply (*subst sup-continuous-lfp-iteration*)
apply (*simp add*: *cflip-is-dist*)
apply (*rule finite-subset*[**where** $B = \{s::cstate \times cstate.\ True\}$])
apply *force*
apply (*metis cstate-rel-UNIV-set finite.emptyI finite.insertI*)
apply (*simp only*: *cflip-drop-initial-segments-eq*[*symmetric*])
apply (*simp only*: *fi*)
by *auto*

4.1.1 Using unique fixed point theorem

lemma *cstate-set-simp*: $\{s::cstate.\ s = (\!|c_v = ctail|\!) \lor s = (\!|c_v = chead|\!)\} = \{(\!|c_v = chead|\!), (\!|c_v = ctail|\!)\}$
by *fastforce*

lemma *cflip-iterdiff-simp*:
shows (*iterdiff 0* $(c^< = ctail)_e$ *cflip* 1_p) = 1_p
(*iterdiff* $(n+1)$ $(c^< = ctail)_e$ *cflip* 1_p) = *prfun-of-rvfun* $((\llbracket c^< = ctail\rrbracket_{\mathcal{I}e} * (1/2)\hat{}«n»)_e)$
proof −
show (*iterdiff 0* $(c^< = ctail)_e$ *cflip* 1_p) = 1_p
by (*auto*)

show (*iterdiff* $(n+1)$ $(c^< = ctail)_e$ *cflip* 1_p) = *prfun-of-rvfun* $((\llbracket c^< = ctail\rrbracket_{\mathcal{I}e} * (1/2)\hat{}«n»)_e)$
apply (*induction n*)
apply (*simp add*: *pfun-defs*)
apply (*subst cflip-altdef*)
apply (*subst ureal-zero*)
apply (*subst ureal-one*)
apply (*subst rvfun-seqcomp-inverse*)
using *cflip-altdef cflip-is-dist* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*metis ureal-is-prob ureal-one*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto add*: *rel assigns-r-def*)
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*simp*)
apply (*subst infsum-constant-finite-states*)
apply (*simp*)
apply (*simp only*: *cstate-set-simp*)
apply (*simp add*: *real2ureal-def*)
apply (*simp only*: *add-Suc*)
apply (*simp only*: *iterdiff.simps*(*2*))
apply (*simp only*: *pcond-def*)
apply (*simp only*: *pseqcomp-def*)
apply (*subst rvfun-seqcomp-inverse*)
using *cflip-altdef cflip-is-dist* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*subst rvfun-inverse*)

apply (*expr-auto add*: *dist-defs*)
apply (*simp add*: *power-le-one*)
apply (*subst cflip-altdef*)
apply (*expr-auto add*: *rel assigns-r-def*)
defer
apply (*simp add*: *pfun-defs*)
apply (*subst ureal-zero*)
apply *simp*
proof −
fix n
let $?lhs = (\sum_\infty v_0{::}cstate.$
$(if\ v_0 = (\!|c_v = ctail|\!) \lor v_0 = (\!|c_v = chead|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
$((if\ c_v\ v_0 = ctail\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) * ((1{::}\mathbb{R}) / (2{::}\mathbb{R}))\ \widehat{}\ n) /$
$(2{::}\mathbb{R}))$
have $?lhs = (\sum_\infty v_0{::}cstate.$
$(if\ (\!|c_v = ctail|\!) = v_0\ then\ ((1{::}\mathbb{R}) / (2{::}\mathbb{R}))\ \widehat{}\ n / 2\ else\ (0{::}\mathbb{R})))$
apply (*rule infsum-cong*)
by *auto*
also have ... $= (((1{::}\mathbb{R}) / (2{::}\mathbb{R}))\ \widehat{}\ n / (2{::}\mathbb{R}))$
apply (*subst infsum-constant-finite-states*)
apply (*simp*)
by *simp*
then show $real2ureal\ ?lhs = real2ureal\ (((1{::}\mathbb{R}) / (2{::}\mathbb{R}))\ \widehat{}\ n / (2{::}\mathbb{R}))$
using *calculation* **by** *presburger*
qed
qed

lemma *cflip-iterdiff-tendsto-0*:
$\forall s{::}cstate \times cstate.\ (\lambda n{::}\mathbb{N}.\ ureal2real\ (iterdiff\ n\ (c^< = ctail)_e\ cflip\ 1_p\ s)) \longrightarrow (0{::}\mathbb{R})$
proof
fix s
have $(\lambda n{::}\mathbb{N}.\ ureal2real\ (iterdiff\ (n{+}1)\ (c^< = ctail)_e\ cflip\ 1_p\ s)) \longrightarrow (0{::}\mathbb{R})$
apply (*subst cflip-iterdiff-simp*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto*)
apply (*subst real2ureal-inverse*)
apply (*simp*)
apply (*simp add*: *power-le-one*)
apply (*simp add*: *LIMSEQ-realpow-zero*)
apply (*subst real2ureal-inverse*)
by (*simp*)+
then show $(\lambda n{::}\mathbb{N}.\ ureal2real\ (iterdiff\ n\ (c^< = ctail)_e\ cflip\ 1_p\ s)) \longrightarrow (0{::}\mathbb{R})$
by (*rule LIMSEQ-offset*[**where** $k = 1$])
qed

lemma *cH-is-fp*: $\mathcal{F}\ (c^< = ctail)_e\ cflip\ (prfun\text{-}of\text{-}rvfun\ cH) = prfun\text{-}of\text{-}rvfun\ cH$
apply (*simp add*: *cH-def loopfunc-def*)
apply (*simp add*: *pfun-defs*)
apply (*subst cflip-altdef*)
apply (*subst rvfun-skip-inverse*)
apply (*subst rvfun-seqcomp-inverse*)
using *cflip-altdef cflip-is-dist* **apply** *presburger*
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*subst rvfun-inverse*)

apply (*expr-auto add*: *dist-defs*)
apply (*expr-auto add*: *prfun-of-rvfun-def skip-def*)
using *Tcoin.exhaust* **apply** *blast*
apply (*pred-auto*)
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*simp*)
apply (*subst infsum-constant-finite-states*)
apply (*simp*)
apply (*smt* (*verit*, *del-insts*) *Collect-cong One-nat-def Suc-1 Tcoin.distinct*(*1*) *UNIV-def card.empty card.insert cstate.ext-inject cstate-UNIV-set dbl-simps*(*3*) *dbl-simps*(*5*) *empty-iff finite.emptyI finite.insertI insert-iff mem-Collect-eq mult-numeral-1-right nonzero-mult-div-cancel-left numeral-One of-nat-1 of-nat-mult of-nat-numeral*)
using *Tcoin.exhaust* **by** *blast*

lemma *coin-flip-loop′*: *cflip-loop* = *prfun-of-rvfun cH*
apply (*simp add*: *cflip-loop-def*)
apply (*subst unique-fixed-point-lfp-gfp′*[**where** *fp* = *prfun-of-rvfun cH*])
using *cflip-is-dist* **apply** *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *Collect-mono-iff cstate-rel-UNIV-set finite.emptyI finite-insert rev-finite-subset*)
using *cflip-iterdiff-tendsto-0* **apply** (*simp*)
using *cH-is-fp* **apply** *blast*
by *simp*

4.1.2 Termination

The probability of c' being *head* is 1, and so almost-sure termination.

lemma *coin-flip-termination-prob*: $cH \ ; \ [\![c^< = chead]\!]_{\mathcal{I}e} = (1)_e$
apply (*simp add*: *cH-def*)
apply (*expr-auto*)
proof −
let $?lhs\text{-}f = \lambda v_0.\ (if\ c_v\ v_0 = chead\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs = (\sum_\infty v_0{::}cstate.\ ?lhs\text{-}f\ v_0 * ?lhs\text{-}f\ v_0\ )$
have $?lhs = (\sum_\infty v_0{::}cstate.\ ?lhs\text{-}f\ v_0)$
apply (*rule infsum-cong*)
by (*auto*)
also have ... = *1*
apply (*subst infsum-constant-finite-states*)
apply (*metis cstate-UNIV-set finite.emptyI finite.insertI rev-finite-subset top-greatest*)
by (*simp add*: *cstate-head*)
then show $?lhs = (1{::}\mathbb{R})$
using *calculation* **by** *presburger*
qed

The probability of c' not being *head* is 0, and so impossible for non-termination.

lemma *coin-flip-nontermination-prob*: $cH \ ; \ [\![\neg c^< = chead]\!]_{\mathcal{I}e} = (0)_e$
apply (*simp add*: *cH-def*)
apply (*expr-auto*)
proof −
let $?lhs\text{-}t = \lambda v_0.\ (if\ c_v\ v_0 = chead\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs\text{-}f = \lambda v_0.\ (if\ \neg c_v\ v_0 = chead\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
let $?lhs = (\sum_\infty v_0{::}cstate.\ ?lhs\text{-}t\ v_0 * ?lhs\text{-}f\ v_0\ )$
have $?lhs = (\sum_\infty v_0{::}cstate.\ 0)$
apply (*rule infsum-cong*)
by (*auto*)

then show *?lhs* = (*0*::ℝ)
by *force*
qed

4.2 Single coin flip (variable probability)

definition *cpflip* :: *ureal* ⇒ *cstate prhfun* **where**
cpflip p = *if*$_p$ «*p*» *then* (*c* := *chead*) *else* (*c* := *ctail*)

definition *cpflip-loop* :: *ureal* ⇒ *cstate prhfun* **where**
cpflip-loop p = *while*$_p$ ($c^<$ = *ctail*)$_e$ *do cpflip p od*

definition *cpH* :: *ureal* ⇒ *cstate rvhfun* **where**
cpH p = ([[$c^>$ = *chead*]]$_{\mathcal{I}e}$)$_e$

definition *cpH'*:: *ureal* ⇒ *cstate rvhfun* **where**
cpH' p = ([[$c^<$ = *chead*]]$_{\mathcal{I}e}$ ∗ ([[$c^>$ = *chead*]]$_{\mathcal{I}e}$) + [[¬$c^<$ = *chead*]]$_{\mathcal{I}e}$ ∗ [[$c^>$ = *chead*]]$_{\mathcal{I}e}$)$_e$

lemma *cpH p* = *cpH' p*
apply (*simp add*: *cpH-def cpH'-def*)
by (*expr-auto*)

lemma *cpflip-is-dist*: *is-final-distribution* (*rvfun-of-prfun* (*cpflip p*))
apply (*simp add*: *cpflip-def pfun-defs*)
apply (*subst rvfun-assignment-inverse*)+
apply (*simp add*: *r-simp*)
apply (*subst rvfun-pchoice-inverse-c*)
apply (*simp add*: *rvfun-assignment-is-prob*)+
apply (*subst rvfun-pchoice-is-dist'*)
by (*simp add*: *rvfun-assignment-is-dist*)+

lemma *cpflip-altdef*: *rvfun-of-prfun* (*cpflip p*) =
([[$c^>$ = *chead*]]$_{\mathcal{I}e}$ ∗ (*ureal2real* «*p*») + [[$c^>$ = *ctail*]]$_{\mathcal{I}e}$ ∗ (*ureal2real* (*1* − «*p*»)))$_e$
apply (*simp add*: *cpflip-def pfun-defs*)
apply (*subst rvfun-assignment-inverse*)+
apply (*simp add*: *r-simp*)
apply (*subst rvfun-pchoice-inverse-c*)
apply (*simp add*: *rvfun-assignment-is-prob*)+
apply (*pred-auto*)
by (*simp add*: *ureal-1-minus-real*)

lemma *cpflip-altdef'*: *rvfun-of-prfun* (*cpflip p*) =
([[*c* := *chead*]]$_{\mathcal{I}e}$ ∗ (*ureal2real* «*p*») + [[*c* := *ctail*]]$_{\mathcal{I}e}$ ∗ (*ureal2real* (*1* − «*p*»)))$_e$
apply (*simp add*: *cpflip-def pfun-defs*)
apply (*subst rvfun-assignment-inverse*)+
apply (*simp add*: *r-simp*)
apply (*subst rvfun-pchoice-inverse-c*)
apply (*simp add*: *rvfun-assignment-is-prob*)+
apply (*pred-auto*)
by (*simp add*: *ureal-1-minus-real*)

4.2.1 Using unique fixed point theorem

lemma *cpflip-sum-1*: ($\sum_\infty v_0$::*cstate*. (*if* c_v v_0 = *chead then 1*::ℝ *else* (*0*::ℝ)) ∗ *ureal2real p* +
(*if* c_v v_0 = *ctail then 1*::ℝ *else* (*0*::ℝ)) ∗ *ureal2real* ((*1*::*ureal*) − *p*)) = (*1*::ℝ)
apply (*subst infsum-add*)

apply (*subst summable-on-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*simp add*: *cstate-head*)+
apply (*subst summable-on-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis cstate-UNIV-set finite.emptyI finite-insert rev-finite-subset top-greatest*)
apply (*simp*)
apply (*subst infsum-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*simp add*: *cstate-head*)+
apply (*subst infsum-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*metis cstate-UNIV-set finite.emptyI finite-insert rev-finite-subset top-greatest*)
apply (*subst infsum-constant-finite-states*)
apply (*simp add*: *cstate-head*)+
apply (*subst infsum-constant-finite-states*)
apply (*simp add*: *cstate-tail*)+
using *ureal-1-minus-real* **by** *fastforce*

lemma *cpflip-iterdiff-simp*:
shows $(iterdiff\ 0\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p) = 1_p$
$(iterdiff\ (n+1)\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p) = prfun\text{-}of\text{-}rvfun\ ((⟦c^< = ctail⟧_{\mathcal{I}e} * (ureal2real\ (1 - «p»))\widehat{\ }«n»)_e)$
proof −
show $(iterdiff\ 0\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p) = 1_p$
by (*auto*)

show $(iterdiff\ (n+1)\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p) = prfun\text{-}of\text{-}rvfun\ ((⟦c^< = ctail⟧_{\mathcal{I}e} * (ureal2real\ (1 - «p»))\widehat{\ }«n»)_e)$
apply (*induction n*)
apply (*simp add*: *pfun-defs*)
apply (*subst cpflip-altdef*)
apply (*subst ureal-zero*)
apply (*subst ureal-one*)
apply (*subst rvfun-seqcomp-inverse*)
using *cpflip-altdef cpflip-is-dist* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*metis ureal-is-prob ureal-one*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto add*: *rel*)
using *cpflip-sum-1* **apply** *presburger*

