

Markov Decision Processes with Rewards

Maximilian Schäffeler and Mohammad Abdulaziz

February 6, 2026

Abstract

We present a formalization of Markov Decision Processes with rewards. In particular we first build on Hölzl's formalization [1] of MDPs and extend them with rewards. We proceed with an analysis of the expected total discounted reward criterion for infinite horizon MDPs. The central result is the construction of the iteration rule for the Bellman operator. We prove the optimality equations for this operator and show the existence of an optimal stationary deterministic solution. The analysis can be used to obtain dynamic programming algorithms such as value iteration and policy iteration to solve Markov Decision Processes with formal guarantees. Our formalization is based upon chapters 5 and 6 in Puterman's book [2].

Contents

| | | |
|----------|--|-----------|
| 1 | Bounded Functions | 3 |
| 1.1 | Definition | 3 |
| 1.2 | Supremum Norm | 5 |
| 1.3 | Complete Space | 7 |
| 1.4 | Order Instance | 8 |
| 1.5 | Miscellaneous | 9 |
| 1.6 | Bounded Functions and Vectors | 9 |
| 2 | Bounded Linear Functions | 10 |
| 2.1 | Composition | 10 |
| 2.2 | Power | 10 |
| 2.3 | Geometric Sum | 11 |
| 2.4 | Inverses | 12 |
| 2.5 | Norm | 14 |
| 2.6 | Miscellaneous | 15 |
| 3 | Auxiliary Lemmas | 18 |
| 3.1 | Summability | 18 |
| 3.2 | Infinite sums | 18 |
| 3.3 | Bounded Functions | 18 |
| 3.4 | Push-Forward of a Bounded Function | 19 |
| 3.5 | Boundedness | 19 |

| | | |
|----------|--|-----------|
| 3.6 | Probability Theory | 19 |
| 3.7 | Argmax | 20 |
| 3.8 | Contraction Mappings | 22 |
| 3.9 | Limits | 22 |
| 3.10 | Supremum | 22 |
| 4 | Least argmax | 23 |
| 5 | Discrete-Time Markov Decision Processes with Arbitrary State Spaces | 24 |
| 5.1 | Definition and Basic Properties | 24 |
| 5.2 | Policies | 26 |
| 5.3 | Successor Policy | 28 |
| 5.4 | Single-Step Distribution | 29 |
| 5.5 | Initial State-Action Distribution | 29 |
| 5.6 | Sequence Space of the MDP | 30 |
| 5.7 | Measurability of the Sequence Space | 31 |
| 5.8 | Iteration Rule | 31 |
| 5.9 | Stream Space of the MDP | 32 |
| 6 | Markov Decision Processes with Discrete State Spaces | 33 |
| 6.1 | Policies | 34 |
| 6.1.1 | Successor Policy | 36 |
| 6.2 | Stream Space of the MDP | 37 |
| 6.2.1 | Initial State-Action Distribution | 37 |
| 6.2.2 | Decomposition of the Stream Space | 38 |
| 6.2.3 | A Denotational View on the Stochastic Process | 39 |
| 6.2.4 | State Process | 40 |
| 6.2.5 | The Conditional Distribution of Actions | 41 |
| 6.2.6 | Action Process | 42 |
| 6.3 | Restriction to Markovian Policies | 42 |
| 6.4 | MDPs without Initial Distribution | 43 |
| 7 | Markov Decision Processes with Rewards | 46 |
| 7.1 | Util | 46 |
| 7.1.1 | Basic Properties of rewards | 46 |
| 7.1.2 | Infinite discounted sums | 47 |
| 7.2 | Total Reward for Single Traces | 47 |
| 7.3 | Expected Finite-Horizon Discounted Reward | 47 |
| 7.4 | Expected Total Discounted Reward | 48 |
| 7.5 | Reward of a Decision Rule | 48 |
| 7.6 | Transition Probability Matrix for MDPs | 49 |
| 7.7 | The Bellman Operator | 51 |
| 7.7.1 | Bellman Operator for Single Actions | 51 |
| 7.8 | Optimality Equations | 52 |
| 7.8.1 | Equivalences involving \mathcal{L}_b | 52 |
| 7.9 | Monotonicity | 53 |
| 7.10 | Optimal Reward | 56 |
| 7.11 | Properties of Solutions of the Optimality Equations | 58 |
| 7.12 | Solutions to the Optimality Equation | 59 |

| | | |
|--------|--|----|
| 7.12.1 | \mathcal{L}_b and L are Contraction Mappings | 59 |
| 7.12.2 | Existence of a Fixpoint of \mathcal{L}_b | 59 |
| 7.13 | Existence of Optimal Policies | 60 |
| 7.13.1 | Conserving Decision Rules are Optimal | 61 |
| 7.13.2 | Deterministic Decision Rules are Optimal | 61 |
| 7.13.3 | Optimal Decision Rules for Finite Action Spaces | 62 |
| 7.13.4 | Existence of Epsilon-Optimal Policies | 62 |
| 7.14 | More Restrictive MDP Locales | 64 |

1 Bounded Functions

theory *Bounded-Functions*

imports

HOL.Topological-Spaces

HOL-Analysis.Uniform-Limit

HOL-Probability.Probability

begin

1.1 Definition

definition *bfun* = {*f*. *bounded* (*range* *f*)}

typedef (**overloaded**) (*'a*, *'b*) *bfun* ($\langle(- \Rightarrow_b -)\rangle$ [22] 21) =

bfun::('a \Rightarrow 'b :: metric-space) set

morphisms *apply-bfun Bfun*

\langle *proof* \rangle

declare [*coercion* *apply-bfun* :: (*'a \Rightarrow_b ('b :: metric-space)*) \Rightarrow *'a \Rightarrow 'b*]]

setup-lifting *type-definition-bfun*

lemma *bounded-apply-bfun*[*intro*, *simp*]: *bounded* ((*apply-bfun* *x*) $\text{' } X$)
 \langle *proof* \rangle

lemma *apply-bfun-bdd-above*[*simp*, *intro*]:

fixes *f* :: *'c \Rightarrow_b real*

shows *bdd-above* (*f* $\text{' } X$)

\langle *proof* \rangle

lemma *bfun-eqI*[*intro*]: ($\bigwedge x$. *apply-bfun* *f* *x* = *apply-bfun* *g* *x*) \Longrightarrow *f* =
g

\langle *proof* \rangle

lemma *bfun-eqD*[*dest*]: *f* = *g* \Longrightarrow ($\bigwedge x$. *apply-bfun* *f* *x* = *apply-bfun* *g*
x)

\langle *proof* \rangle

lemma *bfunE*:

assumes $f \in \text{bfun}$
obtains g where $f = \text{apply-bfun } g$
 $\langle \text{proof} \rangle$

lemma *const-bfun*: $(\lambda x. b) \in \text{bfun}$
 $\langle \text{proof} \rangle$

lift-definition *const-bfun*:: $'b \Rightarrow ('a \Rightarrow_b ('b :: \text{metric-space}))$ **is** $\lambda(c::'b)$
 $\cdot. c$
 $\langle \text{proof} \rangle$

lemma *bounded-dist-le-SUP-dist*:
 $\text{bounded } (\text{range } f) \Longrightarrow \text{bounded } (\text{range } g) \Longrightarrow \text{dist } (f x) (g x) \leq (\text{SUP } x. \text{dist } (f x) (g x))$
 $\langle \text{proof} \rangle$

instantiation *bfun* :: $(\text{type}, \text{metric-space}) \text{ metric-space}$
begin

lift-definition *dist-bfun* :: $('a \Rightarrow_b 'b) \Rightarrow ('a \Rightarrow_b 'b) \Rightarrow \text{real}$
is $\lambda f g. (\text{SUP } x. \text{dist } (f x) (g x))$ $\langle \text{proof} \rangle$

definition *uniformity-bfun* :: $(('a \Rightarrow_b 'b) \times 'a \Rightarrow_b 'b) \text{ filter}$
where *uniformity-bfun* = $(\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x y < e\})$

definition *open-bfun* :: $('a \Rightarrow_b 'b) \text{ set} \Rightarrow \text{bool}$
where *open-bfun* $S = (\forall x \in S. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in S)$

lemma *dist-bounded*:
fixes $f g :: 'a \Rightarrow_b 'b$
shows $\text{dist } (f x) (g x) \leq \text{dist } f g$
 $\langle \text{proof} \rangle$

lemma *dist-bound*:
fixes $f g :: 'a \Rightarrow_b ('b :: \text{metric-space})$
assumes $\bigwedge x. \text{dist } (f x) (g x) \leq b$
shows $\text{dist } f g \leq b$
 $\langle \text{proof} \rangle$

lemma *dist-fun-lt-imp-dist-val-lt*:
fixes $f g :: 'a \Rightarrow_b 'b$
assumes $\text{dist } f g < e$
shows $\text{dist } (f x) (g x) < e$
 $\langle \text{proof} \rangle$

instance
 $\langle \text{proof} \rangle$

end

lift-definition $PiC::'a \text{ set} \Rightarrow ('a \Rightarrow ('b :: \text{metric-space}) \text{ set}) \Rightarrow ('a \Rightarrow_b 'b) \text{ set}$
is $\lambda I X. Pi I X \cap \text{bfun}$
<proof>

lemma $\text{mem-PiC-iff}: x \in PiC I X \longleftrightarrow \text{apply-bfun } x \in Pi I X$
<proof>

lemmas $\text{mem-PiCD} = \text{mem-PiC-iff}[THEN \text{iffD1}]$
and $\text{mem-PiCI} = \text{mem-PiC-iff}[THEN \text{iffD2}]$

lemma $\text{tendsto-bfun-uniform-limit}$:
fixes $f::'i \Rightarrow 'a \Rightarrow_b ('b :: \text{metric-space})$
assumes $(f \longrightarrow l) F$
shows $\text{uniform-limit UNIV } f \ l \ F$
<proof>

lemma $\text{uniform-limit-tendsto-bfun}$:
fixes $f::'i \Rightarrow 'a \Rightarrow_b ('b :: \text{metric-space})$
and $l::'a \Rightarrow_b 'b$
assumes $\text{uniform-limit UNIV } f \ l \ F$
shows $(f \longrightarrow l) F$
<proof>

1.2 Supremum Norm

instantiation $\text{bfun} :: (\text{type}, \text{real-normed-vector}) \text{ real-vector}$
begin

lemma $\text{uminus-cont}: f \in \text{bfun} \Longrightarrow (\lambda x. - f x) \in \text{bfun}$ **for** $f::'a \Rightarrow 'b$
<proof>

lemma $\text{plus-cont}: f \in \text{bfun} \Longrightarrow g \in \text{bfun} \Longrightarrow (\lambda x. f x + g x) \in \text{bfun}$
for $f g::'a \Rightarrow 'b$
<proof>

lemma $\text{minus-cont}: f \in \text{bfun} \Longrightarrow g \in \text{bfun} \Longrightarrow (\lambda x. f x - g x) \in \text{bfun}$
for $f g::'a \Rightarrow 'b$
<proof>

lemma $\text{scaleR-cont}: f \in \text{bfun} \Longrightarrow (\lambda x. a *_R f x) \in \text{bfun}$ **for** $f::'a \Rightarrow 'b$
<proof>

lemma $\text{bfun-normI}[intro]: (\bigwedge x. \text{norm } (f x) \leq b) \Longrightarrow f \in \text{bfun}$

<proof>

lift-definition *uminus-bfun*::('a \Rightarrow_b 'b) \Rightarrow ('a \Rightarrow_b 'b) **is** $\lambda f x. - f x$
<proof>

lift-definition *plus-bfun*::('a \Rightarrow_b 'b) \Rightarrow ('a \Rightarrow_b 'b) \Rightarrow 'a \Rightarrow_b 'b **is** $\lambda f g x. f x + g x$
<proof>

lift-definition *minus-bfun*::('a \Rightarrow_b 'b) \Rightarrow ('a \Rightarrow_b 'b) \Rightarrow 'a \Rightarrow_b 'b **is** $\lambda f g x. f x - g x$
<proof>

lift-definition *zero-bfun*::'a \Rightarrow_b 'b **is** $\lambda-. 0$
<proof>

lemma *const-bfun-0-eq-0[simp]*: *const-bfun 0 = 0*
<proof>

lift-definition *scaleR-bfun*::*real* \Rightarrow ('a \Rightarrow_b 'b) \Rightarrow 'a \Rightarrow_b 'b **is** $\lambda r g x. r *_R g x$
<proof>

lemmas [*simp*] =
const-bfun.rep-eq
uminus-bfun.rep-eq
plus-bfun.rep-eq
minus-bfun.rep-eq
zero-bfun.rep-eq
scaleR-bfun.rep-eq

instance
<proof>
end

lemma *scaleR-cont'*: $f \in \text{bfun} \Longrightarrow (\lambda x. a * f x) \in \text{bfun}$ **for** $f :: 'a \Rightarrow \text{real}$
<proof>

lemma *bfun-norm-le-SUP-norm*:
 $f \in \text{bfun} \Longrightarrow \text{norm} (f x) \leq (\text{SUP } x. \text{norm} (f x))$
<proof>

instantiation *bfun* :: (*type*, *real-normed-vector*) *real-normed-vector*
begin

definition *norm-bfun* :: ('a, 'b) *bfun* \Rightarrow *real*
where *norm-bfun f = dist f 0*

definition $sgn (f::('a,'b) bfun) = f /_R norm f$

instance

$\langle proof \rangle$

end

lemma $norm-bfun-def'$: $norm f = (\bigsqcup x. norm ((f :: 'a \Rightarrow_b 'b :: real-normed-vector) x))$

$\langle proof \rangle$

lemma $norm-le-norm-bfun$: $norm (apply-bfun f x) \leq norm f$

$\langle proof \rangle$

lemma $abs-le-norm-bfun$: $abs (apply-bfun f x) \leq norm f$

$\langle proof \rangle$

lemma $le-norm-bfun$: $apply-bfun f x \leq norm f$

$\langle proof \rangle$

1.3 Complete Space

lemma $tendsto-add$: $P \longrightarrow (L :: 'a :: real-normed-vector) \Longrightarrow (\lambda n.$

$P n + c) \longrightarrow L + c$

$\langle proof \rangle$

lemma $lim-add$: $convergent P \Longrightarrow lim (\lambda n. P n + (c :: 'a :: real-normed-vector))$

$= lim P + c$

$\langle proof \rangle$

lemma $complete-bfun$:

assumes $cauchy-f$: $Cauchy (f :: nat \Rightarrow ('a, 'b :: \{complete-space, real-normed-vector\}) bfun)$

shows $convergent f$

$\langle proof \rangle$

lemma $norm-bound$:

fixes $f :: ('a, 'b :: real-normed-vector) bfun$

assumes $\bigwedge x. norm (apply-bfun f x) \leq b$

shows $norm f \leq b$

$\langle proof \rangle$

lemma $bfun-bounded-norm-range$: $bounded (range (\lambda s. norm (apply-bfun v s)))$

$\langle proof \rangle$

instance $bfun :: (type, banach) banach$

$\langle proof \rangle$

lemma *bfun-prob-space-integrable*:
assumes *prob-space* S $v \in \text{borel-measurable } S$
assumes $(v :: 'a \Rightarrow 'b :: \{\text{second-countable-topology, banach}\}) \in \text{bfun}$
shows *integrable* S v
 $\langle \text{proof} \rangle$

lemma *bfun-integral-bound*:
assumes $(v :: 'a \Rightarrow 'c :: \{\text{euclidean-space}\}) \in \text{bfun}$
shows $(\lambda S. \int x. v \ x \ \partial(S :: 'a \text{ pmf})) \in \text{bfun}$
 $\langle \text{proof} \rangle$

lemma *scale-bfun[intro]*: $f \in \text{bfun} \implies (\lambda x. (k :: \text{real}) * f \ x) \in \text{bfun}$
 $\langle \text{proof} \rangle$

lemma *bfun-spec[intro]*: $f \in \text{bfun} \implies (\lambda x. f \ (g \ x)) \in \text{bfun}$
 $\langle \text{proof} \rangle$

lemma *apply-bfun-bfun[simp]*: *apply-bfun* $f \in \text{bfun}$
 $\langle \text{proof} \rangle$

lemma *bfun-integral-bound'[intro]*: $(v :: 'a \Rightarrow 'c :: \{\text{euclidean-space}\}) \in \text{bfun} \implies$
 $(\lambda S. \int x. v \ x \ \partial((F \ S) :: 'a \ \text{pmf})) \in \text{bfun}$
 $\langle \text{proof} \rangle$

lift-definition *bfun-comp* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow_b 'c :: \text{metric-space}) \Rightarrow$
 $('a \Rightarrow_b 'c)$ **is**
 $\lambda g \ \text{bf} \ x. \ \text{bf} \ (g \ x)$
 $\langle \text{proof} \rangle$

1.4 Order Instance

class *ordered-real-normed-vector* = *real-normed-vector* + *ordered-real-vector*

instance *real* :: *ordered-real-normed-vector*
 $\langle \text{proof} \rangle$

instantiation *bfun* :: $(-, \text{ordered-real-normed-vector}) \text{ ordered-real-normed-vector}$
begin

definition *less-eq-bfun* $f \ g \equiv \forall x. \ \text{apply-bfun} \ f \ x \leq \ \text{apply-bfun} \ g \ x$

definition *less-bfun* $f \ g \equiv \forall x. \ \text{apply-bfun} \ f \ x \leq \ \text{apply-bfun} \ g \ x \wedge (\exists y. \ f \ y < g \ y)$

instance
 $\langle \text{proof} \rangle$
end

lemma *less-eq-bfunI*[*intro*]: $(\bigwedge x. \text{apply-bfun } f \ x \leq \text{apply-bfun } g \ x) \implies f \leq g$
 ⟨*proof*⟩

lemma *less-eq-bfunD*[*dest*]: $f \leq g \implies (\bigwedge x. \text{apply-bfun } f \ x \leq \text{apply-bfun } g \ x)$
 ⟨*proof*⟩

1.5 Miscellaneous

instantiation *bfun* :: (*type*, *one*) *one* **begin**

lift-definition *one-bfun* :: '*s* \Rightarrow_b '*d*::{*metric-space*, *one*} **is** $\lambda x. 1$
 ⟨*proof*⟩

instance
 ⟨*proof*⟩
end

declare *one-bfun.rep-eq* [*simp*]

lemma *apply-bfun-one* [*simp*]: $\text{apply-bfun } (1 :: - \Rightarrow_b \text{real}) \ x = 1$
 ⟨*proof*⟩

lemma *norm-bfun-one*[*simp*]: $\text{norm } (1 :: 'a \Rightarrow_b \text{real}) = 1$
 ⟨*proof*⟩

lemma *range-bfunI*[*intro*]: $\text{bounded } (\text{range } f) \implies f \in \text{bfun}$
 ⟨*proof*⟩

lemma *finite-bfun*[*simp*]: $(\lambda(i:::\text{finite}). f \ i) \in \text{bfun}$
 ⟨*proof*⟩

lemma *bounded-apply-bfun'*:
assumes *bounded* ((*F* :: '*c* \Rightarrow '*d* \Rightarrow_b '*b*::*real-normed-vector*) '*S*)
shows *bounded* (($\lambda b. (F \ b) \ x$) '*S*)
 ⟨*proof*⟩

lemma *bfun-tendsto-apply-bfun*:
assumes *h*: (*F* :: (*nat* \Rightarrow '*a* \Rightarrow_b *real*)) \longrightarrow (*y* :: '*a* \Rightarrow_b *real*)
shows ($\lambda n. F \ n \ x$) \longrightarrow *y* *x*
 ⟨*proof*⟩

