Markov Decision Processes with Rewards

Maximilian Schäffeler and Mohammad Abdulaziz

March 17, 2025

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

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* (‹(- $\Rightarrow_b$ -)› [*22*] *21*) =
  bfun::(*'a* ⇒ *'b* :: *metric-space*) *set*
  morphisms *apply-bfun Bfun*
  ⟨*proof*⟩

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

setup-lifting *type-definition-bfun*

lemma *bounded-apply-bfun*[*intro*, *simp*]: *bounded* ((*apply-bfun x*) ‘ *X*)
  ⟨*proof*⟩

lemma *apply-bfun-bdd-above*[*simp*, *intro*]:
  fixes *f* :: *'c* $\Rightarrow_b$ *real*
  shows *bdd-above* (*f* ‘ *X*)
  ⟨*proof*⟩

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

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

lemma *bfunE*:

assumes $f \in$ *bfun*
obtains g **where** $f =$ *apply-bfun* g
⟨*proof*⟩

lemma *const-bfun*: $(\lambda x.\ b) \in$ *bfun*
⟨*proof*⟩

lift-definition *const-bfun*::$'b \Rightarrow ('a \Rightarrow_b ('b ::$ *metric-space*)) **is** $\lambda(c::'b)$ -. c
⟨*proof*⟩

lemma *bounded-dist-le-SUP-dist*:
bounded (*range* f) $\Longrightarrow$ *bounded* (*range* g) $\Longrightarrow$ *dist* $(f\ x)\ (g\ x) \leq$ (*SUP* x. *dist* $(f\ x)\ (g\ x)$)
⟨*proof*⟩

instantiation *bfun* :: (*type*, *metric-space*) *metric-space*
begin

lift-definition *dist-bfun* :: $('a \Rightarrow_b 'b) \Rightarrow ('a \Rightarrow_b 'b) \Rightarrow$ *real*
is $\lambda f\ g$. (*SUP* x. *dist* $(f\ x)\ (g\ x)$) ⟨*proof*⟩

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

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

lemma *dist-bounded*:
fixes $f\ g :: 'a \Rightarrow_b 'b$
shows *dist* $(f\ x)\ (g\ x) \leq$ *dist* $f\ g$
⟨*proof*⟩

lemma *dist-bound*:
fixes $f\ g :: 'a \Rightarrow_b ('b ::$ *metric-space*)
assumes $\bigwedge x$. *dist* $(f\ x)\ (g\ x) \leq b$
shows *dist* $f\ g \leq b$
⟨*proof*⟩

lemma *dist-fun-lt-imp-dist-val-lt*:
fixes $f\ g :: 'a \Rightarrow_b 'b$
assumes *dist* $f\ g < e$
shows *dist* $(f\ x)\ (g\ x) < e$
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

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

lemmas *mem-PiCD* = *mem-PiC-iff*[*THEN iffD1*]
  and *mem-PiCI* = *mem-PiC-iff*[*THEN iffD2*]

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

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

1.2 Supremum Norm

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

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

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

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

lemma *scaleR-cont*: $f \in bfun \Longrightarrow (\lambda x.\ a *_R f\ x) \in bfun$ **for** $f :: 'a \Rightarrow 'b$
  $\langle proof \rangle$

lemma *bfun-normI*[*intro*]: $(\bigwedge x.\ norm\ (f\ x) \le b) \Longrightarrow f \in bfun$

⟨*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** λ-. *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'*: $f \in bfun \Longrightarrow (\lambda x.\ a * f\ x) \in bfun$ **for** $f :: 'a \Rightarrow real$
⟨*proof*⟩

lemma *bfun-norm-le-SUP-norm*:
$f \in bfun \Longrightarrow norm\ (f\ x) \leq (SUP\ x.\ norm\ (f\ x))$
⟨*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
⟨*proof*⟩
end

lemma *norm-bfun-def'*: $norm\ f\ =\ (\bigsqcup x.\ norm\ ((f\ ::\ 'a\ \Rightarrow_b\ 'b\ ::\ real\text{-}normed\text{-}vector)\ x))$
⟨*proof*⟩

lemma *norm-le-norm-bfun*: $norm\ (apply\text{-}bfun\ f\ x) \leq norm\ f$
⟨*proof*⟩

lemma *abs-le-norm-bfun*: $abs\ (apply\text{-}bfun\ f\ x) \leq norm\ f$
⟨*proof*⟩

lemma *le-norm-bfun*: $apply\text{-}bfun\ f\ x \leq norm\ f$
⟨*proof*⟩

1.3 Complete Space

lemma *tendsto-add*: $P \longrightarrow (L :: 'a :: real\text{-}normed\text{-}vector) \Longrightarrow (\lambda n.\ P\ n + c) \longrightarrow L + c$
⟨*proof*⟩

lemma *lim-add*: $convergent\ P \Longrightarrow lim\ (\lambda n.\ P\ n + (c :: 'a :: real\text{-}normed\text{-}vector)) = lim\ P + c$
⟨*proof*⟩

lemma *complete-bfun*:
assumes *cauchy-f*: $Cauchy\ (f\ ::\ nat \Rightarrow ('a,\ \ 'b\ ::\ \{complete\text{-}space, real\text{-}normed\text{-}vector\})\ bfun)$
shows *convergent f*
⟨*proof*⟩

lemma *norm-bound*:
fixes $f\ ::\ ('a,\ 'b::real\text{-}normed\text{-}vector)\ bfun$
assumes $\bigwedge x.\ norm\ (apply\text{-}bfun\ f\ x) \leq b$
shows $norm\ f \leq b$
⟨*proof*⟩

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

instance $bfun :: (type,\ banach)\ banach$
⟨*proof*⟩

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

  shows *integrable* S v
  ⟨*proof*⟩

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

lemma *scale-bfun*[*intro*!]: $f \in bfun \Longrightarrow (\lambda x.\ (k{::}real) * f\ x) \in bfun$
  ⟨*proof*⟩

lemma *bfun-spec*[*intro*]: $f \in bfun \Longrightarrow (\lambda x.\ f\ (g\ x)) \in bfun$
  ⟨*proof*⟩

lemma *apply-bfun-bfun*[*simp*]: *apply-bfun* $f \in bfun$
  ⟨*proof*⟩

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

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

1.4 Order Instance

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

instance *real* :: *ordered-real-normed-vector*
  ⟨*proof*⟩

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

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

instance
⟨*proof*⟩
end

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

lemma *less-eq-bfunD*[*dest*]: $f \le g \Longrightarrow (\bigwedge x.\ apply\text{-}bfun\ f\ x \le apply\text{-}bfun\ g\ x)$
⟨*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*]: $apply\text{-}bfun\ (1 :: \text{-} \Rightarrow_b real)\ x = 1$
⟨*proof*⟩

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

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

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

lemma *bounded-apply-bfun'*:
assumes $bounded\ ((F :: 'c \Rightarrow 'd \Rightarrow_b 'b::real\text{-}normed\text{-}vector)\ `\ S)$
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*]: $(\$)\ x \in bfun$
⟨*proof*⟩

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

lemma *bounded-linear-bfun-nth*: *bounded-linear f* $\Longrightarrow$ *bounded-linear* (λv. *bfun.Bfun* (($) ($f$ v)))
⟨*proof*⟩

lemma *norm-vec-le-norm-bfun*:
norm (*vec-lambda* (*apply-bfun* (x :: $'d$::*finite* $\Rightarrow_b$ *real*))) $\leq$ *norm x* $*$ *card* (*UNIV* :: $'d$ *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*]:
id-blinfun o_L $f = f$
f o_L *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
blinfunpow $\equiv$ *compow* :: *nat* $\Rightarrow$ ($'a$::*real-normed-vector* $\Rightarrow_L$ $'a$) $\Rightarrow$ ($'a$ $\Rightarrow_L$ $'a$)
begin

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

end

lemma *bounded-pow-blinfun*[*intro*]:
assumes *bounded* (*range* (F::*nat* $\Rightarrow$ $'a$::*real-normed-vector* $\Rightarrow_L$ $'a$))
shows *bounded* (*range* (λt. (F t)$\widehat{\ }\widehat{\ }$(*Suc* n)))
⟨*proof*⟩

lemma *blincomp-scaleR-right*: ($a *_R$ (F :: $'a$ :: *real-normed-vector* $\Rightarrow_L$ $'a$)) $\widehat{\ }\widehat{\ }$ t = a $\widehat{\ }$ t $*_R$ F $\widehat{\ }\widehat{\ }$ t
⟨*proof*⟩
lemma *summable-inv-Q*:
fixes Q :: $'a$ :: *banach* $\Rightarrow_L$ $'a$
assumes *onorm-le*: *norm* (*id-blinfun* $-$ Q) $<$ *1*
shows *summable* (λn. (*id-blinfun* $-$ Q)$\widehat{\ }\widehat{\ }$$n$)
⟨*proof*⟩

lemma *blinfunpow-assoc*: (F::$'a$::*real-normed-vector* $\Rightarrow_L$ $'a$) $\widehat{\ }\widehat{\ }$ (*Suc* n) =(F $\widehat{\ }\widehat{\ }$$n$) o_L F
⟨*proof*⟩

lemma *norm-blinfunpow-le*: *norm* ((f :: $'b$ :: *real-normed-vector* $\Rightarrow_L$ $'b$) $\widehat{\ }\widehat{\ }$ n) $\leq$ *norm* f $\widehat{\ }$ n
⟨*proof*⟩

lemma *blinfunpow-nonneg*:
assumes $\bigwedge v$. *0* $\leq$ v $\Longrightarrow$ *0* $\leq$ *blinfun-apply* (f :: ($'b$ :: {*ord*, *real-normed-vector*} $\Rightarrow_L$ $'b$)) v
shows *0* $\leq$ v $\Longrightarrow$ *0* $\leq$ ($f$$\widehat{\ }\widehat{\ }$$n$) v
⟨*proof*⟩

lemma *blinfunpow-mono*:
assumes $\bigwedge u$ v. u $\leq$ v $\Longrightarrow$ (f :: $'b$ :: {*ord*, *real-normed-vector*} $\Rightarrow_L$ $'b$) u $\leq$ f v
shows u $\leq$ v $\Longrightarrow$ ($f$$\widehat{\ }\widehat{\ }$$n$) u $\leq$ ($f$$\widehat{\ }\widehat{\ }$$n$) v
⟨*proof*⟩

lemma *banach-blinfun*:
fixes C :: $'b$ :: {*real-normed-vector*, *complete-space*} $\Rightarrow_L$ $'b$
assumes *norm* C $<$ *1*
shows $\exists!v$. C v = v $\bigwedge v$. (λn. (C $\widehat{\ }\widehat{\ }$ n) v) $\longrightarrow$ (*THE* v. C v = v)
⟨*proof*⟩