apply (*simp only*: *add-Suc*)
apply (*simp only*: *iterdiff.simps*(*2*))
apply (*simp only*: *pcond-def*)
apply (*simp only*: *pseqcomp-def*)
apply (*subst rvfun-seqcomp-inverse*)
using *cpflip-altdef cpflip-is-dist* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
using *ureal-lower-bound* **apply** *presburger*
apply (*subst power-le-one*)
using *ureal-lower-bound* **apply** *presburger*

using *ureal-upper-bound* **apply** *blast*
apply (*simp*)
apply (*subst cpflip-altdef*)
apply (*expr-auto add*: *rel*)
defer
apply (*simp add*: *pfun-defs*)
apply (*subst ureal-zero*)
apply *simp*
proof −
fix *n*
let *?lhs* = $(\sum_\infty v_0::cstate.$
$((if\ c_v\ v_0 = chead\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ureal2real\ p +$
$(if\ c_v\ v_0 = ctail\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ureal2real\ ((1::ureal) - p)) *$
$((if\ c_v\ v_0 = ctail\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ureal2real\ ((1::ureal) - p)\ \hat{}\ n))$
have *?lhs* = $(\sum_\infty v_0::cstate.$
$(if\ (\!|c_v = ctail|\!) = v_0\ then\ ureal2real\ ((1::ureal) - p)\ \hat{}\ (n+1)\ else\ (0::\mathbb{R})))$
apply (*rule infsum-cong*)
by *auto*
also have ... = $ureal2real\ ((1::ureal) - p)\ \hat{}\ (n+1)$
apply (*subst infsum-constant-finite-states*)
apply (*simp*)
by *simp*
then show *real2ureal ?lhs* = $real2ureal\ (ureal2real\ ((1::ureal) - p) * ureal2real\ ((1::ureal) - p)\ \hat{}\ n)$
using *calculation* **by** *auto*
qed
qed

lemma *cpflip-iterdiff-tendsto-0*:
assumes $p \neq 0$
shows $\forall s::cstate \times cstate.\ (\lambda n::\mathbb{N}.\ ureal2real\ (iterdiff\ n\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p\ s)) \longrightarrow (0::\mathbb{R})$
proof
fix *s*
have $(\lambda n::\mathbb{N}.\ ureal2real\ (iterdiff\ (n+1)\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p\ s)) \longrightarrow (0::\mathbb{R})$
apply (*subst cpflip-iterdiff-simp*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto*)
apply (*subst real2ureal-inverse*)
apply (*simp add*: *ureal-lower-bound*)
apply (*subst power-le-one*)
using *ureal-lower-bound* **apply** *blast*
using *ureal-upper-bound* **apply** *blast*
apply (*simp*)
apply (*subst LIMSEQ-realpow-zero*)
using *ureal-lower-bound* **apply** *blast*
apply (*smt* (*verit*, *best*) *assms real2eureal-inverse ureal2real-eq ureal-1-minus-real ureal-lower-bound zero-ereal-def zero-ureal-def*)
apply (*simp*)
apply (*subst real2ureal-inverse*)
by (*simp*)+

then show $(\lambda n::\mathbb{N}.\ ureal2real\ (iterdiff\ n\ (c^< = ctail)_e\ (cpflip\ p)\ 1_p\ s)) \longrightarrow (0::\mathbb{R})$
by (*rule LIMSEQ-offset*[**where** *k* = *1*])
qed

lemma *cpH-is-fp*: $\mathcal{F}$ ($c^{<}$ = *ctail*)$_e$ (*cpflip p*) (*prfun-of-rvfun* (*cpH p*)) = *prfun-of-rvfun* (*cpH p*)
apply (*simp add*: *cpH-def loopfunc-def*)
apply (*simp add*: *pfun-defs*)
apply (*subst cpflip-altdef*)
apply (*subst rvfun-skip-inverse*)
apply (*subst rvfun-seqcomp-inverse*)
using *cpflip-altdef cpflip-is-dist* **apply** *presburger*
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*expr-auto add*: *prfun-of-rvfun-def skip-def*)
using *Tcoin.exhaust* **apply** *blast*
using *cpflip-sum-1* **apply** *presburger*
using *Tcoin.exhaust* **by** *blast*

Not surprisingly, as long as p is larger than 0, *cpflip-loop* almost surely terminates.

lemma *cpflip-loop*:
assumes $p \neq 0$
shows *cpflip-loop p* = *prfun-of-rvfun* (*cpH p*)
apply (*simp add*: *cpflip-loop-def*)
apply (*subst unique-fixed-point-lfp-gfp'*[**where** *fp* = *prfun-of-rvfun* (*cpH p*)])
using *cpflip-is-dist* **apply** *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *Collect-mono-iff cstate-rel-UNIV-set finite.emptyI finite-insert rev-finite-subset*)
using *cpflip-iterdiff-tendsto-0* **apply** (*simp add*: *assms*)
using *cpH-is-fp* **apply** *blast*
by *simp*

end

5 Throw two six-sided dice

This example is from Section 15 of the Hehner's paper "A probability perspective". The invariant of the program for an equal result is $[\![u' = v']\!]_{\mathcal{I}} * [\![t' \geq t+1]\!]_{\mathcal{I}} * (5/6)\hat{\ }(t'-t-1) * (1/6)$. This program cannot guarantee absolute termination (see Section 2.3 of " Abstraction Refinement and Proof for Probabilistic Systems"), but it is almost-certain termination. The probability for non-termination is $[\![u' \neq v']\!]_{\mathcal{I}} * [\![t' \geq t+1]\!]_{\mathcal{I}} * (5/6)\hat{\ }(t'-t)$. When t' tends to ∞, then the probability tends to 0.

theory *utp-prob-rel-lattice-dices*
imports
UTP-prob-relations.utp-prob-rel
begin

unbundle *UTP-Syntax*

declare [[*show-types*]]

5.1 Finite state space

When choosing a right representation for state space, we need to consider the following factors:

- better to be finite, and it would be easier to prove the second assumption of Theorem $[\![$*is-final-distribution* (*rvfun-of-prfun* (*?P*::*?'s* × *?'s* ⇒ *ureal*)); *finite* {*s*::*?'s* × *?'s*.

ureal2real (*iter*$_p$ (*0*::ℕ) (*?b*::*?'s hrel*) *?P* 0_p *s*) < *ureal2real* (⨆*n*::ℕ. *iter*$_p$ *n ?b ?P* 0_p *s*)}⟧ ⟹ *while*$_p$ *?b do ?P od* = (⨆*n*::ℕ. *iter*$_p$ *n ?b ?P* 0_p);

- the outcome should be numbers, and so we can calculate expectation (such as average outcome) directly. We can use enumerations (such as *datatype Tdice* = *d1* | *d2* | *d3* | *d4* | *d5* | *d6*) for outcomes, then associate each with a weight (for example, *d1* to 1 etc.). But this is an indirect way to calculate expectations.

5.1.1 Type for outcomes: *Tdice*

typedef *Tdice* = {*1*..(*6*::*nat*)}
morphisms *td2nat nat2td*
  apply (*rule-tac x* = *1* **in** *exI*)
  by *auto*

find-theorems *name*: *Tdice*

We use *Tdice*::*'a* as the type for dice outcome, a type definition for natural numbers between 1 and 6.

abbreviation *outcomes* ≡ {*1*..(*6*::*nat*)}

abbreviation *outcomes1* ≡ {*nat2td 1*, *nat2td 2*, *nat2td 3*, *nat2td 4*, *nat2td 5*, *nat2td 6*}

lemma *Tdice-UNIV-eq*: {*x*::*Tdice*. *True*} = *outcomes1*
  apply (*subst set-eq-iff*, *auto*)
proof −
  fix *x*
  assume *a1*: ¬ *x* = *nat2td* (*Suc* (*0*::ℕ))
  assume *a2*: ¬ *x* = *nat2td* (*2*::ℕ)
  assume *a3*: ¬ *x* = *nat2td* (*3*::ℕ)
  assume *a4*: ¬ *x* = *nat2td* (*4*::ℕ)
  assume *a6*: ¬ *x* = *nat2td* (*6*::ℕ)
  show *x* = *nat2td* (*5*::ℕ)
  proof (*rule ccontr*)
    assume *a5*: ¬ *x* = *nat2td* (*5*::ℕ)
    then have *f1*: *td2nat x* ≠ (*Suc* (*0*)) ∧ *td2nat x* ≠ *2* ∧ *td2nat x* ≠ *3* ∧ *td2nat x* ≠ *4* ∧ *td2nat x* ≠ *5* ∧ *td2nat x* ≠ *6*
      by (*metis a1 a2 a3 a4 a6 td2nat-inverse*)
    also have *f2*: *td2nat x* ∈ *outcomes*
      using *td2nat* **by** *blast*
    from *f1 f2* **show** *False*
      by (*auto*)
  qed
qed

lemma *Tdice-UNIV-finite*: *finite* (*UNIV*::*Tdice set*)
  apply (*simp only*: *UNIV-def*)
  apply (*simp only*: *Tdice-UNIV-eq*)
  by *force*

lemma *outcomes1-card*: *card outcomes1* = *6*
  by (*smt* (*verit*, *best*) *One-nat-def Suc-eq-numeral Suc-numeral Tdice-UNIV-eq atLeastAtMost-iff card.empty card.insert finite.emptyI finite.insertI finite-insert insertE insert-absorb insert-not-empty le-Suc-numeral n-not-Suc-n nat2td-inject numeral-1-eq-Suc-0 numeral-2-eq-2*

numeral-3-eq-3 numeral-eq-iff numeral-eq-one-iff one-le-numeral order.refl plus-1-eq-Suc pred-numeral-simps(*2*) *pred-numeral-simps*(*3*) *semiring-norm*(*8*) *semiring-norm*(*84*) *singletonD*)

lemma *Tdice-card*: *card* (*UNIV*::*Tdice set*) = *6*
apply (*simp only*: *UNIV-def*)
apply (*simp only*: *Tdice-UNIV-eq*)
by (*rule outcomes1-card*)

lemma *Tdice-mem*: (*a*::*Tdice*) ∈ *outcomes1*
using *Tdice-UNIV-eq* **by** *auto*

lemma *td2nat-in-1-6*: *td2nat* (*a*::*Tdice*) ≤ *6* ∧ *td2nat* (*a*::*Tdice*) ≥ *1*
using *td2nat* **by** *force*

5.1.2 State space

alphabet *fdstate* =
fd1 :: *Tdice*
fd2 :: *Tdice*

find-theorems *name*: *fdstate*

abbreviation *fd1-pred* :: *fdstate* ⇒ $\mathbb{B}$ **where**
fd1-pred s ≡ ($fd1_v$ *s* = *nat2td* (*Suc* (*0*::$\mathbb{N}$)) ∨ $fd1_v$ *s* = *nat2td* (*2*::$\mathbb{N}$) ∨ $fd1_v$ *s* = *nat2td* (*3*::$\mathbb{N}$) ∨ $fd1_v$ *s* = *nat2td* (*4*::$\mathbb{N}$) ∨ $fd1_v$ *s* = *nat2td* (*5*::$\mathbb{N}$) ∨ $fd1_v$ *s* = *nat2td* (*6*::$\mathbb{N}$))

abbreviation *fd2-pred* :: *fdstate* ⇒ $\mathbb{B}$ **where**
fd2-pred s ≡ ($fd2_v$ *s* = *nat2td* (*Suc* (*0*::$\mathbb{N}$)) ∨ $fd2_v$ *s* = *nat2td* (*2*::$\mathbb{N}$) ∨ $fd2_v$ *s* = *nat2td* (*3*::$\mathbb{N}$) ∨ $fd2_v$ *s* = *nat2td* (*4*::$\mathbb{N}$) ∨ $fd2_v$ *s* = *nat2td* (*5*::$\mathbb{N}$) ∨ $fd2_v$ *s* = *nat2td* (*6*::$\mathbb{N}$))

abbreviation *fdstate-set-1* ≡ {⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 2*⦈,
⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 3*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 4*⦈,
⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 6*⦈}
abbreviation *fdstate-set-2* ≡ {⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 2*⦈,
⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 3*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 4*⦈,
⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 6*⦈}

abbreviation *fdstate-set* ≡ {
⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 2*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 3*⦈,
⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 4*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 1*, $fd2_v$ = *nat2td 6*⦈,
⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 2*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 3*⦈,
⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 4*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 2*, $fd2_v$ = *nat2td 6*⦈,
⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 2*⦈, ⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 3*⦈,
⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 4*⦈, ⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 3*, $fd2_v$ = *nat2td 6*⦈,
⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 1*⦈, ⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 2*⦈, ⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 3*⦈,
⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 4*⦈, ⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 5*⦈, ⦇$fd1_v$ = *nat2td 4*, $fd2_v$ = *nat2td 6*⦈,

