

Markov Models

Johannes Hölzl and Tobias Nipkow

December 14, 2021

Abstract

This is a formalization of various Markov models in Isabelle/HOL. It builds on Isabelle's probability theory. The available models are currently discrete-time and continuous-time Markov chains as well as Markov decision processes. As application of these models we formalize probabilistic model checking of pCTL formulas, analysis of IPv4 address allocation in ZeroConf and an analysis of the anonymity of the Crowds protocol.

Contents

1	Introduction	3
2	Auxiliary Theory	4
3	Discrete-Time Markov Chain	11
3.1	Discrete Markov Kernel	11
3.2	Trace Space for Discrete-Time Markov Chains	13
3.3	Fairness	15
3.4	First Hitting Time	16
3.5	Markov chain with Initial Distribution	17
3.6	Trace space with Restriction	19
3.7	Bisimulation	19
3.8	Reward Structure on Markov Chains	20
3.9	Bisimulation on a relation	21
3.10	Product Construction	22
3.11	Trace Space equal to Markov Chains	22
4	Classifying Markov Chain States	24
4.1	Expected number of visits	24
4.2	Reachability probability	26
4.3	Recurrent states	28
4.4	Stationary distribution	32

5	Markov Decision Processes	35
5.1	Configurations	36
5.2	Configuration with Memoryless Scheduler	38
5.3	MDP Kernel and Induced Configurations	38
5.4	Trace Space	39
5.5	Finite MDPs	43
6	Discrete-time Markov Processes	51
6.1	Constructing Discrete-Time Markov Processes	52
6.2	Strong Markov Property for Discrete-Time Markov Processes	54
7	Continuous-time Markov chains	56
7.1	Trace Operations: relate ($'a \times real$) <i>stream</i> and $real \Rightarrow 'a$	56
7.2	Exponential Distribution	56
7.3	Transition Rates	58
7.4	Continuous-time Kernel	59
7.5	Kernel equals Parallel Choice	60
7.6	Markov Chain Property	61
7.7	Explosion time	62
7.8	Transition probability p_t	62
8	Example A	65
8.1	The essential class $\{C1, C2, C3\}$	66
8.2	The stationary distribution n	66
9	Example B	67
9.1	Enabled, accessible and communicating states	68
9.2	B is aperiodic	68
9.3	The stationary distribution N	68
9.4	Limit behavior and recurrence times	68
10	Adapt Gauss-Jordan elimination to DTMCs	69
11	pCTL model checking	70
11.1	Syntax	70
11.2	Semantics	71
11.3	Implementation of <i>Sat</i>	72
11.3.1	<i>Prob0</i>	72
11.3.2	<i>Prob1</i>	72
11.3.3	<i>ProbU</i> , <i>ExpCumm</i> , and <i>ExpState</i>	73
11.3.4	<i>LES</i>	73
11.3.5	<i>ProbUinfty</i> , compute unbounded until	73
11.3.6	<i>ExpFuture</i> , compute unbounded reward	73
11.3.7	<i>Sat</i>	73
11.3.8	Finite expected reward	74

11.3.9	The expected reward implies a unique LES	75
11.4	Soundness of <i>Sat</i>	76
11.5	Completeness of <i>Sat</i>	76
11.6	Completeness and Soundness <i>Sat</i>	76
12	Probabilistic Guarded Command Language (pGCL)	76
12.1	Syntax	76
12.2	Denotational Semantics	76
12.3	Operational Semantics	77
12.4	Equate Both Semantics	79
13	Formalization of the Crowds-Protocol	79
13.1	Definition of the Crowds-Protocol	79
13.2	Server gets no information	83
13.3	Probability that collaborators gain information	83
13.4	The probability that the sender hits a collaborator	84
13.5	Probability space of hitting a collaborator	85
13.6	Estimate the information to the collaborators	85
13.6.1	Setup random variables for mutual information	85
14	Formalizing the IPv4-address allocation in ZeroConf	86
14.1	Definition of a ZeroConf allocation run	86
14.2	The allocation run is a rewarded DTMC	87
14.3	Probability of a erroneous allocation	88
14.4	An allocation run terminates almost surely	88
14.5	Expected runtime of an allocation run	89
15	Formalization of the Gossip-Broadcast	89
15.1	Definition of the Gossip-Broadcast	90
15.2	The Gossip-Broadcast forms a DTMC	91
16	Certification of Reachability Problems on MDPs	91
16.1	Computable representation	92