2.3 Geometric Sum

lemma *inv-one-sub-Q*:
fixes Q :: $'a$ :: *banach* $\Rightarrow_L$ $'a$
assumes *onorm-le*: *norm* (*id-blinfun* $-$ Q) $<$ *1*
shows (Q o_L ($\sum i$. (*id-blinfun* $-$ Q)$\widehat{\ }\widehat{\ }$$i$)) = *id-blinfun*
and ($\sum i$. (*id-blinfun* $-$ Q)$\widehat{\ }\widehat{\ }$$i$) o_L Q = *id-blinfun*

$\langle proof \rangle$

lemma *inv-norm-le*:
fixes $Q :: 'a :: banach \Rightarrow_L 'a$
assumes $norm\ Q\ < 1$
shows $(id\text{-}blinfun - Q)\ o_L\ (\sum i.\ Q\widehat{\ }\widehat{\ }i) = id\text{-}blinfun$
$(\sum i.\ Q\widehat{\ }\widehat{\ }i)\ o_L\ (id\text{-}blinfun - Q) = id\text{-}blinfun$
$\langle proof \rangle$

lemma *inv-norm-le'*:
fixes $Q :: 'a :: banach \Rightarrow_L 'a$
assumes $norm\ Q\ < 1$
shows $(id\text{-}blinfun - Q)\ ((\sum i.\ Q\widehat{\ }\widehat{\ }i)\ x) = x$
$(\sum i.\ Q\widehat{\ }\widehat{\ }i)\ ((id\text{-}blinfun - Q)\ x) = x$
$\langle proof \rangle$

2.4 Inverses

definition $is\text{-}inverse_L\ X\ Y \longleftrightarrow X\ o_L\ Y = id\text{-}blinfun \wedge Y\ o_L\ X = id\text{-}blinfun$

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

lemma $is\text{-}inverse_L\text{-}I[intro]$:
assumes $X\ o_L\ Y = id\text{-}blinfun\ Y\ o_L\ X = id\text{-}blinfun$
shows $is\text{-}inverse_L\ X\ Y$
$\langle proof \rangle$

lemma $is\text{-}inverse_L\text{-}D[dest]$:
assumes $is\text{-}inverse_L\ X\ Y$
shows $X\ o_L\ Y = id\text{-}blinfun\ Y\ o_L\ X = id\text{-}blinfun$
$\langle proof \rangle$

lemma $invertible_L\text{-}D[dest]$:
assumes $invertible_L\ f$
obtains g **where** $f\ o_L\ g = id\text{-}blinfun\ g\ o_L\ f = id\text{-}blinfun$
$\langle proof \rangle$

lemma $invertible_L\text{-}I[intro]$:
assumes $f\ o_L\ g = id\text{-}blinfun\ g\ o_L\ f = id\text{-}blinfun$
shows $invertible_L\ f$
$\langle proof \rangle$

lemma $is\text{-}inverse_L\text{-}comm$: $is\text{-}inverse_L\ X\ Y \longleftrightarrow is\text{-}inverse_L\ Y\ X$
$\langle proof \rangle$

lemma $is\text{-}inverse_L\text{-}unique$: $is\text{-}inverse_L\ f\ g \Longrightarrow is\text{-}inverse_L\ f\ h \Longrightarrow g = h$
$\langle proof \rangle$

lemma *is-inverse$_L$-ex1*: *is-inverse$_L$ f g* $\Longrightarrow \exists!h.$ *is-inverse$_L$ f h*
⟨*proof*⟩

lemma *is-inverse$_L$-ex1'*: $\exists x.$ *is-inverse$_L$ f x* $\Longrightarrow \exists!x.$ *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* $\Longrightarrow$ (*inv$_L$ X*) *o$_L$ X* = *id-blinfun*
⟨*proof*⟩

lemma *inv-app2*[*simp*]: *invertible$_L$ X* $\Longrightarrow$ *X o$_L$* (*inv$_L$ X*) = *id-blinfun*
⟨*proof*⟩

lemma *inv-app1'*[*simp*]: *invertible$_L$ X* $\Longrightarrow$ *inv$_L$ X* (*X v*) = *v*
⟨*proof*⟩

lemma *inv-app2'*[*simp*]: *invertible$_L$ X* $\Longrightarrow$ *X* (*inv$_L$ X v*) = *v*
⟨*proof*⟩

lemma *inv$_L$-inv$_L$*[*simp*]: *invertible$_L$ X* $\Longrightarrow$ *inv$_L$* (*inv$_L$ X*) = *X*
⟨*proof*⟩

lemma *inv$_L$-cancel-iff*:
assumes *invertible$_L$ f*
shows *f x* = *y* $\longleftrightarrow$ *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* ^^ *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*$_L$ $f \Longrightarrow$ *invertible*$_L$ $g \Longrightarrow$ *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*) $\Longrightarrow$ *bounded* (*f* ‘ *X*′)
  ⟨*proof*⟩

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

lift-definition *bfun-pos* :: (′*d* $\Rightarrow_b$ *real*) ⇒ (′*d* $\Rightarrow_b$ *real*) **is** λ*f i*. *if f i* < *0 then* −*f i else f i*
  ⟨*proof*⟩

lemma *bfun-pos-zero*[*simp*]: *bfun-pos f* = *0* ⟷ *f* = *0*
  ⟨*proof*⟩

lift-definition *bfun-nonneg* :: (′*d* $\Rightarrow_b$ *real*) ⇒ (′*d* $\Rightarrow_b$ *real*) **is** λ*f 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*: $0 \leq$ *bfun-nonneg f*
  ⟨*proof*⟩

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

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

lemma *norm-bfun-pos*[*simp*]: *norm* (*bfun-pos f*) = *norm f*
⟨*proof*⟩

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

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

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

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

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

2.6 Miscellaneous

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

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

lemma *blinfun-apply-mono*: *nonneg-blinfun* $X \Longrightarrow 0 \leq v \Longrightarrow$ *blinfun-le* $X\ Y \Longrightarrow X\ v \leq Y\ v$
⟨*proof*⟩

lemma *nonneg-blinfun-scaleR*[*intro*]: *nonneg-blinfun* $B \Longrightarrow 0 \leq c \Longrightarrow$ *nonneg-blinfun* $(c *_R B)$
⟨*proof*⟩

lemma *nonneg-blinfun-compose*[*intro*]: *nonneg-blinfun* $B \Longrightarrow$ *nonneg-blinfun* $C \Longrightarrow$ *nonneg-blinfun* $(C\ o_L\ B)$
⟨*proof*⟩

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

lemma *bounded-subset*: $Y \subseteq X \Longrightarrow$ *bounded* $(f\ `\ X) \Longrightarrow$ *bounded* $(f\ `\ Y)$
⟨*proof*⟩

lemma *bounded-subset-range*: *bounded* (*range* f) $\Longrightarrow$ *bounded* $(f\ `\ Y)$
⟨*proof*⟩

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

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

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

lemma *bfun-if-zero-le*: $0 \leq v \Longrightarrow$ *bfun-if* $b\ 0\ v \leq v$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *bfun-if-scaleR*: $c *_R$ *bfun-if b v1 v2* = *bfun-if b* ($c *_R$ *v1*) ($c *_R$ *v2*)
⟨*proof*⟩

lemma *summable-blinfun-apply*:
assumes *summable* (f :: *nat* ⇒ *'a::real-normed-vector* $\Rightarrow_L$ *'a*)
shows *summable* ($\lambda n.\ f\ n\ v$)
⟨*proof*⟩

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

3 Auxiliary Lemmas

3.1 Summability

lemma *summable-powser-const*:
fixes c :: *real*
assumes $|c| < 1$
shows *summable* ($\lambda n.\ c \widehat{\ } n * x$)
⟨*proof*⟩

3.2 Infinite sums

lemma *suminf-split-head'*:
summable (f :: *nat* ⇒ *'x* :: *real-normed-vector*) $\Longrightarrow$ *suminf f* = $f\ 0$ + ($\sum n.\ f$ (*Suc n*))
⟨*proof*⟩

lemma *sum-disc-lim*:
assumes $|c$ :: *real*$| < 1$
shows ($\sum x.\ c \widehat{\ } x * B$) = B /($1-c$)
⟨*proof*⟩

3.3 Bounded Functions

lemma *suminf-apply-bfun*:
fixes f :: *nat* ⇒ *'c* $\Rightarrow_b$ *real*

assumes *summable f*
shows $(\sum i.\ f\ i)\ x = (\sum i.\ f\ i\ x)$
⟨*proof*⟩

lemma *sum-apply-bfun*:
fixes f :: *nat* ⇒ *'c* $\Rightarrow_b$ *real*
shows $(\sum i{<}n.\ f\ i)\ x = (\sum i{<}n.\ \textit{apply-bfun}\ (f\ i)\ x)$
⟨*proof*⟩

3.4 Push-Forward of a Bounded Function

lemma *integrable-bfun-prob-space* [*simp*]:
integrable (*measure-pmf P*) (λt. *apply-bfun f* (*F t*) :: *real*)
⟨*proof*⟩

lift-definition *push-exp* :: (*'b* ⇒ *'c pmf*) ⇒ (*'c* $\Rightarrow_b$ *real*) ⇒ (*'b* $\Rightarrow_b$ *real*) **is**
$\lambda c\ f\ s$. *measure-pmf.expectation* (*c s*) *f*
⟨*proof*⟩

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

lemma *norm-push-exp-le-norm*: *norm* (*push-exp d x*) ≤ *norm x*
⟨*proof*⟩

lemma *push-exp-bounded-linear* [*simp*]: *bounded-linear* (*push-exp d*)
⟨*proof*⟩

lemma *onorm-push-exp* [*simp*]: *onorm* (*push-exp d*) = *1*
⟨*proof*⟩

lemma *push-exp-return*[*simp*]: *push-exp return-pmf* = *id*
⟨*proof*⟩

3.5 Boundedness

lemma *bounded-abs*[*intro*]:
bounded (*X'* :: *real set*) ⟹ *bounded* (*abs* ‘ *X'*)
⟨*proof*⟩

lemma *bounded-abs-range*[*intro*]:
bounded (*range f* :: *real set*) ⟹ *bounded* (*range* (λx. *abs* (*f x*)))
⟨*proof*⟩