$(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 1|)$, $(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 2|)$, $(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 3|)$,
$(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 4|)$, $(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 5|)$, $(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 6|)$,
$(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 1|)$, $(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 2|)$, $(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 3|)$,
$(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 4|)$, $(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 5|)$, $(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 6|)$
}

abbreviation *fdstate-set-d1d2-eq* $\equiv \{(|fd1_v = nat2td\ 1,\ fd2_v = nat2td\ 1|)$,
$(|fd1_v = nat2td\ 2,\ fd2_v = nat2td\ 2|)$, $(|fd1_v = nat2td\ 3,\ fd2_v = nat2td\ 3|)$,
$(|fd1_v = nat2td\ 4,\ fd2_v = nat2td\ 4|)$, $(|fd1_v = nat2td\ 5,\ fd2_v = nat2td\ 5|)$,
$(|fd1_v = nat2td\ 6,\ fd2_v = nat2td\ 6|)\}$

lemma *fdstate-set-finite*: *finite fdstate-set*
by *force*

lemma *fd1-mem*: $fd1_v\ x \in outcomes1$
apply (*simp only*: *Tdice-UNIV-eq*[*symmetric*])
by *simp*

lemma *fd2-mem*: $fd2_v\ x \in outcomes1$
apply (*simp only*: *Tdice-UNIV-eq*[*symmetric*])
by *simp*

lemma *fdstate-set-eq*: $\{x::fdstate.\ True\} = fdstate\text{-}set$
apply (*simp*)
apply (*subst set-eq-iff*)
apply (*auto*)
apply (*rule ccontr*)
proof −
fix *x::fdstate*
assume *a1* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (Suc\ (0::\mathbb{N}))|)$
assume *a2* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (2::\mathbb{N})|)$
assume *a3* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (3::\mathbb{N})|)$
assume *a4* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (4::\mathbb{N})|)$
assume *a5* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (5::\mathbb{N})|)$
assume *a6* : $\neg\ x = (|fd1_v = nat2td\ (Suc\ (0::\mathbb{N})),\ fd2_v = nat2td\ (6::\mathbb{N})|)$
assume *a7* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (Suc\ (0::\mathbb{N}))|)$
assume *a8* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (2::\mathbb{N})|)$
assume *a9* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (3::\mathbb{N})|)$
assume *a10* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (4::\mathbb{N})|)$
assume *a11* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (5::\mathbb{N})|)$
assume *a12* : $\neg\ x = (|fd1_v = nat2td\ (2::\mathbb{N}),\ fd2_v = nat2td\ (6::\mathbb{N})|)$
assume *a13* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (Suc\ (0::\mathbb{N}))|)$
assume *a14* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (2::\mathbb{N})|)$
assume *a15* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (3::\mathbb{N})|)$
assume *a16* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (4::\mathbb{N})|)$
assume *a17* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (5::\mathbb{N})|)$
assume *a18* : $\neg\ x = (|fd1_v = nat2td\ (3::\mathbb{N}),\ fd2_v = nat2td\ (6::\mathbb{N})|)$
assume *a19* : $\neg\ x = (|fd1_v = nat2td\ (4::\mathbb{N}),\ fd2_v = nat2td\ (Suc\ (0::\mathbb{N}))|)$
assume *a20* : $\neg\ x = (|fd1_v = nat2td\ (4::\mathbb{N}),\ fd2_v = nat2td\ (2::\mathbb{N})|)$
assume *a21* : $\neg\ x = (|fd1_v = nat2td\ (4::\mathbb{N}),\ fd2_v = nat2td\ (3::\mathbb{N})|)$

assume *a22* : ¬ x = (|$fd1_v$ = $nat2td$ (4::ℕ), $fd2_v$ = $nat2td$ (4::ℕ)|)
assume *a23* : ¬ x = (|$fd1_v$ = $nat2td$ (4::ℕ), $fd2_v$ = $nat2td$ (5::ℕ)|)
assume *a24* : ¬ x = (|$fd1_v$ = $nat2td$ (4::ℕ), $fd2_v$ = $nat2td$ (6::ℕ)|)
assume *a25* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (Suc (0::ℕ))|)
assume *a26* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (2::ℕ)|)
assume *a27* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (3::ℕ)|)
assume *a28* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (4::ℕ)|)
assume *a29* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (5::ℕ)|)
assume *a30* : ¬ x = (|$fd1_v$ = $nat2td$ (5::ℕ), $fd2_v$ = $nat2td$ (6::ℕ)|)
assume *a31* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (Suc (0::ℕ))|)
assume *a32* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (2::ℕ)|)
assume *a33* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (3::ℕ)|)
assume *a34* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (4::ℕ)|)
assume *a35* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (6::ℕ)|)
assume *a36* : ¬ x = (|$fd1_v$ = $nat2td$ (6::ℕ), $fd2_v$ = $nat2td$ (5::ℕ)|)

have *f1*: $fd1_v$ x ∈ (*UNIV*)
  by *simp*

have *f2*: $fd1_v$ x ∉ *outcomes1*
  apply (*auto*)
  using *fd2-mem a1 a2 a3 a4 a5 a6*
    apply (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)
  using *fd2-mem a7 a8 a9 a10 a11 a12*
    apply (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)
  using *fd2-mem a13 a14 a15 a16 a17 a18*
    apply (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)
  using *fd2-mem a19 a20 a21 a22 a23 a24*
    apply (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)
  using *fd2-mem a25 a26 a27 a28 a29 a30*
    apply (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)
  using *fd2-mem a31 a32 a33 a34 a35 a36*
  by (*metis* (*mono-tags*, *lifting*) *One-nat-def fdstate.surjective insert-iff old.unit.exhaust singletonD*)

from *f1 f2* **show** *False*
  using *Tdice-UNIV-eq* **by** *blast*
qed

lemma *fdstate-neq*: (x::*fdstate*) ≠ y ⟷ ($fd1_v$ x ≠ $fd1_v$ y) ∨ ($fd2_v$ x ≠ $fd2_v$ y)
  by (*auto*)

term x <+> y
term *Inl a*
lemma *card* (*fdstate-set-1*) = *6*
  apply (*simp*)
  by (*smt* (*verit*) *Suc-numeral add-cancel-right-right card.empty card-insert-if eval-nat-numeral*(*3*) *fdstate.simps*(*2*) *finite.emptyI finite-insert insertCI insertE insert-absorb numeral-3-eq-3 numeral-eq-iff outcomes1-card plus-1-eq-Suc semiring-norm*(*8*) *singletonD zero-neq-numeral*)

lemma *card-fdstate-set*: *card* (*fdstate-set*) = *36*

proof −
let $?f = \lambda x{::}fdstate.\ 6 * (td2nat\ (fd1_v\ x) - 1) + td2nat\ (fd2_v\ x)$
have *f-inj-on*: *inj-on ?f fdstate-set*
apply (*subst inj-on-def*)
apply (*clarify*)
apply (*rule ccontr*)
proof −
fix $x\ y$
assume *a1*: $x \in fdstate\text{-}set$
assume *a2*: $y \in fdstate\text{-}set$
assume *a3*: $(6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ x) = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ y)$
assume *a4*: $\neg\ x = y$
then have *f1*: $\neg(fd1_v\ x) = (fd1_v\ y) \vee \neg(fd2_v\ x) = (fd2_v\ y)$
by (*simp add*: *fdstate-neq*)
have *f2*: $\neg(fd1_v\ x) = (fd1_v\ y) \Longrightarrow \neg(6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ x) = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ y)$
proof (*cases* $td2nat\ (fd1_v\ x) > td2nat\ (fd1_v\ y)$)
case *True*
then have *f20*: $(6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ x) = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) + (td2nat\ (fd1_v\ x) - td2nat\ (fd1_v\ y)) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ x)$
by *simp*
have *f21*: $... = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) - (1{::}\mathbb{N})) + 6 * (td2nat\ (fd1_v\ x) - td2nat\ (fd1_v\ y)) + td2nat\ (fd2_v\ x)$
using *diff-mult-distrib2 td2nat-in-1-6* **by** *force*
have *f22*: $6 * (td2nat\ (fd1_v\ x) - td2nat\ (fd1_v\ y)) \geq 6$
using *True* **by** *simp*
then have *f23*: $6 * (td2nat\ (fd1_v\ x) - td2nat\ (fd1_v\ y)) + td2nat\ (fd2_v\ x) > 6$
by (*metis diff-add-inverse diff-is-0-eq le-eq-less-or-eq le-zero-eq td2nat-in-1-6 trans-le-add1 zero-neq-one*)
have *f24*: $6 * (td2nat\ (fd1_v\ x) - td2nat\ (fd1_v\ y)) + td2nat\ (fd2_v\ x) \neq td2nat\ (fd2_v\ y)$
using *f23 td2nat-in-1-6* **by** (*metis linorder-not-less*)
then show *?thesis*
using *f21 f20* **by** *linarith*
next
case *False*
assume *a11*: $\neg\ fd1_v\ x = fd1_v\ y$
assume *a12*: $\neg\ td2nat\ (fd1_v\ y) < td2nat\ (fd1_v\ x)$
from *False* **have** $td2nat\ (fd1_v\ y) \geq td2nat\ (fd1_v\ x)$
by *simp*
then have *f0*: $td2nat\ (fd1_v\ y) > td2nat\ (fd1_v\ x)$
using *a11 le-neq-implies-less td2nat-inject* **by** *presburger*
then have *f20*: $(6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ y) = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) + (td2nat\ (fd1_v\ y) - td2nat\ (fd1_v\ x)) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ y)$
by *simp*
have *f21*: $... = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) - (1{::}\mathbb{N})) + 6 * (td2nat\ (fd1_v\ y) - td2nat\ (fd1_v\ x)) + td2nat\ (fd2_v\ y)$
using *diff-mult-distrib2 td2nat-in-1-6* **by** *force*
have *f22*: $6 * (td2nat\ (fd1_v\ y) - td2nat\ (fd1_v\ x)) \geq 6$
using *f0* **by** *simp*
then have *f23*: $6 * (td2nat\ (fd1_v\ y) - td2nat\ (fd1_v\ x)) + td2nat\ (fd2_v\ y) > 6$
by (*metis diff-add-inverse diff-is-0-eq le-eq-less-or-eq le-zero-eq td2nat-in-1-6 trans-le-add1 zero-neq-one*)

have *f24*: $6 * (td2nat\ (fd1_v\ y) - td2nat\ (fd1_v\ x)) + td2nat\ (fd2_v\ y) \neq td2nat\ (fd2_v\ x)$
using *f23 td2nat-in-1-6* **by** (*metis linorder-not-less*)
then show *?thesis*
using *f21 f20* **by** *linarith*
qed
have *f3*: $\neg(fd2_v\ x) = (fd2_v\ y) \Longrightarrow \neg(6{::}\mathbb{N}) * (td2nat\ (fd1_v\ x) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ x) = (6{::}\mathbb{N}) * (td2nat\ (fd1_v\ y) - (1{::}\mathbb{N})) + td2nat\ (fd2_v\ y)$
proof (*cases* $(fd1_v\ x) = (fd1_v\ y)$)
case *True*
then show *?thesis*
using *f1 td2nat-inject* **by** *force*
next
case *False*
then show *?thesis*
using *f2* **by** *blast*
qed
show *False*
using *f1 f2 f3 a3* **by** *blast*
qed

have *inj-set*: *?f ‘ fdstate-set* = $\{(1{::}\mathbb{N})..36\}$
apply (*simp add*: *image-def*)
apply (*simp add*: *nat2td-inverse*)
apply (*auto*)
by *presburger*
have *card-eq*: *card fdstate-set* = *card*(*?f ‘ fdstate-set*)
using *inj-on-iff-eq-card f-inj-on* **by** (*metis* (*no-types*, *lifting*) *fdstate-set-finite*)
have *card-inj-eq*: ... = *card* ($\{(1{::}\mathbb{N})..36\}$)
using *inj-set* **by** *presburger*
have ... = *36*
by *simp*
then show *?thesis*
using *card-eq inj-set* **by** *presburger*
qed

lemma *fdstate-set-d1-d2-eq*: $\{x{::}fdstate.\ fd1_v\ x = fd2_v\ x\}$ = *fdstate-set-d1d2-eq*
apply (*auto*)
by (*smt* (*verit*, *best*) *Tdice-UNIV-eq empty-iff fdstate.cases fdstate.select-convs*(*1*) *fdstate.select-convs*(*2*) *insert-iff mem-Collect-eq numeral-1-eq-Suc-0 one-eq-numeral-iff*)

lemma *fdstate-set-d1d2-eq-card*: *card* $\{x{::}fdstate.\ fd1_v\ x = fd2_v\ x\} = 6$
apply (*simp add*: *fdstate-set-d1-d2-eq*)
by (*smt* (*verit*) *Suc-numeral add-cancel-right-right card.empty card-insert-if eval-nat-numeral*(*3*) *fdstate.simps*(*2*) *finite.emptyI finite-insert insertCI insertE insert-absorb numeral-3-eq-3 numeral-eq-iff outcomes1-card plus-1-eq-Suc semiring-norm*(*8*) *singletonD zero-neq-numeral*)

lemma *fdstate-set-d1d2-eq-card′*: *card fdstate-set-d1d2-eq* = *6*
using *fdstate-set-d1-d2-eq fdstate-set-d1d2-eq-card* **by** *auto*

lemma *fdstate-set-d1d2-neq*: $\{x{::}fdstate.\ \neg fd1_v\ x = fd2_v\ x\} = \{x{::}fdstate.\ True\} - \{x{::}fdstate.\ fd1_v\ x = fd2_v\ x\}$
by *auto*

lemma *fdstate-set-d1d2-neq′*: $\{x{::}fdstate.\ \neg fd1_v\ x = fd2_v\ x\}$ = *fdstate-set* − *fdstate-set-d1d2-eq*
apply (*simp only*: *fdstate-set-d1d2-neq*)

by (*simp only*: *fdstate-set-eq fdstate-set-d1-d2-eq*)

lemma *fdstate-set-d1d2-neq-card*: *card* $\{x::fdstate.\ \neg fd1_v\ x = fd2_v\ x\} = 30$
proof −
have *card* $\{x::fdstate.\ \neg fd1_v\ x = fd2_v\ x\}$ = *card* (*fdstate-set* − *fdstate-set-d1d2-eq*)
by (*simp add*: *fdstate-set-d1d2-neq′*)
also have ... = *card* (*fdstate-set*) − *card* (*fdstate-set-d1d2-eq*)
by (*smt* (*verit*) *One-nat-def UNIV-def card-Diff-subset card-fdstate-set fdstate-set-d1-d2-eq fdstate-set-d1d2-neq fdstate-set-eq fdstate-set-finite finite-subset insert-commute numeral-1-eq-Suc-0 top.extremum*)
also have ... = *30*
apply (*simp only*: *card-fdstate-set fdstate-set-d1-d2-eq*[*symmetric*])
by (*simp only*: *fdstate-set-d1d2-eq-card*)
then show *?thesis*
using *calculation* **by** *presburger*
qed

lemma *fdstate-finite*: *finite* (*UNIV*::*fdstate set*)
apply (*simp only*: *UNIV-def*)
using *fdstate-set-eq fdstate-set-finite* **by** *presburger*

lemma *fdstate-pred-univ*: $\{s::fdstate.\ (fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd1_v\ s = nat2td\ (2::\mathbb{N}) \lor$
$fd1_v\ s = nat2td\ (3::\mathbb{N}) \lor fd1_v\ s = nat2td\ (4::\mathbb{N}) \lor fd1_v\ s = nat2td\ (5::\mathbb{N}) \lor fd1_v\ s = nat2td\ (6::\mathbb{N})) \land$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd2_v\ s = nat2td\ (2::\mathbb{N}) \lor$
$fd2_v\ s = nat2td\ (3::\mathbb{N}) \lor fd2_v\ s = nat2td\ (4::\mathbb{N}) \lor fd2_v\ s = nat2td\ (5::\mathbb{N}) \lor fd2_v\ s = nat2td\ (6::\mathbb{N}))\} = fdstate\text{-}set$
apply (*subst set-eq-iff*)
apply (*rule allI*, *rule iffI*)
using *fdstate-set-eq* **apply** *auto*[*1*]
by *force*

lemma *fdstate-pred-d1d2-neq*: $\{s::fdstate.\ (fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd1_v\ s = nat2td\ (2::\mathbb{N}) \lor$
$fd1_v\ s = nat2td\ (3::\mathbb{N}) \lor fd1_v\ s = nat2td\ (4::\mathbb{N}) \lor fd1_v\ s = nat2td\ (5::\mathbb{N}) \lor fd1_v\ s = nat2td\ (6::\mathbb{N})) \land$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd2_v\ s = nat2td\ (2::\mathbb{N}) \lor$
$fd2_v\ s = nat2td\ (3::\mathbb{N}) \lor fd2_v\ s = nat2td\ (4::\mathbb{N}) \lor fd2_v\ s = nat2td\ (5::\mathbb{N}) \lor fd2_v\ s = nat2td\ (6::\mathbb{N}))$
$\land \neg fd1_v\ s = fd2_v\ s\} =$
$\{s::fdstate.\ \neg fd1_v\ s = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*rule allI*, *rule iffI*)
using *fdstate-set-eq* **apply** *auto*[*1*]
using *fdstate-pred-univ fdstate-set-eq* **by** *auto*