1.6 Bounded Functions and Vectors

lemma *vec-bfun*[*simp*, *intro*]: $(\$) \ x \in \text{bfun}$
 ⟨*proof*⟩

lemma *norm-bfun-le-norm-vec*: $\text{norm } (\text{bfun.Bfun } ((\$) (x :: \text{real}^c :: \text{finite}))) \leq \text{norm } x$
 ⟨proof⟩

lemma *bounded-linear-bfun-nth*: $\text{bounded-linear } f \implies \text{bounded-linear } (\lambda v. \text{bfun.Bfun } ((\$) (f v)))$
 ⟨proof⟩

lemma *norm-vec-le-norm-bfun*:
 $\text{norm } (\text{vec-lambda } (\text{apply-bfun } (x :: 'd::\text{finite} \Rightarrow_b \text{real}))) \leq \text{norm } x * \text{card } (\text{UNIV} :: 'd \text{ set})$
 ⟨proof⟩

end

2 Bounded Linear Functions

theory *Blinfun-Util*
imports
 HOL-Analysis.Bounded-Linear-Function
 Bounded-Functions
begin

2.1 Composition

lemma *blinfun-compose-id[simp]*:
 $\text{id-blinfun } o_L f = f$
 $f o_L \text{id-blinfun} = f$
 ⟨proof⟩

lemma *blinfun-compose-assoc*: $F o_L G o_L H = F o_L (G o_L H)$
 ⟨proof⟩

lemma *blinfun-compose-diff-right*: $f o_L (g - h) = (f o_L g) - (f o_L h)$
 ⟨proof⟩

2.2 Power

overloading
 $\text{blinfunpow} \equiv \text{compow} :: \text{nat} \Rightarrow ('a::\text{real-normed-vector} \Rightarrow_L 'a) \Rightarrow ('a \Rightarrow_L 'a)$
begin

primrec $\text{blinfunpow} :: \text{nat} \Rightarrow ('a::\text{real-normed-vector} \Rightarrow_L 'a) \Rightarrow ('a \Rightarrow_L 'a)$
where
 $\text{blinfunpow } 0 f = \text{id-blinfun}$
 $|\ \text{blinfunpow } (\text{Suc } n) f = f o_L \text{blinfunpow } n f$

end

lemma *bounded-pow-blinfun*[intro]:

assumes *bounded* (*range* ($F :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector} \Rightarrow_L 'a$))
shows *bounded* (*range* ($\lambda t. (F t) \hat{\sim} (\text{Suc } n)$))
<proof>

lemma *blincomp-scaleR-right*: ($a *_R (F :: 'a :: \text{real-normed-vector} \Rightarrow_L 'a)$) $\hat{\sim} t = a \hat{\sim} t *_R F \hat{\sim} t$

<proof>

lemma *summable-inv-Q*:

fixes $Q :: 'a :: \text{banach} \Rightarrow_L 'a$
assumes *onorm-le*: $\text{norm } (\text{id-blinfun} - Q) < 1$
shows *summable* ($\lambda n. (\text{id-blinfun} - Q) \hat{\sim} n$)
<proof>

lemma *blinfunpow-assoc*: ($F :: 'a :: \text{real-normed-vector} \Rightarrow_L 'a$) $\hat{\sim} (\text{Suc } n) = (F \hat{\sim} n) \circ_L F$
<proof>

lemma *norm-blinfunpow-le*: $\text{norm } ((f :: 'b :: \text{real-normed-vector} \Rightarrow_L 'b) \hat{\sim} n) \leq \text{norm } f \hat{\sim} n$
<proof>

lemma *blinfunpow-nonneg*:

assumes $\bigwedge v. 0 \leq v \implies 0 \leq \text{blinfun-apply } (f :: ('b :: \{\text{ord}, \text{real-normed-vector}\} \Rightarrow_L 'b)) v$
shows $0 \leq v \implies 0 \leq (f \hat{\sim} n) v$
<proof>

lemma *blinfunpow-mono*:

assumes $\bigwedge u v. u \leq v \implies (f :: 'b :: \{\text{ord}, \text{real-normed-vector}\} \Rightarrow_L 'b) u \leq f v$
shows $u \leq v \implies (f \hat{\sim} n) u \leq (f \hat{\sim} n) v$
<proof>

lemma *banach-blinfun*:

fixes $C :: 'b :: \{\text{real-normed-vector}, \text{complete-space}\} \Rightarrow_L 'b$
assumes *norm* $C < 1$
shows $\exists! v. C v = v \bigwedge v. (\lambda n. (C \hat{\sim} n) v) \longrightarrow (\text{THE } v. C v = v)$
<proof>

2.3 Geometric Sum

lemma *inv-one-sub-Q*:

fixes $Q :: 'a :: \text{banach} \Rightarrow_L 'a$
assumes *onorm-le*: $\text{norm } (\text{id-blinfun} - Q) < 1$
shows $(Q \circ_L (\sum i. (\text{id-blinfun} - Q) \hat{\sim} i)) = \text{id-blinfun}$
and $(\sum i. (\text{id-blinfun} - Q) \hat{\sim} i) \circ_L Q = \text{id-blinfun}$

$\langle proof \rangle$

lemma *inv-norm-le*:

fixes $Q :: 'a :: banach \Rightarrow_L 'a$

assumes $norm\ Q < 1$

shows $(id\text{-blinfun}\text{-}Q) \circ_L (\sum i. Q \hat{\sim} i) = id\text{-blinfun}$

$(\sum i. Q \hat{\sim} i) \circ_L (id\text{-blinfun}\text{-}Q) = id\text{-blinfun}$

$\langle proof \rangle$

lemma *inv-norm-le'*:

fixes $Q :: 'a :: banach \Rightarrow_L 'a$

assumes $norm\ Q < 1$

shows $(id\text{-blinfun}\text{-}Q) ((\sum i. Q \hat{\sim} i) x) = x$

$(\sum i. Q \hat{\sim} i) ((id\text{-blinfun}\text{-}Q) x) = x$

$\langle proof \rangle$

2.4 Inverses

definition $is\text{-inverse}_L\ X\ Y \iff X \circ_L Y = id\text{-blinfun} \wedge Y \circ_L X = id\text{-blinfun}$

abbreviation $invertible_L\ X \equiv \exists X'. is\text{-inverse}_L\ X\ X'$

lemma *is-inverse_L-I[intro]*:

assumes $X \circ_L Y = id\text{-blinfun}$ $Y \circ_L X = id\text{-blinfun}$

shows $is\text{-inverse}_L\ X\ Y$

$\langle proof \rangle$

lemma *is-inverse_L-D[dest]*:

assumes $is\text{-inverse}_L\ X\ Y$

shows $X \circ_L Y = id\text{-blinfun}$ $Y \circ_L X = id\text{-blinfun}$

$\langle proof \rangle$

lemma *invertible_L-D[dest]*:

assumes $invertible_L\ f$

obtains g **where** $f \circ_L g = id\text{-blinfun}$ $g \circ_L f = id\text{-blinfun}$

$\langle proof \rangle$

lemma *invertible_L-I[intro]*:

assumes $f \circ_L g = id\text{-blinfun}$ $g \circ_L f = id\text{-blinfun}$

shows $invertible_L\ f$

$\langle proof \rangle$

lemma *is-inverse_L-comm*: $is\text{-inverse}_L\ X\ Y \iff is\text{-inverse}_L\ Y\ X$

$\langle proof \rangle$

lemma *is-inverse_L-unique*: $is\text{-inverse}_L\ f\ g \implies is\text{-inverse}_L\ f\ h \implies g = h$

$\langle proof \rangle$

lemma *is-inverse_L-ex1*: $is_inverse_L f g \implies \exists! h. is_inverse_L f h$
 ⟨proof⟩

lemma *is-inverse_L-ex1'*: $\exists x. is_inverse_L f x \implies \exists! x. is_inverse_L f x$
 ⟨proof⟩

definition $inv_L f = (THE g. is_inverse_L f g)$

lemma *inv_L-eq*:
 assumes $is_inverse_L f g$
 shows $inv_L f = g$
 ⟨proof⟩

lemma *inv_L-I*:
 assumes $f o_L g = id_blinfun g o_L f = id_blinfun$
 shows $g = inv_L f$
 ⟨proof⟩

lemma *inv-app1[simp]*: $invertible_L X \implies (inv_L X) o_L X = id_blinfun$
 ⟨proof⟩

lemma *inv-app2[simp]*: $invertible_L X \implies X o_L (inv_L X) = id_blinfun$
 ⟨proof⟩

lemma *inv-app1'[simp]*: $invertible_L X \implies inv_L X (X v) = v$
 ⟨proof⟩

lemma *inv-app2'[simp]*: $invertible_L X \implies X (inv_L X v) = v$
 ⟨proof⟩

lemma *inv_L-inv_L[simp]*: $invertible_L X \implies inv_L (inv_L X) = X$
 ⟨proof⟩

lemma *inv_L-cancel-iff*:
 assumes $invertible_L f$
 shows $f x = y \iff x = inv_L f y$
 ⟨proof⟩

lemma *invertible_L-inf-sum*:
 assumes $norm (X :: 'b :: banach \Rightarrow_L 'b) < 1$
 shows $invertible_L (id_blinfun - X)$
 ⟨proof⟩

lemma *inv_L-inf-sum*:
 fixes $X :: 'b :: banach \Rightarrow_L -$
 assumes $norm X < 1$
 shows $inv_L (id_blinfun - X) = (\sum i. X \hat{=} i)$
 ⟨proof⟩

lemma *is-inverse_L-compose*:

assumes *invertible_L f invertible_L g*

shows *is-inverse_L (f o_L g) (inv_L g o_L inv_L f)*

<proof>

lemma *invertible_L-compose*: *invertible_L f \implies invertible_L g \implies invertible_L (f o_L g)*

<proof>

lemma *inv_L-compose*:

assumes *invertible_L f invertible_L g*

shows *inv_L (f o_L g) = (inv_L g) o_L (inv_L f)*

<proof>

lemma *inv_L-id-blinfun[simp]*: *inv_L id-blinfun = id-blinfun*

<proof>

2.5 Norm

lemma *bounded-range-subset*:

bounded (range f :: real set) \implies bounded (f ‘ X)

<proof>

lemma *bounded-const*: *bounded ((λ-. x) ‘ X)*

<proof>

lift-definition *bfun-pos* :: *(‘d \Rightarrow_b real) \Rightarrow (‘d \Rightarrow_b real) is λf i. if f i < 0 then -f i else f i*

<proof>

lemma *bfun-pos-zero[simp]*: *bfun-pos f = 0 \longleftrightarrow f = 0*

<proof>

lift-definition *bfun-nonneg* :: *(‘d \Rightarrow_b real) \Rightarrow (‘d \Rightarrow_b real) is λf i. if f i ≤ 0 then 0 else f i*

<proof>

lemma *bfun-nonneg-split*: *bfun-nonneg x - bfun-nonneg (- x) = x*

<proof>

lemma *blinfun-split*: *blinfun-apply f x = f (bfun-nonneg x) - f (bfun-nonneg (- x))*

<proof>

lemma *bfun-nonneg-pos*: *bfun-nonneg x + bfun-nonneg (-x) = bfun-pos x*

<proof>

lemma *bfun-nonneg*: $0 \leq \text{bfun-nonneg } f$
 ⟨proof⟩

lemma *bfun-pos-eq-nonneg*: $\text{bfun-pos } n = \text{bfun-nonneg } n + \text{bfun-nonneg } (-n)$
 ⟨proof⟩

lemma *blinfun-mono-norm-pos*:
fixes $f :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('d \Rightarrow_b \text{real})$
assumes $\bigwedge v :: 'c \Rightarrow_b \text{real}. v \geq 0 \implies f v \geq 0$
shows $\text{norm } (f n) \leq \text{norm } (f (\text{bfun-pos } n))$
 ⟨proof⟩

lemma *norm-bfun-pos[simp]*: $\text{norm } (\text{bfun-pos } f) = \text{norm } f$
 ⟨proof⟩

lemma *norm-blinfun-mono-eq-nonneg*:
fixes $f :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('d \Rightarrow_b \text{real})$
assumes $\bigwedge v :: 'c \Rightarrow_b \text{real}. v \geq 0 \implies f v \geq 0$
shows $\text{norm } f = (\bigsqcup v \in \{v. v \geq 0\}. \text{norm } (f v) / \text{norm } v)$
 ⟨proof⟩

lemma *norm-blinfun-normalized-le*: $\text{norm } (\text{blinfun-apply } f v) / \text{norm } v \leq \text{norm } f$
 ⟨proof⟩

lemma *norm-blinfun-mono-eq-nonneg'*:
fixes $f :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('d \Rightarrow_b \text{real})$
assumes $\bigwedge v :: 'c \Rightarrow_b \text{real}. 0 \leq v \implies 0 \leq f v$
shows $\text{norm } f = (\bigsqcup x \in \{x. \text{norm } x = 1 \wedge x \geq 0\}. \text{norm } (f x))$
 ⟨proof⟩

lemma *norm-blinfun-mono-le-norm-one*:
fixes $f :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('d \Rightarrow_b \text{real})$
assumes $\bigwedge v :: 'c \Rightarrow_b \text{real}. v \geq 0 \implies f v \geq 0$
assumes $\text{norm } x = 1 \implies 0 \leq x$
shows $\text{norm } (f x) \leq \text{norm } (f 1)$
 ⟨proof⟩

lemma *norm-blinfun-mono-eq-one*:
fixes $f :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('d \Rightarrow_b \text{real})$
assumes $\bigwedge v :: 'c \Rightarrow_b \text{real}. v \geq 0 \implies f v \geq 0$
shows $\text{norm } f = \text{norm } (f 1)$
 ⟨proof⟩

2.6 Miscellaneous

lemma *bounded-linear-apply-bfun*: *bounded-linear* $(\lambda x. \text{apply-bfun } x i)$
 ⟨proof⟩

lemma *lim-blinfun-apply: convergent* $X \implies (\lambda n. \text{blinfun-apply } (X \ n) \ u) \longrightarrow \text{lim } X \ u$
 ⟨proof⟩

lemma *bounded-apply-blinfun:*
assumes *bounded* $((F :: 'c \Rightarrow 'd::\text{real-normed-vector} \Rightarrow_L 'b::\text{real-normed-vector})$
 $' S)$
shows *bounded* $((\lambda b. \text{blinfun-apply } (F \ b) \ x) ' S)$
 ⟨proof⟩

lemma *tendsto-blinfun-apply:* $(\lambda n. X \ n) \longrightarrow L \implies (\lambda n. \text{blinfun-apply } (X \ n) \ u) \longrightarrow L \ u$
 ⟨proof⟩

definition *nonneg-blinfun* $(Q :: -::\{\text{ordered-real-normed-vector}\} \Rightarrow_L -::\{\text{ordered-ab-group-add, ordered-real-normed-vector}\}) \equiv (\forall v \geq 0. \text{blinfun-apply } Q \ v \geq 0)$

definition *blinfun-le* $Q \ R = \text{nonneg-blinfun } (R - Q)$

lemma *nonneg-blinfun-nonneg[dest]:* *nonneg-blinfun* $Q \implies 0 \leq v \implies 0 \leq Q \ v$
 ⟨proof⟩

lemma *nonneg-blinfun-mono[dest]:* *nonneg-blinfun* $Q \implies u \leq v \implies Q \ u \leq Q \ v$
 ⟨proof⟩

lemma *nonneg-id-blinfun:* *nonneg-blinfun* *id-blinfun*
 ⟨proof⟩

lemma *blinfun-nonneg-eq:*
assumes $\forall v \geq 0. \text{blinfun-apply } (f::('c \Rightarrow_b \text{real}) \Rightarrow_L ('c \Rightarrow_b \text{real})) \ v = \text{blinfun-apply } g \ v$
shows $f = g$
 ⟨proof⟩

lemma *bfun-zero-le-one:* $0 \leq (1 :: 'c \Rightarrow_b \text{real})$
 ⟨proof⟩

lemma *norm-nonneg-blinfun-one:*
assumes *nonneg-blinfun* $(X :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('c \Rightarrow_b \text{real}))$
shows $\text{norm } X = \text{norm } (\text{blinfun-apply } X \ 1)$

⟨proof⟩

lemma *blinfun-apply-mono*: $\text{nonneg-blinfun } X \implies 0 \leq v \implies \text{blinfun-le } X \ Y \implies X \ v \leq Y \ v$
⟨proof⟩

lemma *nonneg-blinfun-scaleR[intro]*: $\text{nonneg-blinfun } B \implies 0 \leq c \implies \text{nonneg-blinfun } (c *_{\mathbb{R}} B)$
⟨proof⟩

lemma *nonneg-blinfun-compose[intro]*: $\text{nonneg-blinfun } B \implies \text{nonneg-blinfun } C \implies \text{nonneg-blinfun } (C \circ_L B)$
⟨proof⟩

lemma *matrix-le-norm-mono*:
assumes $\text{nonneg-blinfun } (C :: ('c \Rightarrow_b \text{real}) \Rightarrow_L ('c \Rightarrow_b \text{real}))$
and $\text{nonneg-blinfun } (D - C)$
shows $\text{norm } C \leq \text{norm } D$
⟨proof⟩

lemma *bounded-subset*: $Y \subseteq X \implies \text{bounded } (f \text{ ` } X) \implies \text{bounded } (f \text{ ` } Y)$
⟨proof⟩

lemma *bounded-subset-range*: $\text{bounded } (\text{range } f) \implies \text{bounded } (f \text{ ` } Y)$
⟨proof⟩

lift-definition *bfun-if* :: $('b \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow_b 'c :: \text{metric-space}) \Rightarrow ('b \Rightarrow_b 'c) \Rightarrow ('b \Rightarrow_b 'c)$ **is** $\lambda b \ u \ v \ s. \text{if } b \ s \ \text{then } u \ s \ \text{else } v \ s$
⟨proof⟩

lemma *bfun-if-add*: $\text{bfun-if } b \ (w + z) \ (u + v) = \text{bfun-if } b \ w \ u + \text{bfun-if } b \ z \ v$
⟨proof⟩

lemma *bfun-if-zero-add*: $\text{bfun-if } b \ 0 \ (u + v) = \text{bfun-if } b \ 0 \ u + \text{bfun-if } b \ 0 \ v$
⟨proof⟩

lemma *bfun-if-zero-le*: $0 \leq v \implies \text{bfun-if } b \ 0 \ v \leq v$
⟨proof⟩

lemma *bfun-if-eq*: $(\bigwedge i. P \ i \implies \text{apply-bfun } v \ i = \text{apply-bfun } u \ i) \implies (\bigwedge i. \neg P \ i \implies v \ i = \text{apply-bfun } w \ i) \implies \text{bfun-if } P \ u \ w = v$

$\langle proof \rangle$

lemma *bfun-if-scaleR*: $c *_{\mathbb{R}} \text{bfun-if } b \ v1 \ v2 = \text{bfun-if } b \ (c *_{\mathbb{R}} \ v1) \ (c *_{\mathbb{R}} \ v2)$
 $\langle proof \rangle$

lemma *summable-blinfun-apply*:
assumes *summable* $(f :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector} \Rightarrow_L 'a)$
shows *summable* $(\lambda n. f \ n \ v)$
 $\langle proof \rangle$

lemma *blinfun-apply-suminf*:
assumes *summable* $(f :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector} \Rightarrow_L 'a)$
shows $(\sum k. \text{blinfun-apply } (f \ k) \ v) = (\sum k. f \ k) \ v$
 $\langle proof \rangle$
end
theory *MDP-reward-Util*
imports *Blinfun-Util*
begin