3.6 Probability Theory

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

$\langle proof \rangle$

lemma *lemma-4-3-1'*:
assumes *set-pmf p* $\subseteq$ *W*
and *bounded* $((w :: {'c} \Rightarrow real)$ ‘ $W)$
and $W \neq \{\}$
and *measure-pmf.expectation p w* $= (\bigsqcup p \in \{p.$ *set-pmf p* $\subseteq W\}.$ *measure-pmf.expectation p w*)
shows $\exists\, x \in W.$ *measure-pmf.expectation p w* $= w\ x$
$\langle proof \rangle$

lemma *lemma-4-3-1*:
assumes *set-pmf p* $\subseteq$ *W integrable* (*measure-pmf p*) *w bounded* $((w :: {'c} \Rightarrow real)$ ‘ $W)$
shows *measure-pmf.expectation p w* $\leq \bigsqcup (w$ ‘ $W)$
$\langle proof \rangle$

lemma *bounded-integrable*:
assumes *bounded* (*range v*) $v \in$ *borel-measurable* (*measure-pmf p*)
shows *integrable* (*measure-pmf p*) $(v :: {'c} \Rightarrow real)$
$\langle proof \rangle$

3.7 Argmax

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

lemma *finite-arg-max-le*:
assumes *finite* $(X :: {'c}\ set)$ $X \neq \{\}$
shows $s \in X \Longrightarrow (f :: {'c} \Rightarrow real)\ s \leq f$ (*arg-max-on* $(f :: {'c} \Rightarrow real)$ $X)$
$\langle proof \rangle$

lemma *arg-max-on-in*:
assumes *finite* $(X :: {'c}\ set)$ $X \neq \{\}$
shows (*arg-max-on* $(f :: {'c} \Rightarrow real)$ $X) \in X$
$\langle proof \rangle$

lemma *finite-arg-max-eq-Max*:
assumes *finite* $(X :: {'c}\ set)$ $X \neq \{\}$
shows $(f :: {'c} \Rightarrow real)$ (*arg-max-on f X*) $=$ *Max* $(f$ ‘ $X)$
$\langle proof \rangle$

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

definition $\textit{has-max}\ X \equiv \exists\, x \in X.\ \forall\, x' \in X.\ x' \leq x$
definition $\textit{has-arg-max}\ f\ X \equiv \exists\, x.\ \textit{is-arg-max}\ f\ (\lambda x.\ x \in X)\ x$

lemma $\textit{has-max}\ ((f :: 'b \Rightarrow \textit{real})\ `\ X) \longleftrightarrow \textit{has-arg-max}\ f\ X$
⟨*proof*⟩

lemma *has-arg-max-is-arg-max*: $\textit{has-arg-max}\ f\ X \Longrightarrow \textit{is-arg-max}\ f\ (\lambda x.\ x \in X)\ (\textit{arg-max}\ f\ (\lambda x.\ x \in X))$
⟨*proof*⟩

lemma *has-arg-max-arg-max*: $\textit{has-arg-max}\ f\ X \Longrightarrow (\textit{arg-max}\ f\ (\lambda x.\ x \in X)) \in X$
⟨*proof*⟩

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

lemma *app-arg-max-eq-SUP*: $\textit{has-arg-max}\ (f :: 'b \Rightarrow \textit{real})\ X \Longrightarrow f\ (\textit{arg-max-on}\ f\ X) = \bigsqcup (f\ `\ X)$
⟨*proof*⟩

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

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

lemma *is-arg-max-linorderD*[*dest*]: **fixes** $f :: 'c \Rightarrow 'b :: \textit{linorder}$
assumes $\textit{is-arg-max}\ f\ P\ x$
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 $\textit{is-arg-max}\ f\ P\ x \longleftrightarrow \textit{is-arg-max}\ g\ P\ x$
⟨*proof*⟩

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

lemma *is-arg-max-congI*:

assumes *is-arg-max f P x* $\bigwedge x.\ P\ x \Longrightarrow f\ x = g\ x$
shows *is-arg-max g P x*
$\langle proof \rangle$

3.8 Contraction Mappings

definition *is-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-contraction C*
shows $\exists!v.\ C\ v = v \bigwedge v.\ (\lambda n.\ (C\ \widehat{}\widehat{}\ n)\ v) \longrightarrow (THE\ v.\ C\ v = v)$
$\langle proof \rangle$

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

3.9 Limits

lemma *tendsto-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-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-SUP-diff'*:
assumes *ne*: $X \neq \{\}$
and *bdd*: *bounded* $(B\ `\ X)$ *bounded* $(A'\ `\ X)$
and *sup-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-SUP-diff*:
fixes $A' :: 'a \Rightarrow real$

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

lemma *bounded-SUP-mul*[*simp*]:
$X \neq \{\} \Longrightarrow 0 \leq l \Longrightarrow$ *bounded* $(f$ ‘ $X) \Longrightarrow (\bigsqcup x \in X.\ (l :: real) * f\ x) = (l * (\bigsqcup x \in X.\ f\ x))$
⟨*proof*⟩

lemma *abs-cSUP-le*[*intro*]:
$X \neq \{\} \Longrightarrow$ *bounded* $(F$ ‘ $X) \Longrightarrow |\bigsqcup x \in X.\ (F\ x) :: real| \leq (\bigsqcup x \in X.\ |F\ x|)$
⟨*proof*⟩

4 Least argmax

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

lemma *least-arg-max-prop*: $\exists\, x::'a::wellorder.\ P\ x \Longrightarrow$ *finite* $\{x.\ P\ x\} \Longrightarrow P$ (*least-arg-max* $(f :: \text{-} \Rightarrow real)\ P)$
⟨*proof*⟩

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

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

lemma *apply-arg-max-eq-max*: *finite* $\{x\ .\ P\ x\} \Longrightarrow$ *is-arg-max* $(f :: \text{-} \Rightarrow \text{-} :: linorder)\ P\ x \Longrightarrow f\ x = Max\ (f$ ‘ $\{x.\ P\ x\})$
⟨*proof*⟩

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

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

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

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

assumes $\bigwedge x.$ *is-arg-max f P x* = *is-arg-max g P x*
shows *least-arg-max f P* = *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 $\bigwedge i\ h.\ h \in space\ (PiM\ \{0..<i\}\ N) \Longrightarrow P\ i\ h = P'\ i\ h$
assumes *h*: *Ionescu-Tulcea P N Ionescu-Tulcea P′ N*
shows *Ionescu-Tulcea.C P N 0 n* ($\lambda x.$ *undefined*) = *Ionescu-Tulcea.C P′ N 0 n* ($\lambda x.$ *undefined*)
⟨*proof*⟩

lemma *Ionescu-Tulcea-CI-eq*:
assumes $\bigwedge i\ h.\ h \in space\ (PiM\ \{0..<i\}\ N) \Longrightarrow P\ i\ h = P'\ i\ h$
assumes *h*: *Ionescu-Tulcea P N Ionescu-Tulcea P′ N*
shows *Ionescu-Tulcea.CI P N* = *Ionescu-Tulcea.CI P′ N*
⟨*proof*⟩

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

lemma *distr-cong-simp*:
$M = K \Longrightarrow sets\ N = sets\ L \Longrightarrow (\bigwedge x.\ x \in space\ M$ =simp=> $f\ x = g\ x) \Longrightarrow distr\ M\ N\ f = distr\ K\ L\ g$
⟨*proof*⟩

5.1 Definition and Basic Properties

locale *discrete-MDP* =

fixes *Ms* :: *'s measure*
  and *Ma* :: *'a measure*
  and *A* :: *'s* ⇒ *'a set*
  and *K* :: *'s* × *'a* ⇒ *'s measure*

assumes *A-s*: $\bigwedge s.\ A\ s \in sets\ Ma$

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

assumes *ex-pol*: $\exists\,\delta \in Ms \rightarrow_M Ma.\ \forall\, s.\ \delta\ s \in A\ s$

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

lemma *space-prodI*[*intro*]: $x \in space\ A' \Longrightarrow y \in space\ B \Longrightarrow (x,y) \in space\ (A' \bigotimes_M B)$
  ⟨*proof*⟩

abbreviation $M \equiv Ms \bigotimes_M Ma$
abbreviation *Ma-A* $s \equiv$ *restrict-space Ma* $(A\ s)$

lemma *space-ma*[*intro*]: $s \in space\ Ms \Longrightarrow a \in space\ Ma \Longrightarrow (s,a) \in space\ M$
  ⟨*proof*⟩

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

lemma *A-subs-Ma*: $A\ s \subseteq space\ Ma$
  ⟨*proof*⟩

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

lemma *K-restrict* [*measurable*]: $K \in (Ms \bigotimes_M Ma\text{-}A\ s) \rightarrow_M prob\text{-}algebra\ Ms$
  ⟨*proof*⟩

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

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

lemma *space-K*[*simp*]: $sa \in space\ M \Longrightarrow space\ (K\ sa) = space\ Ms$
  ⟨*proof*⟩

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

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

lemma *sets-K*: $sa \in space\ M \Longrightarrow sets\ (K\ sa) = sets\ Ms$
⟨*proof*⟩