5.1.3 Definitions

definition *fdice-throw*:: *fdstate prhfun* **where**
fdice-throw = *prfun-of-rvfun* (*fd1* $\mathcal{U}$ *outcomes1*) ; *prfun-of-rvfun* (*fd2* $\mathcal{U}$ *outcomes1*)

definition *fdice-throw-loop* **where**
fdice-throw-loop = $while_p\ (fd1^< \neq fd2^<)_e$ *do fdice-throw od*

definition *fH*:: *fdstate rvhfun* **where**
$fH = ((\llbracket fd1^{<} = fd2^{<} \rrbracket_{\mathcal{I}e} * \llbracket fd1^{>} = fd1^{<} \wedge fd2^{>} = fd2^{<} \rrbracket_{\mathcal{I}e}) + \llbracket \neg fd1^{<} = fd2^{<} \rrbracket_{\mathcal{I}e} * \llbracket fd1^{>} = fd2^{>} \rrbracket_{\mathcal{I}e}$
$/ \; 6)_e$

definition *fdice-iterate-n* :: $\mathbb{N} \Rightarrow$ *fdstate prhfun* **where**
fdice-iterate-n $= (\lambda n.\ iter_p\ n\ (fd1^{<} \neq fd2^{<})_e$ *fdice-throw* $0_p)$

5.1.4 Theorems

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

lemma *fd1-uni-is-dist*: *is-final-distribution* (*rvfun-of-prfun* (*prfun-of-rvfun* (*fd1* $\mathcal{U}$ *outcomes1*)))
 apply (*subst rvfun-uniform-dist-is-dist'*)
 apply *blast*
 by *simp+*

lemma *fd2-uni-is-dist*: *is-final-distribution* (*rvfun-of-prfun* (*prfun-of-rvfun* (*fd2* $\mathcal{U}$ *outcomes1*)))
 apply (*subst rvfun-uniform-dist-is-dist'*)
 apply *blast*
 by *simp+*

lemma *fdice-throw-is-dist*: *is-final-distribution* (*rvfun-of-prfun fdice-throw*)
 apply (*simp only*: *fdice-throw-def pseqcomp-def*)
 apply (*subst rvfun-seqcomp-inverse*)
 using *fd1-uni-is-dist* **apply** *blast*
 using *ureal-is-prob* **apply** *blast*
 apply (*subst rvfun-seqcomp-is-dist*)
 using *fd1-uni-is-dist* **apply** *blast*
 using *fd2-uni-is-dist* **by** *blast+*

lemma *fdice-throw-altdef*: *rvfun-of-prfun fdice-throw* $= (\llbracket fd1^{>} \in outcomes1 \rrbracket_{\mathcal{I}e} * \llbracket fd2^{>} \in outcomes1 \rrbracket_{\mathcal{I}e}$
$/ \; 36)_e$
 apply (*simp add*: *fdice-throw-def pseqcomp-def*)
 apply (*subst rvfun-uniform-dist-inverse*)
 apply (*simp*)+
 apply (*subst rvfun-uniform-dist-inverse*)
 apply (*simp*)+
 apply (*subst rvfun-seqcomp-inverse*)
 apply (*simp add*: *rvfun-uniform-dist-is-dist*)
 using *fd2-vwb-lens rvfun-uniform-dist-is-prob* **apply** (*metis finite.emptyI finite.insertI*)
 apply (*subst rvfun-uniform-dist-altdef*)
 apply (*simp*)+
 apply (*subst rvfun-uniform-dist-altdef*)
 apply (*simp*)+
 apply (*expr-simp-1 add*: *rel assigns-r-def*)
 apply (*subst fun-eq-iff*)
 apply (*rule allI*)
proof −
 fix *x*::*fdstate* × *fdstate*
 let *?lhs1-b* $= \lambda v_0.\ v_0 = fst\ x(\!|fd1_v := nat2td\ (Suc\ (0::\mathbb{N}))|\!) \vee$
  $v_0 = fst\ x(\!|fd1_v := nat2td\ (2::\mathbb{N})|\!) \vee$
  $v_0 = fst\ x(\!|fd1_v := nat2td\ (3::\mathbb{N})|\!) \vee$
  $v_0 = fst\ x(\!|fd1_v := nat2td\ (4::\mathbb{N})|\!) \vee$
  $v_0 = fst\ x(\!|fd1_v := nat2td\ (5::\mathbb{N})|\!) \vee$

$v_0 = fst\ x(|fd1_v := nat2td\ (6::\mathbb{N})|)$

let $?lhs1\text{-}b' = \lambda v_0.\ ((fst\ x(|fd1_v := (nat2td\ (Suc\ (0::\mathbb{N})))|) = v_0) \vee$
$(fst\ x(|fd1_v := nat2td\ (2::\mathbb{N})|) = v_0) \vee$
$(fst\ x(|fd1_v := nat2td\ (3::\mathbb{N})|) = v_0) \vee$
$(fst\ x(|fd1_v := nat2td\ (4::\mathbb{N})|) = v_0) \vee$
$(fst\ x(|fd1_v := nat2td\ (5::\mathbb{N})|) = v_0) \vee$
$(fst\ x(|fd1_v := nat2td\ (6::\mathbb{N})|) = v_0))$

let $?lhs1 = \lambda v_0.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$

let $?lhs2\text{-}b = \lambda v_0.\ snd\ x = v_0(|fd2_v := nat2td\ (Suc\ (0::\mathbb{N}))|) \vee$
$snd\ x = v_0(|fd2_v := nat2td\ (2::\mathbb{N})|) \vee$
$snd\ x = v_0(|fd2_v := nat2td\ (3::\mathbb{N})|) \vee$
$snd\ x = v_0(|fd2_v := nat2td\ (4::\mathbb{N})|) \vee$
$snd\ x = v_0(|fd2_v := nat2td\ (5::\mathbb{N})|) \vee$
$snd\ x = v_0(|fd2_v := nat2td\ (6::\mathbb{N})|)$

let $?lhs2\text{-}b' = \lambda v_0.\ v_0(|fd2_v := nat2td\ (Suc\ (0::\mathbb{N}))|) = snd\ x \vee$
$v_0(|fd2_v := nat2td\ (2::\mathbb{N})|) = snd\ x \vee$
$v_0(|fd2_v := nat2td\ (3::\mathbb{N})|) = snd\ x \vee$
$v_0(|fd2_v := nat2td\ (4::\mathbb{N})|) = snd\ x \vee$
$v_0(|fd2_v := nat2td\ (5::\mathbb{N})|) = snd\ x \vee v_0(|fd2_v := nat2td\ (6::\mathbb{N})|) = snd\ x$

let $?lhs2 = \lambda v_0.\ ((if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$

let $?lhs3 = (real\ (card\ \{nat2td\ (Suc\ (0::\mathbb{N})),\ nat2td\ (2::\mathbb{N}),\ nat2td\ (3::\mathbb{N}),\ nat2td\ (4::\mathbb{N}),\ nat2td\ (5::\mathbb{N}),\ nat2td\ (6::\mathbb{N})\}) *$
$real\ (card\ \{nat2td\ (Suc\ (0::\mathbb{N})),\ nat2td\ (2::\mathbb{N}),\ nat2td\ (3::\mathbb{N}),\ nat2td\ (4::\mathbb{N}),\ nat2td\ (5::\mathbb{N}),\ nat2td\ (6::\mathbb{N})\}))$

let $?lhs = (\sum_\infty v_0::fdstate.\ ?lhs1\ v_0 * ?lhs2\ v_0\ /\ ?lhs3)$

have *lhs3-simp*: $?lhs3 = 36$
using *outcomes1-card* **by** *fastforce*

let $?rhs1 = (if\ fd1_v\ (snd\ x) = nat2td\ (Suc\ (0::\mathbb{N})) \vee$
$fd1_v\ (snd\ x) = nat2td\ (2::\mathbb{N}) \vee$
$fd1_v\ (snd\ x) = nat2td\ (3::\mathbb{N}) \vee$
$fd1_v\ (snd\ x) = nat2td\ (4::\mathbb{N}) \vee$
$fd1_v\ (snd\ x) = nat2td\ (5::\mathbb{N}) \vee$
$fd1_v\ (snd\ x) = nat2td\ (6::\mathbb{N})$
$then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$

let $?rhs2 = (if\ fd2_v\ (snd\ x) = nat2td\ (Suc\ (0::\mathbb{N})) \vee$
$fd2_v\ (snd\ x) = nat2td\ (2::\mathbb{N}) \vee$
$fd2_v\ (snd\ x) = nat2td\ (3::\mathbb{N}) \vee$
$fd2_v\ (snd\ x) = nat2td\ (4::\mathbb{N}) \vee$
$fd2_v\ (snd\ x) = nat2td\ (5::\mathbb{N}) \vee$
$fd2_v\ (snd\ x) = nat2td\ (6::\mathbb{N})$
$then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$

let $?rhs = ?rhs1 * ?rhs2\ /\ 36$

have *lhs1-lhs2-simp*: $\forall v_0::fdstate.\ (?lhs1\ v_0 * ?lhs2\ v_0 = (if\ (?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0)\ then\ 1\ else\ 0))$
by (*auto*)

have *lhs1b-lhs2b-simp*: $\forall v_0.\ (?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0) = (v_0 = (|fd1_v = fd1_v\ (snd\ x),\ fd2_v = fd2_v\ (fst\ x)|))$
apply (*rule allI*)
proof −
fix $v_0::fdstate$
have *f1*: $?lhs1\text{-}b\ v_0 \longrightarrow fd2_v\ v_0 = fd2_v\ (fst\ x)$
by *auto*
have *f2*: $?lhs2\text{-}b\ v_0 \longrightarrow fd1_v\ v_0 = fd1_v\ (snd\ x)$

by (*smt* (*verit*, *ccfv-threshold*) *fdstate.ext-inject fdstate.surjective fdstate.update-convs*(*2*))
show (*?lhs1-b* v_0 ∧ *?lhs2-b* v_0) = (v_0 = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|))
apply (*rule iffI*)
using *f1 f2* **apply** *force*
apply (*auto*)
proof −
assume *a1*: ¬ (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*Suc* (*0*::ℕ))|)
assume *a2*: ¬ (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*2*::ℕ)|)
assume *a3*: ¬ (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*3*::ℕ)|)
assume *a4*: ¬ (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*4*::ℕ)|)
assume *a6*: ¬ (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*6*::ℕ)|)
from *a1* **have** *f11*: ¬*fd1*$_v$ (*snd x*) = *nat2td* (*Suc* (*0*::ℕ))
by *force*
from *a2* **have** *f12*: ¬*fd1*$_v$ (*snd x*) = *nat2td* (*2*::ℕ)
by *force*
from *a3* **have** *f13*: ¬*fd1*$_v$ (*snd x*) = *nat2td* (*3*::ℕ)
by *force*
from *a4* **have** *f14*: ¬*fd1*$_v$ (*snd x*) = *nat2td* (*4*::ℕ)
by *force*
from *a6* **have** *f16*: ¬*fd1*$_v$ (*snd x*) = *nat2td* (*6*::ℕ)
by *force*
have *fd1*$_v$ (*snd x*) = *nat2td* (*5*::ℕ)
using *f11 f12 f13 f14 f16 fd1-mem* **by** (*metis One-nat-def insertE singletonD*)
then show (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *fd2*$_v$ (*fst x*)|) = *fst x*(|*fd1*$_v$:= *nat2td* (*5*::ℕ)|)
by *simp*
next
assume *a1*: ¬ *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*Suc* (*0*::ℕ))|)
assume *a2*: ¬ *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*2*::ℕ)|)
assume *a3*: ¬ *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*3*::ℕ)|)
assume *a4*: ¬ *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*4*::ℕ)|)
assume *a6*: ¬ *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*6*::ℕ)|)
from *a1* **have** *f11*: ¬*fd2*$_v$ (*snd x*) = *nat2td* (*Suc* (*0*::ℕ))
by *force*
from *a2* **have** *f12*: ¬*fd2*$_v$ (*snd x*) = *nat2td* (*2*::ℕ)
by *force*
from *a3* **have** *f13*: ¬*fd2*$_v$ (*snd x*) = *nat2td* (*3*::ℕ)
by *force*
from *a4* **have** *f14*: ¬*fd2*$_v$ (*snd x*) = *nat2td* (*4*::ℕ)
by *force*
from *a6* **have** *f16*: ¬*fd2*$_v$ (*snd x*) = *nat2td* (*6*::ℕ)
by *force*
have *fd2*$_v$ (*snd x*) = *nat2td* (*5*::ℕ)
using *f11 f12 f13 f14 f16 fd2-mem* **by** (*metis One-nat-def insertE singletonD*)
then show *snd x* = (|*fd1*$_v$ = *fd1*$_v$ (*snd x*), *fd2*$_v$ = *nat2td* (*5*::ℕ)|)
by *simp*
qed
qed

have *f1*: ($\sum_\infty$ v_0::*fdstate*. *?lhs1* v_0 ∗ *?lhs2* v_0) =
($\sum_\infty$ v_0::*fdstate*. (*if* (*?lhs1-b* v_0 ∧ *?lhs2-b* v_0) *then 1 else 0*))
using *lhs1-lhs2-simp infsum-cong* **by** *auto*
also have *f2*: ... = *card* {v_0. (*?lhs1-b* v_0 ∧ *?lhs2-b* v_0)}
apply (*subst infsum-constant-finite-states*)
apply (*subst finite-subset*[**where** *B* = {*s*::*fdstate*. *True*}])
apply (*simp*)

using *fdstate-finite* **apply** *fastforce*
by (*simp*)+
also have *f3*: ... = *1*
by (*simp add*: *lhs1b-lhs2b-simp*)