3 Auxiliary Lemmas

3.1 Summability

lemma *summable-powser-const*:
fixes $c :: \text{real}$
assumes $|c| < 1$
shows *summable* $(\lambda n. c^{\wedge} n * x)$
 $\langle proof \rangle$

3.2 Infinite sums

lemma *suminf-split-head'*:
summable $(f :: \text{nat} \Rightarrow 'x :: \text{real-normed-vector}) \implies \text{suminf } f = f \ 0$
 $+ (\sum n. f \ (\text{Suc } n))$
 $\langle proof \rangle$

lemma *sum-disc-lim*:
assumes $|c :: \text{real}| < 1$
shows $(\sum x. c^{\wedge} x * B) = B / (1 - c)$
 $\langle proof \rangle$

3.3 Bounded Functions

lemma *suminf-apply-bfun*:
fixes $f :: \text{nat} \Rightarrow 'c \Rightarrow_b \text{real}$

assumes *summable f*
shows $(\sum i. f i) x = (\sum i. f i x)$
 $\langle proof \rangle$

lemma *sum-apply-bfun*:
fixes $f :: nat \Rightarrow 'c \Rightarrow_b real$
shows $(\sum i < n. f i) x = (\sum i < n. apply-bfun f i) x$
 $\langle proof \rangle$

3.4 Push-Forward of a Bounded Function

lemma *integrable-bfun-prob-space [simp]*:
integrable (measure-pmf P) ($\lambda t. apply-bfun f (F t) :: real$)
 $\langle proof \rangle$

lift-definition *push-exp* :: $('b \Rightarrow 'c \text{ pmf}) \Rightarrow ('c \Rightarrow_b real) \Rightarrow ('b \Rightarrow_b real)$ **is**
 $\lambda c f s. measure-pmf.expectation (c s) f$
 $\langle proof \rangle$

declare *push-exp.rep-eq*[*simp*]

lemma *norm-push-exp-le-norm*: $norm (push-exp d x) \leq norm x$
 $\langle proof \rangle$

lemma *push-exp-bounded-linear [simp]*: *bounded-linear (push-exp d)*
 $\langle proof \rangle$

lemma *onorm-push-exp [simp]*: *onorm (push-exp d) = 1*
 $\langle proof \rangle$

lemma *push-exp-return*[*simp*]: *push-exp return-pmf = id*
 $\langle proof \rangle$

3.5 Boundedness

lemma *bounded-abs*[*intro*]:
bounded (X' :: real set) \implies bounded (abs ' X')
 $\langle proof \rangle$

lemma *bounded-abs-range*[*intro*]:
bounded (range f :: real set) \implies bounded (range ($\lambda x. abs (f x)$))
 $\langle proof \rangle$

3.6 Probability Theory

lemma *integral-measure-pmf-bind*:
assumes $(\bigwedge x. |f :: 'b \Rightarrow real| x| \leq B)$
shows $(\int x. f x \partial((measure-pmf M) \gg (\lambda x. measure-pmf (N x))))$
 $= (\int x. \int y. f y \partial N x \partial M)$

$\langle \text{proof} \rangle$

lemma *lemma-4-3-1'*:

assumes *set-pmf* $p \subseteq W$

and *bounded* $((w :: 'c \Rightarrow \text{real}) \text{ ' } W)$

and $W \neq \{\}$

and *measure-pmf.expectation* $p \ w = (\bigsqcup p \in \{p. \text{set-pmf } p \subseteq W\}.$

measure-pmf.expectation $p \ w)$

shows $\exists x \in W. \text{measure-pmf.expectation } p \ w = w \ x$

$\langle \text{proof} \rangle$

lemma *lemma-4-3-1*:

assumes *set-pmf* $p \subseteq W$ *integrable* $(\text{measure-pmf } p) \ w$ *bounded* $((w :: 'c \Rightarrow \text{real}) \text{ ' } W)$

shows *measure-pmf.expectation* $p \ w \leq \bigsqcup (w \text{ ' } W)$

$\langle \text{proof} \rangle$

lemma *bounded-integrable*:

assumes *bounded* $(\text{range } v) \ v \in \text{borel-measurable } (\text{measure-pmf } p)$

shows *integrable* $(\text{measure-pmf } p) \ (v :: 'c \Rightarrow \text{real})$

$\langle \text{proof} \rangle$

3.7 Argmax

lemma *finite-is-arg-max*: *finite* $X \Longrightarrow X \neq \{\} \Longrightarrow \exists x. \text{is-arg-max } (f :: 'c \Rightarrow \text{real}) \ (\lambda x. x \in X) \ x$

$\langle \text{proof} \rangle$

lemma *finite-arg-max-le*:

assumes *finite* $(X :: 'c \text{ set}) \ X \neq \{\}$

shows $s \in X \Longrightarrow (f :: 'c \Rightarrow \text{real}) \ s \leq f (\text{arg-max-on } (f :: 'c \Rightarrow \text{real}) \ X)$

$\langle \text{proof} \rangle$

lemma *arg-max-on-in*:

assumes *finite* $(X :: 'c \text{ set}) \ X \neq \{\}$

shows $(\text{arg-max-on } (f :: 'c \Rightarrow \text{real}) \ X) \in X$

$\langle \text{proof} \rangle$

lemma *finite-arg-max-eq-Max*:

assumes *finite* $(X :: 'c \text{ set}) \ X \neq \{\}$

shows $(f :: 'c \Rightarrow \text{real}) \ (\text{arg-max-on } f \ X) = \text{Max } (f \text{ ' } X)$

$\langle \text{proof} \rangle$

lemma *arg-max-SUP*: *is-arg-max* $(f :: 'b \Rightarrow \text{real}) \ (\lambda x. x \in X) \ m \Longrightarrow f \ m = (\bigsqcup (f \text{ ' } X))$

$\langle \text{proof} \rangle$

definition *has-max* $X \equiv \exists x \in X. \forall x' \in X. x' \leq x$

definition *has-arg-max* $f X \equiv \exists x. \text{is-arg-max } f (\lambda x. x \in X) x$

lemma *has-max* $((f :: 'b \Rightarrow \text{real}) ' X) \longleftrightarrow \text{has-arg-max } f X$
<proof>

lemma *has-arg-max-is-arg-max*: *has-arg-max* $f X \Longrightarrow \text{is-arg-max } f$
 $(\lambda x. x \in X) (\text{arg-max } f (\lambda x. x \in X))$
<proof>

lemma *has-arg-max-arg-max*: *has-arg-max* $f X \Longrightarrow (\text{arg-max } f (\lambda x. x \in X)) \in X$
<proof>

lemma *app-arg-max-ge*: *has-arg-max* $(f :: 'b \Rightarrow \text{real}) X \Longrightarrow x \in X$
 $\Longrightarrow f x \leq f (\text{arg-max-on } f X)$
<proof>

lemma *app-arg-max-eq-SUP*: *has-arg-max* $(f :: 'b \Rightarrow \text{real}) X \Longrightarrow f$
 $(\text{arg-max-on } f X) = \bigsqcup (f ' X)$
<proof>

lemma *SUP-is-arg-max*:
assumes $x \in X$ *bdd-above* $(f ' X) (f :: 'c \Rightarrow \text{real}) x = \bigsqcup (f ' X)$
shows *is-arg-max* $f (\lambda x. x \in X) x$
<proof>

lemma *is-arg-max-linorderI*[*intro*]: **fixes** $f :: 'c \Rightarrow 'b :: \text{linorder}$
assumes $P x \wedge y. (P y \Longrightarrow f x \geq f y)$
shows *is-arg-max* $f P x$
<proof>

lemma *is-arg-max-linorderD*[*dest*]: **fixes** $f :: 'c \Rightarrow 'b :: \text{linorder}$
assumes *is-arg-max* $f P x$
shows $P x (P y \Longrightarrow f x \geq f y)$
<proof>

lemma *is-arg-max-cong*:
assumes $\bigwedge x. P x \Longrightarrow f x = g x$
shows *is-arg-max* $f P x \longleftrightarrow \text{is-arg-max } g P x$
<proof>

lemma *is-arg-max-cong'*:
assumes $\bigwedge x. P x \Longrightarrow f x = g x$
shows *is-arg-max* $f P = \text{is-arg-max } g P$
<proof>

lemma *is-arg-max-congI*:

assumes $is\text{-arg-max } f P x \wedge x. P x \implies f x = g x$
shows $is\text{-arg-max } g P x$
 $\langle proof \rangle$

3.8 Contraction Mappings

definition $is\text{-contraction } C \equiv \exists l. 0 \leq l \wedge l < 1 \wedge (\forall v u. dist (C v) (C u) \leq l * dist v u)$

lemma $banach'$:

fixes $C :: 'b :: complete\text{-space} \Rightarrow 'b$
assumes $is\text{-contraction } C$
shows $\exists! v. C v = v \wedge v. (\lambda n. (C \overset{\sim}{\sim} n) v) \longrightarrow (THE v. C v = v)$
 $\langle proof \rangle$

lemma $contraction\text{-dist}$:

fixes $C :: 'b :: complete\text{-space} \Rightarrow 'b$
assumes $\wedge v u. dist (C v) (C u) \leq c * dist v u$
assumes $0 \leq c < 1$
shows $(1 - c) * dist v (THE v. C v = v) \leq dist v (C v)$
 $\langle proof \rangle$

3.9 Limits

lemma $tendsto\text{-bfun-sandwich}$:

assumes
 $(f :: nat \Rightarrow 'b \Rightarrow_b real) \longrightarrow x (g :: nat \Rightarrow 'b \Rightarrow_b real) \longrightarrow x$
 $eventually (\lambda n. f n \leq h n) sequentially eventually (\lambda n. h n \leq g n)$
 $sequentially$
shows $(h :: nat \Rightarrow 'b \Rightarrow_b real) \longrightarrow x$
 $\langle proof \rangle$

3.10 Supremum

lemma $SUP\text{-add-le}$:

assumes $X \neq \{\}$ $bounded (B ' X) bounded (A' ' X)$
shows $(\bigsqcup c \in X. (B :: 'a \Rightarrow real) c + A' c) \leq (\bigsqcup b \in X. B b) + (\bigsqcup a \in X. A' a)$
 $\langle proof \rangle$

lemma $le\text{-SUP-diff}'$:

assumes $ne: X \neq \{\}$
and $bdd: bounded (B ' X) bounded (A' ' X)$
and $sup\text{-le}: (\bigsqcup a \in X. (A' :: 'a \Rightarrow real) a) \leq (\bigsqcup b \in X. B b)$
shows $(\bigsqcup b \in X. B b) - (\bigsqcup a \in X. (A' :: 'a \Rightarrow real) a) \leq (\bigsqcup c \in X. B c - A' c)$
 $\langle proof \rangle$

lemma $le\text{-SUP-diff}$:

fixes $A' :: 'a \Rightarrow real$

assumes $X \neq \{\}$ *bounded* $(B \text{ ' } X)$ *bounded* $(A' \text{ ' } X)$ $(\bigsqcup a \in X. A' a) \leq (\bigsqcup b \in X. B b)$
shows $0 \leq (\bigsqcup c \in X. B c - A' c)$
 $\langle \text{proof} \rangle$

lemma *bounded-SUP-mul[simp]*:
 $X \neq \{\} \implies 0 \leq l \implies \text{bounded } (f \text{ ' } X) \implies (\bigsqcup x \in X. (l :: \text{real}) * f x) = (l * (\bigsqcup x \in X. f x))$
 $\langle \text{proof} \rangle$

lemma *abs-cSUP-le[intro]*:
 $X \neq \{\} \implies \text{bounded } (F \text{ ' } X) \implies |\bigsqcup x \in X. (F x) :: \text{real}| \leq (\bigsqcup x \in X. |F x|)$
 $\langle \text{proof} \rangle$

4 Least argmax

definition *least-arg-max* $f P = (\text{LEAST } x. \text{is-arg-max } f P x)$

lemma *least-arg-max-prop*: $\exists x::'a::\text{wellorder}. P x \implies \text{finite } \{x. P x\} \implies P (\text{least-arg-max } (f :: - \Rightarrow \text{real}) P)$
 $\langle \text{proof} \rangle$

lemma *is-arg-max-apply-eq*: *is-arg-max* $(f :: - \Rightarrow - :: \text{linorder}) P x \implies \text{is-arg-max } f P y \implies f x = f y$
 $\langle \text{proof} \rangle$

lemma *least-arg-max-apply*:
assumes *is-arg-max* $(f :: - \Rightarrow - :: \text{linorder}) P (x:::\text{wellorder})$
shows $f (\text{least-arg-max } f P) = f x$
 $\langle \text{proof} \rangle$

lemma *apply-arg-max-eq-max*: *finite* $\{x. P x\} \implies \text{is-arg-max } (f :: - \Rightarrow - :: \text{linorder}) P x \implies f x = \text{Max } (f \text{ ' } \{x. P x\})$
 $\langle \text{proof} \rangle$

lemma *apply-arg-max-eq-max'*: *finite* $X \implies \text{is-arg-max } (f :: - \Rightarrow - :: \text{linorder}) (\lambda x. x \in X) x \implies (\text{MAX } x \in X. f x) = f x$
 $\langle \text{proof} \rangle$

lemma *least-arg-max-is-arg-max*: $P \neq \{\} \implies \text{finite } P \implies \text{is-arg-max } f (\lambda x:::\text{wellorder}. x \in P) (\text{least-arg-max } (f :: - \Rightarrow \text{real}) (\lambda x. x \in P))$
 $\langle \text{proof} \rangle$

lemma *is-arg-max-const*: *is-arg-max* $(f :: - \Rightarrow - :: \text{linorder}) (\lambda y. y = c) x \longleftrightarrow x = c$
 $\langle \text{proof} \rangle$

lemma *least-arg-max-cong'*:

```

assumes  $\bigwedge x. \text{is-arg-max } f \ P \ x = \text{is-arg-max } g \ P \ x$ 
shows  $\text{least-arg-max } f \ P = \text{least-arg-max } g \ P$ 
<proof>

```

end

5 Discrete-Time Markov Decision Processes with Arbitrary State Spaces

In this file we construct discrete-time Markov decision processes, e.g. with arbitrary state spaces. Proofs and definitions are adapted from `Markov_Models.Discrete_Time_Markov_Process`.

```

theory MDP-cont
imports HOL-Probability.Probability
begin

```

```

lemma Ionescu-Tulcea-C-eq:
assumes  $\bigwedge i \ h. h \in \text{space } (PiM \ \{0..<i\} \ N) \implies P \ i \ h = P' \ i \ h$ 
assumes  $h: \text{Ionescu-Tulcea } P \ N \ \text{Ionescu-Tulcea } P' \ N$ 
shows  $\text{Ionescu-Tulcea.C } P \ N \ 0 \ n \ (\lambda x. \text{undefined}) = \text{Ionescu-Tulcea.C } P' \ N \ 0 \ n \ (\lambda x. \text{undefined})$ 
<proof>

```

```

lemma Ionescu-Tulcea-CI-eq:
assumes  $\bigwedge i \ h. h \in \text{space } (PiM \ \{0..<i\} \ N) \implies P \ i \ h = P' \ i \ h$ 
assumes  $h: \text{Ionescu-Tulcea } P \ N \ \text{Ionescu-Tulcea } P' \ N$ 
shows  $\text{Ionescu-Tulcea.CI } P \ N = \text{Ionescu-Tulcea.CI } P' \ N$ 
<proof>

```

```

lemma measure-eqI-PiM-sequence:
fixes  $M :: \text{nat} \Rightarrow 'a \ \text{measure}$ 
assumes  $*[simp]: \text{sets } P = PiM \ UNIV \ M \ \text{sets } Q = PiM \ UNIV \ M$ 
assumes  $eq: \bigwedge A \ n. (\bigwedge i. A \ i \in \text{sets } (M \ i)) \implies$ 
 $P \ (\text{prod-emb } UNIV \ M \ \{..n\} \ (Pi_E \ \{..n\} \ A)) = Q \ (\text{prod-emb } UNIV$ 
 $M \ \{..n\} \ (Pi_E \ \{..n\} \ A))$ 
assumes  $A: \text{finite-measure } P$ 
shows  $P = Q$ 
<proof>

```

```

lemma distr-cong-simp:
 $M = K \implies \text{sets } N = \text{sets } L \implies (\bigwedge x. x \in \text{space } M = \text{simp} \implies f \ x =$ 
 $g \ x) \implies \text{distr } M \ N \ f = \text{distr } K \ L \ g$ 
<proof>

```

5.1 Definition and Basic Properties

```

locale discrete-MDP =

```

fixes $Ms :: 's \text{ measure}$
and $Ma :: 'a \text{ measure}$
and $A :: 's \Rightarrow 'a \text{ set}$
and $K :: 's \times 'a \Rightarrow 's \text{ measure}$

assumes $A\text{-}s: \bigwedge s. A \ s \in \text{sets } Ma$

assumes $A\text{-}ne: \bigwedge s. A \ s \neq \{\}$

assumes $ex\text{-}pol: \exists \delta \in Ms \rightarrow_M Ma. \forall s. \delta \ s \in A \ s$

assumes $K[\text{measurable}]: K \in Ms \otimes_M Ma \rightarrow_M \text{prob-algebra } Ms$
begin

lemma $space\text{-}prodI[\text{intro}]: x \in \text{space } A' \Longrightarrow y \in \text{space } B \Longrightarrow (x,y) \in \text{space } (A' \otimes_M B)$
 $\langle \text{proof} \rangle$

abbreviation $M \equiv Ms \otimes_M Ma$
abbreviation $Ma\text{-}A \ s \equiv \text{restrict-space } Ma \ (A \ s)$

lemma $space\text{-}ma[\text{intro}]: s \in \text{space } Ms \Longrightarrow a \in \text{space } Ma \Longrightarrow (s,a) \in \text{space } M$
 $\langle \text{proof} \rangle$

lemma $space\text{-}x0[\text{simp}]: x0 \in \text{space } (\text{prob-algebra } Ms) \Longrightarrow \text{space } x0 = \text{space } Ms$
 $\langle \text{proof} \rangle$

lemma $A\text{-}subs\text{-}Ma: A \ s \subseteq \text{space } Ma$
 $\langle \text{proof} \rangle$

lemma $space\text{-}Ma\text{-}A\text{-}subset: s \in \text{space } Ms \Longrightarrow \text{space } (Ma\text{-}A \ s) \subseteq A \ s$
 $\langle \text{proof} \rangle$

lemma $K\text{-}restrict \ [\text{measurable}]: K \in (Ms \otimes_M Ma\text{-}A \ s) \rightarrow_M \text{prob-algebra } Ms$
 $\langle \text{proof} \rangle$

lemma $measurable\text{-}K\text{-}act[\text{measurable}, \text{intro}]: s \in \text{space } Ms \Longrightarrow (\lambda a. K \ (s, a)) \in Ma \rightarrow_M \text{prob-algebra } Ms$
 $\langle \text{proof} \rangle$

lemma $measurable\text{-}K\text{-}st[\text{measurable}, \text{intro}]: a \in \text{space } Ma \Longrightarrow (\lambda s. K \ (s, a)) \in Ms \rightarrow_M \text{prob-algebra } Ms$
 $\langle \text{proof} \rangle$

lemma $space\text{-}K[\text{simp}]: sa \in \text{space } M \Longrightarrow \text{space } (K \ sa) = \text{space } Ms$
 $\langle \text{proof} \rangle$

lemma *space-K2[simp]*: $s \in \text{space } Ms \implies a \in \text{space } Ma \implies \text{space } (K (s, a)) = \text{space } Ms$
 ⟨proof⟩

lemma *space-K-E*: $s' \in \text{space } (K (s, a)) \implies s \in \text{space } Ms \implies a \in \text{space } Ma \implies s' \in \text{space } Ms$
 ⟨proof⟩

lemma *sets-K*: $sa \in \text{space } M \implies \text{sets } (K sa) = \text{sets } Ms$
 ⟨proof⟩

lemma *sets-K'[measurable-cong]*: $s \in \text{space } Ms \implies a \in \text{space } Ma \implies \text{sets } (K (s, a)) = \text{sets } Ms$
 ⟨proof⟩

lemma *prob-space-K[intro]*: $sa \in \text{space } M \implies \text{prob-space } (K sa)$
 ⟨proof⟩

lemma *prob-space-K2[intro]*: $s \in \text{space } Ms \implies a \in \text{space } Ma \implies \text{prob-space } (K (s, a))$
 ⟨proof⟩

lemma *K-in-space [intro]*: $m \in \text{space } M \implies K m \in \text{space } (\text{prob-algebra } Ms)$
 ⟨proof⟩