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

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

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

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

5.2 Policies

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

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

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

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

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

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

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

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

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

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

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

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

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

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

context
fixes *p* **assumes** *p*[*simp*]: *is-policy p*
begin

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

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

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

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

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

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

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

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

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

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

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

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

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

5.3 Successor Policy

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

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

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

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

lemma *Suc-policy-measurable-step*[*measurable*]:
assumes *is-policy* p
shows $(\lambda x.\ Suc\text{-}policy\ p\ (fst\ (fst\ x))\ n\ (snd\ (fst\ x),\ snd\ x)) \in$
$(M \bigotimes_M Pi_M\ \{0..<n\}\ (\lambda\text{-}.\ M)) \bigotimes_M Ms \rightarrow_M prob\text{-}algebra\ Ma$
⟨*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 $K' :: ('s, 'a)\ pol \Rightarrow 's\ measure \Rightarrow nat \Rightarrow (nat \Rightarrow ('s \times 'a)) \Rightarrow ('s \times 'a)\ measure$
where
$K'\ p\ s0\ n\ \omega = do\ \{$
$s \leftarrow case\text{-}nat\ s0\ (K \circ \omega)\ n;$
$a \leftarrow p\ n\ (\omega,\ s);$
$return\ M\ (s,\ a)$
$\}$

lemma *prob-space-K'*:
assumes p: *is-policy* p **and** x: $x0 \in space\ (prob\text{-}algebra\ Ms)$ **and** h: $h \in space\ (H\ n)$
shows $prob\text{-}space\ (K'\ p\ x0\ n\ h)$
⟨*proof*⟩

lemma *measurable-K'*[*measurable*]:
assumes p: *is-policy* p **and** x: $x \in space\ (prob\text{-}algebra\ Ms)$
shows $K'\ p\ x\ i \in H\ i \rightarrow_M prob\text{-}algebra\ M$
⟨*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'*:
$K0\ p\ s0 = do\ \{$
$s \leftarrow s0;$
$a \leftarrow p0\ p\ s;$
$return\ M\ (s,\ a)\}$
⟨*proof*⟩

lemma *K0-prob*[*measurable*]: $is\text{-}policy\ p \Longrightarrow K0\ p \in prob\text{-}algebra\ Ms \rightarrow_M prob\text{-}algebra\ M$
⟨*proof*⟩

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

lemma *space-K0*[*simp*]: $is\text{-}policy\ p \Longrightarrow s \in space\ (prob\text{-}algebra\ Ms) \Longrightarrow space\ (K0\ p\ s) = space\ M$
⟨*proof*⟩

lemma *sets-K0*[*measurable-cong*]:
  assumes *is-policy p s* $\in$ *space* (*prob-algebra Ms*)
  shows *sets* (*K0 p s*) = *sets M*
  ⟨*proof*⟩

lemma *K0-return-eq-p0*:
  assumes *is-policy p s* $\in$ *space Ms*
  shows *K0 p* (*return Ms s*) = *p0 p s* $\gg=$ (λa. *return M* (*s*,*a*))
  ⟨*proof*⟩

lemma *M-ne-policy*[*intro*]: *is-policy p* $\Longrightarrow$ *s* $\in$ *space* (*prob-algebra Ms*) $\Longrightarrow$ *space M* $\neq$ {}
  ⟨*proof*⟩

5.6 Sequence Space of the MDP

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

lemma *IT-K'*: *is-policy p* $\Longrightarrow$ *x* $\in$ *space* (*prob-algebra Ms*) $\Longrightarrow$ *Ionescu-Tulcea* (K' *p x*) (λ-. *M*)
  ⟨*proof*⟩

definition *lim-sequence* :: ('*s*, '*a*) *pol* $\Rightarrow$ '*s measure* $\Rightarrow$ (*nat* $\Rightarrow$ ('*s* $\times$ '*a*)) *measure*
  where
    lim-sequence p x = *projective-family.lim UNIV* (*Ionescu-Tulcea.CI* (K' *p x*) (λ-. *M*)) (λ-. *M*)

lemma
  assumes *x*: *x* $\in$ *space* (*prob-algebra Ms*) **and** *p*: *is-policy p*
  shows *space-lim-sequence*: *space* (*lim-sequence p x*) = *space* (Π_M *i*$\in$*UNIV*. *M*)
    and *sets-lim-sequence*[*measurable-cong*]: *sets* (*lim-sequence p x*) = *sets* (Π_M *i*$\in$*UNIV*. *M*)
    and *emeasure-lim-sequence-emb*: $\bigwedge$*J X*. *finite J* $\Longrightarrow$ *X* $\in$ *sets* (Π_M *j*$\in$*J*. *M*) $\Longrightarrow$
      emeasure (*lim-sequence p x*) (*prod-emb UNIV* (λ-. *M*) *J X*) =
      emeasure (*Ionescu-Tulcea.CI* (K' *p x*) (λ-. *M*) *J*) *X*
    and *emeasure-lim-sequence-emb-I0o*: $\bigwedge$*n X*. *X* $\in$ *sets* (Π_M *i* $\in$ {*0..*<*n*}. *M*) $\Longrightarrow$
      emeasure (*lim-sequence p x*) (*prod-emb UNIV* (λ-. *M*) {*0..*<*n*} *X*) =
      emeasure (*Ionescu-Tulcea.C* (K' *p x*) (λ-. *M*) *0 n* (λx. *undefined*)) *X*
⟨*proof*⟩

lemma *lim-sequence-prob-space*:
  assumes *is-policy p s* $\in$ *space* (*prob-algebra Ms*)
  shows *prob-space* (*lim-sequence p s*)

⟨*proof*⟩

5.7 Measurablility of the Sequence Space

lemma *lim-sequence*[*measurable*]:
assumes *p*: *is-policy p*
shows *lim-sequence p* $\in$ *prob-algebra Ms* $\to_M$ *prob-algebra* (Π_M $i \in UNIV$. *M*)
⟨*proof*⟩

lemma *lim-sequence-aux*[*measurable*]:
assumes *p*: *is-policy p*
assumes *f* : $\bigwedge x$. $x \in$ *space M* $\Longrightarrow$ *is-policy* (*f x*)
assumes *f'*: $\bigwedge n$. (λx. *f* (*fst* (*fst x*)) *n* (*snd* (*fst x*), *snd x*)) $\in$
(*M* $\bigotimes_M$ *Pi*$_M$ {*0..<n*} (λ-. *M*)) $\bigotimes_M$ *Ms* $\to_M$ *prob-algebra Ma*
assumes *gm*: *g* $\in$ *M* $\to_M$ *prob-algebra Ms*
shows (λx. *lim-sequence* (*f x*) (*g x*)) $\in$ *M* $\to_M$ *prob-algebra* (*Pi*$_M$ *UNIV* (λ-. *M*))
⟨*proof*⟩

lemma *lim-sequence-Suc-return*[*measurable*]:
assumes *p*: *is-policy p*
assumes *s*: *s* $\in$ *space Ms*
shows (λx. *lim-sequence* (*Suc-policy p* (*s*, *snd x*)) (*return Ms* (*fst x*))) $\in$
M $\to_M$ *prob-algebra* (*Pi*$_M$ *UNIV* (λ-. *M*))
⟨*proof*⟩

lemma *lim-sequence-Suc-K*[*measurable*]:
assumes *is-policy p*
shows (λx. *lim-sequence* (*Suc-policy p x*) (*K x*)) $\in$ *M* $\to_M$ *prob-algebra* (*Pi*$_M$ *UNIV* (λ-. *M*))
⟨*proof*⟩

5.8 Iteration Rule

lemma *step-C*:
assumes *x*: *x* $\in$ *space* (*prob-algebra Ms*) **and** *p*: *is-policy p*
shows *Ionescu-Tulcea.C* (*K' p x*) (λ-. *M*) *0 1* (λ-. *undefined*) $\gg=$
Ionescu-Tulcea.C (*K' p x*) (λ-. *M*) *1 n* =
K0 p x $\gg=$ (λa. *Ionescu-Tulcea.C* (*K' p x*) (λ-. *M*) *1 n* (*case-nat a* (λ-. *undefined*)))
⟨*proof*⟩

lemma *lim-sequence-eq*:
assumes *x*: *x* $\in$ *space* (*prob-algebra Ms*) **assumes** *p*: *is-policy p*
shows *lim-sequence p x* =
K0 p x $\gg=$ (λy. *distr* (*lim-sequence* (*Suc-policy p y*) (*K y*)) (Π_M -$\in UNIV$. *M*) (*case-nat y*))
(**is** - = *?B p x*)

⟨*proof*⟩

5.9 Stream Space of the MDP

definition *lim-stream* :: $('s, 'a)$ *pol* ⇒ $'s$ *measure* ⇒ $('s \times 'a)$ *stream measure*
where
lim-stream p x = *distr* (*lim-sequence p x*) (*stream-space M*) *to-stream*

lemma *space-lim-stream*: *space* (*lim-stream p x*) = *streams* (*space M*)
⟨*proof*⟩

lemma *sets-lim-stream*[*measurable-cong*]: *sets* (*lim-stream p x*) = *sets* (*stream-space M*)
⟨*proof*⟩

lemma *lim-stream*[*measurable*]:
assumes *is-policy p*
shows *lim-stream p* ∈ *prob-algebra Ms* $\to_M$ *prob-algebra* (*stream-space M*)
⟨*proof*⟩

lemma *lim-stream-Suc*[*measurable*]:
assumes *p*: *is-policy p*
shows (λa. *lim-stream* (*Suc-policy p a*) (*K a*)) ∈ *M* $\to_M$ *prob-algebra* (*stream-space M*)
⟨*proof*⟩

lemma *space-stream-space-M-ne*: *x* ∈ *space M* ⟹ *space* (*stream-space M*) ≠ {}
⟨*proof*⟩

lemma *prob-space-lim-stream*[*intro*]:
assumes *is-policy p x* ∈ *space* (*prob-algebra Ms*)
shows *prob-space* (*lim-stream p x*)
⟨*proof*⟩

lemma *prob-space-step*:
assumes *is-policy p x* ∈ *space M*
shows *prob-space* (*lim-stream* (*Suc-policy p x*) (*K x*))
⟨*proof*⟩

lemma *lim-stream-eq*:
assumes *p*: *is-policy p*
assumes *x*: *x* ∈ *space* (*prob-algebra Ms*)
shows *lim-stream p x* = *do* {
y ← *K0 p x*;
ω ← *lim-stream* (*Suc-policy p y*) (*K y*);

return (*stream-space M*) (*y ## ω*)
}
⟨*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* ⇒ (*'b* :: {*banach*, *second-countable-topology*,*real-normed-vector*})
assumes *int-f*: *integrable* (*stream-space M*) *f*
assumes [*measurable*]: *f* ∈ *borel-measurable* (*stream-space M*)
shows (∫ *X*. *f X* ∂*stream-space M*) = (∫ *x*. (∫ *X*. *f* (*x ## X*) ∂*stream-space M*) ∂*M*)
⟨*proof*⟩

lemma *prefix-cons*:
Omega-Words-Fun.prefix (*Suc n*) *seq* = *seq 0#Omega-Words-Fun.prefix n* (λ*n*. *seq* (*Suc n*))
⟨*proof*⟩

lemma *restrict-Suc*: *restrict y* {*0..<Suc i*} (*Suc n*) = (*restrict* (λ*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 ∈ Pi_M {*0..<i*}
(λ-. *count-space* (*UNIV* :: (*'s* ::*countable* × *'a*::*countable*) *set*)) $\to_M$ *count-space UNIV*
⟨*proof*⟩

no-notation *Omega-Words-Fun.build* (**infixr** ‹##› *65*)

locale *discrete-MDP* =
fixes *A* :: *'s*::*countable* ⇒ *'a*::*countable set* — enabled actions

and K :: $'s \times 'a \Rightarrow 's\ pmf$ — 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 $('c, 'd)\ dec = 'c \Rightarrow 'd\ pmf$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Type synonym for policies.