have $(\sum_\infty v_0::fdstate.\ ?lhs1\ v_0 * ?lhs2\ v_0) = ?rhs1 * ?rhs2$
apply (*subst infsum-finite*)
apply (*simp add*: *fdstate-finite*)
by (*smt* (*z3*) *calculation f1 f3 fdstate.select-convs*(*1*) *fdstate.select-convs*(*2*) *fdstate.surjective fdstate.update-convs*(*1*) *fdstate.update-convs*(*2*) *fdstate-finite infsum-0 infsum-finite lhs1b-lhs2b-simp mult-cancel-right1*)
then show *?lhs* = *?rhs*
apply (*simp only*: *lhs3-simp*)
apply (*subst infsum-cdiv-left*)
apply (*subst summable-on-finite*)
using *Tdice-UNIV-finite* **apply** (*metis UNIV-def fdstate-set-eq fdstate-set-finite*)
apply (*simp*)
by *presburger*
qed

lemma *fdice-throw-drop-initial-segments-eq*:
$(\bigsqcup n::\mathbb{N}.\ iter_p\ (n+2)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p) = (\bigsqcup n::\mathbb{N}.\ iter_p\ (n)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p)$
apply (*rule increasing-chain-sup-subset-eq*)
apply (*rule iterate-increasing-chain*)
by (*simp add*: *fdice-throw-is-dist*)

abbreviation $sum\text{-}5\text{-}6 \equiv \lambda n.\ (1 - (5\ /\ 6)\ \hat{}\ (n+1))\ /\ (1 - ((5::\mathbb{R})\ /\ 6))$

lemma *sum-geometric-series-5-6*: $(sum\ ((\hat{})\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0..n\}) = sum\text{-}5\text{-}6\ n$
apply (*induction n*)
apply (*simp*)
by (*metis Suc-eq-plus1 atLeast0AtMost eq-divide-eq-numeral1*(*1*) *mult-cancel-right1 numeral-eq-iff semiring-norm*(*88*) *sum-gp0 zero-neq-numeral*)

lemma *sum-5-6-in-0-6*: $sum\text{-}5\text{-}6\ n \geq 1 \wedge sum\text{-}5\text{-}6\ n \leq 6$
apply (*rule conjI*)
apply (*simp-all*)
apply (*induction n*)
apply (*simp*)
by *simp*

lemma *sum-5-6-in-0-6'*: $sum\text{-}5\text{-}6\ n \leq 6$
using *sum-5-6-in-0-6* **by** *blast*

lemma *iterate-fdice-throw-bottom-simp*:
shows $iter_p\ 0\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p = 0_p$
$iter_p\ (Suc\ 0)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p$
$= ([\![\$fd1^< = \$fd2^<]\!]_{\mathcal{I}e} * [\![\$fd1^> = \$fd1^< \wedge \$fd2^> = \$fd2^<]\!]_{\mathcal{I}e})_e$
$iter_p\ (n+2)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p =$
$(([\![\$fd1^< = \$fd2^<]\!]_{\mathcal{I}e} * [\![\$fd1^> = \$fd1^< \wedge \$fd2^> = \$fd2^<]\!]_{\mathcal{I}e}) +$
$[\![\neg\$fd1^< = \$fd2^<]\!]_{\mathcal{I}e} * [\![\$fd1^> = \$fd2^>]\!]_{\mathcal{I}e}\ /\ 36 * (\sum i\in\{0..«n»\}.\ (5/6)\hat{}i))_e$
proof −
show $iter_p\ 0\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p = 0_p$
by *auto*

show $iter_p$ (*Suc 0*) ($fd1^< \neq fd2^<)_e$ *fdice-throw* 0_p = ($[\![\$fd1^< = \$fd2^<]\!]_{\mathcal{I}e} * [\![\$fd1^> = \$fd1^< \wedge \$fd2^> = \$fd2^<]\!]_{\mathcal{I}e})_e$
  apply (*auto*)
  apply (*simp add*: *loopfunc-def*)
  apply (*simp add*: *prfun-zero-right'*)
  apply (*simp add*: *pfun-defs*)
  apply (*subst rvfun-skip-inverse*)
  apply (*subst ureal-zero*)
  apply (*simp add*: *ureal-defs*)
  apply (*subst fun-eq-iff*)
  by (*pred-auto*)

let $?lhs1\text{-}b = \lambda v_0{::}fdstate.\ fd1_v\ v_0 = nat2td\ (Suc\ (0{::}\mathbb{N})) \vee$
  $fd1_v\ v_0 = nat2td\ (2{::}\mathbb{N}) \vee$
  $fd1_v\ v_0 = nat2td\ (3{::}\mathbb{N}) \vee$
  $fd1_v\ v_0 = nat2td\ (4{::}\mathbb{N}) \vee$
  $fd1_v\ v_0 = nat2td\ (5{::}\mathbb{N}) \vee$
  $fd1_v\ v_0 = nat2td\ (6{::}\mathbb{N})$
let $?lhs2\text{-}b = \lambda v_0{::}fdstate.\ fd2_v\ v_0 = nat2td\ (Suc\ (0{::}\mathbb{N})) \vee$
  $fd2_v\ v_0 = nat2td\ (2{::}\mathbb{N}) \vee$
  $fd2_v\ v_0 = nat2td\ (3{::}\mathbb{N}) \vee$
  $fd2_v\ v_0 = nat2td\ (4{::}\mathbb{N}) \vee$
  $fd2_v\ v_0 = nat2td\ (5{::}\mathbb{N}) \vee$
  $fd2_v\ v_0 = nat2td\ (6{::}\mathbb{N})$

have *card-lhs-eq*: $\{v_0{::}fdstate.\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge$
  $v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\} = \{v_0{::}fdstate.\ v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\}$
  apply (*subst set-eq-iff*)
  apply (*auto*)
  using *Tdice-mem* **apply** *auto*[*1*]
  using *Tdice-mem* **by** *auto*[*1*]
then have *card-lhs-1*: *card* $\{v_0{::}fdstate.\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge$
  $v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\} = 1$
  by (*simp add*: *numeral-1-eq-Suc-0 numerals*(*1*))

have *f7*: $\forall\, v_0{::}fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
  $(if\ ?lhs2\text{-}b\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
  $(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 0{::}\mathbb{R}\ else\ if\ v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) =$
  $(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))$
  apply (*rule allI*)
  by (*auto*)
then have *f8*: $(\sum_\infty v_0{::}fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
  $(if\ ?lhs2\text{-}b\ v_0\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) *$
  $(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 0{::}\mathbb{R}\ else\ if\ v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) / 36)$
  $= (\sum_\infty v_0{::}fdstate.$
  $(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R})) / 36)$
  using *infsum-cong* **by** *presburger*
have *f9*: ... $= (\sum_\infty v_0{::}fdstate.$
  $(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = a,\ fd2_v = a|\!)\ then\ 1{::}\mathbb{R}\ else\ (0{::}\mathbb{R}))) / 36$
  apply (*subst infsum-cdiv-left*)

apply (*rule infsum-cond-finite-states-summable*)
using *fdstate-finite finite-subset top-greatest* **apply** *blast*
by *simp*
have *f10*: ... = *card* $\{v_0$::*fdstate*. *?lhs1-b* $v_0 \wedge$ *?lhs2-b* $v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = a, fd2_v = a|\!)\}$ / *36*
apply (*subst infsum-constant-finite-states*)
using *fdstate-finite finite-subset top-greatest* **apply** *blast*
by *simp*
have *f11*: ... = *1* / *36*
using *card-lhs-1* **by** *linarith*

show $iter_p$ $(n+2)$ $(fd1^< \neq fd2^<)_e$ *fdice-throw* 0_p =
$((\llbracket \$fd1^< = \$fd2^< \rrbracket_{\mathcal{I}e} * \llbracket \$fd1^> = \$fd1^< \wedge \$fd2^> = \$fd2^< \rrbracket_{\mathcal{I}e}) +$
$\llbracket \neg \$fd1^< = \$fd2^< \rrbracket_{\mathcal{I}e} * \llbracket \$fd1^> = \$fd2^> \rrbracket_{\mathcal{I}e}$ / *36* $* (\sum i \in \{0..«n»\}. (5/6)\hat{\ }i))_e$
apply (*induct-tac n*)
apply (*simp*)
apply (*simp add*: *loopfunc-def*)
apply (*simp add*: *prfun-zero-right'*)
apply (*simp add*: *pfun-defs*)
apply (*subst rvfun-skip-inverse*)+
apply (*subst ureal-zero*)
apply (*subst rvfun-pcond-inverse*)
apply (*metis ureal-is-prob ureal-zero*)
apply (*simp add*: *rvfun-skip-f-is-prob*)
apply (*subst fdice-throw-altdef*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*simp add*: *expr-defs rel lens-defs*)
apply (*rule allI*)+
apply (*rule conjI*)
apply (*simp add*: *infsum-nonneg iverson-bracket-def*)
apply (*subst rvfun-skip_f-simp*)
apply (*simp only*: *ureal-rzero-0*)
apply (*auto*)
defer
apply (*expr-auto add*: *prfun-of-rvfun-def*)
apply (*simp add*: *real2ureal-def skip-def*)+
apply (*subst rvfun-skip_f-simp*)
apply (*simp only*: *ureal-rzero-0 snd-conv*)
apply (*auto*)
defer
apply (*subst rvfun-skip_f-simp*)
apply (*simp only*: *ureal-rzero-0 snd-conv*)
apply (*auto*)
apply (*simp add*: *infsum-0 real2ureal-def*)

apply (*subst loopfunc-def*)
apply (*subst pseqcomp-def*)
apply (*subst pcond-def*)
apply (*subst fdice-throw-altdef*)
apply (*subst rvfun-inverse*)
apply (*simp add*: *dist-defs*)
apply (*simp add*: *expr-defs rel lens-defs*)
apply (*rule allI*)+
apply (*rule conjI*)

apply (*simp add*: *infsum-nonneg prfun-in-0-1'*)
apply (*simp add*: *rvfun-of-prfun-def*)
apply (*auto*)
prefer *3*
apply (*simp only*: *rvfun-of-prfun-def prfun-of-rvfun-def*)
apply (*expr-auto*)
apply (*metis ereal-eq-1*(*1*) *one-ureal-def prfun-skip-id real2ureal-def ureal2rereal-inverse*)
apply (*simp add*: *prfun-skip-not-id real2ureal-def ureal2rereal-inverse zero-ereal-def zero-ureal-def*)+
defer
apply (*smt* (*verit*, *best*) *divide-eq-0-iff infsum-0 mult-cancel-left1 mult-cancel-right1 o-apply real2ureal-def real-of-ereal-0 ureal2real-def zero-ereal-def zero-ureal.rep-eq zero-ureal-def*)
prefer *4*
prefer *4*
proof −
  fix *b*::*fdstate*
  let *?lhs* = ($\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
    (*if ?lhs2-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
    (*if* ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 0*::ℝ *else if* v_0 = *b then 1*::ℝ *else* (*0*::ℝ)) / (*36*::ℝ))
  have *card-lhs-leq*: *card* {v_0::*fdstate*. *?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ v_0 = *b*}
  ≤ *card* {v_0::*fdstate*. v_0 = *b*}
    apply (*subst card-mono*)
    apply *simp*
    apply *force*
    by *simp*
  have *card-lhs-leq'*: ... = *1*
    by *simp*

  have *f1*: ∀ v_0::*fdstate*. (*if ?lhs1-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
      (*if ?lhs2-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
      (*if* ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 0*::ℝ *else if* v_0 = *b then 1*::ℝ *else* (*0*::ℝ)) =
      (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ v_0 = *b then 1*::ℝ *else* (*0*::ℝ))
    apply (*rule allI*)
    by (*auto*)
  then have *f2*: *?lhs* = ($\sum_\infty v_0$::*fdstate*.
    (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ v_0 = *b then 1*::ℝ *else* (*0*::ℝ)) / *36*)
    using *infsum-cong* **by** *presburger*
  have *f3*: ... = ($\sum_\infty v_0$::*fdstate*.
    (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ v_0 = *b then 1*::ℝ *else* (*0*::ℝ))) / *36*
    apply (*subst infsum-cdiv-left*)
    apply (*rule infsum-cond-finite-states-summable*)
    using *fdstate-finite finite-subset top-greatest* **apply** *blast*
    by *simp*
  have *f4*: ... = *card* {v_0::*fdstate*. *?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ v_0 = *b*} / *36*
    apply (*subst infsum-constant-finite-states*)
    using *fdstate-finite finite-subset top-greatest* **apply** *blast*
    by *simp*
  have *f5*: ... ≤ *1*
    using *card-lhs-leq card-lhs-leq'* **by** *linarith*

  show *?lhs* ≤ (*1*::ℝ)
    using *f2 f3 f4 f5* **by** *presburger*
next
  fix *fd1 fd2* $fd2_v$*'*::*Tdice*

  have *card-lhs-eq*: {v_0::*fdstate*. *?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧

$v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\} = \{v_0::fdstate.\ v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\}$
apply (*subst set-eq-iff*)
apply (*auto*)
using *Tdice-mem* **apply** *auto*[*1*]
using *Tdice-mem* **by** *auto*[*1*]
then have *card-lhs-1*: $card\ \{v_0::fdstate.\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge$
$v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\} = 1$
by (*simp add*: *numeral-1-eq-Suc-0 numerals*(*1*))

have *f01*: $(\sum_\infty v_0::fdstate.(if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 0::\mathbb{R}\ else$
$if\ v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (36::\mathbb{R})) =$
$(\sum_\infty v_0::fdstate.$
$(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R}))\ /\ 36)$
apply (*subst infsum-cong*[**where** $g = \lambda v_0.\ (if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge$
$v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ 36$])
by *auto*
have *f02*: $... = (\sum_\infty v_0::fdstate.$
$(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R})))\ /\ 36$
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-cond-finite-states-summable*)
using *fdstate-finite finite-subset top-greatest* **apply** *blast*
by *simp*
have *f03*: $... = card\ \{v_0::fdstate.\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\}\ /\ 36$
apply (*subst infsum-constant-finite-states*)
using *fdstate-finite finite-subset top-greatest* **apply** *blast*
by *simp*
have *f04*: $... = 1\ /\ 36$
using *card-lhs-1* **by** *linarith*

then show *ereal2ureal*
(*ereal*
$(\sum_\infty v_0::fdstate.(if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 0::\mathbb{R}\ else$
$if\ v_0 = (\!|fd1_v = fd2_v', fd2_v = fd2_v'|\!)\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (36::\mathbb{R}))) =$
$ereal2ureal\ (ereal\ ((1::\mathbb{R})\ /\ (36::\mathbb{R})))$
using *f01 f02 f03* **by** *presburger*
next
fix $n::\mathbb{N}$ **and** $b::fdstate$
let $?lhs3 = \lambda v_0.\ ureal2real\ (ereal2ureal\ (ereal$
$((if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ fd2_v\ b = fd1_v\ v_0 \wedge fd2_v\ b = fd2_v\ v_0\ then$
$1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /$
$(36::\mathbb{R}))))$
let $?lhs = (\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ?lhs3\ v_0\ /\ (36::\mathbb{R}))$
have *lhs1'-set-eq*: $\{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd1_v\ s = nat2td\ (2::\mathbb{N}) \vee fd1_v\ s = nat2td\ (3::\mathbb{N}) \vee fd1_v\ s =$
$nat2td\ (4::\mathbb{N}) \vee fd1_v\ s = nat2td\ (5::\mathbb{N}) \vee fd1_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd2_v\ s = nat2td\ (2::\mathbb{N}) \vee fd2_v\ s = nat2td\ (3::\mathbb{N}) \vee fd2_v\ s =$
$nat2td\ (4::\mathbb{N}) \vee fd2_v\ s = nat2td\ (5::\mathbb{N}) \vee fd2_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$fd1_v\ s = fd2_v\ s \wedge fd2_v\ b = fd1_v\ s \wedge fd2_v\ b = fd2_v\ s\} = \{s::fdstate.\ fd2_v\ b = fd1_v\ s \wedge fd2_v\ b =$

$fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
using *fd1-mem* **apply** *auto*[*1*]
using *fd1-mem* **by** *auto*[*1*]
have *lhs1'-set-card*: *card* $\{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd1_v\ s = nat2td\ (2::\mathbb{N}) \lor fd1_v\ s = nat2td\ (3::\mathbb{N}) \lor fd1_v\ s = nat2td\ (4::\mathbb{N}) \lor fd1_v\ s = nat2td\ (5::\mathbb{N}) \lor fd1_v\ s = nat2td\ (6::\mathbb{N})) \land$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd2_v\ s = nat2td\ (2::\mathbb{N}) \lor fd2_v\ s = nat2td\ (3::\mathbb{N}) \lor fd2_v\ s = nat2td\ (4::\mathbb{N}) \lor fd2_v\ s = nat2td\ (5::\mathbb{N}) \lor fd2_v\ s = nat2td\ (6::\mathbb{N})) \land$
$fd1_v\ s = fd2_v\ s \land fd2_v\ b = fd1_v\ s \land fd2_v\ b = fd2_v\ s\} = Suc\ 0$
apply (*subst lhs1'-set-eq*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac* $x = (\!|fd1_v = fd2_v\ b,\ fd2_v = fd2_v\ b|\!)$ **in** *exI*)
by (*auto*)
have *lhs1'-simp*: $(\sum_\infty v_0::fdstate.\ ($
$(if\ ?lhs1\text{-}b\ v_0 \land ?lhs2\text{-}b\ v_0 \land fd1_v\ v_0 = fd2_v\ v_0 \land fd2_v\ b = fd1_v\ v_0 \land fd2_v\ b = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ 36)) = 1\ /\ 36$
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*simp*)
apply (*subst infsum-constant-finite-states*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
using *lhs1'-set-card* **by** *linarith*

have *lhs2'-card*: *card* $\{s::fdstate.\ ?lhs1\text{-}b\ s \land ?lhs2\text{-}b\ s \land \neg\ fd1_v\ s = fd2_v\ s\} = 30$
proof −
have $\{x::fdstate.\ \neg fd1_v\ x = fd2_v\ x\} = \{s::fdstate.\ ?lhs1\text{-}b\ s \land ?lhs2\text{-}b\ s \land \neg\ fd1_v\ s = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
apply (*metis One-nat-def fd1-mem insert-iff singletonD*)
by (*metis One-nat-def fd2-mem insert-iff singletonD*)
then show *?thesis*
using *fdstate-set-d1d2-neq-card* **by** *presburger*
qed
have *lhs2'-simp*: $(\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0 \land ?lhs2\text{-}b\ v_0 \land \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R}))$
$= (\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0 \land ?lhs2\text{-}b\ v_0 \land \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R})))$
by *auto*
have *lhs2'-simp'*: ... =
$(30) * sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R})$
apply (*subst infsum-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*subst infsum-constant-finite-states*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
by (*simp add*: *lhs2'-card*)

have *f1*: $\forall\ v_0.\ ?lhs3\ v_0$
$= ((if\ fd1_v\ v_0 = fd2_v\ v_0 \land fd2_v\ b = fd1_v\ v_0 \land fd2_v\ b = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R}))$

apply (*auto*)
apply (*simp add*: *sum-geometric-series-5-6*)
apply (*subst real2eureal-inverse*)
apply (*induction n*)
apply (*simp*)
apply *fastforce*
apply (*simp*)
apply (*smt* (*verit*, *del-insts*) *divide-nonneg-nonneg one-le-power power-divide*)
apply (*simp*)
using *real2eureal-inverse* **apply** *auto*[*1*]
using *real2eureal-inverse* **by** *auto*[*1*]

have *f2*: $\forall v_0$. (*if ?lhs1-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
(*if ?lhs2-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗ *?lhs3* v_0
= (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ $fd2_v$ *b* = $fd1_v$ v_0 ∧ $fd2_v$ *b* = $fd2_v$ v_0 *then* *1*::ℝ *else* (*0*::ℝ)) +
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
sum ((^) ((*5*::ℝ) / (*6*::ℝ))) {*0*::ℕ..*n*} / (*36*::ℝ)
apply (*rule allI*)
apply (*subst f1*)
by *simp*

have *f3*: *?lhs* = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ $fd2_v$ *b* = $fd1_v$ v_0 ∧ $fd2_v$ *b* = $fd2_v$ v_0 *then* *1*::ℝ *else* (*0*::ℝ))
+ (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
sum ((^) ((*5*::ℝ) / (*6*::ℝ))) {*0*::ℕ..*n*} / (*36*::ℝ)) / (*36*::ℝ))
using *f2 infsum-cong* **by** *presburger*
have *f4*: ... = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ $fd2_v$ *b* = $fd1_v$ v_0 ∧ $fd2_v$ *b* = $fd2_v$ v_0 *then* *1*::ℝ *else* (*0*::ℝ)) / *36*
+ (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
sum ((^) ((*5*::ℝ) / (*6*::ℝ))) {*0*::ℕ..*n*} / (*36*::ℝ) / (*36*::ℝ)))
apply (*rule infsum-cong*)
using *add-divide-distrib* **by** *blast*
have *f5*: ... = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd1_v$ v_0 = $fd2_v$ v_0 ∧ $fd2_v$ *b* = $fd1_v$ v_0 ∧ $fd2_v$ *b* = $fd2_v$ v_0 *then* *1*::ℝ *else* (*0*::ℝ)) / *36*))
+ ($\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
sum ((^) ((*5*::ℝ) / (*6*::ℝ))) {*0*::ℕ..*n*} / (*36*::ℝ) / (*36*::ℝ))
apply (*subst infsum-add*)
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
by *simp*
have *f6*: ... = *1* / *36* + (*30*) ∗ *sum* ((^) ((*5*::ℝ) / (*6*::ℝ))) {*0*::ℕ..*n*} / (*36*::ℝ) / (*36*::ℝ)
by (*simp only*: *lhs1′-simp lhs2′-simp lhs2′-simp′*)
have *f7*: ... ≤ *1*
apply (*subst sum-geometric-series-5-6*)
apply (*simp*)

apply (*induction n*)
apply *force*
proof −
fix *nb* :: ℕ
have $(180 - 150 * ((5::\mathbb{R}) / 6) \hat{}\ Suc\ nb + (180 - 150 * (5 / 6) \hat{}\ Suc\ nb)) / 1296 = (180 - 150 * (5 / 6) \hat{}\ Suc\ nb) / 1296 + (180 - 150 * (5 / 6) \hat{}\ Suc\ nb) / 1296$
using *add-divide-distrib* **by** *blast*
then show $(1::\mathbb{R}) / 36 + (180 - 150 * (5 / 6) \hat{}\ Suc\ nb) / 1296 \le 1$
by (*smt* (*z3*) *add-divide-distrib divide-le-eq-1-pos divide-nonneg-nonneg one-le-power power-divide*)
qed
then show *?lhs* ≤ *1*
using *f3 f4 f5 f6* **by** *presburger*
next
fix *n*::ℕ **and** *b*::*fdstate*
assume *a1*: $\neg\ fd1_v\ b = fd2_v\ b$
let *?lhs3* = $\lambda v_0.\ ureal2real\ (ereal2ureal\ (ereal\ ((if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))$
$* (if\ fd1_v\ b = fd1_v\ v_0 \wedge fd2_v\ b = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))))$
let *?lhs* = $(\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ?lhs3\ v_0 / 36)$
have *f1*: $\forall v_0.\ ?lhs3\ v_0 = 0$
apply (*subst real2eureal-inverse*)
apply *auto*[*1*]
apply *simp*
using *a1* **by** *force*
have *f2*: $\forall v_0.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ?lhs3\ v_0 / 36 = 0$
apply (*subst f1*)
by *simp*
then show *?lhs* ≤ *1*
by (*simp add*: *infsum-0*)
next
fix *n*::ℕ **and** *fd1 fd2* $fd2_v'$::*Tdice*
let *?lhs3* = $\lambda v_0.\ ureal2real\ (ereal2ureal\ (ereal$
$((if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ fd2_v' = fd1_v\ v_0 \wedge fd2_v' = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * sum\ ((\hat{})\ ((5::\mathbb{R}) / (6::\mathbb{R})))\ \{0::\mathbb{N}..n\} / (36::\mathbb{R}))))$
let *?lhs* = $(\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ?lhs3\ v_0 / (36::\mathbb{R}))$

have *lhs1'-set-eq*: $\{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd1_v\ s = nat2td\ (2::\mathbb{N}) \vee fd1_v\ s = nat2td\ (3::\mathbb{N}) \vee fd1_v\ s = nat2td\ (4::\mathbb{N}) \vee fd1_v\ s = nat2td\ (5::\mathbb{N}) \vee fd1_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd2_v\ s = nat2td\ (2::\mathbb{N}) \vee fd2_v\ s = nat2td\ (3::\mathbb{N}) \vee fd2_v\ s = nat2td\ (4::\mathbb{N}) \vee fd2_v\ s = nat2td\ (5::\mathbb{N}) \vee fd2_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$fd1_v\ s = fd2_v\ s \wedge fd2_v' = fd1_v\ s \wedge fd2_v' = fd2_v\ s\} = \{s::fdstate.\ fd2_v' = fd1_v\ s \wedge fd2_v' = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
using *fd2-mem* **apply** *auto*[*1*]
using *fd2-mem* **by** *auto*[*1*]
have *lhs1'-set-card*: *card* $\{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd1_v\ s = nat2td\ (2::\mathbb{N}) \vee fd1_v\ s = nat2td\ (3::\mathbb{N}) \vee fd1_v\ s = nat2td\ (4::\mathbb{N}) \vee fd1_v\ s = nat2td\ (5::\mathbb{N}) \vee fd1_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \vee fd2_v\ s = nat2td\ (2::\mathbb{N}) \vee fd2_v\ s = nat2td\ (3::\mathbb{N}) \vee fd2_v\ s =$

$nat2td$ $(4::\mathbb{N}) \vee fd2_v\ s = nat2td\ (5::\mathbb{N}) \vee fd2_v\ s = nat2td\ (6::\mathbb{N})) \wedge$
$fd1_v\ s = fd2_v\ s \wedge fd2_v{}' = fd1_v\ s \wedge fd2_v{}' = fd2_v\ s\} = Suc\ 0$
apply (*subst lhs1′-set-eq*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac* $x = (\!|fd1_v = fd2_v{}', fd2_v = fd2_v{}'|\!)$ **in** *exI*)
by (*auto*)
have *lhs1′-simp*: $(\sum_\infty v_0::fdstate.$ (
$(if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge fd1_v\ v_0 = fd2_v\ v_0 \wedge fd2_v{}' = fd1_v\ v_0 \wedge fd2_v{}' = fd2_v\ v_0\ then\ 1::\mathbb{R}$
$else\ (0::\mathbb{R}))\ /\ 36)) = 1\ /\ 36$
apply (*subst infsum-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*simp*)
apply (*subst infsum-constant-finite-states*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
using *lhs1′-set-card* **by** *linarith*

have *lhs2′-card*: $card\ \{s::fdstate.\ ?lhs1\text{-}b\ s \wedge ?lhs2\text{-}b\ s \wedge \neg\ fd1_v\ s = fd2_v\ s\} = 30$
proof −
have $\{x::fdstate.\ \neg fd1_v\ x = fd2_v\ x\} = \{s::fdstate.\ ?lhs1\text{-}b\ s \wedge ?lhs2\text{-}b\ s \wedge \neg\ fd1_v\ s = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
apply (*metis One-nat-def fd1-mem insert-iff singletonD*)
by (*metis One-nat-def fd2-mem insert-iff singletonD*)
then show *?thesis*
using *fdstate-set-d1d2-neq-card* **by** *presburger*
qed
have *lhs2′-simp*: $(\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else$
$(0::\mathbb{R})) *$
$sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R}))$
$= (\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0 \wedge ?lhs2\text{-}b\ v_0 \wedge \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R})))$
by *auto*
have *lhs2′-simp′*: ... =
$(30) * sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /\ (36::\mathbb{R})\ /\ (36::\mathbb{R})$
apply (*subst infsum-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*subst infsum-constant-finite-states*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
by (*simp add*: *lhs2′-card*)

have *f1*: $\forall\, v_0.\ ?lhs3\ v_0$
$= ((if\ fd1_v\ v_0 = fd2_v\ v_0 \wedge fd2_v{}' = fd1_v\ v_0 \wedge fd2_v{}' = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * sum\ ((\widehat{\ })\ ((5::\mathbb{R})\ /\ (6::\mathbb{R})))\ \{0::\mathbb{N}..n\}\ /$
$(36::\mathbb{R}))$
apply (*auto*)
apply (*simp add*: *sum-geometric-series-5-6*)
apply (*subst real2eureal-inverse*)
apply (*induction n*)
apply (*simp*)
apply *fastforce*
apply (*simp*)
apply (*smt* (*verit*, *del-insts*) *divide-nonneg-nonneg one-le-power power-divide*)
apply (*simp*)

using *real2eureal-inverse* **apply** *auto*[*1*]
using *real2eureal-inverse* **by** *auto*[*1*]