5.2 Policies

type-synonym $('c, 'd) \text{ pol} = \text{nat} \Rightarrow ((\text{nat} \Rightarrow 'c \times 'd) \times 'c) \Rightarrow 'd$
measure

abbreviation $H \ i \equiv Pi_M \ \{0..<i\} \ (\lambda-. M)$

abbreviation $Hs \ i \equiv H \ i \ \otimes_M \ Ms$

lemma *space-H1*: $j < (i :: \text{nat}) \implies \omega \in \text{space } (H \ i) \implies \omega \ j \in \text{space } M$
 ⟨proof⟩

lemma *space-case-nat[intro]*:
assumes $\omega \in \text{space } (H \ i) \ s \in \text{space } Ms$
shows $\text{case-nat } s \ (\text{fst} \circ \omega) \ i \in \text{space } Ms$
 ⟨proof⟩

lemma *undefined-in-H0*: $(\lambda-. \text{undefined}) \in \text{space } (H \ (0 :: \text{nat}))$
 ⟨proof⟩

lemma *sets-K-Suc*[*measurable-cong*]: $h \in \text{space } (H \text{ (Suc } n)) \implies \text{sets } (K \text{ (h } n)) = \text{sets } Ms$
 ⟨*proof*⟩

A decision rule is a function from states to distributions over enabled actions.

definition *is-dec0* $d \equiv d \in Ms \rightarrow_M \text{prob-algebra } Ma$

definition *is-dec* ($t :: \text{nat}$) $d \equiv (d \in Hs \ t \rightarrow_M \text{prob-algebra } Ma)$

lemma *is-dec0* $d \implies \text{is-dec } t \ (\lambda(-,s). \ d \ s)$
 ⟨*proof*⟩

A policy is a function from histories to valid decision rules.

definition *is-policy* :: $('s, 'a) \text{pol} \Rightarrow \text{bool}$ **where**
is-policy $p \equiv \forall i. \text{is-dec } i \ (p \ i)$

abbreviation *p0* :: $('s, 'a) \text{pol} \Rightarrow 's \Rightarrow 'a \text{ measure}$ **where**
p0 $p \ s \equiv p \ (0 :: \text{nat}) \ (\lambda-. \ \text{undefined}, \ s)$

context

fixes p **assumes** $p[\text{simp}]$: *is-policy* p

begin

lemma *is-policyD*[*measurable*]: $p \ i \in Hs \ i \rightarrow_M \text{prob-algebra } Ma$
 ⟨*proof*⟩

lemma *space-policy*[*simp*]: $hs \in \text{space } (Hs \ i) \implies \text{space } (p \ i \ hs) = \text{space } Ma$
 ⟨*proof*⟩

lemma *space-policy'*[*simp*]: $h \in \text{space } (H \ i) \implies s \in \text{space } Ms \implies \text{space } (p \ i \ (h,s)) = \text{space } Ma$
 ⟨*proof*⟩

lemma *space-policyI*[*intro*]:
assumes $s \in \text{space } Ms \ h \in \text{space } (H \ i) \ a \in \text{space } Ma$
shows $a \in \text{space } (p \ i \ (h,s))$
 ⟨*proof*⟩

lemma *sets-policy*[*simp*]: $hs \in \text{space } (Hs \ i) \implies \text{sets } (p \ i \ hs) = \text{sets } Ma$
 ⟨*proof*⟩

lemma *sets-policy'*[*measurable-cong, simp*]:
 $h \in \text{space } (H \ i) \implies s \in \text{space } Ms \implies \text{sets } (p \ i \ (h,s)) = \text{sets } Ma$
 ⟨*proof*⟩

lemma *sets-policy'*[*measurable-cong, simp*]:
 $h \in \text{space } ((Pi_M \{ \} (\lambda \cdot M))) \implies s \in \text{space } Ms \implies \text{sets } (p \ 0 \ (h,s))$
 $= \text{sets } Ma$
 ⟨*proof*⟩

lemma *policy-prob-space*: $hs \in \text{space } (Hs \ i) \implies \text{prob-space } (p \ i \ hs)$
 ⟨*proof*⟩

lemma *policy-prob-space'*: $h \in \text{space } (H \ i) \implies s \in \text{space } Ms \implies$
 $\text{prob-space } (p \ i \ (h,s))$
 ⟨*proof*⟩

lemma *prob-space-p0*: $x \in \text{space } Ms \implies \text{prob-space } (p0 \ p \ x)$
 ⟨*proof*⟩

lemma *p0-sets*[*measurable-cong*]: $x \in \text{space } Ms \implies \text{sets } (p \ 0 \ (\lambda \cdot$
 $\text{undefined},x)) = \text{sets } Ma$
 ⟨*proof*⟩

lemma *space-p0*[*simp*]: $s \in \text{space } Ms \implies \text{space } (p0 \ p \ s) = \text{space } Ma$
 ⟨*proof*⟩

lemma *return-policy-prob-algebra* [*measurable*]:
 $h \in \text{space } (H \ n) \implies x \in \text{space } Ms \implies (\lambda a. \text{return } M \ (x, a)) \in p \ n$
 $(h, x) \rightarrow_M \text{prob-algebra } M$
 ⟨*proof*⟩
end

5.3 Successor Policy

To shift the policy by one step, we provide a single state-action pair as history

definition *Suc-policy* $p \ sa = (\lambda i \ (h, s). \ p \ (Suc \ i) \ (\lambda i'. \ \text{case-nat } sa \ h \ i', s))$

lemma *p-as-Suc-policy*: $p \ (Suc \ i) \ (h, s) = \text{Suc-policy } p \ ((h \ 0)) \ i \ (\lambda i. \ h \ (Suc \ i), s)$
 ⟨*proof*⟩

lemma *is-policy-Suc-policy*[*intro*]:
assumes $s: sa \in \text{space } M$ **and** $p: \text{is-policy } p$
shows $\text{is-policy } (Suc\text{-policy } p \ sa)$
 ⟨*proof*⟩

lemma *Suc-policy-measurable-step*[*measurable*]:
assumes $\text{is-policy } p$
shows $(\lambda x. \ \text{Suc-policy } p \ (fst \ (fst \ x)) \ n \ (snd \ (fst \ x), \ snd \ x)) \in$
 $(M \ \otimes_M \ Pi_M \ \{0..<n\} \ (\lambda \cdot M)) \ \otimes_M \ Ms \rightarrow_M \text{prob-algebra } Ma$
 ⟨*proof*⟩

5.4 Single-Step Distribution

K' takes a policy, a distribution over s , the epoch, and a history, produces a distribution over the next state-action pair.

definition $K' :: ('s, 'a) \text{ pol} \Rightarrow 's \text{ measure} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow ('s \times 'a)) \Rightarrow ('s \times 'a) \text{ measure}$

where

```

 $K' p s0 n \omega = \text{do} \{$ 
 $s \leftarrow \text{case-nat } s0 (K \circ \omega) n;$ 
 $a \leftarrow p n (\omega, s);$ 
 $\text{return } M (s, a)$ 

```

}

lemma *prob-space- K'* :

assumes p : *is-policy* p **and** x : $x0 \in \text{space} (\text{prob-algebra } Ms)$ **and** h : $h \in \text{space} (H n)$

shows *prob-space* $(K' p x0 n h)$

<proof>

lemma *measurable- K' [measurable]*:

assumes p : *is-policy* p **and** x : $x \in \text{space} (\text{prob-algebra } Ms)$

shows $K' p x i \in H i \rightarrow_M \text{prob-algebra } M$

<proof>

5.5 Initial State-Action Distribution

$K0$ produces the initial state-action distribution from a state distribution and a policy.

definition $K0 p s0 = K' p s0 0 (\lambda-. \text{undefined})$

lemma *$K0$ -def'*:

```

 $K0 p s0 = \text{do} \{$ 
 $s \leftarrow s0;$ 
 $a \leftarrow p0 p s;$ 
 $\text{return } M (s, a)\}$ 

```

<proof>

lemma *$K0$ -prob[measurable]*: *is-policy* $p \implies K0 p \in \text{prob-algebra } Ms$

$\rightarrow_M \text{prob-algebra } M$

<proof>

lemma *prob-space- $K0$* : *is-policy* $p \implies x0 \in \text{space} (\text{prob-algebra } Ms)$

$\implies \text{prob-space} (K0 p x0)$

<proof>

lemma *space- $K0$ [simp]*: *is-policy* $p \implies s \in \text{space} (\text{prob-algebra } Ms)$

$\implies \text{space} (K0 p s) = \text{space } M$

<proof>

lemma *sets-K0[measurable-cong]*:
assumes *is-policy* p $s \in \text{space}$ (*prob-algebra* M s)
shows *sets* ($K0$ p s) = *sets* M
 $\langle \text{proof} \rangle$

lemma *K0-return-eq-p0*:
assumes *is-policy* p $s \in \text{space}$ M s
shows $K0$ p (*return* M s) = $p0$ p $s \gg\gg (\lambda a. \text{return } M (s,a))$
 $\langle \text{proof} \rangle$

lemma *M-ne-policy[intro]*: *is-policy* $p \implies s \in \text{space}$ (*prob-algebra* M s)
 $\implies \text{space } M \neq \{\}$
 $\langle \text{proof} \rangle$

5.6 Sequence Space of the MDP

We can instantiate *Ionescu-Tulcea* with K' .

lemma *IT-K'*: *is-policy* $p \implies x \in \text{space}$ (*prob-algebra* M s) \implies *Ionescu-Tulcea*
 $(K' p x) (\lambda-. M)$
 $\langle \text{proof} \rangle$

definition *lim-sequence* :: $(s, 'a)$ *pol* \Rightarrow $'s$ *measure* \Rightarrow $(\text{nat} \Rightarrow ('s \times 'a))$ *measure*

where

lim-sequence $p x = \text{projective-family.lim UNIV (Ionescu-Tulcea.CI (K' p x) (\lambda-. M)) (\lambda-. M)}$

lemma

assumes $x: x \in \text{space}$ (*prob-algebra* M s) **and** $p: \text{is-policy } p$
shows *space-lim-sequence*: *space* (*lim-sequence* $p x$) = *space* $(\Pi_M$
 $i \in \text{UNIV}. M)$

and *sets-lim-sequence[measurable-cong]*: *sets* (*lim-sequence* $p x$) =
sets $(\Pi_M i \in \text{UNIV}. M)$

and *emeasure-lim-sequence-emb*: $\bigwedge J X. \text{finite } J \implies X \in \text{sets} (\Pi_M$
 $j \in J. M) \implies$

emeasure (*lim-sequence* $p x$) (*prod-emb* $\text{UNIV} (\lambda-. M) J X$) =
emeasure (*Ionescu-Tulcea.CI* $(K' p x) (\lambda-. M) J X$)

and *emeasure-lim-sequence-emb-I0o*: $\bigwedge n X. X \in \text{sets} (\Pi_M i \in$
 $\{0..<n\}. M) \implies$

emeasure (*lim-sequence* $p x$) (*prod-emb* $\text{UNIV} (\lambda-. M) \{0..<n\}$
 X) =

emeasure (*Ionescu-Tulcea.C* $(K' p x) (\lambda-. M) 0 n (\lambda x. \text{undefined})$)

X

$\langle \text{proof} \rangle$

lemma *lim-sequence-prob-space*:

assumes *is-policy* p $s \in \text{space}$ (*prob-algebra* M s)

shows *prob-space* (*lim-sequence* $p s$)

$\langle proof \rangle$

5.7 Measurability of the Sequence Space

lemma *lim-sequence[measurable]*:

assumes p : *is-policy* p

shows *lim-sequence* $p \in \text{prob-algebra } Ms \rightarrow_M \text{prob-algebra } (\Pi_M$
 $i \in UNIV. M)$

$\langle proof \rangle$

lemma *lim-sequence-aux[measurable]*:

assumes p : *is-policy* p

assumes f : $\bigwedge x. x \in \text{space } M \implies \text{is-policy } (f x)$

assumes f' : $\bigwedge n. (\lambda x. f (fst (fst x)) n (snd (fst x), snd x)) \in$

$(M \otimes_M Pi_M \{0..<n\} (\lambda-. M)) \otimes_M Ms \rightarrow_M \text{prob-algebra } Ma$

assumes gm : $g \in M \rightarrow_M \text{prob-algebra } Ms$

shows $(\lambda x. \text{lim-sequence } (f x) (g x)) \in M \rightarrow_M \text{prob-algebra } (Pi_M$
 $UNIV (\lambda-. M))$

$\langle proof \rangle$

lemma *lim-sequence-Suc-return[measurable]*:

assumes p : *is-policy* p

assumes s : $s \in \text{space } Ms$

shows $(\lambda x. \text{lim-sequence } (\text{Suc-policy } p (s, snd x)) (\text{return } Ms (fst$
 $x))) \in$

$M \rightarrow_M \text{prob-algebra } (Pi_M UNIV (\lambda-. M))$

$\langle proof \rangle$

lemma *lim-sequence-Suc-K[measurable]*:

assumes *is-policy* p

shows $(\lambda x. \text{lim-sequence } (\text{Suc-policy } p x) (K x)) \in M \rightarrow_M \text{prob-algebra}$
 $(Pi_M UNIV (\lambda-. M))$

$\langle proof \rangle$

5.8 Iteration Rule

lemma *step-C*:

assumes x : $x \in \text{space } (\text{prob-algebra } Ms)$ **and** p : *is-policy* p

shows *Ionescu-Tulcea.C* $(K' p x) (\lambda-. M) 0 1 (\lambda-. \text{undefined}) \gg=$

Ionescu-Tulcea.C $(K' p x) (\lambda-. M) 1 n =$

$K 0 p x \gg= (\lambda a. \text{Ionescu-Tulcea.C } (K' p x) (\lambda-. M) 1 n (\text{case-nat}$
 $a (\lambda-. \text{undefined})))$

$\langle proof \rangle$

lemma *lim-sequence-eq*:

assumes x : $x \in \text{space } (\text{prob-algebra } Ms)$ **assumes** p : *is-policy* p

shows *lim-sequence* $p x =$

$K 0 p x \gg= (\lambda y. \text{distr } (\text{lim-sequence } (\text{Suc-policy } p y) (K y)) (\Pi_M$
 $- \in UNIV. M) (\text{case-nat } y))$

$(\text{is } - = ?B p x)$

⟨proof⟩

5.9 Stream Space of the MDP

definition $\text{lim-stream} :: ('s, 'a) \text{pol} \Rightarrow 's \text{ measure} \Rightarrow ('s \times 'a) \text{ stream measure}$

where

$\text{lim-stream } p \ x = \text{distr } (\text{lim-sequence } p \ x) \ (\text{stream-space } M) \ \text{to-stream}$

lemma space-lim-stream : $\text{space } (\text{lim-stream } p \ x) = \text{streams } (\text{space } M)$
⟨proof⟩

lemma $\text{sets-lim-stream}[\text{measurable-cong}]$: $\text{sets } (\text{lim-stream } p \ x) = \text{sets } (\text{stream-space } M)$
⟨proof⟩

lemma $\text{lim-stream}[\text{measurable}]$:

assumes $\text{is-policy } p$

shows $\text{lim-stream } p \in \text{prob-algebra } Ms \rightarrow_M \text{prob-algebra } (\text{stream-space } M)$

⟨proof⟩

lemma $\text{lim-stream-Suc}[\text{measurable}]$:

assumes p : $\text{is-policy } p$

shows $(\lambda a. \text{lim-stream } (\text{Suc-policy } p \ a) \ (K \ a)) \in M \rightarrow_M \text{prob-algebra } (\text{stream-space } M)$

⟨proof⟩

lemma $\text{space-stream-space-M-ne}$: $x \in \text{space } M \implies \text{space } (\text{stream-space } M) \neq \{\}$

⟨proof⟩

lemma $\text{prob-space-lim-stream}[\text{intro}]$:

assumes $\text{is-policy } p \ x \in \text{space } (\text{prob-algebra } Ms)$

shows $\text{prob-space } (\text{lim-stream } p \ x)$

⟨proof⟩

lemma prob-space-step :

assumes $\text{is-policy } p \ x \in \text{space } M$

shows $\text{prob-space } (\text{lim-stream } (\text{Suc-policy } p \ x) \ (K \ x))$

⟨proof⟩

lemma lim-stream-eq :

assumes p : $\text{is-policy } p$

assumes x : $x \in \text{space } (\text{prob-algebra } Ms)$

shows $\text{lim-stream } p \ x = \text{do } \{$

$y \leftarrow K0 \ p \ x;$

$\omega \leftarrow \text{lim-stream } (\text{Suc-policy } p \ y) \ (K \ y);$

```

    return (stream-space M) (y ##  $\omega$ )
  }
  <proof>

end
end

```

```

theory MDP-disc
  imports
    MDP-cont
    HOL-Library.Omega-Words-Fun
begin

```

6 Markov Decision Processes with Discrete State Spaces

```

lemma (in prob-space) integral-stream-space:
  fixes f :: 'a stream  $\Rightarrow$  ('b :: {banach, second-countable-topology, real-normed-vector})
  assumes int-f: integrable (stream-space M) f
  assumes [measurable]: f  $\in$  borel-measurable (stream-space M)
  shows ( $\int X. f X \partial$ stream-space M) = ( $\int x. (\int X. f (x ## X) \partial$ stream-space M)  $\partial$ M)
  <proof>

```

```

lemma prefix-cons:
  Omega-Words-Fun.prefix (Suc n) seq = seq 0 # Omega-Words-Fun.prefix
  n ( $\lambda n. seq (Suc n)$ )
  <proof>

```

```

lemma restrict-Suc: restrict y {0..<Suc i} (Suc n) = (restrict ( $\lambda n. y$ 
  (Suc n)) {0..<i}) n
  <proof>

```

```

lemma prefix-restrict: Omega-Words-Fun.prefix i (restrict y {0..<i})
  = Omega-Words-Fun.prefix i y
  <proof>

```

```

lemma prefix-measurable[measurable]:
  Omega-Words-Fun.prefix i  $\in$  Pi_M {0..<i}
  ( $\lambda. count-space (UNIV :: ('s :: countable \times 'a :: countable) set)$ )  $\rightarrow_M$ 
  count-space UNIV
  <proof>

```

```

no-notation Omega-Words-Fun.build (infixr <##> 65)