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

A policy assigns a decision rule to each observed past.

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

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

Deterministic policies

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

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

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

Markovian policies

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

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

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

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

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

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

abbreviation $\Pi_{MD} \equiv \{p.\ \forall\, n{::}\textit{nat}.\ p\ n \in D_D\}$
abbreviation $\Pi_{MR} \equiv \{p.\ \forall\, n.\ p\ n \in D_R\}$

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

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

⟨*proof*⟩

lemma Π_{MD}-*to-MR*[*intro*]: $p \in \Pi_{MD} \Longrightarrow (\lambda n.\ \textit{mk-dec-det}\ (p\ n)) \in \Pi_{MR}$
⟨*proof*⟩

lemma *p-n-π-MD*[*intro*]: $p \in \Pi_{MD} \Longrightarrow p\ n \in D_D$
⟨*proof*⟩

lemma *p-n-π-MR*[*intro*]: $p \in \Pi_{MR} \Longrightarrow p\ n \in D_R$
⟨*proof*⟩

lemma Π_{MD}-*ne*[*simp*]: $\Pi_{MD} \neq \{\}$
⟨*proof*⟩

lemma Π_{MR}-*ne*[*simp*]: $\Pi_{MR} \neq \{\}$
⟨*proof*⟩

lemma *policies-ne*[*simp*, *intro*]: $\Pi_{HR} \neq \{\}$
⟨*proof*⟩

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* $(\lambda\text{-}.\ d) =$ *is-dec* d
⟨*proof*⟩

lemma *is-stationary-const*[*intro*]: *is-dec* $d \Longrightarrow$ *is-stationary* $(\lambda\text{-}.\ d)$
⟨*proof*⟩

abbreviation *mk-stationary* $p \equiv$ *mk-markovian* $(\lambda\text{-}.\ p)$
abbreviation *mk-stationary-det* $d \equiv$ *mk-markovian* $(\lambda\text{-}.\ \textit{mk-dec-det}\ d)$

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 π-*Suc* :: $('s,\ 'a)\ pol \Rightarrow 's \times 'a \Rightarrow ('s,\ 'a)\ pol$
where
$\pi\text{-}Suc\ p\ sa\ h = p\ (sa\#h)$

lemma *is-policy-π-Suc* [*intro*]: *is-policy* $p \Longrightarrow$ *is-policy* $(\pi\text{-}Suc\ p\ sa)$
⟨*proof*⟩

lemma *Suc-mk-markovian*[*simp*]: π-*Suc* (*mk-markovian p*) x = *mk-markovian* ($\lambda n.\ p$ (*Suc n*))
⟨*proof*⟩

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* (‹K_0›)

lemma *K0-iff*: *K0 d S0* = *S0* $\gg\!=$ ($\lambda s.$ *map-pmf* ($\lambda a.\ (s,a)$) ($d\ s$))
⟨*proof*⟩

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

lemma *pmf-K0* [*simp*]: *pmf* (*K0 d S0*) (s,a) = *pmf S0 s* $*$ *pmf* ($d\ s$) a
⟨*proof*⟩

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

lemma *fst-K0*[*simp*]: *map-pmf fst* (*K0 p S0*) = *S0*
⟨*proof*⟩

abbreviation $S \equiv$ *stream-space* (*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 count-space UNIV A K*
⟨*proof*⟩

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

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* (λ*n* (*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* (λ*n* (*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 (λ*x* (*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 $s0'$
  Then the iteration repeats with the updated policy π-*Suc* $p\ (s,\ a)$.
  The result carries over from $[\![$*MDP-cont.is-policy ?p*; $?x \in$ *space* (*prob-algebra* (*count-space UNIV*))$]\!] \Longrightarrow$ *MDP-cont.lim-stream ?p ?x* $=$ *MDP-cont.K0 ?p ?x* $\gg\!=$ (λy. *MDP-cont.lim-stream* (*MDP-cont.Suc-policy ?p y*) (*measure-pmf* (*K y*)) $\gg\!=$ ($\lambda\omega$. *return* (*stream-space MDP-cont.M*) (y ## ω))).

lemma *T-eq*:
shows *T p S0* = *do* {
sa $\leftarrow$ *measure-pmf* (*K0* (*p* []) *S0*);
$\omega \leftarrow$ *T* (π-*Suc p sa*) (*K sa*);
return S (*sa* ## ω)
}
$\langle$*proof*$\rangle$

lemma *T-eq-distr*:
shows *T p S0* = *measure-pmf* (*K0* (*p* []) *S0*) $\gg\!=$ (λ*sa*. *distr* (*T* (π-*Suc p sa*) (*K sa*)) *S* ((##) *sa*))
$\langle$*proof*$\rangle$

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

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

lemma *nn-integral-T*:
assumes f: $f \in$ *borel-measurable S*
shows $(\int^+ t.\ f\ t\ \partial T\ p\ x) = (\int^+ sa.\ \int^+\ t'.\ f\ (sa\#\#t')\ \partial T\ (\pi\text{-}Suc\ p\ sa)\ (K\ sa)\ \partial K0\ (p\ [])\ x)$
$\langle$*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 $Pn :: ('s, 'a)\ pol \Rightarrow 's\ pmf \Rightarrow nat \Rightarrow ('s \times 'a)\ pmf$ **where**
$Pn\ p\ S0\ 0 = K0\ (p\ [])\ S0$
$|\ Pn\ p\ S0\ (Suc\ n) = K0\ (p\ [])\ S0 \gg= (\lambda sa.\ Pn\ (\pi\text{-}Suc\ p\ sa)\ (K\ sa)\ n)$

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

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

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

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

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

6.2.4 State Process

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

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

lemma *X0* [*simp*]: $Xn\ p\ S0\ 0 = S0$
⟨*proof*⟩

lemma *Xn-Suc*: $Xn\ p\ S0\ (Suc\ n) = Pn\ p\ S0\ n \gg= K$
⟨*proof*⟩

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

lemma *Xn-Suc'*: $Xn\ p\ S0\ (Suc\ n) = K0\ (p\ [])\ S0 \gg= (\lambda sa.\ Xn\ (\pi\text{-}Suc\ p\ sa)\ (K\ sa)\ n)$
⟨*proof*⟩

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

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

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-cond-X p S0 n x* = *map-pmf snd* (*cond-pmf* (*Pn p S0 n*) {(*s*,*a*). *s* = *x*})

lemma *prob-K0-X* [*simp*]: *measure-pmf.prob* (*K0 p S0*) {(*s*, *a*). *s* = *x*} = *pmf S0 x*
⟨*proof*⟩

lemma *prob-Pn-X*[*simp*]: *measure-pmf.prob* (*Pn p S0 n*) {(*s*, *a*). *s* = *x*} = *pmf* (*Xn p S0 n*) *x*
⟨*proof*⟩

lemma *pmf-Pn-pair*:
assumes *sa* ∈ *set-pmf* (*Pn p S0 n*)
shows *pmf* (*Pn p S0 n*) *sa* = *pmf* (*Y-cond-X p S0 n* (*fst sa*)) (*snd sa*) ∗ *pmf* (*Xn p S0 n*) (*fst sa*)
⟨*proof*⟩

lemma *pmf-Pn*:
assumes *x* ∈ *set-pmf* (*Xn p S0 n*)
shows *pmf* (*Pn p S0 n*) (*x*,*a*) = *pmf* (*Y-cond-X p S0 n x*) *a* ∗ *pmf* (*Xn p S0 n*) *x*
⟨*proof*⟩

lemma *pmf-Y-cond-X*:
assumes *x* ∈ *set-pmf* (*Xn p S0 n*)
shows *pmf* (*Y-cond-X p S0 n x*) *a* = *pmf* (*Pn p S0 n*) (*x*,*a*) / *pmf* (*Xn p S0 n*) *x*
⟨*proof*⟩

lemma *Y-cond-X-0*[*simp*]:
assumes *x* ∈ *set-pmf S0*
shows *Y-cond-X p S0 0 x* = *p* [] *x*
⟨*proof*⟩

lemma *Y-cond-X-markovian*[*simp*]:
assumes *h*: *x* ∈ *Xn* (*mk-markovian p*) *S0 n*
shows *Y-cond-X* (*mk-markovian p*) *S0 n x* = *p n x*
⟨*proof*⟩

lemma *Pn-eq-Xn-Y-cond*: *Pn p S0 n* = *Xn p S0 n* ≫= (λ*x*. *map-pmf* (λ*a*. (*x*, *a*)) (*Y-cond-X p S0 n x*))
⟨*proof*⟩

lemma *Pn-eq-Xn-Y-cond′*:
$Pn\ p\ S0\ n = Xn\ p\ S0\ n \gg= (\lambda s.\ Y\text{-}cond\text{-}X\ p\ S0\ n\ s \gg= (\lambda a.\ return\text{-}pmf\ (s,a)))$
⟨*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 $Yn\ p\ S0\ n = map\text{-}pmf\ snd\ (Pn\ p\ S0\ n)$

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

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

lemma *is-*Π_{MR}*-as-markovian*:
assumes *p*: *is-policy p*
shows $as\text{-}markovian\ p\ S0 \in \Pi_{MR}$
⟨*proof*⟩

lemma *is-policy-as-markovian*: $is\text{-}policy\ p \Longrightarrow is\text{-}policy\ (mk\text{-}markovian\ (as\text{-}markovian\ p\ S0))$
⟨*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 we the initial state is of the form *return-pmf s*.