1 Introduction

This is a formalization of probabilistic models in Isabelle/HOL. It builds on Isabelle's probability theory (HOL-Probability). It provides formalizations for the following models:

- Discrete-time Markov processes with measurable state spaces [2]
- Markov decision processes on discrete spaces [5]
- Continuous-time Markov chains on discrete spaces [2]

As application of these models we formalize

- a probabilistic model checking of pCTL formulas [4],
- an analysis of IPv4 address allocation in ZeroConf [3],
- an analysis of the anonymity of the Crowds protocol [3],
- the reachability analysis on finite-state MDPs [5], and
- expected running-time semantics for pGCL [1].

The formalization of rewarded DTMCs and pCTL model checking is discussed in detail in our paper.

2 Auxiliary Theory

Parts of it should be moved to the Isabelle repository

theory *Markov-Models-Auxiliary*

imports

HOL-Probability.Probability

HOL-Library.Rewrite

HOL-Library.Linear-Temporal-Logic-on-Streams

Coinductive.Coinductive-Stream

Coinductive.Coinductive-Nat

begin

lemma *lfp-upperbound*: $(\bigwedge y. x \leq f y) \implies x \leq \text{lfp } f$
 $\langle \text{proof} \rangle$

lemma *lfp-arg*: $(\lambda t. \text{lfp } (F t)) = \text{lfp } (\lambda x t. F t (x t))$
 $\langle \text{proof} \rangle$

lemma *lfp-pair*: $\text{lfp } (\lambda f (a, b). F (\lambda a b. f (a, b)) a b) (a, b) = \text{lfp } F a b$
 $\langle \text{proof} \rangle$

lemma *all-Suc-split*: $(\forall i. P i) \longleftrightarrow (P 0 \wedge (\forall i. P (\text{Suc } i)))$
 $\langle \text{proof} \rangle$

definition *with* $P f d = (\text{if } \exists x. P x \text{ then } f (\text{SOME } x. P x) \text{ else } d)$

lemma *withI*[*case-names default exists*]:

$((\bigwedge x. \neg P x) \implies Q d) \implies (\bigwedge x. P x \implies Q (f x)) \implies Q (\text{with } P f d)$
 $\langle \text{proof} \rangle$

context *order*

begin

definition

$$\text{maximal } f S = \{x \in S. \forall y \in S. f y \leq f x\}$$

lemma maximalI: $x \in S \implies (\bigwedge y. y \in S \implies f y \leq f x) \implies x \in \text{maximal } f S$
(proof)

lemma maximalI-trans: $x \in \text{maximal } f S \implies f x \leq f y \implies y \in S \implies y \in \text{maximal } f S$
(proof)

lemma maximalD1: $x \in \text{maximal } f S \implies x \in S$
(proof)

lemma maximalD2: $x \in \text{maximal } f S \implies y \in S \implies f y \leq f x$
(proof)

lemma maximal-inject: $x \in \text{maximal } f S \implies y \in \text{maximal } f S \implies f x = f y$
(proof)

lemma maximal-empty[simp]: $\text{maximal } f \{\} = \{\}$
(proof)

lemma maximal-singleton[simp]: $\text{maximal } f \{x\} = \{x\}$
(proof)

lemma maximal-in-S: $\text{maximal } f S \subseteq S$
(proof)

end

context *linorder*

begin

lemma maximal-ne:

assumes *finite* $S S \neq \{\}$

shows $\text{maximal } f S \neq \{\}$

(proof)

end

lemma mono-les:

fixes $s S N$ **and** $l1 l2 :: 'a \Rightarrow \text{real}$ **and** $K :: 'a \Rightarrow 'a \text{ pmf}$

defines $\Delta x \equiv l2 x - l1 x$

assumes $s: s \in S$ **and** $S: (\bigcup s \in S. \text{set-pmf } (K s)) \subseteq S \cup N$

assumes *int-l1*[simp]: $\bigwedge s. s \in S \implies \text{integrable } (K s) l1$

assumes *int-l2*[simp]: $\bigwedge s. s \in S \implies \text{integrable } (K s) l2$

assumes *to-N*: $\bigwedge s. s \in S \implies \exists t \in N. (s, t) \in (\text{SIGMA } s:UNIV. K s)^*$