```

```

locale discrete-MDP =
  fixes A :: 's :: countable  $\Rightarrow$  'a :: countable set — enabled actions

```

and $K :: 's \times 'a \Rightarrow 's \text{ pmf}$ — MDP kernel, transition probabilities
assumes
 $A\text{-ne}: \bigwedge s. A\ s \neq \{\}$ — set of enabled actions is nonempty
begin

6.1 Policies

Type synonym for decision rules.

type-synonym $('c, 'd) \text{ dec} = 'c \Rightarrow 'd \text{ pmf}$

definition $is\text{-dec} :: ('s, 'a) \text{ dec} \Rightarrow \text{bool}$ **where**
 $is\text{-dec}\ d \equiv \forall s. d\ s \subseteq A\ s$

lemma $is\text{-decI}$ [intro]:
 $(\bigwedge s. \text{set-pmf}\ (d\ s) \subseteq A\ s) \Longrightarrow is\text{-dec}\ d$
 $\langle \text{proof} \rangle$

abbreviation $D_R \equiv \{d. is\text{-dec}\ d\}$

definition $is\text{-dec-det} :: ('s \Rightarrow 'a) \Rightarrow \text{bool}$ **where**
 $is\text{-dec-det}\ d \equiv \forall s. d\ s \in A\ s$

abbreviation $D_D \equiv \{d. is\text{-dec-det}\ d\}$

definition $mk\text{-dec-det}\ d\ s = \text{return-pmf}\ (d\ s)$

lemma $is\text{-dec-mk-dec-det-iff}$ [simp]: $is\text{-dec}\ (mk\text{-dec-det}\ d) \longleftrightarrow is\text{-dec-det}\ d$
 $\langle \text{proof} \rangle$

lemma $D\text{-det-to-MR}$ [intro]: $is\text{-dec-det}\ d \Longrightarrow is\text{-dec}\ (mk\text{-dec-det}\ d)$
 $\langle \text{proof} \rangle$

Due to the assumption $A\ ?s \neq \{\}$, a deterministic decision rule always exists. It immediately follows via $is\text{-dec}\ (mk\text{-dec-det}\ ?d) = is\text{-dec-det}\ ?d$ that a randomized decision rule also exists.

lemma $SOME\text{-is-dec-det}$: $is\text{-dec-det}\ (\lambda s. SOME\ a. a \in A\ s)$
 $\langle \text{proof} \rangle$

lemma $ex\text{-dec-det}$ [simp]: $\exists d. is\text{-dec-det}\ d$
 $\langle \text{proof} \rangle$

lemma $D\text{-det-ne}$ [simp]: $D_D \neq \{\}$
 $\langle \text{proof} \rangle$

lemma $D_R\text{-ne}$ [simp]: $D_R \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *ex-dec*[*intro, simp*]: $\exists d. \text{is-dec } d$
 ⟨*proof*⟩

Type synonym for policies.

type-synonym $('c, 'd) \text{ pol} = ('c \times 'd) \text{ list} \Rightarrow ('c, 'd) \text{ dec}$

A policy assigns a decision rule to each observed past.

definition *is-policy* :: $('s, 'a) \text{ pol} \Rightarrow \text{bool}$ **where**
is-policy $p \equiv \forall hs. \text{is-dec } (p \text{ } hs)$

abbreviation $\Pi_{HR} \equiv \{p. \text{is-policy } p\}$

Deterministic policies

definition *is-deterministic* $p \equiv \text{is-policy } p \wedge (\forall h \ s. \exists a. p \ h \ s = \text{return-pmf } a)$

definition *mk-det* $p \ h \ s \equiv \text{return-pmf } (p \ h \ s)$

abbreviation $\Pi_{HD} \equiv \{p. \forall h. p \ h \in D_D\}$

Markovian policies

definition *is-markovian* $p \equiv \text{is-policy } p \wedge (\forall h \ h'. \text{length } h = \text{length } h' \longrightarrow p \ h = p \ h')$

definition *mk-markovian* :: $(\text{nat} \Rightarrow ('s, 'a) \text{ dec}) \Rightarrow ('s, 'a) \text{ pol}$ **where**
mk-markovian $p \equiv (\lambda h. p \ (\text{length } h))$

lemma *is-markovian-mk-iff*[*simp*]: $\text{is-markovian } (mk\text{-markovian } p) \longleftrightarrow (\forall n. \text{is-dec } (p \ n))$
 ⟨*proof*⟩

lemma *is-markovian-mk*[*intro*]: $\forall n. \text{is-dec } (p \ n) \Longrightarrow \text{is-markovian } (mk\text{-markovian } p)$
 ⟨*proof*⟩

lemma *mk-markovian-nil* [*simp*]: $mk\text{-markovian } p \ [] = p \ 0$
 ⟨*proof*⟩

definition *mk-markovian-det* $p \equiv (\lambda h \ s. \text{return-pmf } (p \ (\text{length } h) \ s))$

abbreviation $\Pi_{MD} \equiv \{p. \forall n::\text{nat}. p \ n \in D_D\}$

abbreviation $\Pi_{MR} \equiv \{p. \forall n. p \ n \in D_R\}$

lemma $\Pi_{MR}\text{-imp-policies}$ [*intro*]: $p \in \Pi_{MR} \Longrightarrow \text{mk-markovian } p \in \Pi_{HR}$
 ⟨*proof*⟩

lemma $\Pi_{MD}\text{-MR-iff}$ [*simp*]: $(\lambda n. mk\text{-dec-det } (p \ n)) \in \Pi_{MR} \longleftrightarrow p \in \Pi_{MD}$

$\langle proof \rangle$

lemma $\Pi_{MD-to-MR}[intro]: p \in \Pi_{MD} \implies (\lambda n. mk-dec-det (p n)) \in \Pi_{MR}$
 $\langle proof \rangle$

lemma $p-n-\pi-MD[intro]: p \in \Pi_{MD} \implies p n \in D_D$
 $\langle proof \rangle$

lemma $p-n-\pi-MR[intro]: p \in \Pi_{MR} \implies p n \in D_R$
 $\langle proof \rangle$

lemma $\Pi_{MD-ne}[simp]: \Pi_{MD} \neq \{\}$
 $\langle proof \rangle$

lemma $\Pi_{MR-ne}[simp]: \Pi_{MR} \neq \{\}$
 $\langle proof \rangle$

lemma $policies-ne[simp, intro]: \Pi_{HR} \neq \{\}$
 $\langle proof \rangle$

Stationary policies

definition $is-stationary p \equiv is-policy p \wedge (\forall h h'. p h = p h')$

lemma $is-stationary-const-iff[simp]: is-stationary (\lambda-. d) = is-dec d$
 $\langle proof \rangle$

lemma $is-stationary-const[intro]: is-dec d \implies is-stationary (\lambda-. d)$
 $\langle proof \rangle$

abbreviation $mk-stationary p \equiv mk-markovian (\lambda-. p)$

abbreviation $mk-stationary-det d \equiv mk-markovian (\lambda-. mk-dec-det d)$

6.1.1 Successor Policy

After taking the first step in the MDP, we will know which state and which action got selected during the initial epoch. To obtain a policy that acts as if the current epoch was the initial one, we prepend the observed state-action pair to the history. The result is again a policy, i.e. it satisfies *is-policy*.

definition $\pi-Suc :: ('s, 'a) pol \Rightarrow 's \times 'a \Rightarrow ('s, 'a) pol$

where

$\pi-Suc p sa h = p (sa\#h)$

lemma $is-policy-\pi-Suc [intro]: is-policy p \implies is-policy (\pi-Suc p sa)$
 $\langle proof \rangle$

lemma *Suc-mk-markovian[simp]*: π -*Suc* (*mk-markovian* p) $x = \text{mk-markovian}$
 $(\lambda n. p (Suc\ n))$
 $\langle \text{proof} \rangle$

6.2 Stream Space of the MDP

6.2.1 Initial State-Action Distribution

If we fix a decision rule d and an initial distribution of states $S0$, we obtain a distribution over state-action pairs in the following way: First, the initial state s is sampled from $S0$, then an action a is selected from $d\ s$.

definition *K0* $d\ S0 = do$ {
 $s \leftarrow S0$;
 $a \leftarrow d\ s$;
return-pmf (s, a)
}

notation *K0* ($\langle K_0 \rangle$)

lemma *K0-iff*: $K0\ d\ S0 = S0 \gg= (\lambda s. \text{map-pmf } (\lambda a. (s, a)) (d\ s))$
 $\langle \text{proof} \rangle$

lemma *vimage-pair[simp]*: *Pair* $x - \{p\} = (\text{if } x = \text{fst } p \text{ then } \{\text{snd } p\}$
 $\text{else } \{\})$
 $\langle \text{proof} \rangle$

lemma *pmf-K0 [simp]*: $\text{pmf } (K0\ d\ S0) (s, a) = \text{pmf } S0\ s * \text{pmf } (d\ s)$
 a
 $\langle \text{proof} \rangle$

lemma *set-pmf-K0*: $\text{set-pmf } (K0\ p\ S0) = \{(s, a). s \in S0 \wedge a \in p\ s\}$
 $\langle \text{proof} \rangle$

lemma *fst-K0[simp]*: $\text{map-pmf } \text{fst } (K0\ p\ S0) = S0$
 $\langle \text{proof} \rangle$

abbreviation $S \equiv \text{stream-space } (\text{count-space } UNIV)$

We inherit the trace space from MDPs with continuous state-action spaces

interpretation *MDP-cont*: *MDP-cont*.*discrete-MDP* *count-space* *UNIV*
 $\text{count-space } UNIV\ A\ K$
 $\langle \text{proof} \rangle$

lemma *count-space-M[simp]*: *MDP-cont*. $M = \text{count-space } UNIV$
 $\langle \text{proof} \rangle$

lemma *space-M[simp]*: *space MDP-cont.M = UNIV*
 ⟨*proof*⟩

We reuse the stream space provided by *MDP-cont.lim-stream*

definition *T* :: ('s, 'a) *pol* ⇒ 's *pmf* ⇒ ('s × 'a) *stream measure*
where *T p* = *MDP-cont.lim-stream* (λn (h,s). *p* (*Omega-Words-Fun.prefix* n h) s)

lemma *sets-T[measurable-cong]*:
sets (T p x) = sets S
 ⟨*proof*⟩

lemma *space-stream-space-ne[simp]*: *space S ≠ {}*
 ⟨*proof*⟩

lemma *space-T[simp]*: *space (T p S0) = space S*
 ⟨*proof*⟩

lemma *is-policy-MDP-cont[intro]*:
fixes *p* :: ('s × 'a) *list* ⇒ 's ⇒ 'a *pmf*
shows *MDP-cont.is-policy* (λn (h,s). *p* (*Omega-Words-Fun.prefix* n h) s)
 ⟨*proof*⟩

lemma *prob-space-T[intro, simp]*: *prob-space (T p x)*
 ⟨*proof*⟩

lemma *T-subprob[simp]*:
T p S0 ∈ space (subprob-algebra S)
 ⟨*proof*⟩

lemma *T-subprob-space [simp]*: *subprob-space (T p S0)*
 ⟨*proof*⟩

lemma *K0-MDP-cont-eq*:
MDP-cont.K0 (λx (h,s). *measure-pmf* (p (*Omega-Words-Fun.prefix* x h) s)) (*measure-pmf* S0) =
K0 (p []) S0
 ⟨*proof*⟩

6.2.2 Decomposition of the Stream Space

The distribution of traces/walks the MDP allows should intuitively satisfy the following rule:

1. select the initial state *s* from *S0*
2. pass it to the decision rule *p []* to determine a distribution over actions

3. select the action a

- finally pass the state-action pair (s, a) to the kernel K to get a new distribution over states s_0'

Then the iteration repeats with the updated policy $\pi\text{-Suc } p(s, a)$.

The result carries over from $\llbracket \text{MDP-cont.is-policy } ?p; ?x \in \text{space (prob-algebra (count-space UNIV))} \rrbracket \implies \text{MDP-cont.lim-stream } ?p \text{ } ?x = \text{MDP-cont.K0 } ?p \text{ } ?x \ggg (\lambda y. \text{MDP-cont.lim-stream } (\text{MDP-cont.Suc-policy } ?p \text{ } y) (\text{measure-pmf } (K \text{ } y)) \ggg (\lambda \omega. \text{return (stream-space MDP-cont.M) (y \#\#\ \omega)}))$.

lemma *T-eq*:

shows $T \text{ } p \text{ } S0 = \text{do } \{$
 $\text{ } sa \leftarrow \text{measure-pmf } (K0 \text{ } (p \text{ []}) \text{ } S0);$
 $\text{ } \omega \leftarrow T (\pi\text{-Suc } p \text{ } sa) (K \text{ } sa);$
 $\text{ } \text{return } S (sa \text{ \#\#\ } \omega)$
 $\}$
 $\langle \text{proof} \rangle$

lemma *T-eq-distr*:

shows $T \text{ } p \text{ } S0 = \text{measure-pmf } (K0 \text{ } (p \text{ []}) \text{ } S0) \ggg (\lambda sa. \text{distr } (T (\pi\text{-Suc } p \text{ } sa) (K \text{ } sa)) \text{ } S ((\#\#\ \text{ } sa)))$
 $\langle \text{proof} \rangle$

The iteration rule lets us nicely decompose integrals (expected values) over functions on traces of the MDP.

lemma *integral-T*:

fixes $f :: ('s \times 'a) \text{ stream} \Rightarrow \text{real}$
assumes $f\text{-bounded: } \bigwedge x. |f \text{ } x| \leq B$
assumes $f: f \in \text{borel-measurable } S$
shows $(\int t. f \text{ } t \text{ } \partial T \text{ } p \text{ } x) = \int sa. \int t'. f (sa \text{ \#\#\ } t') \text{ } \partial T (\pi\text{-Suc } p \text{ } sa) (K \text{ } sa) \text{ } \partial K0 \text{ } (p \text{ []}) \text{ } x$
 $\langle \text{proof} \rangle$

lemma *nn-integral-T*:

assumes $f: f \in \text{borel-measurable } S$
shows $(\int ^+ t. f \text{ } t \text{ } \partial T \text{ } p \text{ } x) = (\int ^+ sa. \int ^+ t'. f (sa \text{ \#\#\ } t') \text{ } \partial T (\pi\text{-Suc } p \text{ } sa) (K \text{ } sa) \text{ } \partial K0 \text{ } (p \text{ []}) \text{ } x)$
 $\langle \text{proof} \rangle$

6.2.3 A Denotational View on the Stochastic Process

Many definitions on MDPs do not rely on the individual traces but only on the distribution of states and actions at each epoch.

We define this view on the trace space as the repeated iteration of K_0 and K . It coincides with the definition of T .

primrec $Pn :: ('s, 'a) \text{pol} \Rightarrow 's \text{ pmf} \Rightarrow \text{nat} \Rightarrow ('s \times 'a) \text{ pmf}$ **where**
 $Pn \ p \ S0 \ 0 = K0 \ (p \ []) \ S0$
 $| Pn \ p \ S0 \ (Suc \ n) = K0 \ (p \ []) \ S0 \ggg (\lambda sa. Pn \ (\pi\text{-Suc} \ p \ sa) \ (K \ sa) \ n)$

declare $Pn.\text{simps}(2)[\text{simp del}]$

lemma $Pn\text{-eq-}T$: $\text{measure-pmf} \ (Pn \ p \ S0 \ n) = \text{distr} \ (T \ p \ S0) \ (\text{count-space} \ UNIV) \ (\lambda t. t \ !! \ n)$
 $\langle \text{proof} \rangle$

The definition of Pn also allows us to easily prove that only enabled actions can occur in the traces of the MDP.

lemma $Pn\text{-in-}A$: $\text{is-policy} \ p \Longrightarrow (s, a) \in Pn \ p \ S0 \ n \Longrightarrow a \in A \ s$
 $\langle \text{proof} \rangle$

lemma $T\text{-in-}A$:
assumes $\text{is-policy} \ p$
shows $AE \ t \ \text{in} \ T \ p \ S0. \ \text{snd} \ (t \ !! \ n) \in A \ (\text{fst} \ (t \ !! \ n))$
 $\langle \text{proof} \rangle$

6.2.4 State Process

Alongside Pn , we also define the state and action distributions as projections.

definition $Xn \ p \ S0 \ n = \text{map-pmf} \ \text{fst} \ (Pn \ p \ S0 \ n)$

lemma $X0$ $[\text{simp}]$: $Xn \ p \ S0 \ 0 = S0$
 $\langle \text{proof} \rangle$

lemma $Xn\text{-Suc}$: $Xn \ p \ S0 \ (Suc \ n) = Pn \ p \ S0 \ n \ggg K$
 $\langle \text{proof} \rangle$

lemma $Pn\text{-markovian-eq-}Xn\text{-bind}$: $Pn \ (\text{mk-markovian} \ p) \ S0 \ n = K0 \ (p \ n) \ (Xn \ (\text{mk-markovian} \ p) \ S0 \ n)$
 $\langle \text{proof} \rangle$

lemma $Xn\text{-Suc}'$: $Xn \ p \ S0 \ (Suc \ n) = K0 \ (p \ []) \ S0 \ggg (\lambda sa. Xn \ (\pi\text{-Suc} \ p \ sa) \ (K \ sa) \ n)$
 $\langle \text{proof} \rangle$

lemma $\text{set-pmf-}X0$ $[\text{simp}]$: $\text{set-pmf} \ (Xn \ p \ S0 \ 0) = S0$
 $\langle \text{proof} \rangle$

lemma $\text{set-pmf-}PSuc$: $\text{set-pmf} \ (Pn \ (\text{mk-markovian} \ p) \ S0 \ n) = \{(s, a). s \in \text{set-pmf} \ (Xn \ (\text{mk-markovian} \ p) \ S0 \ n) \wedge a \in p \ n \ s\}$
 $\langle \text{proof} \rangle$

6.2.5 The Conditional Distribution of Actions

Actions are selected wrt. the whole history of state-action pairs encountered so far. The following definition defines the expected action selection when only the current state is given.