have *f2*: $\forall v_0$. (*if ?lhs1-b* v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$
(*if ?lhs2-b* v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$ *?lhs3* v_0
= (*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $fd1_v$ v_0 = $fd2_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd1_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) +
(*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $\neg$ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$
sum (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} / (*36*::$\mathbb{R}$)
apply (*rule allI*)
apply (*subst f1*)
by *simp*

have *f3*: *?lhs* = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $fd1_v$ v_0 = $fd2_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd1_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$))
+ (*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $\neg$ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$
sum (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} / (*36*::$\mathbb{R}$)) / (*36*::$\mathbb{R}$))
using *f2 infsum-cong* **by** *presburger*
have *f4*: ... = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $fd1_v$ v_0 = $fd2_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd1_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / *36*
+ (*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $\neg$ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$
sum (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} / (*36*::$\mathbb{R}$) / (*36*::$\mathbb{R}$)))
apply (*rule infsum-cong*)
using *add-divide-distrib* **by** *blast*
have *f5*: ... = ($\sum_\infty v_0$::*fdstate*. (
(*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $fd1_v$ v_0 = $fd2_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd1_v$ v_0 $\wedge$ $fd2_v{}'$ = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) / *36*))
+ ($\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 $\wedge$ *?lhs2-b* v_0 $\wedge$ $\neg$ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::$\mathbb{R}$ *else* (*0*::$\mathbb{R}$)) $*$
sum (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} / (*36*::$\mathbb{R}$) / (*36*::$\mathbb{R}$))
apply (*subst infsum-add*)
apply (*rule summable-on-cdiv-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cdiv-left*)
apply (*rule summable-on-cmult-left*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
by *simp*
have *f6*: ... = *1* / *36* + (*30*) $*$ *sum* (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} / (*36*::$\mathbb{R}$) / (*36*::$\mathbb{R}$)
by (*simp only*: *lhs1'-simp lhs2'-simp lhs2'-simp'*)
have *f7*: ... = ((*sum* (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*} + (*5*::$\mathbb{R}$) $*$ ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$)) $\widehat{\ }$ *n* / (*6*::$\mathbb{R}$)) / (*36*::$\mathbb{R}$))
apply (*subst sum-geometric-series-5-6*)+
by (*auto*)
then show *ereal2ureal* (*ereal ?lhs*) = *ereal2ureal* (*ereal* ((*sum* (($\widehat{\ }$) ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$))) {*0*::$\mathbb{N}$..*n*}
+ (*5*::$\mathbb{R}$) $*$ ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$)) $\widehat{\ }$ *n* / (*6*::$\mathbb{R}$)) / (*36*::$\mathbb{R}$)))
using *f3 f4 f5 f6* **by** *presburger*
qed
qed

lemma *sum-5-6-by-36-tendsto-1-6*:
(λn::$\mathbb{N}$. *ureal2real* (*ereal2ureal* (*ereal* (((*6*::$\mathbb{R}$) − (*5*::$\mathbb{R}$) $*$ ((*5*::$\mathbb{R}$) / (*6*::$\mathbb{R}$)) $\widehat{\ }$ *n*) / (*36*::$\mathbb{R}$))))) $\longrightarrow$

$(1::\mathbb{R})$ / 6
proof −
have $f0$: $(\lambda n::\mathbb{N}.\ ureal2real\ (ereal2ureal\ (ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ n) / (36::\mathbb{R})))))$
$=$
$(\lambda n::\mathbb{N}.\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ n) / (36::\mathbb{R})))$
apply (*subst fun-eq-iff*)
apply (*auto*)
apply (*simp add*: *ureal-defs*)
apply (*subst real2uereal-inverse*)
apply (*meson max.cobounded1 min.boundedI zero-less-one-ereal*)
apply *simp*
proof −
fix x
have $((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x \leq 1$
by (*simp add*: *power-le-one-iff*)
then have $f1$: $(max\ (0::ereal)\ (ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x) / (36::\mathbb{R})))) =$
$(ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x) / (36::\mathbb{R})))$
by (*simp add*: *max-def*)
have $f2$: $(min\ (max\ (0::ereal)\ (ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x) / (36::\mathbb{R}))))\ (1::ereal))$
$=$
$(ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x) / (36::\mathbb{R})))$
apply (*simp add*: *f1 min-def*)
by (*smt* (*verit*, *best*) *divide-nonneg-nonneg one-le-power power-divide*)
show $real\text{-}of\text{-}ereal\ (min\ (max\ (0::ereal)\ (ereal\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x) / (36::\mathbb{R}))))$
$(1::ereal)) * (36::\mathbb{R}) =$
$(6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ x$
by (*simp add*: *f2*)
qed

have $f1$: $(\lambda n::\mathbb{N}.\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ n) / (36::\mathbb{R}))) \longrightarrow (1::\mathbb{R}) / 6$
proof −
have $f0$: $(\lambda n::\mathbb{N}.\ (((6::\mathbb{R}) - (5::\mathbb{R}) * ((5::\mathbb{R}) / (6::\mathbb{R}))\ \hat{}\ n) / (36::\mathbb{R}))) = (\lambda n::\mathbb{N}.\ (1::\mathbb{R}) / 6 -$
$((5::\mathbb{R}) / ((6::\mathbb{R})\hat{}2) * (5/6)\ \hat{}\ n))$
apply (*subst fun-eq-iff*)
by (*auto*)
have $f1$: $(\lambda n::\mathbb{N}.\ (1::\mathbb{R}) / 6 - ((5::\mathbb{R}) / ((6::\mathbb{R})\hat{}2) * (5/6)\ \hat{}\ n)) \longrightarrow (1/6 - 0)$
apply (*rule tendsto-diff*)
apply (*auto*)
apply (*rule LIMSEQ-power-zero*)
by *simp*
show *?thesis*
using *f0 f1* **by** *auto*
qed

show *?thesis*
apply (*simp add*: *f0*)
by (*simp add*: *f1*)
qed

lemma *fdice-throw-iterate-limit-fH*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (fd1^{<} \neq fd2^{<})_e\ fdice\text{-}throw\ 0_p))$
shows $(\lambda n.\ ureal2real\ (f\ n\ s)) \longrightarrow (fH\ s)$
apply (*simp only*: *assms fH-def*)
apply (*subst iterate-fdice-throw-bottom-simp*(*3*))
apply (*subst sum-geometric-series-5-6*)

apply (*pred-auto*)
apply (*simp add*: *real2eureal-inverse*)
apply (*metis comp-def real-of-ereal-0 tendsto-const ureal2real-def zero-ereal-def zero-ureal.rep-eq zero-ureal-def*)
apply (*simp add*: *sum-5-6-by-36-tendsto-1-6*)
by (*simp add*: *real2eureal-inverse*)+

lemma *fdice-throw-iterate-limit-sup*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p))$
shows $(\lambda n.\ ureal2real\ (f\ n\ s)) \longrightarrow (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$
apply (*simp only*: *assms*)
apply (*subst LIMSEQ-ignore-initial-segment*[**where** $k = 2$])
apply (*subst increasing-chain-sup-subset-eq*[**where** $m = 2$])
apply (*rule increasing-chain-fun*)
apply (*rule iterate-increasing-chain*)
apply (*simp add*: *fdice-throw-is-dist*)
apply (*subst increasing-chain-limit-is-lub'*)
apply (*simp add*: *increasing-chain-def*)
apply (*auto*)
apply (*rule le-funI*)
by (*smt* (*verit*, *ccfv-threshold*) *fdice-throw-is-dist iterate-increasing2 le-fun-def*)

lemma *fH-eq-sup-by-limit*:
assumes $f = (\lambda n.\ (iter_p\ (n+2)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p))$
shows $(fH\ s) = (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$
apply (*subst LIMSEQ-unique*[**where** $X = (\lambda n.\ ureal2real\ (f\ n\ s))$ **and** $a = (fH\ s)$ **and** $b = (ureal2real\ (\bigsqcup n::\mathbb{N}.\ f\ n\ s))$])
using *fdice-throw-iterate-limit-fH* **apply** (*simp add*: *assms*)
using *fdice-throw-iterate-limit-sup* **apply** (*simp add*: *assms*)
by *auto*

lemma *fH-eq-sup-by-limit'*: $(\bigsqcup n::\mathbb{N}.\ iter_p\ (n+2)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 0_p) = (\lambda x::fdstate \times fdstate.\ ereal2ureal\ (ereal\ (fH\ x)))$
apply (*simp only*: *fH-eq-sup-by-limit*)
apply (*simp add*: *ureal2rereal-inverse*)
using *SUP-apply* **by** *fastforce*

lemma *fdice-throw-loop*: *fdice-throw-loop* = *prfun-of-rvfun fH*
apply (*simp add*: *fdice-throw-loop-def fH-def prfun-of-rvfun-def real2ureal-def*)
apply (*subst sup-continuous-lfp-iteration*)
apply (*simp add*: *fdice-throw-is-dist*)
apply (*rule finite-subset*[**where** $B = \{s::fdstate \times fdstate.\ True\}$])
apply *force*
using *fdstate-finite finite-Prod-UNIV* **apply** *auto*[*1*]
apply (*simp only*: *fdice-throw-drop-initial-segments-eq*[*symmetric*])
apply (*simp only*: *fH-eq-sup-by-limit' fH-def*)
by *auto*

5.1.5 Using unique fixed point theorem

lemma *fdice-throw-iterdiff-simp*:
shows $(iterdiff\ 0\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 1_p) = 1_p$
$(iterdiff\ (n+1)\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 1_p) = prfun\text{-}of\text{-}rvfun\ ((\llbracket fd1^< \neq fd2^< \rrbracket_{\mathcal{I}e} * (5/6)\hat{~}«n»)_e)$
proof −
show $(iterdiff\ 0\ (fd1^< \neq fd2^<)_e\ fdice\text{-}throw\ 1_p) = 1_p$
by (*auto*)

have *f1*: $(\sum_\infty v_0$::*fdstate*. (*if fd1-pred* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
(*if fd2-pred* v_0 *then 1*::ℝ *else* (*0*::ℝ)) / (*36*::ℝ)) =
$(\sum_\infty v_0$::*fdstate*. (*if fd1-pred* v_0 ∧ *fd2-pred* v_0 *then 1/36 else* (*0*::ℝ)))
apply (*rule infsum-cong*)
by (*simp*)
have *f2*: ... = *1*
apply (*subst infsum-constant-finite-states*)
apply (*meson fdstate-finite rev-finite-subset subset-UNIV*)
apply (*simp add*: *fdstate-pred-univ*)
using *card-fdstate-set* **by** *auto*

show (*iterdiff* (*n*+*1*) $(fd1^< \neq fd2^<)_e$ *fdice-throw* 1_p) = *prfun-of-rvfun* $((⟦fd1^< \neq fd2^<⟧_{\mathcal{I}e} * (5/6)$^«*n*»$)_e)$
apply (*induction n*)
apply (*simp add*: *pfun-defs*)
apply (*subst fdice-throw-altdef*)
apply (*subst ureal-zero*)
apply (*subst ureal-one*)
apply (*subst rvfun-seqcomp-inverse*)
using *fdice-throw-altdef fdice-throw-is-dist* **apply** *presburger*
apply (*metis ureal-is-prob ureal-one*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto add*: *rel*)
using *f1 f2* **apply** *presburger*
apply (*simp only*: *add-Suc*)
apply (*simp only*: *iterdiff.simps*(*2*))
apply (*simp only*: *pcond-def*)
apply (*simp only*: *pseqcomp-def*)
apply (*subst rvfun-seqcomp-inverse*)
using *fdice-throw-altdef fdice-throw-is-dist* **apply** *presburger*
apply (*simp add*: *ureal-is-prob*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*simp add*: *power-le-one*)
apply (*subst fdice-throw-altdef*)
apply (*expr-auto add*: *rel*)
defer
apply (*simp add*: *pfun-defs*)
apply (*subst ureal-zero*)
apply *simp*
proof −
fix *n fd1 fd2*
let *?lhs-3* = λv_0. ((*if* ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗ ((*5*::ℝ) / (*6*::ℝ)) ^ *n*)
let *?lhs* = $(\sum_\infty v_0$::*fdstate*. (*if fd1-pred* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
(*if fd2-pred* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗ *?lhs-3* v_0 / (*36*::ℝ))
have *f1*: *?lhs* = $(\sum_\infty v_0$::*fdstate*.
(*if fd1-pred* v_0 ∧ *fd2-pred* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then* ((*5*::ℝ) / (*6*::ℝ)) ^ *n* / (*36*::ℝ) *else*
(*0*::ℝ)))
apply (*rule infsum-cong*)
by *auto*
also have *f2*: ... = *30* ∗ ((*5*::ℝ) / (*6*::ℝ)) ^ *n* / (*36*::ℝ)
apply (*subst infsum-constant-finite-states*)
using *fdstate-finite infinite-super top-greatest* **apply** *blast*
by (*simp add*: *fdstate-pred-d1d2-neq fdstate-set-d1d2-neq-card*)

then show *real2ureal ?lhs* = *real2ureal* ((*5*::ℝ) ∗ ((*5*::ℝ) / (*6*::ℝ)) ^ *n* / (*6*::ℝ))
by (*simp add*: *f1 f2*)
qed
qed

lemma *fdice-throw-iterdiff-tendsto-0*:
$\forall$ *s*::*fdstate* × *fdstate*. (λ*n*::ℕ. *ureal2real* (*iterdiff n* ($fd1^< \neq fd2^<$)$_e$ *fdice-throw* 1_p *s*)) ⟶ (*0*::ℝ)
proof
fix *s*
have (λ*n*::ℕ. *ureal2real* (*iterdiff* (*n*+*1*) ($fd1^< \neq fd2^<$)$_e$ *fdice-throw* 1_p *s*)) ⟶ (*0*::ℝ)
apply (*subst fdice-throw-iterdiff-simp*)
apply (*simp add*: *prfun-of-rvfun-def*)
apply (*expr-auto*)
apply (*subst real2ureal-inverse*)
apply (*simp*)
apply (*simp add*: *power-le-one*)
apply (*simp add*: *LIMSEQ-realpow-zero*)
apply (*subst real2ureal-inverse*)
by (*simp*)+
then show (λ*n*::ℕ. *ureal2real* (*iterdiff n* ($fd1^< \neq fd2^<$)$_e$ *fdice-throw* 1_p *s*)) ⟶ (*0*::ℝ)
by (*rule LIMSEQ-offset*[**where** *k* = *1*])
qed