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

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

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

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

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

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

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

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

lemma *measurable-ident-Suc*$'$[*measurable*]:
$(\lambda x.\ x) \in \mathcal{T}\ (\pi\text{-}Suc\ p\ sa)\ s' \to_M S$
$\langle proof \rangle$

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

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

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

definition $Pn'\ p\ s = Pn\ p\ (return\text{-}pmf\ s)$
definition $Xn'\ p\ s = Xn\ p\ (return\text{-}pmf\ s)$
definition $Yn'\ p\ s = Yn\ p\ (return\text{-}pmf\ s)$
definition $K0'\ d\ s \equiv map\text{-}pmf\ (\lambda a.\ (s,\ a))\ (d\ s)$

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

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

K-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*$'$[*simp*]: $Pn'\ p\ s\ 0 = K0'\ (p\ [])\ s$
$\langle proof \rangle$

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

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

S0 n
⟨*proof*⟩

lemma *PSuc′*: $Pn'\ p\ s\ (Suc\ n) = K0'\ (p\ [])\ s \gg= (\lambda sa.\ K\ sa \gg= (\lambda s'.\ Pn'\ (\pi\text{-}Suc\ p\ sa)\ s'\ n))$
⟨*proof*⟩

lemma *PSuc′-markovian*:
$Pn'\ (mk\text{-}markovian\ p)\ s\ (Suc\ n) = K\text{-}st\ (p\ 0)\ s \gg= (\lambda s'.\ Pn'\ (mk\text{-}markovian\ (p \circ Suc))\ s'\ n)$
⟨*proof*⟩

lemma *Xn′-Suc*: $Xn'\ p\ s\ (Suc\ n) = Pn'\ p\ s\ n \gg= K$
⟨*proof*⟩

lemma *Xn′-Pn′*: $Xn'\ p\ s\ n = map\text{-}pmf\ fst\ (Pn'\ p\ s\ n)$
⟨*proof*⟩

lemma *Suc-Xn′*: $Xn'\ p\ s\ (Suc\ n) = p\ []\ s \gg= (\lambda a.\ K\ (s,a) \gg= (\lambda s'.\ Xn'\ (\pi\text{-}Suc\ p\ (s,a))\ s'\ n))$
⟨*proof*⟩

lemma *Suc-Xn′-markovian*:
$Xn'\ (mk\text{-}markovian\ p)\ s\ (Suc\ n) = K\text{-}st\ (p\ 0)\ s \gg= (\lambda s'.\ Xn'\ (mk\text{-}markovian\ (\lambda n.\ p\ (Suc\ n)))\ s'\ n)$
⟨*proof*⟩

lemma *Xn′-split*: $Xn'\ (mk\text{-}markovian\ p)\ s\ (n + m) =$
$Xn'\ (mk\text{-}markovian\ p)\ s\ n \gg= (\lambda s.\ Xn'\ (mk\text{-}markovian\ (\lambda i.\ p\ (i + n)))\ s\ m)$
⟨*proof*⟩

lemma *Yn′-markovian*: $Yn'\ (mk\text{-}markovian\ p)\ s\ n = Xn'\ (mk\text{-}markovian\ p)\ s\ n \gg= p\ n$
⟨*proof*⟩

lemma *Pn′-markovian-eq-Xn′-bind*: $Pn'\ (mk\text{-}markovian\ p)\ s\ n = Xn'\ (mk\text{-}markovian\ p)\ s\ n \gg= K0'\ (p\ n)$
⟨*proof*⟩

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

end
end

theory *MDP-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**
$K :: 's :: countable \times 'a :: countable \Rightarrow 's\ pmf$ +
fixes
$r :: ('s \times 'a) \Rightarrow real$ **and**
$l :: real$
assumes
zero-le-disc [*simp*]: $0 \leq l$ **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*: $r \in bfun$
⟨*proof*⟩

lemma *r-bounded′*: *bounded* (r ‘ X)
⟨*proof*⟩

definition $r_M = (\bigsqcup sa.\ |r\ sa|)$

lemma *abs-r-le-*r_M: $|r\ sa| \leq r_M$
⟨*proof*⟩

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

lemma r_M*-nonneg*: $0 \leq r_M$
⟨*proof*⟩

lemma *measurable-r-nth* [*measurable*]: $(\lambda t.\ r\ (t\ !!\ i)) \in$ *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* $d\ (\lambda a.\ |r\ (s,\ a)|) \le r_M$
  ⟨*proof*⟩

lemma *abs-exp-r-le*: $|$*measure-pmf.expectation* $d\ r| \le r_M$
  ⟨*proof*⟩

7.1.2 Infinite disounted sums

lemma *abs-disc-eq*[*simp*]: $|l\ \widehat{}\ i * x| = l\ \widehat{}\ i * |x|$
  ⟨*proof*⟩

lemma *norm-l-pow-eq*[*simp*]: *norm* $(l\widehat{}t *_R F) = l\widehat{}t *$ *norm* F
  ⟨*proof*⟩

7.2 Total Reward for Single Traces

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

lemma *abs-*ν*-trace-fin-le*: $|\nu$-*trace-fin* $t\ N| \le (\sum i < N.\ l\widehat{}i * r_M)$
  ⟨*proof*⟩

lemma *measurable-suminf-reward*[*measurable*]: ν-*trace* $\in$ *borel-measurable* S
  ⟨*proof*⟩

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

context
  fixes p :: $('s,\ 'a)$ *pol*
begin

7.3 Expected Finite-Horizon Discounted Reward

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

lemma *abs-*ν*-fin-le*: $|\nu$-*fin* $N\ s| \le (\sum i{<}N.\ l\widehat{}i * r_M)$
  ⟨*proof*⟩

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

⟨*proof*⟩

lift-definition ν_b-*fin* :: *nat* ⇒ *'s* $\Rightarrow_b$ *real* **is** ν-*fin*
⟨*proof*⟩

lemma ν-*fin-Suc*[*simp*]: ν-*fin* (*Suc n*) *s* = ν-*fin n s* + *l* ^ *n* ∗ ∫ *t*. *r* (*t* !! *n*) ∂$\mathcal{T}$ *p s*
⟨*proof*⟩

lemma ν-*fin-zero*[*simp*]: ν-*fin 0 s* = *0*
⟨*proof*⟩

lemma ν-*fin-eq-Pn*: ν-*fin n s* = (∑ *i*<*n*. *l*^*i* ∗ *measure-pmf.expectation* (*Pn' p s i*) *r*)
⟨*proof*⟩
end

7.4 Expected Total Discounted Reward

definition ν *p s* = *lim* (λ*n*. ν-*fin p n s*)

lemmas ν-*eq-lim* = ν-*def*

lemma ν-*eq-Pn*: ν *p s* = (∑ *i*. *l*^*i* ∗ *measure-pmf.expectation* (*Pn' p s i*) *r*)
⟨*proof*⟩

7.5 Reward of a Decision Rule

context
fixes *d* :: (*'s*, *'a*) *dec*
begin
abbreviation *r-dec s* ≡ ∫ *a*. *r* (*s*, *a*) ∂*d s*

lemma *abs-r-dec-le*: |*r-dec s*| ≤ r_M
⟨*proof*⟩

lemma *r-dec-eq-r-K0*: *r-dec s* = *measure-pmf.expectation* (*K0' d s*) *r*
⟨*proof*⟩

lemma *r-dec-bfun*: *r-dec* ∈ *bfun*
⟨*proof*⟩

lift-definition *r-dec*$_b$ :: *'s* $\Rightarrow_b$ *real* **is** *r-dec*
⟨*proof*⟩

declare *r-dec*$_b$*.rep-eq*[*simp*] *bfun.Bfun-inverse*[*simp*]

lemma *norm-r-dec-le*: *norm r-dec*$_b$ ≤ r_M
⟨*proof*⟩

end

lemma *r-dec-det* [*simp*]: *r-dec* (*mk-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 $\mathcal{P}_X$-*0*[*simp*]: $\mathcal{P}_X\ 0 = id$
⟨*proof*⟩

lemma $\mathcal{P}_X$-*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* ($\mathcal{P}_X\ n\ x$) ≤ *norm* *x*
⟨*proof*⟩

lemma $\mathcal{P}_X$-*bound-r*: *norm* ($\mathcal{P}_X\ t$ (*r-dec*$_b$ (*p* *t*))) ≤ r_M
⟨*proof*⟩

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

end

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

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

lemma ν-*fin-eq*-$\mathcal{P}_X$: ν-*fin* (*mk-markovian* *p*) *n* = ($\sum i{<}n.\ l\hat{}i *_R \mathcal{P}_X$ *p* *i* (*r-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 $\mathcal{P}_1$ :: $('s \Rightarrow_b real) \Rightarrow_L ('s \Rightarrow_b real)$ **is** *push-exp* (*K-st d*)
⟨*proof*⟩

lemma $\mathcal{P}_1$*-bfun-one* [*simp*]:$\mathcal{P}_1$ *1* = *1*
⟨*proof*⟩

lemma $\mathcal{P}_1$*-pow-bfun-one* [*simp*]: $(\mathcal{P}_1$ ^^$t)$ *1* = *1*
⟨*proof*⟩

lemma $\mathcal{P}_1$*-pow*: *blinfun-apply* $(\mathcal{P}_1$ ^^ $n)$ = *blinfun-apply* $\mathcal{P}_1$ ^^ n
⟨*proof*⟩

lemma *norm-*$\mathcal{P}_1$ [*simp*]: *norm* $\mathcal{P}_1$ = *1*
⟨*proof*⟩
end

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

lemma $\mathcal{P}_X$*-Suc'*: $\mathcal{P}_X$ p (*Suc* n) v = $\mathcal{P}_X$ p n ($\mathcal{P}_1$ (p n) v)
⟨*proof*⟩

lemma $\mathcal{P}_X$*-const*: $\mathcal{P}_X$ (λ-. d) n = $\mathcal{P}_1$ d ^^ n
⟨*proof*⟩

lemma $\mathcal{P}_X$*-sconst*: $\mathcal{P}_X$ (λ-. p) n = $\mathcal{P}_1$ p ^^n
⟨*proof*⟩

lemma *norm-P-n*[*simp*]: *onorm* ($\mathcal{P}_1$ d ^^ n) = *1*
⟨*proof*⟩

lemma *norm-*$\mathcal{P}_1$*-pow* [*simp*]: *norm* ($\mathcal{P}_1$ d ^^ t) = *1*
⟨*proof*⟩

lemma $\mathcal{P}_X$*-Suc-n-elem*: $\mathcal{P}_X$ p n ($\mathcal{P}_1$ (p n) v) = $\mathcal{P}_X$ p (*Suc* n) v
⟨*proof*⟩

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

lemma $\mathcal{P}_1$*-pos*: $0 \le u \Longrightarrow 0 \le \mathcal{P}_1\ d\ u$
⟨*proof*⟩