definition $Y\text{-cond-}X\ p\ S0\ n\ x = \text{map-pmf}\ \text{snd}\ (\text{cond-pmf}\ (Pn\ p\ S0\ n)\ \{(s,a).\ s = x\})$

lemma $\text{prob-}K0\text{-}X\ [\text{simp}]: \text{measure-pmf}.\text{prob}\ (K0\ p\ S0)\ \{(s, a).\ s = x\} = \text{pmf}\ S0\ x$
 $\langle \text{proof} \rangle$

lemma $\text{prob-}Pn\text{-}X[\text{simp}]: \text{measure-pmf}.\text{prob}\ (Pn\ p\ S0\ n)\ \{(s, a).\ s = x\} = \text{pmf}\ (Xn\ p\ S0\ n)\ x$
 $\langle \text{proof} \rangle$

lemma $\text{pmf-}Pn\text{-pair}$:

assumes $sa \in \text{set-pmf}\ (Pn\ p\ S0\ n)$

shows $\text{pmf}\ (Pn\ p\ S0\ n)\ sa = \text{pmf}\ (Y\text{-cond-}X\ p\ S0\ n\ (\text{fst}\ sa))\ (\text{snd}\ sa) * \text{pmf}\ (Xn\ p\ S0\ n)\ (\text{fst}\ sa)$
 $\langle \text{proof} \rangle$

lemma $\text{pmf-}Pn$:

assumes $x \in \text{set-pmf}\ (Xn\ p\ S0\ n)$

shows $\text{pmf}\ (Pn\ p\ S0\ n)\ (x,a) = \text{pmf}\ (Y\text{-cond-}X\ p\ S0\ n\ x)\ a * \text{pmf}\ (Xn\ p\ S0\ n)\ x$
 $\langle \text{proof} \rangle$

lemma $\text{pmf-}Y\text{-cond-}X$:

assumes $x \in \text{set-pmf}\ (Xn\ p\ S0\ n)$

shows $\text{pmf}\ (Y\text{-cond-}X\ p\ S0\ n\ x)\ a = \text{pmf}\ (Pn\ p\ S0\ n)\ (x,a) / \text{pmf}\ (Xn\ p\ S0\ n)\ x$
 $\langle \text{proof} \rangle$

lemma $Y\text{-cond-}X\ 0[\text{simp}]:$

assumes $x \in \text{set-pmf}\ S0$

shows $Y\text{-cond-}X\ p\ S0\ 0\ x = p\ \square\ x$
 $\langle \text{proof} \rangle$

lemma $Y\text{-cond-}X\ \text{markovian}[\text{simp}]:$

assumes $h: x \in Xn\ (\text{mk-markovian}\ p)\ S0\ n$

shows $Y\text{-cond-}X\ (\text{mk-markovian}\ p)\ S0\ n\ x = p\ n\ x$
 $\langle \text{proof} \rangle$

lemma $Pn\ \text{eq}\ Xn\ \text{Y-cond}$: $Pn\ p\ S0\ n = Xn\ p\ S0\ n \gg (\lambda x. \text{map-pmf}\ (\lambda a. (x, a))\ (Y\text{-cond-}X\ p\ S0\ n\ x))$
 $\langle \text{proof} \rangle$

lemma *Pn-eq-Xn-Y-cond'*:

$Pn\ p\ S0\ n = Xn\ p\ S0\ n \gg= (\lambda s. Y\text{-cond-}X\ p\ S0\ n\ s \gg= (\lambda a. \text{return-pmf}\ (s,a)))$
 ⟨proof⟩

lemma *Pn-markovian-Suc*: $Pn\ (mk\text{-markovian}\ p)\ S0\ (Suc\ n) =$

$Pn\ (mk\text{-markovian}\ p)\ S0\ n \gg= (\lambda sa. K0\ (p\ (Suc\ n))\ (K\ sa))$
 ⟨proof⟩

6.2.6 Action Process

The distribution of actions.

definition $Yn\ p\ S0\ n = \text{map-pmf}\ \text{snd}\ (Pn\ p\ S0\ n)$

lemma *Y0*: $Yn\ p\ S0\ 0 = S0 \gg= p\ []$

⟨proof⟩

For markovian policies, the decision rules at each epoch are independent of each other, hence we may express Yn solely in terms of Xn and the current decision rule.

lemma *Yn-markovian*: $Yn\ (mk\text{-markovian}\ p)\ S0\ n = Xn\ (mk\text{-markovian}\ p)\ S0\ n \gg= p\ n$

⟨proof⟩

6.3 Restriction to Markovian Policies

abbreviation $as\text{-markovian}\ p\ S0\ n\ x \equiv$

$\text{if } x \in (Xn\ p\ S0\ n) \text{ then } Y\text{-cond-}X\ p\ S0\ n\ x \text{ else } \text{return-pmf}\ (SOME\ a. a \in A\ x)$

For states which cannot occur we choose an arbitrary enabled action, as in this case we cannot make any statements about $Y\text{-cond-}X$ (a distribution conditioned on an event with probability 0).

lemma *is- Π_{MR} -as-markovian*:

assumes p : *is-policy* p

shows *as-markovian* $p\ S0 \in \Pi_{MR}$

⟨proof⟩

lemma *is-policy-as-markovian*: $\text{is-policy}\ p \implies \text{is-policy}\ (mk\text{-markovian}\ (as\text{-markovian}\ p\ S0))$

⟨proof⟩

theorem *Pn-as-markovian-eq*: $Pn\ (mk\text{-markovian}\ (as\text{-markovian}\ p\ S0))\ S0 = Pn\ p\ S0$

⟨proof⟩

6.4 MDPs without Initial Distribution

From now on, we assume a known, deterministic initial state. All results from the previous discussion carry over as we are now in the special case where the initial state is of the form *return-pmf s*.

definition $\mathcal{T} p s \equiv T p (\text{return-pmf } s)$

lemma *\mathcal{T} -eq-return-distr*: $\mathcal{T} p s =$
 $\text{measure-pmf } (p \square s) \gg (\lambda a. \text{distr } (T (\pi\text{-Suc } p (s,a))) (K (s,a))) S$
 $((\#\#) (s,a))$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -eq-return*:
shows $\mathcal{T} p s = \text{do } \{$
 $y \leftarrow \text{measure-pmf } (p \square s);$
 $\omega \leftarrow T (\pi\text{-Suc } p (s,y)) (K (s,y));$
 $\text{return } S ((s,y) \#\# \omega)$
 $\}$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -return*:
shows $T p S0 = \text{measure-pmf } S0 \gg \mathcal{T} p$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -return-eq*:
 $\mathcal{T} p s = \text{do } \{$
 $a \leftarrow \text{measure-pmf } (p \square s);$
 $s' \leftarrow \text{measure-pmf } (K (s,a));$
 $w \leftarrow T (\pi\text{-Suc } p (s,a)) (\text{return-pmf } s');$
 $\text{return } S ((s,a) \#\# w)$
 $\}$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -eq*:
shows $\mathcal{T} p s = \text{do } \{$
 $a \leftarrow \text{measure-pmf } (p \square s);$
 $s' \leftarrow \text{measure-pmf } (K (s,a));$
 $w \leftarrow \mathcal{T} (\pi\text{-Suc } p (s,a)) s';$
 $\text{return } S ((s,a) \#\# w)$
 $\}$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -prob-space[intro]*: *prob-space* $(\mathcal{T} p s)$
 $\langle \text{proof} \rangle$

lemma *\mathcal{T} -sets[measurable-cong]*:
 $\text{sets } (\mathcal{T} p s) = \text{sets } S$
 $\langle \text{proof} \rangle$

lemma *measurable-ident-Suc'*[*measurable*]:

$(\lambda x. x) \in \mathcal{T} (\pi\text{-Suc } p \text{ } sa) \text{ } s' \rightarrow_M S$
 $\langle \text{proof} \rangle$

lemma *nn-integral-T*:

fixes $f :: ('s \times 'a) \text{ stream} \Rightarrow \text{real}$
assumes $f[\text{measurable}]$: $f \in \text{borel-measurable } S$
shows $(\int^{+t}. f \text{ } t \text{ } \partial \mathcal{T} \text{ } p \text{ } s)$
 $= \int^{+a}. \int^{+s'}. \int^{+t'}. f ((s,a)\#\#t') \text{ } \partial \mathcal{T} (\pi\text{-Suc } p (s,a)) \text{ } s' \text{ } \partial K (s,a)$
 $\partial p \text{ } \square \text{ } s$
 $\langle \text{proof} \rangle$

lemma *integral-T*:

fixes $f :: ('s \times 'a) \text{ stream} \Rightarrow \text{real}$
assumes $f\text{-bounded}$: $\bigwedge x. |f x| \leq B$
assumes $f[\text{measurable}]$: $f \in \text{borel-measurable } S$
shows $(\int t. f \text{ } t \text{ } \partial \mathcal{T} \text{ } p \text{ } s)$
 $= \int a. \int s'. \int t'. f ((s,a)\#\#t') \text{ } \partial \mathcal{T} (\pi\text{-Suc } p (s,a)) \text{ } s' \text{ } \partial K (s,a) \text{ } \partial p$
 $\square \text{ } s$
 $\langle \text{proof} \rangle$

lemma *integrable-T-bounded*[*intro*]:

fixes $f :: ('s \times 'a) \text{ stream} \Rightarrow 'd :: \{\text{second-countable-topology, banach}\}$
assumes $f[\text{measurable}]$: $f \in \text{borel-measurable } S$
assumes b : *bounded* (*range* f)
shows *integrable* $(\mathcal{T} \text{ } p \text{ } s) \text{ } f$
 $\langle \text{proof} \rangle$

definition $Pn' \text{ } p \text{ } s = Pn \text{ } p \text{ } (\text{return-pmf } s)$

definition $Xn' \text{ } p \text{ } s = Xn \text{ } p \text{ } (\text{return-pmf } s)$

definition $Yn' \text{ } p \text{ } s = Yn \text{ } p \text{ } (\text{return-pmf } s)$

definition $K0' \text{ } d \text{ } s \equiv \text{map-pmf } (\lambda a. (s, a)) \text{ } (d \text{ } s)$

definition $K\text{-st } d \text{ } s \equiv d \text{ } s \gg (\lambda a. K (s, a))$

lemma *pmf-K-st*: $\text{pmf } (K\text{-st } d \text{ } s) \text{ } t = \int a. \text{pmf } (K(s, a)) \text{ } t \text{ } \partial d \text{ } s$
 $\langle \text{proof} \rangle$

$K\text{-st}$ defines the distribution over the successor states for a given decision rule and state. It is mostly useful for markovian policies, as the information which action was selected is lost.

lemma $P0'[\text{simp}]$: $Pn' \text{ } p \text{ } s \text{ } 0 = K0' (p \text{ } \square) \text{ } s$
 $\langle \text{proof} \rangle$

lemma $X0'[\text{simp}]$: $Xn' \text{ } p \text{ } s \text{ } 0 = \text{return-pmf } s$
 $\langle \text{proof} \rangle$

lemma $Pn\text{-return-pmf}$: $S0 \gg (\lambda s'. Pn \text{ } p \text{ } (\text{return-pmf } s') \text{ } n) = Pn \text{ } p$

$S0\ n$
 $\langle proof \rangle$

lemma $PSuc'$: $Pn'\ p\ s\ (Suc\ n) = K0'\ (p\ [])\ s \ggg (\lambda sa. K\ sa \ggg (\lambda s'. Pn'\ (\pi\text{-}Suc\ p\ sa)\ s'\ n))$
 $\langle proof \rangle$

lemma $PSuc'$ -markovian:
 $Pn'\ (mk\text{-}markovian\ p)\ s\ (Suc\ n) = K\text{-}st\ (p\ 0)\ s \ggg (\lambda s'. Pn'\ (mk\text{-}markovian\ (p\ \circ\ Suc))\ s'\ n)$
 $\langle proof \rangle$

lemma Xn' - Suc : $Xn'\ p\ s\ (Suc\ n) = Pn'\ p\ s\ n \ggg K$
 $\langle proof \rangle$

lemma Xn' - Pn' : $Xn'\ p\ s\ n = map\text{-}pmf\ fst\ (Pn'\ p\ s\ n)$
 $\langle proof \rangle$

lemma Suc - Xn' : $Xn'\ p\ s\ (Suc\ n) = p\ []\ s \ggg (\lambda a. K\ (s,a) \ggg (\lambda s'. Xn'\ (\pi\text{-}Suc\ p\ (s,a))\ s'\ n))$
 $\langle proof \rangle$

lemma Suc - Xn' -markovian:
 $Xn'\ (mk\text{-}markovian\ p)\ s\ (Suc\ n) = K\text{-}st\ (p\ 0)\ s \ggg (\lambda s'. Xn'\ (mk\text{-}markovian\ (\lambda n. p\ (Suc\ n)))\ s'\ n)$
 $\langle proof \rangle$

lemma Xn' -split: $Xn'\ (mk\text{-}markovian\ p)\ s\ (n + m) = Xn'\ (mk\text{-}markovian\ p)\ s\ n \ggg (\lambda s. Xn'\ (mk\text{-}markovian\ (\lambda i. p\ (i + n)))\ s\ m)$
 $\langle proof \rangle$

lemma Yn' -markovian: $Yn'\ (mk\text{-}markovian\ p)\ s\ n = Xn'\ (mk\text{-}markovian\ p)\ s\ n \ggg p\ n$
 $\langle proof \rangle$

lemma Pn' -markovian-eq- Xn' -bind: $Pn'\ (mk\text{-}markovian\ p)\ s\ n = Xn'\ (mk\text{-}markovian\ p)\ s\ n \ggg K0'\ (p\ n)$
 $\langle proof \rangle$

lemma Pn' -eq- \mathcal{T} : $measure\text{-}pmf\ (Pn'\ p\ s\ n) = distr\ (\mathcal{T}\ p\ s)\ (count\text{-}space\ UNIV)\ (\lambda t. t\ !!\ n)$
 $\langle proof \rangle$

end
end

theory $MDP\text{-}reward$

```

imports
  Bounded-Functions
  MDP-reward-Util
  Blinfun-Util
  MDP-disc
begin

```

7 Markov Decision Processes with Rewards

```

locale MDP-reward = discrete-MDP A K
for
  A and
  K :: 's :: countable × 'a :: countable ⇒ 's pmf +
fixes
  r :: ('s × 'a) ⇒ real and
  l :: real
assumes
  zero-le-disc [simp]: 0 ≤ l and
  r-bounded: bounded (range r)
begin

```

This extension to the basic MDPs is formalized with another locale. It assumes the existence of a reward function r which takes a state-action pair to a real number. We assume that the function is bounded r -bounded.

Furthermore, we fix a discounting factor l , where $0 \leq l \wedge l < 1$.

7.1 Util

7.1.1 Basic Properties of rewards

```

lemma r-bfun: r ∈ bfun
  ⟨proof⟩

```

```

lemma r-bounded': bounded (r ' X)
  ⟨proof⟩

```

```

definition r_M = (⊔ sa. |r sa|)

```

```

lemma abs-r-le-r_M: |r sa| ≤ r_M
  ⟨proof⟩

```

```

lemma abs-r_M-eq-r_M [simp]: |r_M| = r_M
  ⟨proof⟩

```

```

lemma r_M-nonneg: 0 ≤ r_M
  ⟨proof⟩

```

lemma *measurable-r-nth* [*measurable*]: $(\lambda t. r (t !! i)) \in \text{borel-measurable } S$

<proof>

lemma *integrable-r-nth* [*simp*]: *integrable* $(\mathcal{T} p s) (\lambda t. r (t !! i))$

<proof>

lemma *expectation-abs-r-le*: *measure-pmf.expectation* $d (\lambda a. |r (s, a)|) \leq r_M$

<proof>

lemma *abs-exp-r-le*: $| \text{measure-pmf.expectation } d r | \leq r_M$

<proof>

7.1.2 Infinite discounted sums

lemma *abs-disc-eq*[*simp*]: $|l \hat{ } i * x| = l \hat{ } i * |x|$

<proof>

lemma *norm-l-pow-eq*[*simp*]: $\text{norm } (l \hat{ } t *_{R} F) = l \hat{ } t * \text{norm } F$

<proof>

7.2 Total Reward for Single Traces

abbreviation *ν -trace-fin* $t N \equiv \sum_{i < N}. l \hat{ } i * r (t !! i)$

abbreviation *ν -trace* $t \equiv \sum_{i}. l \hat{ } i * r (t !! i)$

lemma *abs- ν -trace-fin-le*: $|\nu\text{-trace-fin } t N| \leq (\sum_{i < N}. l \hat{ } i * r_M)$

<proof>

lemma *measurable-suminf-reward*[*measurable*]: *ν -trace* $\in \text{borel-measurable } S$

<proof>

lemma *integrable- ν -trace-fin*: *integrable* $(\mathcal{T} p s) (\lambda t. \nu\text{-trace-fin } t N)$

<proof>

context

fixes $p :: ('s, 'a) \text{pol}$

begin

7.3 Expected Finite-Horizon Discounted Reward

definition *ν -fin* $n s = \int t. \nu\text{-trace-fin } t n \partial \mathcal{T} p s$

lemma *abs- ν -fin-le*: $|\nu\text{-fin } N s| \leq (\sum_{i < N}. l \hat{ } i * r_M)$

<proof>

lemma *ν -fin-bfun*: $(\lambda s. \nu\text{-fin } N s) \in \text{bfun}$

$\langle proof \rangle$

lift-definition $\nu_b\text{-fin} :: \text{nat} \Rightarrow 's \Rightarrow_b \text{real}$ is $\nu\text{-fin}$
 $\langle proof \rangle$

lemma $\nu\text{-fin}\text{-Suc}[simp]: \nu\text{-fin} (\text{Suc } n) s = \nu\text{-fin } n s + l \hat{\wedge} n * \int t. r$
 $(t !! n) \partial \mathcal{T} p s$
 $\langle proof \rangle$

lemma $\nu\text{-fin}\text{-zero}[simp]: \nu\text{-fin } 0 s = 0$
 $\langle proof \rangle$

lemma $\nu\text{-fin}\text{-eq}\text{-Pn}: \nu\text{-fin } n s = (\sum i < n. l \hat{\wedge} i * \text{measure}\text{-pmf}.\text{expectation}$
 $(Pn' p s i) r)$
 $\langle proof \rangle$
end

7.4 Expected Total Discounted Reward

definition $\nu p s = \text{lim} (\lambda n. \nu\text{-fin } p n s)$

lemmas $\nu\text{-eq}\text{-lim} = \nu\text{-def}$

lemma $\nu\text{-eq}\text{-Pn}: \nu p s = (\sum i. l \hat{\wedge} i * \text{measure}\text{-pmf}.\text{expectation} (Pn' p$
 $s i) r)$
 $\langle proof \rangle$

7.5 Reward of a Decision Rule

context

fixes $d :: ('s, 'a) \text{dec}$

begin

abbreviation $r\text{-dec } s \equiv \int a. r (s, a) \partial d s$

lemma $\text{abs}\text{-}r\text{-dec}\text{-le}: |r\text{-dec } s| \leq r_M$
 $\langle proof \rangle$

lemma $r\text{-dec}\text{-eq}\text{-}r\text{-}K0: r\text{-dec } s = \text{measure}\text{-pmf}.\text{expectation} (K0' d s) r$
 $\langle proof \rangle$

lemma $r\text{-dec}\text{-bfun}: r\text{-dec} \in \text{bfun}$
 $\langle proof \rangle$

lift-definition $r\text{-dec}_b :: 's \Rightarrow_b \text{real}$ is $r\text{-dec}$
 $\langle proof \rangle$

declare $r\text{-dec}_b.\text{rep}\text{-eq}[simp] \text{ bfun}.\text{Bfun}\text{-inverse}[simp]$

lemma $\text{norm}\text{-}r\text{-dec}\text{-le}: \text{norm } r\text{-dec}_b \leq r_M$
 $\langle proof \rangle$

end

lemma *r-dec-det* [*simp*]: $r\text{-dec } (mk\text{-dec-det } d) s = r (s, d s)$
<proof>