lemma *fH-is-fp*: $\mathcal{F}$ ($fd1^< \neq fd2^<$)$_e$ *fdice-throw* (*prfun-of-rvfun fH*) = *prfun-of-rvfun fH*
apply (*simp add*: *fH-def loopfunc-def*)
apply (*simp add*: *pfun-defs*)
apply (*subst fdice-throw-altdef*)
apply (*subst rvfun-skip-inverse*)
apply (*subst rvfun-seqcomp-inverse*)
using *fdice-throw-altdef fdice-throw-is-dist* **apply** *presburger*
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*subst rvfun-inverse*)
apply (*expr-auto add*: *dist-defs*)
apply (*expr-auto add*: *prfun-of-rvfun-def skip-def*)
defer
apply (*subst infsum-0*)
prefer *2*
apply *simp*
proof −
fix *fd1 fd2* $fd1_v{}'$ $fd2_v{}'$::*Tdice* **and** *x*::*fdstate*
assume *a1*: ¬ $fd1_v{}' = fd2_v{}'$
have ((*if* $fd1_v$ *x* = $fd2_v$ *x then 1*::ℝ *else* (*0*::ℝ)) ∗ (*if* $fd1_v{}'$ = $fd1_v$ *x* ∧ $fd2_v{}'$ = $fd2_v$ *x then 1*::ℝ *else* (*0*::ℝ))) = *0*
using *a1* **by** *auto*
then show (*if* $fd1_v$ *x* = *nat2td* (*Suc* (*0*::ℕ)) ∨
$fd1_v$ *x* = *nat2td* (*2*::ℕ) ∨ $fd1_v$ *x* = *nat2td* (*3*::ℕ) ∨ $fd1_v$ *x* = *nat2td* (*4*::ℕ) ∨ $fd1_v$ *x* = *nat2td* (*5*::ℕ) ∨ $fd1_v$ *x* = *nat2td* (*6*::ℕ)
then 1::ℝ *else* (*0*::ℝ)) ∗
(*if* $fd2_v$ *x* = *nat2td* (*Suc* (*0*::ℕ)) ∨
$fd2_v$ *x* = *nat2td* (*2*::ℕ) ∨ $fd2_v$ *x* = *nat2td* (*3*::ℕ) ∨ $fd2_v$ *x* = *nat2td* (*4*::ℕ) ∨ $fd2_v$ *x* = *nat2td* (*5*::ℕ) ∨ $fd2_v$ *x* = *nat2td* (*6*::ℕ)
then 1::ℝ *else* (*0*::ℝ)) ∗
((*if* $fd1_v$ *x* = $fd2_v$ *x then 1*::ℝ *else* (*0*::ℝ)) ∗ (*if* $fd1_v{}'$ = $fd1_v$ *x* ∧ $fd2_v{}'$ = $fd2_v$ *x then 1*::ℝ *else* (*0*::ℝ))) /

$(36::\mathbb{R}) = (0::\mathbb{R})$
by *fastforce*
next
fix $fd1\ fd2\ fd2_v{}'::Tdice$
let $?lhs1\text{-}b = \lambda v_0::fdstate.\ fd1_v\ v_0 = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd1_v\ v_0 = nat2td\ (2::\mathbb{N}) \lor$
$fd1_v\ v_0 = nat2td\ (3::\mathbb{N}) \lor$
$fd1_v\ v_0 = nat2td\ (4::\mathbb{N}) \lor$
$fd1_v\ v_0 = nat2td\ (5::\mathbb{N}) \lor$
$fd1_v\ v_0 = nat2td\ (6::\mathbb{N})$
let $?lhs2\text{-}b = \lambda v_0::fdstate.\ fd2_v\ v_0 = nat2td\ (Suc\ (0::\mathbb{N})) \lor$
$fd2_v\ v_0 = nat2td\ (2::\mathbb{N}) \lor$
$fd2_v\ v_0 = nat2td\ (3::\mathbb{N}) \lor$
$fd2_v\ v_0 = nat2td\ (4::\mathbb{N}) \lor$
$fd2_v\ v_0 = nat2td\ (5::\mathbb{N}) \lor$
$fd2_v\ v_0 = nat2td\ (6::\mathbb{N})$
let $?lhs3 = \lambda v_0.\ ((if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * (if\ fd2_v{}' = fd1_v\ v_0 \land fd2_v{}' = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))\ /\ (6::\mathbb{R}))$
let $?lhs = (\sum_\infty v_0::fdstate.\ (if\ ?lhs1\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
$(if\ ?lhs2\text{-}b\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) * ?lhs3\ v_0\ /\ (36::\mathbb{R}))$
have *lhs3-simp*: $\forall v_0.\ ?lhs3\ v_0 = ((if\ fd2_v{}' = fd1_v\ v_0 \land fd2_v{}' = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) +$
$(if\ \neg\ fd1_v\ v_0 = fd2_v\ v_0\ then\ ((1::\mathbb{R})\ /\ (6::\mathbb{R}))\ else\ (0::\mathbb{R})))$
by *force*

have *lhs1-set-eq*: $\{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd1_v\ s = nat2td\ (2::\mathbb{N}) \lor fd1_v\ s = nat2td\ (3::\mathbb{N}) \lor fd1_v\ s = nat2td\ (4::\mathbb{N}) \lor fd1_v\ s = nat2td\ (5::\mathbb{N}) \lor fd1_v\ s = nat2td\ (6::\mathbb{N})) \land$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd2_v\ s = nat2td\ (2::\mathbb{N}) \lor fd2_v\ s = nat2td\ (3::\mathbb{N}) \lor fd2_v\ s = nat2td\ (4::\mathbb{N}) \lor fd2_v\ s = nat2td\ (5::\mathbb{N}) \lor fd2_v\ s = nat2td\ (6::\mathbb{N})) \land$
$fd2_v{}' = fd1_v\ s \land fd2_v{}' = fd2_v\ s\} = \{s::fdstate.\ fd2_v{}' = fd1_v\ s \land fd2_v{}' = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
using *fd2-mem* **apply** *auto*[*1*]
using *fd2-mem* **by** *auto*[*1*]

have *lhs1-set-card*: $card\ \{s::fdstate.$
$(fd1_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd1_v\ s = nat2td\ (2::\mathbb{N}) \lor fd1_v\ s = nat2td\ (3::\mathbb{N}) \lor fd1_v\ s = nat2td\ (4::\mathbb{N}) \lor fd1_v\ s = nat2td\ (5::\mathbb{N}) \lor fd1_v\ s = nat2td\ (6::\mathbb{N})) \land$
$(fd2_v\ s = nat2td\ (Suc\ (0::\mathbb{N})) \lor fd2_v\ s = nat2td\ (2::\mathbb{N}) \lor fd2_v\ s = nat2td\ (3::\mathbb{N}) \lor fd2_v\ s = nat2td\ (4::\mathbb{N}) \lor fd2_v\ s = nat2td\ (5::\mathbb{N}) \lor fd2_v\ s = nat2td\ (6::\mathbb{N})) \land$
$fd2_v{}' = fd1_v\ s \land fd2_v{}' = fd2_v\ s\} = Suc\ 0$
apply (*subst lhs1-set-eq*)
apply (*subst card-1-singleton-iff*)
apply (*rule-tac* $x = (\!|fd1_v = fd2_v{}',\ fd2_v = fd2_v{}'|\!)$ **in** *exI*)
by (*auto*)

have *lhs2-card*: $card\ \{s::fdstate.\ ?lhs1\text{-}b\ s \land ?lhs2\text{-}b\ s \land \neg\ fd1_v\ s = fd2_v\ s\} = 30$
proof −
have $\{x::fdstate.\ \neg fd1_v\ x = fd2_v\ x\} = \{s::fdstate.\ ?lhs1\text{-}b\ s \land ?lhs2\text{-}b\ s \land \neg\ fd1_v\ s = fd2_v\ s\}$
apply (*subst set-eq-iff*)
apply (*auto*)
apply (*metis One-nat-def fd1-mem insert-iff singletonD*)
by (*metis One-nat-def fd2-mem insert-iff singletonD*)
then show *?thesis*

using *fdstate-set-d1d2-neq-card* **by** *presburger*
qed
have *f1*: *?lhs* = $(\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
((*if* $fd2_v{}'$ = $fd1_v$ v_0 ∧ $fd2_v{}'$ = $fd2_v$ v_0 *then 1*::ℝ *else* (*0*::ℝ)) +
(*if* ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then* ((*1*::ℝ) / (*6*::ℝ)) *else* (*0*::ℝ))) / (*36*::ℝ))
apply (*rule infsum-cong*)
by *force*
have *f2*: ... = $(\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 *then 1*::ℝ *else* (*0*::ℝ)) ∗
((*if* $fd2_v{}'$ = $fd1_v$ v_0 ∧ $fd2_v{}'$ = $fd2_v$ v_0 *then 1* / (*36*::ℝ) *else* (*0*::ℝ)) +
(*if* ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then* ((*1*::ℝ) / (*6*::ℝ)) / (*36*::ℝ) *else* (*0*::ℝ))))
apply (*rule infsum-cong*)
by (*smt* (*verit*, *best*) *add-cancel-left-right div-0 mult-cancel-left2 mult-cancel-right2*)
have *f3*: ... = $(\sum_\infty v_0$::*fdstate*.
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd2_v{}'$ = $fd1_v$ v_0 ∧ $fd2_v{}'$ = $fd2_v$ v_0 *then 1* / (*36*::ℝ) *else* (*0*::ℝ)) +
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then* ((*1*::ℝ) / (*6*::ℝ)) / *36 else* (*0*::ℝ)))
apply (*rule infsum-cong*)
by *force*
have *f4*: ... = $(\sum_\infty v_0$::*fdstate*.
(*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ $fd2_v{}'$ = $fd1_v$ v_0 ∧ $fd2_v{}'$ = $fd2_v$ v_0 *then 1* / (*36*::ℝ) *else* (*0*::ℝ))) +
$(\sum_\infty v_0$::*fdstate*. (*if ?lhs1-b* v_0 ∧ *?lhs2-b* v_0 ∧ ¬ $fd1_v$ v_0 = $fd2_v$ v_0 *then* ((*1*::ℝ) / (*6*::ℝ)) / *36 else*
(*0*::ℝ)))
apply (*rule infsum-add*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*rule finite-subset*[**where** *B* = *UNIV*])
apply (*simp*)
apply (*simp add*: *fdstate-finite*)
apply (*rule infsum-constant-finite-states-summable*)
apply (*rule finite-subset*[**where** *B* = *UNIV*])
apply (*simp*)
by (*simp add*: *fdstate-finite*)
have *f5*: ... = *1*/*6*
apply (*subst infsum-constant-finite-states*)
apply (*rule finite-subset*[**where** *B* = *UNIV*])
apply (*simp*)
apply (*simp add*: *fdstate-finite*)
apply (*subst infsum-constant-finite-states*)
apply (*rule finite-subset*[**where** *B* = *UNIV*])
apply (*simp*)
apply (*simp add*: *fdstate-finite*)
by (*simp add*: *lhs2-card lhs1-set-card*)

then show *real2ureal ?lhs* = *real2ureal* ((*1*::ℝ) / (*6*::ℝ))
using *f1 f2 f3 f4* **by** *presburger*
qed

lemma *fdice-throw-loop′*: *fdice-throw-loop* = *prfun-of-rvfun fH*
apply (*simp add*: *fdice-throw-loop-def*)
apply (*subst unique-fixed-point-lfp-gfp′*[**where** *fp* = *prfun-of-rvfun fH*])
using *fdice-throw-is-dist* **apply** *auto*[*1*]
apply (*subst finite-subset*[**where** *B* = *UNIV*])
apply *simp*
using *fdstate-finite finite-prod* **apply** *blast*
apply (*simp*)
using *fdice-throw-iterdiff-tendsto-0* **apply** (*simp*)
using *fH-is-fp* **apply** *blast*

by *simp*

5.1.6 Termination

lemma *fdice-throw-termination-prob*: $fH\ ;\ [\![fd1^< = fd2^<]\!]_{\mathcal{I}e} = (1)_e$
  apply (*simp add*: *fH-def*)
  apply (*expr-auto*)
proof −
  fix *fd1 fd2*
  have *f0*: $\{s::fdstate.\ fd1_v\ s = fd2 \wedge fd2_v\ s = fd2 \wedge fd1_v\ s = fd2_v\ s\} = \{(\!|fd1_v = fd2, fd2_v = fd2|\!)\}$
    apply (*subst set-eq-iff*)
    by (*expr-auto*)
  have $(\sum_\infty v_0::fdstate.\ (if\ fd1_v\ v_0 = fd2 \wedge fd2_v\ v_0 = fd2\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
      $(if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$
    $= (\sum_\infty v_0::fdstate.\ (if\ fd1_v\ v_0 = fd2 \wedge fd2_v\ v_0 = fd2 \wedge fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})))$
    apply (*rule infsum-cong*)
    by *auto*
  also have ... = *1*
    apply (*subst infsum-constant-finite-states*)
    using *fdstate-finite infinite-super subset-UNIV* **apply** *blast*
    by (*simp add*: *f0*)
  then show $(\sum_\infty v_0::fdstate.$
      $(if\ fd1_v\ v_0 = fd2 \wedge fd2_v\ v_0 = fd2\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
      $(if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R}))) = (1::\mathbb{R})$
    using *calculation* **by** *presburger*

  have *f1*: $(\sum_\infty v_0::fdstate.\ (if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
     $(if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) / (6::\mathbb{R}))$
   $= (\sum_\infty v_0::fdstate.\ (if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) / (6::\mathbb{R}))$
    apply (*rule infsum-cong*)
    by (*auto*)
  have *f2*: ... = *1*
    apply (*subst infsum-cdiv-left*)
    apply (*simp add*: *fdstate-finite*)
    apply (*subst infsum-constant-finite-states*)
    apply (*meson fdstate-finite rev-finite-subset top-greatest*)
    by (*simp add*: *fdstate-set-d1d2-eq-card*)

  then show $(\sum_\infty v_0::fdstate.\ (if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) *$
     $(if\ fd1_v\ v_0 = fd2_v\ v_0\ then\ 1::\mathbb{R}\ else\ (0::\mathbb{R})) / (6::\mathbb{R})) = (1::\mathbb{R})$
    using *f1* **by** *presburger*
qed

lemma *fdice-throw-nontermination-prob*: $fH\ ;\ [\![\neg fd1^< = fd2^<]\!]_{\mathcal{I}e} = (0)_e$
  apply (*simp add*: *fH-def*)
  apply (*expr-auto*)
  apply (*smt* (*verit*) *infsum-0 mult-not-zero*)
  by (*simp add*: *infsum-0*)

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