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

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

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

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

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

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

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

7.7.1 Bellman Operator for Single Actions

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

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

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

lemma L_a*-int*:
fixes $d :: 'a\ pmf$ **and** $v :: 's \Rightarrow_b real$
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: $L\ d\ v\ s = measure\text{-}pmf.expectation\ (d\ s)\ (\lambda a.\ L_a\ a\ v\ s)$
⟨*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*: $measure\text{-}pmf.expectation\ p\ (\lambda a.\ L_a\ a\ (apply\text{-}bfun\ v)\ s) =$
$L\ (\lambda t.\ if\ t = s\ then\ p\ else\ return\text{-}pmf\ (SOME\ a.\ a \in A\ t))\ v\ s$
⟨*proof*⟩

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

lemma *L_a-le′*: $L_a\ a\ (apply\text{-}bfun\ v)\ s \le r_M + l * norm\ v$
⟨*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]*: $bounded\ ((\lambda p.\ L\ p\ v\ s)\ `\ X)$
⟨*proof*⟩

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

lemma *L-le-$\mathcal{L}_b$*: $is\text{-}dec\ d \Longrightarrow L\ d\ v \le \mathcal{L}_b\ v$
⟨*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) (λa. L_a a (*apply-bfun* v) s)
$\langle proof \rangle$

lemma *SUP-L_a-eq-det*:
fixes v :: $'s \Rightarrow_b$ *real*
shows ($\bigsqcup p{\in}\{p.$ *set-pmf* $p \subseteq A\ s\}$. $\int a.\ L_a\ a\ v\ s\ \partial$*measure-pmf* p) = ($\bigsqcup a{\in}A\ s.\ L_a\ a\ v\ s$)
$\langle proof \rangle$

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

7.9 Monotonicity

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

lemma *$\mathcal{P}_1$-mono*[*intro*]: $a \leq b \Longrightarrow \mathcal{P}_1\ p\ a \leq \mathcal{P}_1\ p\ b$
$\langle proof \rangle$

lemma *L-mono*[*intro*]: $u \leq v \Longrightarrow L\ d\ u \leq L\ d\ v$
$\langle proof \rangle$

lemma *$\mathcal{L}_b$-mono*[*intro*]: $u \leq v \Longrightarrow \mathcal{L}_b\ u \leq \mathcal{L}_b\ v$
$\langle proof \rangle$

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-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 $\mathcal{P}_X\ p\ n\ (L\ (p\ n)\ v) \leq \mathcal{P}_X\ p\ n\ v$
⟨*proof*⟩

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

lemma *summable-disc* [*intro*, *simp*]: *summable* ($\lambda i.\ l\ \widehat{}\ i * x$)
⟨*proof*⟩

lemma *summable-r-disc*[*intro*, *simp*]:
summable ($\lambda i.\ |l\ \widehat{}\ i * r\ (sa\ i)|$)
summable ($\lambda i.\ l\ \widehat{}\ i * |r\ (sa\ i)|$)
summable ($\lambda i.\ l\ \widehat{}\ i * r\ (sa\ i)$)
⟨*proof*⟩

lemma *summable-norm-disc-I*[*intro*]:
assumes *summable* ($\lambda t.$ ($l\widehat{}t$ * *norm* F))
shows *summable* ($\lambda t.$ *norm* ($l\widehat{}t *_R F$))
⟨*proof*⟩

lemma *summable-norm-disc-I'*[*intro*]:
assumes *summable* ($\lambda t.$ ($l\widehat{}t$ * *norm* ($F\ t$)))
shows *summable* ($\lambda t.$ *norm* ($l\widehat{}t *_R F\ t$))
⟨*proof*⟩

lemma *summable-discI* [*intro*]:
assumes *bounded* (*range* F)
shows *summable* ($\lambda t.\ l\widehat{}t$ * *norm* ($F\ t$))
⟨*proof*⟩

lemma *summable-disc-reward* [*intro*]:
assumes *bounded* (*range* (F :: *nat* ⇒ $'b$ :: *banach*))
shows *summable* ($\lambda t.\ l\widehat{}t *_R (F\ t)$)
⟨*proof*⟩

lemma *summable-norm-bfun-disc*: *summable* ($\lambda t.\ l\widehat{}t$ * *norm* (*apply-bfun* $f\ t$))
⟨*proof*⟩

lemma *summable-bfun-disc* [*simp*]: *summable* ($\lambda t.$ l^t $*$ (*apply-bfun f* t))
⟨*proof*⟩

lemma *norm-bfun-disc-le*: *norm f* $\leq B \Longrightarrow (\sum x.$ l^x $*$ *norm* (*apply-bfun* f x)) $\leq (\sum x.$ l^x $*$ B)
⟨*proof*⟩

lemma *norm-bfun-disc-le'*: *norm* $f \leq B \Longrightarrow (\sum x.$ l^x $*$ (*apply-bfun* f x)) $\leq (\sum x.$ l^x $*$ B)
⟨*proof*⟩

lemma *sum-disc-lim-l*: $(\sum x.$ l^x $*$ $B) = B$ /(*1*−l)
⟨*proof*⟩

lemma *sum-disc-bound*: $(\sum x.$ l^x $*$ *apply-bfun f x*) $\leq$ (*norm f*) /(*1*−l)
⟨*proof*⟩

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

lemma *abs-ν-trace-le*: |*ν-trace t*| $\leq (\sum i.$ l ^ i $*$ r_M)
⟨*proof*⟩

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

context
fixes p :: (*'s*, *'a*) *pol*
begin

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

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

lemma *ν-le*: ν p s $\leq (\sum i.$ l^i $*$ r_M)
⟨*proof*⟩

lemma *ν-bfun*: ν $p \in$ *bfun*
⟨*proof*⟩

lift-definition ν_b :: $'s \Rightarrow_b$ *real* **is** ν *p*
⟨*proof*⟩

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

lemma *ν-as-markovian*: ν (*mk-markovian* (*as-markovian p* (*return-pmf s*))) *s* = ν *p s*
⟨*proof*⟩

lemma *ν_b-as-markovian*: ν_b (*mk-markovian* (*as-markovian p* (*return-pmf s*))) *s* = ν_b *p s*
⟨*proof*⟩

7.10 Optimal Reward

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

lemma *ν-opt-bfun*: *ν-opt* $\in$ *bfun*
⟨*proof*⟩

lift-definition *ν_b-opt* :: $'s \Rightarrow_b$ *real* **is** *ν-opt*
⟨*proof*⟩

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

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

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

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

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

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

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

lemma *summable-norm-disc-reward'*[*simp*]: *summable* ($\lambda t.\ l\widehat{\ }t * norm\ (\mathcal{P}_X\ p\ t\ (r\text{-}dec_b\ (p\ t))))$)
⟨*proof*⟩

lemma *summable-disc-reward-*$\mathcal{P}_X$ [*simp*]: *summable* ($\lambda t.\ l\widehat{\ }t *_R \mathcal{P}_X\ p\ t\ (r\text{-}dec_b\ (p\ t)))$
⟨*proof*⟩

lemma *disc-reward-tendsto*:
$(\lambda n.\ \sum t{<}n.\ l\widehat{\ }t *_R \mathcal{P}_X\ p\ t\ (r\text{-}dec_b\ (p\ t))) \longrightarrow (\sum t.\ l\widehat{\ }t *_R \mathcal{P}_X\ p\ t\ (r\text{-}dec_b\ (p\ t)))$
⟨*proof*⟩

lemma ν*-eq-*$\mathcal{P}_X$: ν *(mk-markovian p)* $= (\sum i.\ l\widehat{\ }i *_R \mathcal{P}_X\ p\ i\ (r\text{-}dec_b\ (p\ i)))$
⟨*proof*⟩

lemma ν_b*-eq-*$\mathcal{P}_X$: ν_b *(mk-markovian p)* $= (\sum i.\ l\widehat{\ }i *_R \mathcal{P}_X\ p\ i\ (r\text{-}dec_b\ (p\ i)))$
⟨*proof*⟩

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

lemma *norm-*$\mathcal{P}_1$*-l-less*: $norm\ (l *_R \mathcal{P}_1\ d) < 1$
⟨*proof*⟩
lemma *disc-*$\mathcal{P}_1$*-tendsto*: $(\lambda n.\ (\sum t{\le}n.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t)) \longrightarrow (\sum t.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t)$
⟨*proof*⟩

lemma *disc-*$\mathcal{P}_1$*-lim*: $lim\ (\lambda n.\ (\sum t{\le}n.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t)) = (\sum t.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t)$
⟨*proof*⟩

lemma *convergent-disc-*$\mathcal{P}_1$: *convergent* $(\lambda n.\ (\sum t{\le}n.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t))$
⟨*proof*⟩

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

lemma $\mathcal{P}_1$*-suminf-pos*:
assumes $0 \le u$
shows $0 \le (\sum t.\ l\widehat{\ }t *_R \mathcal{P}_1\ d\ \widehat{\ }\widehat{\ }t)\ u$

$\langle proof \rangle$

lemma *lemma-6-1-2-b*:
assumes $v \leq u$
shows $(\sum t.\ l\hat{}t *_R \mathcal{P}_1\ d\ \hat{}\hat{}t)\ v \leq (\sum t.\ l\hat{}t *_R \mathcal{P}_1\ d\ \hat{}\hat{}t)\ u$
$\langle proof \rangle$

lemma *ν-stationary*: $\nu_b\ (mk\text{-}stationary\ d) = (\sum t.\ l\hat{}t *_R (\mathcal{P}_1\ d\ \hat{}\hat{}t))\ (r\text{-}dec_b\ d)$
$\langle proof \rangle$

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

lemma *bounded-P*: *bounded* $(\mathcal{P}_1$ ' $X)$
⟨*proof*⟩

7.12 Solutions to the Optimality Equation

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

declare *bounded-apply-blinfun*[*intro*] *bounded-apply-bfun'*[*intro*]

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

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

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

lemma *is-contraction-L*: *is-contraction* $(L\ p)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

7.13 Existence of Optimal Policies

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

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

lemma ν*-improving-D-MR*[*dest*]: $\nu\text{-}improving\ v\ d \Longrightarrow d \in D_R$
⟨*proof*⟩

lemma ν*-improving-ge*: $\nu\text{-}improving\ v\ d \Longrightarrow d' \in D_R \Longrightarrow L\ d'\ v\ s \leq L\ d\ v\ s$
⟨*proof*⟩

lemma ν*-improving-imp-*$\mathcal{L}_b$: $\nu\text{-}improving\ v\ d \Longrightarrow \mathcal{L}_b\ v = L\ d\ v$
⟨*proof*⟩