7.6 Transition Probability Matrix for MDPs

context

fixes $p :: nat \Rightarrow ('s, 'a) dec$

begin

definition $\mathcal{P}_X n = push\text{-exp } (\lambda s. Xn' (mk\text{-markovian } p) s n)$

lemma $\mathcal{P}_X\text{-}0$ [*simp*]: $\mathcal{P}_X 0 = id$
<proof>

lemma $\mathcal{P}_X\text{-bounded-linear}$ [*simp*]: *bounded-linear* ($\mathcal{P}_X t$)
<proof>

lemma *norm- \mathcal{P}_X* [*simp*]: *onorm* ($\mathcal{P}_X t$) = 1
<proof>

lemma *norm- \mathcal{P}_X -apply* [*simp*]: *norm* ($\mathcal{P}_X n x$) $\leq norm x$
<proof>

lemma $\mathcal{P}_X\text{-bound-r}$: *norm* ($\mathcal{P}_X t (r\text{-dec}_b (p t))$) $\leq r_M$
<proof>

lemma $\mathcal{P}_X\text{-bounded-r}$: *bounded* (*range* ($\lambda t. (\mathcal{P}_X t (r\text{-dec}_b (p t)))$))
<proof>

end

lemma $\nu\text{-fin-elem}$: $\nu\text{-fin } (mk\text{-markovian } p) n s = (\sum i < n. l\hat{i} * \mathcal{P}_X p i (r\text{-dec}_b (p i)) s)$
<proof>

lemma $\nu_b\text{-fin-eq-}\mathcal{P}_X$: $\nu_b\text{-fin } (mk\text{-markovian } p) n = (\sum i < n. l\hat{i} *_R \mathcal{P}_X p i (r\text{-dec}_b (p i)))$
<proof>

lemma $\nu\text{-fin-eq-}\mathcal{P}_X$: $\nu\text{-fin } (mk\text{-markovian } p) n = (\sum i < n. l\hat{i} *_R \mathcal{P}_X p i (r\text{-dec}_b (p i)))$
<proof>

$\mathcal{P}_1 d v$ defines for each state the expected value of v after taking a single step in the MDP according to the decision rule d .

context

fixes $d :: ('s, 'a) dec$

begin

lift-definition $\mathcal{P}_1 :: ('s \Rightarrow_b \text{real}) \Rightarrow_L ('s \Rightarrow_b \text{real})$ is push-exp (K-st
d)

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-bfun-one}$ [simp]: $\mathcal{P}_1 1 = 1$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-pow-bfun-one}$ [simp]: $(\mathcal{P}_1 \hat{\sim} t) 1 = 1$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-pow}$: $\text{blinfun-apply } (\mathcal{P}_1 \hat{\sim} n) = \text{blinfun-apply } \mathcal{P}_1 \hat{\sim} n$

$\langle \text{proof} \rangle$

lemma $\text{norm-}\mathcal{P}_1$ [simp]: $\text{norm } \mathcal{P}_1 = 1$

$\langle \text{proof} \rangle$

end

lemma $\mathcal{P}_X\text{-Suc}$: $\mathcal{P}_X p (\text{Suc } n) v = \mathcal{P}_1 (p 0) ((\mathcal{P}_X (\lambda n. p (\text{Suc } n))$
 $n) v)$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_X\text{-Suc}'$: $\mathcal{P}_X p (\text{Suc } n) v = \mathcal{P}_X p n (\mathcal{P}_1 (p n) v)$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_X\text{-const}$: $\mathcal{P}_X (\lambda-. d) n = \mathcal{P}_1 d \hat{\sim} n$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_X\text{-sconst}$: $\mathcal{P}_X (\lambda-. p) n = \mathcal{P}_1 p \hat{\sim} n$

$\langle \text{proof} \rangle$

lemma $\text{norm-}\mathcal{P}\text{-}n$ [simp]: $\text{onorm } (\mathcal{P}_1 d \hat{\sim} n) = 1$

$\langle \text{proof} \rangle$

lemma $\text{norm-}\mathcal{P}_1\text{-pow}$ [simp]: $\text{norm } (\mathcal{P}_1 d \hat{\sim} t) = 1$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_X\text{-Suc-n-elem}$: $\mathcal{P}_X p n (\mathcal{P}_1 (p n) v) = \mathcal{P}_X p (\text{Suc } n) v$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-eq-}\mathcal{P}_X\text{-one}$: $\text{blinfun-apply } (\mathcal{P}_1 (p 0)) = \mathcal{P}_X p 1$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-pos}$: $0 \leq u \implies 0 \leq \mathcal{P}_1 d u$

$\langle \text{proof} \rangle$

lemma $\mathcal{P}_1\text{-nonneg}$: $\text{nonneg-blinfun } (\mathcal{P}_1 d)$

$\langle \text{proof} \rangle$

lemma \mathcal{P}_1 -*n-pos*: $0 \leq u \implies 0 \leq (\mathcal{P}_1 \ d \ \widehat{\sim} \ n) \ u$
 ⟨proof⟩

lemma \mathcal{P}_1 -*n-nonneg*: *nonneg-blinfun* $(\mathcal{P}_1 \ d \ \widehat{\sim} \ n)$
 ⟨proof⟩

lemma \mathcal{P}_1 -*n-disc-pos*: $0 \leq u \implies 0 \leq (l \widehat{\sim} \ n \ * _R \ \mathcal{P}_1 \ d \ \widehat{\sim} \ n) \ u$
 ⟨proof⟩

lemma \mathcal{P}_1 -*sum-pos*: $0 \leq u \implies 0 \leq (\sum t \leq n. l \widehat{\sim} \ t \ * _R \ (\mathcal{P}_1 \ d \ \widehat{\sim} \ t)) \ u$
 ⟨proof⟩

lemma \mathcal{P}_1 -*sum-ge*:
assumes $0 \leq u$
shows $u \leq (\sum t \leq n. l \widehat{\sim} \ t \ * _R \ \mathcal{P}_1 \ d \ \widehat{\sim} \ t) \ u$
 ⟨proof⟩

7.7 The Bellman Operator

definition $L \ d \ v \equiv r\text{-dec}_b \ d + l \ * _R \ \mathcal{P}_1 \ d \ v$

lemma *norm-L-le*: *norm* $(L \ d \ v) \leq r_M + l \ * \ \text{norm} \ v$
 ⟨proof⟩

lemma *abs-L-le*: $|L \ d \ v \ s| \leq r_M + l \ * \ \text{norm} \ v$
 ⟨proof⟩

7.7.1 Bellman Operator for Single Actions

abbreviation $L_a \ a \ v \ s \equiv r \ (s, a) + l \ * \ \text{measure-pmf.expectation} \ (K \ (s, a)) \ v$

lemma L_a -*le*:
fixes $v :: 's \Rightarrow_b \ \text{real}$
shows $|L_a \ a \ v \ s| \leq r_M + l \ * \ \text{norm} \ v$
 ⟨proof⟩

lemma L_a -*bounded*:
bounded $(\text{range} \ (\lambda a. L_a \ a \ (\text{apply-bfun} \ v) \ s))$
 ⟨proof⟩

lemma L_a -*int*:
fixes $d :: 'a \ \text{pmf}$ **and** $v :: 's \Rightarrow_b \ \text{real}$
shows $(\int a. L_a \ a \ v \ s \ \partial d) = (\int a. r \ (s, a) \ \partial d) + l \ * \ \int a. \int s'. v \ s'$
 $\partial K \ (s, a) \ \partial d$
 ⟨proof⟩

lemma L -*eq- L_a* : $L \ d \ v \ s = \text{measure-pmf.expectation} \ (d \ s) \ (\lambda a. L_a \ a \ v \ s)$
 ⟨proof⟩

lemma L -eq- L_a -det: $L (mk\text{-}dec\text{-}det\ d) v\ s = L_a (d\ s) v\ s$
 ⟨proof⟩

lemma L_a -eq- L : $measure\text{-}pmf.\text{expectation}\ p (\lambda a. L_a\ a\ (apply\text{-}bfun\ v)\ s) =$
 $L (\lambda t. \text{if } t = s \text{ then } p \text{ else } return\text{-}pmf\ (SOME\ a. a \in A\ t)) v\ s$
 ⟨proof⟩

lemma L -le: $L\ d\ v\ s \leq r_M + l * norm\ v$
 ⟨proof⟩

lemma L_a -le': $L_a\ a\ (apply\text{-}bfun\ v)\ s \leq r_M + l * norm\ v$
 ⟨proof⟩

7.8 Optimality Equations

definition $\mathcal{L} (v :: 's \Rightarrow_b\ real) s = (\bigsqcup d \in D_R. L\ d\ v\ s)$

lemma \mathcal{L} -bfun: $\mathcal{L}\ v \in bfun$
 ⟨proof⟩

lift-definition $\mathcal{L}_b :: ('s \Rightarrow_b\ real) \Rightarrow 's \Rightarrow_b\ real$ is \mathcal{L}
 ⟨proof⟩

lemma L -bounded[simp, intro]: $bounded\ (range\ (\lambda p. L\ p\ v\ s))$
 ⟨proof⟩

lemma L -bounded'[simp, intro]: $bounded\ ((\lambda p. L\ p\ v\ s) \text{ ' } X)$
 ⟨proof⟩

lemma L -bdd-above[simp, intro]: $bdd\text{-}above\ ((\lambda p. L\ p\ v\ s) \text{ ' } X)$
 ⟨proof⟩

lemma L -le- \mathcal{L}_b : $is\text{-}dec\ d \implies L\ d\ v \leq \mathcal{L}_b\ v$
 ⟨proof⟩

7.8.1 Equivalences involving \mathcal{L}_b

lemma SUP-step-MR-eq:
 $\mathcal{L}\ v\ s = (\bigsqcup pa \in \{pa. set\text{-}pmf\ pa \subseteq A\ s\}. (\int a. L_a\ a\ v\ s\ \partial measure\text{-}pmf\ pa))$
 ⟨proof⟩

lemma \mathcal{L}_b -eq-SUP- L_a : $\mathcal{L}_b\ v\ s = (\bigsqcup p \in \{p. set\text{-}pmf\ p \subseteq A\ s\}. \int a. L_a\ a\ v\ s\ \partial measure\text{-}pmf\ p)$
 ⟨proof⟩

lemma SUP-step-det-eq: $(\bigsqcup d \in D_D. L (mk\text{-}dec\text{-}det\ d) v\ s) = (\bigsqcup a \in A\ s. L_a\ a\ v\ s)$

$\langle proof \rangle$

lemma *integrable- L_a* : *integrable (measure-pmf x) ($\lambda a. L_a a$ (apply-bfun v) s)*

$\langle proof \rangle$

lemma *SUP- L_a -eq-det*:

fixes $v :: 's \Rightarrow_b \text{real}$

shows $(\bigsqcup p \in \{p. \text{set-pmf } p \subseteq A \text{ s}\}. \int a. L_a a v s \partial \text{measure-pmf } p) = (\bigsqcup a \in A \text{ s}. L_a a v s)$

$\langle proof \rangle$

lemma *\mathcal{L} -eq-SUP-det*: $\mathcal{L} v s = (\bigsqcup d \in D_D. L (mk\text{-dec-det } d) v s)$

$\langle proof \rangle$

lemma *\mathcal{L}_b -eq-SUP-det*: $\mathcal{L}_b v s = (\bigsqcup d \in D_D. L (mk\text{-dec-det } d) v s)$

$\langle proof \rangle$

7.9 Monotonicity

lemma *\mathcal{P}_X -mono[*intro*]*: $a \leq b \implies \mathcal{P}_X p n a \leq \mathcal{P}_X p n b$

$\langle proof \rangle$

lemma *\mathcal{P}_1 -mono[*intro*]*: $a \leq b \implies \mathcal{P}_1 p a \leq \mathcal{P}_1 p b$

$\langle proof \rangle$

lemma *L -mono[*intro*]*: $u \leq v \implies L d u \leq L d v$

$\langle proof \rangle$

lemma *\mathcal{L}_b -mono[*intro*]*: $u \leq v \implies \mathcal{L}_b u \leq \mathcal{L}_b v$

$\langle proof \rangle$

lemma *step-mono*:

assumes $\mathcal{L}_b v \leq v d \in D_R$

shows $L d v \leq v$

$\langle proof \rangle$

lemma *step-mono-elem-det*:

assumes $v \leq \mathcal{L}_b v e > 0$

shows $\exists d \in D_D. v \leq L (mk\text{-dec-det } d) v + e *_R 1$

$\langle proof \rangle$

lemma *step-mono-elem*:

assumes $v \leq \mathcal{L}_b v e > 0$

shows $\exists d \in D_R. v \leq L d v + e *_R 1$

$\langle proof \rangle$

lemma *\mathcal{P}_X - L -le*:

assumes $\mathcal{L}_b v \leq v p \in \Pi_{MR}$

shows $\mathcal{P}_X p n (L (p n) v) \leq \mathcal{P}_X p n v$
 $\langle \text{proof} \rangle$

end

locale *MDP-reward-disc* = *MDP-reward* *A K r l*
for

A **and**

K :: '*s* :: countable × '*a* :: countable ⇒ '*s* pmf **and**

r l +

assumes

disc-lt-one [*simp*]: $l < 1$

begin

definition *is-opt-act* *v s* = *is-arg-max* ($\lambda a. L_a a v s$) ($\lambda a. a \in A s$)

abbreviation *opt-acts* *v s* $\equiv \{a. \text{is-opt-act } v s a\}$

lemma *summable-disc* [*intro*, *simp*]: *summable* ($\lambda i. l \hat{\ } i * x$)
 $\langle \text{proof} \rangle$

lemma *summable-r-disc* [*intro*, *simp*]:

summable ($\lambda i. |l \hat{\ } i * r (sa i)|$)

summable ($\lambda i. l \hat{\ } i * |r (sa i)|$)

summable ($\lambda i. l \hat{\ } i * r (sa i)$)

$\langle \text{proof} \rangle$

lemma *summable-norm-disc-I* [*intro*]:

assumes *summable* ($\lambda t. (l \hat{\ } t * \text{norm } F)$)

shows *summable* ($\lambda t. \text{norm } (l \hat{\ } t *_{R} F)$)

$\langle \text{proof} \rangle$

lemma *summable-norm-disc-I'* [*intro*]:

assumes *summable* ($\lambda t. (l \hat{\ } t * \text{norm } (F t))$)

shows *summable* ($\lambda t. \text{norm } (l \hat{\ } t *_{R} F t)$)

$\langle \text{proof} \rangle$

lemma *summable-discI* [*intro*]:

assumes *bounded* (*range* *F*)

shows *summable* ($\lambda t. l \hat{\ } t * \text{norm } (F t)$)

$\langle \text{proof} \rangle$

lemma *summable-disc-reward* [*intro*]:

assumes *bounded* (*range* (*F* :: *nat* ⇒ '*b* :: *banach*))

shows *summable* ($\lambda t. l \hat{\ } t *_{R} (F t)$)

$\langle \text{proof} \rangle$

lemma *summable-norm-bfun-disc*: *summable* ($\lambda t. l \hat{\ } t * \text{norm } (\text{apply-bfun } f t)$)

$\langle \text{proof} \rangle$

lemma *summable-bfun-disc* [*simp*]: *summable* ($\lambda t. l \hat{\wedge} t * (\text{apply-bfun } f \ t)$)
 ⟨*proof*⟩

lemma *norm-bfun-disc-le*: $\text{norm } f \leq B \implies (\sum x. l \hat{\wedge} x * \text{norm } (\text{apply-bfun } f \ x)) \leq (\sum x. l \hat{\wedge} x * B)$
 ⟨*proof*⟩

lemma *norm-bfun-disc-le'*: $\text{norm } f \leq B \implies (\sum x. l \hat{\wedge} x * (\text{apply-bfun } f \ x)) \leq (\sum x. l \hat{\wedge} x * B)$
 ⟨*proof*⟩

lemma *sum-disc-lim-l*: $(\sum x. l \hat{\wedge} x * B) = B / (1-l)$
 ⟨*proof*⟩

lemma *sum-disc-bound*: $(\sum x. l \hat{\wedge} x * \text{apply-bfun } f \ x) \leq (\text{norm } f) / (1-l)$
 ⟨*proof*⟩

lemma *sum-disc-bound'*:
fixes $f :: \text{nat} \Rightarrow 'b \Rightarrow_b \text{real}$
assumes $h: \forall n. \text{norm } (f \ n) \leq B$
shows $\text{norm } (\sum x. l \hat{\wedge} x *_R f \ x) \leq B / (1-l)$
 ⟨*proof*⟩

lemma *abs- ν -trace-le*: $|\nu\text{-trace } t| \leq (\sum i. l \hat{\wedge} i * r_M)$
 ⟨*proof*⟩

lemma *integrable- ν -trace*: *integrable* ($\mathcal{T} \ p \ s$) $\nu\text{-trace}$
 ⟨*proof*⟩

context
fixes $p :: ('s, 'a) \text{pol}$
begin

lemma *ν -eq- ν -trace*: $\nu \ p \ s = \int t. \nu\text{-trace } t \ \partial \mathcal{T} \ p \ s$
 ⟨*proof*⟩

lemma *abs- ν -le*: $|\nu \ p \ s| \leq (\sum i. l \hat{\wedge} i * r_M)$
 ⟨*proof*⟩

lemma *ν -le*: $\nu \ p \ s \leq (\sum i. l \hat{\wedge} i * r_M)$
 ⟨*proof*⟩

lemma *ν -bfun*: $\nu \ p \in \text{bfun}$
 ⟨*proof*⟩

lift-definition $\nu_b :: 's \Rightarrow_b \text{real is } \nu p$
 ⟨proof⟩

lemma *norm- ν -le*: $\text{norm } \nu_b \leq r_M / (1-l)$
 ⟨proof⟩
end

lemma *ν -as-markovian*: $\nu (\text{mk-markovian } (\text{as-markovian } p (\text{return-pmf } s))) s = \nu p s$
 ⟨proof⟩

lemma *ν_b -as-markovian*: $\nu_b (\text{mk-markovian } (\text{as-markovian } p (\text{return-pmf } s))) s = \nu_b p s$
 ⟨proof⟩

7.10 Optimal Reward

definition *ν -MD* $s \equiv \bigsqcup p \in \Pi_{MD}. \nu (\text{mk-markovian-det } p) s$
definition *ν -opt* $s \equiv \bigsqcup p \in \Pi_{HR}. \nu p s$

lemma *ν -opt-bfun*: $\nu\text{-opt} \in \text{bfun}$
 ⟨proof⟩

lift-definition $\nu_b\text{-opt} :: 's \Rightarrow_b \text{real is } \nu\text{-opt}$
 ⟨proof⟩

lemma *ν_b -opt-eq*: $\nu_b\text{-opt } s = (\bigsqcup p \in \Pi_{HR}. \nu_b p s)$
 ⟨proof⟩

lemma *ν -le- ν -opt* [*intro*]:
assumes *is-policy* p
shows $\nu p s \leq \nu\text{-opt } s$
 ⟨proof⟩

lemma *ν_b -le-opt* [*intro*]: $p \in \Pi_{HR} \implies \nu_b p \leq \nu_b\text{-opt}$
 ⟨proof⟩

lemma *ν_b -le-opt-MD* [*intro*]: $p \in \Pi_{MD} \implies \nu_b (\text{mk-markovian-det } p) \leq \nu_b\text{-opt}$
 ⟨proof⟩

lemma *ν_b -le-opt-DD* [*intro*]: *is-dec-det* $d \implies \nu_b (\text{mk-stationary-det } d) \leq \nu_b\text{-opt}$
 ⟨proof⟩

lemma *ν_b -le-opt-DR* [*intro*]: *is-dec* $d \implies \nu_b (\text{mk-stationary } d) \leq \nu_b\text{-opt}$
 ⟨proof⟩

lemma ν_b -opt-eq-MR: ν_b -opt $s = (\bigsqcup p \in \Pi_{MR}. \nu_b (mk\text{-markovian } p)$
 $s)$
 $\langle proof \rangle$

lemma *summable-norm-disc-reward'*[simp]: *summable* ($\lambda t. l \hat{\sim} t * norm$
 $(\mathcal{P}_X p t (r\text{-dec}_b (p t)))$)
 $\langle proof \rangle$

lemma *summable-disc-reward- \mathcal{P}_X* [simp]: *summable* ($\lambda t. l \hat{\sim} t *_{R} \mathcal{P}_X p$
 $t (r\text{-dec}_b (p t))$)
 $\langle proof \rangle$

lemma *disc-reward-tendsto*:
 $(\lambda n. \sum t < n. l \hat{\sim} t *_{R} \mathcal{P}_X p t (r\text{-dec}_b (p t))) \longrightarrow (\sum t. l \hat{\sim} t *_{R} \mathcal{P}_X$
 $p t (r\text{-dec}_b (p t)))$
 $\langle proof \rangle$

lemma ν -eq- \mathcal{P}_X : $\nu (mk\text{-markovian } p) = (\sum i. l \hat{\sim} i *_{R} \mathcal{P}_X p i (r\text{-dec}_b$
 $(p i)))$
 $\langle proof \rangle$

lemma ν_b -eq- \mathcal{P}_X : $\nu_b (mk\text{-markovian } p) = (\sum i. l \hat{\sim} i *_{R} \mathcal{P}_X p i (r\text{-dec}_b$
 $(p i)))$
 $\langle proof \rangle$

lemma ν_b -fin-tendsto- ν_b : (ν_b -fin (*mk-markovian* p)) $\longrightarrow \nu_b$ (*mk-markovian*
 p)
 $\langle proof \rangle$

lemma *norm- \mathcal{P}_1 -l-less*: *norm* ($l *_{R} \mathcal{P}_1 d$) < 1
 $\langle proof \rangle$

lemma *disc- \mathcal{P}_1 -tendsto*: ($\lambda n. (\sum t \leq n. l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t)$) $\longrightarrow (\sum t.$
 $l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t)$
 $\langle proof \rangle$

lemma *disc- \mathcal{P}_1 -lim*: *lim* ($\lambda n. (\sum t \leq n. l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t)$) = ($\sum t. l \hat{\sim} t$
 $*_{R} \mathcal{P}_1 d \hat{\sim} t)$
 $\langle proof \rangle$

lemma *convergent-disc- \mathcal{P}_1* : *convergent* ($\lambda n. (\sum t \leq n. l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t)$)
 $\langle proof \rangle$

lemma \mathcal{P}_1 -suminf-ge:
assumes $0 \leq u$ **shows** $u \leq (\sum t. l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t) u$
 $\langle proof \rangle$

lemma \mathcal{P}_1 -suminf-pos:
assumes $0 \leq u$
shows $0 \leq (\sum t. l \hat{\sim} t *_{R} \mathcal{P}_1 d \hat{\sim} t) u$

$\langle proof \rangle$

lemma *lemma-6-1-2-b*:

assumes $v \leq u$

shows $(\sum t. l \hat{t} *_R \mathcal{P}_1 d \sim t) v \leq (\sum t. l \hat{t} *_R \mathcal{P}_1 d \sim t) u$

$\langle proof \rangle$

lemma *ν -stationary*: $\nu_b (mk\text{-stationary } d) = (\sum t. l \hat{t} *_R (\mathcal{P}_1 d \sim t)) (r\text{-dec}_b d)$

$\langle proof \rangle$

lemma *ν -stationary-inv*: $\nu_b (mk\text{-stationary } d) = inv_L (id\text{-blinfun } - l *_R \mathcal{P}_1 d) (r\text{-dec}_b d)$

$\langle proof \rangle$

The value of a markovian policy can be expressed in terms of L .

lemma *ν -step*: $\nu_b (mk\text{-markovian } p) = L (p 0) (\nu_b (mk\text{-markovian } (\lambda n. p (Suc n))))$

$\langle proof \rangle$

lemma *L - ν -fix*: $\nu_b (mk\text{-stationary } d) = L d (\nu_b (mk\text{-stationary } d))$

$\langle proof \rangle$

lemma *L -fix- ν* :

assumes $L p v = v$

shows $v = \nu_b (mk\text{-stationary } p)$

$\langle proof \rangle$

lemma *L - ν -fix-iff*: $L d v = v \longleftrightarrow v = \nu_b (mk\text{-stationary } d)$

$\langle proof \rangle$

7.11 Properties of Solutions of the Optimality Equations

abbreviation $\mathcal{P}_d p n v \equiv l \hat{n} *_R \mathcal{P}_X p n v$

lemma *\mathcal{P}_d -lim*: $(\lambda n. (\mathcal{P}_d p n v)) \longrightarrow 0$

$\langle proof \rangle$

lemma *\mathcal{L} -dec-ge-opt*:

assumes $\mathcal{L}_b v \leq v$

shows $\nu_b\text{-opt} \leq v$

$\langle proof \rangle$

lemma *\mathcal{L} -inc-le-opt*:

assumes $v \leq \mathcal{L}_b v$
shows $v \leq \nu_b\text{-opt}$
 ⟨proof⟩
lemma $\mathcal{L}\text{-fix-imp-opt}$:
assumes $v = \mathcal{L}_b v$
shows $v = \nu_b\text{-opt}$
 ⟨proof⟩

lemma bounded-P : $\text{bounded } (P_1 \text{ ' } X)$
 ⟨proof⟩

7.12 Solutions to the Optimality Equation

7.12.1 \mathcal{L}_b and L are Contraction Mappings

declare $\text{bounded-apply-blinfun}$ [intro] $\text{bounded-apply-bfun}$ '[intro]

lemma contraction-L : $\text{dist } (\mathcal{L}_b v) (\mathcal{L}_b u) \leq l * \text{dist } v u$
 ⟨proof⟩

lemma is-contraction-L : $\text{is-contraction } \mathcal{L}_b$
 ⟨proof⟩

lemma contraction-L : $\text{dist } (L p v) (L p u) \leq l * \text{dist } v u$
 ⟨proof⟩

lemma is-contraction-L : $\text{is-contraction } (L p)$
 ⟨proof⟩

7.12.2 Existence of a Fixpoint of \mathcal{L}_b

lemma $\mathcal{L}_b\text{-conv}$:
 $\exists! v. \mathcal{L}_b v = v$ $(\lambda n. (\mathcal{L}_b \text{ } \tilde{n}) v) \longrightarrow (\text{THE } v. \mathcal{L}_b v = v)$
 ⟨proof⟩

lemma $\mathcal{L}_b\text{-fix-iff-opt}$ [simp]: $\mathcal{L}_b v = v \longleftrightarrow v = \nu_b\text{-opt}$
 ⟨proof⟩

lemma $\nu_b\text{-opt-fix}$: $\nu_b\text{-opt} = (\text{THE } v. \mathcal{L}_b v = v)$
 ⟨proof⟩

lemma $\mathcal{L}_b\text{-opt}$ [simp]: $\mathcal{L}_b \nu_b\text{-opt} = \nu_b\text{-opt}$
 ⟨proof⟩

lemma $\mathcal{L}_b\text{-lim}$: $(\lambda n. (\mathcal{L}_b \text{ } \tilde{n}) v) \longrightarrow \nu_b\text{-opt}$
 ⟨proof⟩

lemma thm-6-2-6 : $\nu_b p = \nu_b\text{-opt} \longleftrightarrow \mathcal{L}_b (\nu_b p) = \nu_b p$
 ⟨proof⟩

lemma *thm-6-2-6'*: $\nu p = \nu\text{-opt} \iff \mathcal{L}_b(\nu_b p) = \nu_b p$
 ⟨proof⟩

7.13 Existence of Optimal Policies

definition ν -improving $v d \iff (\forall s. \text{is-arg-max } (\lambda d. (L d v) s) (\lambda d. d \in D_R) d)$

lemma ν -improving-iff: ν -improving $v d \iff d \in D_R \wedge (\forall d' \in D_R. \forall s. L d' v s \leq L d v s)$
 ⟨proof⟩

lemma ν -improving-D-MR[dest]: ν -improving $v d \implies d \in D_R$
 ⟨proof⟩

lemma ν -improving-ge: ν -improving $v d \implies d' \in D_R \implies L d' v s \leq L d v s$
 ⟨proof⟩

lemma ν -improving-imp- \mathcal{L}_b : ν -improving $v d \implies \mathcal{L}_b v = L d v$
 ⟨proof⟩

lemma \mathcal{L}_b -imp- ν -improving:
 assumes $d \in D_R \mathcal{L}_b v = L d v$
 shows ν -improving $v d$
 ⟨proof⟩

lemma ν -improving-alt:
 assumes $d \in D_R$
 shows ν -improving $v d \iff \mathcal{L}_b v = L d v$
 ⟨proof⟩

definition ν -conserving $d = \nu$ -improving $(\nu_b\text{-opt}) d$

lemma ν -conserving-iff: ν -conserving $d \iff d \in D_R \wedge (\forall d' \in D_R. \forall s. L d' \nu_b\text{-opt} s \leq L d \nu_b\text{-opt} s)$
 ⟨proof⟩

lemma ν -conserving-ge: ν -conserving $d \implies d' \in D_R \implies L d' \nu_b\text{-opt} s \leq L d \nu_b\text{-opt} s$
 ⟨proof⟩

lemma ν -conserving-imp- \mathcal{L}_b [simp]: ν -conserving $d \implies L d \nu_b\text{-opt} = \nu_b\text{-opt}$
 ⟨proof⟩

lemma \mathcal{L}_b -imp- ν -conserving:
 assumes $d \in D_R \mathcal{L}_b \nu_b\text{-opt} = L d \nu_b\text{-opt}$
 shows ν -conserving d

$\langle proof \rangle$

lemma ν -conserving-alt:

assumes $d \in D_R$

shows ν -conserving $d \iff \mathcal{L}_b \nu_b\text{-opt} = L d \nu_b\text{-opt}$

$\langle proof \rangle$

lemma ν -conserving-alt':

assumes $d \in D_R$

shows ν -conserving $d \iff L d \nu_b\text{-opt} = \nu_b\text{-opt}$

$\langle proof \rangle$

7.13.1 Conserving Decision Rules are Optimal

theorem *ex-improving-imp-conserving*:

assumes $\bigwedge v. \exists d. \nu$ -improving v (*mk-dec-det* d)

shows $\exists d. \nu$ -conserving (*mk-dec-det* d)

$\langle proof \rangle$

theorem *conserving-imp-opt[simp]*:

assumes ν -conserving (*mk-dec-det* d)

shows ν_b (*mk-stationary-det* d) = $\nu_b\text{-opt}$

$\langle proof \rangle$

lemma *conserving-imp-opt'*:

assumes $\exists d. \nu$ -conserving (*mk-dec-det* d)

shows $\exists d \in D_D. (\nu_b$ (*mk-stationary-det* d)) = $\nu_b\text{-opt}$

$\langle proof \rangle$

theorem *improving-att-imp-det-opt*:

assumes $\bigwedge v. \exists d. \nu$ -improving v (*mk-dec-det* d)

shows $\nu_b\text{-opt } s = (\bigsqcup d \in D_D. \nu_b$ (*mk-stationary-det* d) s)

$\langle proof \rangle$

lemma \mathcal{L}_b -sup-att-dec:

assumes $d \in D_R \mathcal{L}_b v = L d v$

shows $\exists d' \in D_D. \mathcal{L}_b v = L$ (*mk-dec-det* d') v

$\langle proof \rangle$

lemma \mathcal{L}_b -sup-att-dec':

assumes $d \in D_R \mathcal{L}_b v = L d v$

shows $\exists d' \in D_D. \nu$ -improving v (*mk-dec-det* d')

$\langle proof \rangle$

7.13.2 Deterministic Decision Rules are Optimal

lemma *opt-imp-opt-dec-det*:

assumes $p \in \Pi_{HR} \nu_b p = \nu_b\text{-opt}$

shows $\exists d \in D_D. \nu_b$ (*mk-stationary-det* d) = $\nu_b\text{-opt}$

$\langle proof \rangle$

7.13.3 Optimal Decision Rules for Finite Action Spaces

lemma *ex-opt-act*:

assumes $\bigwedge s. finite (A s)$

shows $\exists a \in A s. L_a a (v :: - \Rightarrow_b -) s = \mathcal{L}_b v s$

$\langle proof \rangle$

lemma *ex-opt-dec-det*:

assumes $\bigwedge s. finite (A s)$

shows $\exists d \in D_D. L (mk-dec-det d) (v :: - \Rightarrow_b -) = \mathcal{L}_b v$

$\langle proof \rangle$

lemma *thm-6-2-10*:

assumes $\bigwedge s. finite (A s)$

shows $\exists d \in D_D. \nu_b-opt = \nu_b (mk-stationary-det d)$

$\langle proof \rangle$

7.13.4 Existence of Epsilon-Optimal Policies

lemma *ex-det-eps*:

assumes $0 < e$

shows $\exists d \in D_D. \mathcal{L}_b v \leq L (mk-dec-det d) v + e *_R 1$

$\langle proof \rangle$

lemma *thm-6-2-11*:

assumes $eps > 0$

shows $\exists d \in D_D. \nu_b-opt \leq \nu_b (mk-stationary-det d) + eps *_R 1$

$\langle proof \rangle$

lemma *ex-det-dist-eps*:

assumes $0 < (e :: real)$

shows $\exists d \in D_D. dist (\mathcal{L}_b v) (L (mk-dec-det d) v) \leq e$

$\langle proof \rangle$

lemma *less-imp-ex-add-le*: $(x :: real) < y \implies \exists eps > 0. x + eps \leq y$

$\langle proof \rangle$

lemma *$\nu_b-opt-le-det$* : $\nu_b-opt s \leq (\bigsqcup d \in D_D. \nu_b (mk-stationary-det d) s)$

$\langle proof \rangle$

lemma *$\nu_b-opt-eq-det$* : $\nu_b-opt s = (\bigsqcup d \in D_D. \nu_b (mk-stationary-det d) s)$

$\langle proof \rangle$

lemma *lemma-6-3-1-a*:

assumes $v0 \in bfun$

shows *uniform-limit UNIV* $(\lambda n. ((\lambda v. \mathcal{L} (Bfun v)) \overset{\sim}{\sim} n) v0) \nu\text{-opt}$
sequentially

$\langle proof \rangle$

lemma *dist-Suc-tendsto-zero*:

assumes $(\lambda n. f n) \longrightarrow (y::\text{real-normed-vector})$

shows $(\lambda n. \text{dist } (f n) (f (Suc n))) \longrightarrow 0$

$\langle proof \rangle$

lemma *dist- \mathcal{L}_b -tendsto*: $(\lambda n. \text{dist } ((\mathcal{L}_b \overset{\sim}{\sim} n) v) ((\mathcal{L}_b \overset{\sim}{\sim} (Suc n)) v))$
 $\longrightarrow 0$

$\langle proof \rangle$

definition *max-L-ex s v* $\equiv \text{has-arg-max } (\lambda a. L_a a v s) (A s)$

lemma *ν_b -fin-zero[simp]*: $\nu_b\text{-fin } p \ 0 = 0$

$\langle proof \rangle$

lemma *ν_b -fin-Suc[simp]*:

$\nu_b\text{-fin } (mk\text{-stationary } d) (Suc n) = \nu_b\text{-fin } (mk\text{-stationary } d) n + ((l$
 $*_R \mathcal{P}_1 d) \overset{\sim}{\sim} n) (r\text{-dec}_b d)$

$\langle proof \rangle$

lemma *ν_b -fin-eq*: $\nu_b\text{-fin } (mk\text{-stationary } d) n = (\sum i < n. ((l *_R \mathcal{P}_1$
 $d) \overset{\sim}{\sim} i)) (r\text{-dec}_b d)$

$\langle proof \rangle$

lemma *L-iter*: $(L d \overset{\sim}{\sim} m) v = \nu_b\text{-fin } (mk\text{-stationary } d) m + ((l *_R$
 $\mathcal{P}_1 d) \overset{\sim}{\sim} m) v$

$\langle proof \rangle$

lemma *bounded-stationary- ν_b -fin*: *bounded* $((\lambda x. (\nu_b\text{-fin } (mk\text{-stationary}$
 $x) N) s) ' X)$

$\langle proof \rangle$

lemma *bounded-disc- \mathcal{P}_1* : *bounded* $((\lambda x. (((l *_R \mathcal{P}_1 x) \overset{\sim}{\sim} m) v) s) ' X)$

$\langle proof \rangle$

lemma *bounded-disc- \mathcal{P}_1'* : *bounded* $((\lambda x. ((\mathcal{P}_1 x \overset{\sim}{\sim} m) v) s) ' X)$

$\langle proof \rangle$

lemma *L-iter-le- \mathcal{L}_b* : *is-dec* $d \implies (L d \overset{\sim}{\sim} n) v \leq (\mathcal{L}_b \overset{\sim}{\sim} n) v$

$\langle proof \rangle$

end

7.14 More Restrictive MDP Locales

locale *MDP-fin-acts* = *discrete-MDP* +
assumes $\bigwedge s. \text{finite } (A \ s)$

locale *MDP-att- \mathcal{L}* = *MDP-reward-disc* *A K r l*
for
A **and**
K :: 's :: countable \times 'a :: countable \Rightarrow 's pmf **and**
r **and** *l* +
assumes *Sup-att: max-L-ex* (*s* :: 's) *v*

begin

theorem *\mathcal{L}_b -eq-argmax- L_a* :

fixes *v* :: 's \Rightarrow_b real

assumes *is-arg-max* ($\lambda a. L_a \ a \ v \ s$) ($\lambda a. a \in A \ s$) *a*

shows $\mathcal{L}_b \ v \ s = L_a \ a \ v \ s$

<proof>

lemma *L_a -le-arg-max*: $a \in A \ s \Rightarrow L_a \ a \ v \ s \leq L_a \ (\text{arg-max-on } (\lambda a. L_a \ a \ v \ s) \ (A \ s)) \ v \ s$

<proof>

lemma *arg-max-on-in*: *has-arg-max* *f Q* \Rightarrow *arg-max-on* *f Q* $\in Q$

<proof>

lemma *\mathcal{L}_b -eq- L_a -max*: $\mathcal{L}_b \ v \ s = L_a \ (\text{arg-max-on } (\lambda a. L_a \ a \ v \ s) \ (A \ s)) \ v \ s$

<proof>

lemma *ex-opt-det*: $\exists d \in D_D. \mathcal{L}_b \ v = L \ (\text{mk-dec-det } d) \ v$

<proof>

lemma *ex-improving-det*: $\exists d \in D_D. \nu$ -improving *v* (*mk-dec-det* *d*)

<proof>

end

locale *MDP-act* = *discrete-MDP* *A K* **for** *A* :: 's :: countable \Rightarrow 'a :: countable
set **and** *K* +

fixes *arb-act* :: 'a *set* \Rightarrow 'a

assumes *arb-act-in[simp]*: $X \neq \{\}$ \Rightarrow *arb-act* $X \in X$

locale *MDP-act-disc* = *MDP-act* *A K* + *MDP-att- \mathcal{L}* *A K r l*

for *A* :: 's :: countable \Rightarrow 'a :: countable *set* **and** *K r l*

begin

lemma *is-opt-act-some*: *is-opt-act* *v s* (*arb-act* (*opt-acts* *v s*))

<proof>

lemma *some-opt-acts-in-A*: *arb-act* (*opt-acts* *v s*) $\in A \ s$

<proof>

lemma *ν -improving-opt-acts: ν -improving $v0$ (mk-dec-det (λs . arb-act (opt-acts (apply-bfun $v0$) s)))*
<proof>

end

locale *MDP-finite-type = MDP-reward-disc $A K r l$*
for *A and $K :: 's :: finite \times 'a :: finite \Rightarrow 's pmf$ and $r l$*

end

References

- [1] J. Hölzl and T. Nipkow. Markov models. *Archive of Formal Proofs*, Jan. 2012. https://isa-afp.org/entries/Markov_Models.html, Formal proof development.
- [2] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.