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

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

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

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

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

lemma ν*-conserving-imp-*$\mathcal{L}_b$ [*simp*]: $\nu\text{-}conserving\ d \Longrightarrow L\ d\ \nu_b\text{-}opt = \nu_b\text{-}opt$
⟨*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 $\nu\text{-}conserving\ d$

$\langle proof \rangle$

lemma *ν-conserving-alt*:
assumes $d \in D_R$
shows ν-*conserving* $d \longleftrightarrow \mathcal{L}_b\ \nu_b\text{-}opt = L\ d\ \nu_b\text{-}opt$
$\langle proof \rangle$

lemma *ν-conserving-alt$'$*:
assumes $d \in D_R$
shows ν-*conserving* $d \longleftrightarrow L\ d\ \nu_b\text{-}opt = \nu_b\text{-}opt$
$\langle proof \rangle$

7.13.1 Conserving Decision Rules are Optimal

theorem *ex-improving-imp-conserving*:
assumes $\bigwedge v.\ \exists\, d.\ \nu\text{-}improving\ v\ (mk\text{-}dec\text{-}det\ d)$
shows $\exists\, d.\ \nu\text{-}conserving\ (mk\text{-}dec\text{-}det\ d)$
$\langle proof \rangle$

theorem *conserving-imp-opt*[*simp*]:
assumes $\nu\text{-}conserving\ (mk\text{-}dec\text{-}det\ d)$
shows $\nu_b\ (mk\text{-}stationary\text{-}det\ d) = \nu_b\text{-}opt$
$\langle proof \rangle$

lemma *conserving-imp-opt$'$*:
assumes $\exists\, d.\ \nu\text{-}conserving\ (mk\text{-}dec\text{-}det\ d)$
shows $\exists\, d \in D_D.\ (\nu_b\ (mk\text{-}stationary\text{-}det\ d)) = \nu_b\text{-}opt$
$\langle proof \rangle$

theorem *improving-att-imp-det-opt*:
assumes $\bigwedge v.\ \exists\, d.\ \nu\text{-}improving\ v\ (mk\text{-}dec\text{-}det\ d)$
shows $\nu_b\text{-}opt\ s = (\bigsqcup d \in D_D.\ \nu_b\ (mk\text{-}stationary\text{-}det\ d)\ s)$
$\langle proof \rangle$

lemma *$\mathcal{L}_b$-sup-att-dec*:
assumes $d \in D_R$ $\mathcal{L}_b\ v = L\ d\ v$
shows $\exists\, d' \in D_D.\ \mathcal{L}_b\ v = L\ (mk\text{-}dec\text{-}det\ d')\ v$
$\langle proof \rangle$

lemma *$\mathcal{L}_b$-sup-att-dec$'$*:
assumes $d \in D_R$ $\mathcal{L}_b\ v = L\ d\ v$
shows $\exists\, d' \in D_D.\ \nu\text{-}improving\ v\ (mk\text{-}dec\text{-}det\ d')$
$\langle proof \rangle$

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

$\langle proof \rangle$

7.13.3 Optimal Decision Rules for Finite Action Spaces

lemma *ex-opt-act*:
assumes $\bigwedge s.\ \mathit{finite}\ (A\ s)$
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 $\bigwedge s.\ \mathit{finite}\ (A\ s)$
shows $\exists\, d \in D_D.\ L\ (\mathit{mk\text{-}dec\text{-}det}\ d)\ (v :: - \Rightarrow_b -) = \mathcal{L}_b\ v$
$\langle proof \rangle$

lemma *thm-6-2-10*:
assumes $\bigwedge s.\ \mathit{finite}\ (A\ s)$
shows $\exists\, d \in D_D.\ \nu_b\text{-}\mathit{opt} = \nu_b\ (\mathit{mk\text{-}stationary\text{-}det}\ d)$
$\langle proof \rangle$

7.13.4 Existence of Epsilon-Optimal Policies

lemma *ex-det-eps*:
assumes $0 < e$
shows $\exists\, d \in D_D.\ \mathcal{L}_b\ v \le L\ (\mathit{mk\text{-}dec\text{-}det}\ d)\ v + e *_R 1$
$\langle proof \rangle$

lemma *thm-6-2-11*:
assumes $\mathit{eps} > 0$
shows $\exists\, d \in D_D.\ \nu_b\text{-}\mathit{opt} \le \nu_b\ (\mathit{mk\text{-}stationary\text{-}det}\ d) + \mathit{eps} *_R 1$
$\langle proof \rangle$

lemma *ex-det-dist-eps*:
assumes $0 < (e :: \mathit{real})$
shows $\exists\, d \in D_D.\ \mathit{dist}\ (\mathcal{L}_b\ v)\ (L\ (\mathit{mk\text{-}dec\text{-}det}\ d)\ v) \le e$
$\langle proof \rangle$

lemma *less-imp-ex-add-le*: $(x :: \mathit{real}) < y \Longrightarrow \exists\, \mathit{eps}{>}0.\ x + \mathit{eps} \le y$
$\langle proof \rangle$

lemma ν_b*-opt-le-det*: $\nu_b\text{-}\mathit{opt}\ s \le (\bigsqcup d \in D_D.\ \nu_b\ (\mathit{mk\text{-}stationary\text{-}det}\ d)\ s)$
$\langle proof \rangle$

lemma ν_b*-opt-eq-det*: $\nu_b\text{-}\mathit{opt}\ s = (\bigsqcup d \in D_D.\ \nu_b\ (\mathit{mk\text{-}stationary\text{-}det}\ d)\ s)$
$\langle proof \rangle$

lemma *lemma-6-3-1-a*:
assumes $v0 \in \mathit{bfun}$

shows *uniform-limit UNIV* ($\lambda n.$ (($\lambda v.$ $\mathcal{L}$ (*Bfun v*)) $\frown\frown$ *n*) *v0*) ν*-opt sequentially*
⟨*proof*⟩

lemma *dist-Suc-tendsto-zero*:
assumes ($\lambda n.$ *f n*) $\longrightarrow$ (*y*::-::*real-normed-vector*)
shows ($\lambda n.$ *dist* (*f n*) (*f* (*Suc n*))) $\longrightarrow$ *0*
⟨*proof*⟩

lemma *dist-*$\mathcal{L}_b$*-tendsto*: ($\lambda n.$ *dist* (($\mathcal{L}_b$ $\frown\frown$ *n*) *v*) (($\mathcal{L}_b$ $\frown\frown$ (*Suc n*)) *v*)) $\longrightarrow$ *0*
⟨*proof*⟩

definition *max-L-ex s v* $\equiv$ *has-arg-max* ($\lambda a.$ L_a *a v s*) (*A s*)

lemma ν_b*-fin-zero*[*simp*]: ν_b*-fin p 0* = *0*
⟨*proof*⟩

lemma ν_b*-fin-Suc*[*simp*]:
ν_b*-fin* (*mk-stationary d*) (*Suc n*) = ν_b*-fin* (*mk-stationary d*) *n* + ((*l* $*_R$ $\mathcal{P}_1$ *d*) $\frown\frown$ *n*) (*r-dec*$_b$ *d*)
⟨*proof*⟩

lemma ν_b*-fin-eq*: ν_b*-fin* (*mk-stationary d*) *n* = ($\sum i < n.$ ((*l* $*_R$ $\mathcal{P}_1$ *d*) $\frown\frown$ *i*)) (*r-dec*$_b$ *d*)
⟨*proof*⟩

lemma *L-iter*: (*L d* $\frown\frown$ *m*) *v* = ν_b*-fin* (*mk-stationary d*) *m* + ((*l* $*_R$ $\mathcal{P}_1$ *d*) $\frown\frown$ *m*) *v*
⟨*proof*⟩

lemma *bounded-stationary-*ν_b*-fin*: *bounded* (($\lambda x.$ (ν_b*-fin* (*mk-stationary x*) *N*) *s*) ‘ *X*)
⟨*proof*⟩

lemma *bounded-disc-*$\mathcal{P}_1$: *bounded* (($\lambda x.$ (((*l* $*_R$ $\mathcal{P}_1$ *x*) $\frown\frown$ *m*) *v*) *s*) ‘ *X*)
⟨*proof*⟩

lemma *bounded-disc-*$\mathcal{P}_1$*'*: *bounded* (($\lambda x.$ (($\mathcal{P}_1$ *x* $\frown\frown$ *m*) *v*) *s*) ‘ *X*)
⟨*proof*⟩

lemma *L-iter-le-*$\mathcal{L}_b$: *is-dec d* $\Longrightarrow$ (*L d* $\frown\frown$ *n*) *v* $\leq$ ($\mathcal{L}_b$ $\frown\frown$ *n*) *v*
⟨*proof*⟩

end

7.14 More Restrictive MDP Locales

locale *MDP-fin-acts* = *discrete-MDP* +
  assumes $\bigwedge s.$ *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*: $a \in A\ s \Longrightarrow L_a\ a\ v\ s \le L_a$ (*arg-max-on* $(\lambda a.$ $L_a\ a\ v\ s)\ (A\ s))\ v\ s$
  ⟨*proof*⟩

lemma *arg-max-on-in*: *has-arg-max* $f\ Q \Longrightarrow$ *arg-max-on* $f\ Q \in Q$
  ⟨*proof*⟩

lemma *$\mathcal{L}_b$-eq-L_a-max*: $\mathcal{L}_b\ v\ s = L_a$ (*arg-max-on* $(\lambda a.\ L_a\ a\ v\ s)\ (A\ s))$ $v\ s$
  ⟨*proof*⟩

lemma *ex-opt-det*: $\exists\, d \in D_D.\ \mathcal{L}_b\ v = L$ (*mk-dec-det* d) v
⟨*proof*⟩

lemma *ex-improving-det*: $\exists\, d \in D_D.$ *ν-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*]: $X \neq \{\} \Longrightarrow$ *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* $v\ s$) $\in A\ s$

$\langle proof \rangle$

lemma *ν-improving-opt-acts*: *ν-improving v0* (*mk-dec-det* (λ*s. arb-act* (*opt-acts* (*apply-bfun v0*) *s*)))
$\langle proof \rangle$

end

locale *MDP-finite-type* = *MDP-reward-disc A K r l*
for *A* **and** *K* :: *'s* :: *finite* × *'a* :: *finite* ⇒ *'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.