assumes $l1$: $\bigwedge s. s \in S \implies (\int t. l1 t \partial K s) + c s \leq l1 s$

assumes $l2$: $\bigwedge s. s \in S \implies l2 s \leq (\int t. l2 t \partial K s) + c s$

assumes $eq: \bigwedge s. s \in N \implies l2\ s \leq l1\ s$
assumes *finitary*: *finite* ($\Delta\ ' (S \cup N)$)
shows $l2\ s \leq l1\ s$
<proof>

lemma *unique-les*:

fixes $s\ S\ N$ **and** $l1\ l2 :: 'a \Rightarrow real$ **and** $K :: 'a \Rightarrow 'a\ pmf$
defines $\Delta\ x \equiv l2\ x - l1\ x$
assumes $s: s \in S$ **and** $S: (\bigcup s \in S. set\ pmf\ (K\ s)) \subseteq S \cup N$
assumes $\bigwedge s. s \in S \implies integrable\ (K\ s)\ l1$
assumes $\bigwedge s. s \in S \implies integrable\ (K\ s)\ l2$
assumes $\bigwedge s. s \in S \implies \exists t \in N. (s, t) \in (SIGMA\ s: UNIV. K\ s)^*$
assumes $\bigwedge s. s \in S \implies l1\ s = (\int t. l1\ t\ \partial K\ s) + c\ s$
assumes $\bigwedge s. s \in S \implies l2\ s = (\int t. l2\ t\ \partial K\ s) + c\ s$
assumes $\bigwedge s. s \in N \implies l2\ s = l1\ s$
assumes $1: finite\ (\Delta\ ' (S \cup N))$
shows $l2\ s = l1\ s$
<proof>

lemma *inf-continuous-suntil-disj*[*order-continuous-intros*]:

assumes $Q: inf\ continuous\ Q$
assumes *disj*: $\bigwedge x\ \omega. \neg (P\ \omega \wedge Q\ x\ \omega)$
shows *inf-continuous* ($\lambda x. P\ suntil\ Q\ x$)
<proof>

lemma *inf-continuous-nxt*[*order-continuous-intros*]: *inf-continuous* $P \implies inf\ continuous$
 $(\lambda x. nxt\ (P\ x)\ \omega)$
<proof>

lemma *sup-continuous-nxt*[*order-continuous-intros*]: *sup-continuous* $P \implies sup\ continuous$
 $(\lambda x. nxt\ (P\ x)\ \omega)$
<proof>

lemma *mcont-ennreal-of-enat*: *mcont* $Sup\ (\leq)\ Sup\ (\leq)\ ennreal\ of\ enat$
<proof>

lemma *mcont2mcont-ennreal-of-enat*[*cont-intro*]:
mcont $lub\ ord\ Sup\ (\leq)\ f \implies mcont\ lub\ ord\ Sup\ (\leq)\ (\lambda x. ennreal\ of\ enat\ (f\ x))$
<proof>

declare *stream.exhaust*[*cases type: stream*]

lemma *scount-eq-emeasure*: *scount* $P\ \omega = emeasure\ (count\ space\ UNIV)\ \{i. P\ (sdrop\ i\ \omega)\}$
<proof>

lemma *measurable-scount*[*measurable*]:

assumes [*measurable*]: *Measurable.pred* (*stream-space* M) P
shows *scount* $P \in measurable\ (stream\ space\ M)\ (count\ space\ UNIV)$

<proof>

lemma *measurable-sfirst2*:

assumes [*measurable*]: *Measurable.pred* ($N \otimes_M \text{stream-space } M$) ($\lambda(x, \omega). P x$
 ω)

shows ($\lambda(x, \omega). \text{sfirst } (P x) \omega$) $\in \text{measurable } (N \otimes_M \text{stream-space } M)$ (*count-space UNIV*)

<proof>

lemma *measurable-sfirst2'* [*measurable (raw)*]:

assumes [*measurable (raw)*]: $f \in N \rightarrow_M \text{stream-space } M$ *Measurable.pred* (N
 $\otimes_M \text{stream-space } M$) ($\lambda x. P (\text{fst } x) (\text{snd } x)$)

shows ($\lambda x. \text{sfirst } (P x) (f x)$) $\in \text{measurable } N$ (*count-space UNIV*)

<proof>

lemma *measurable-sfirst* [*measurable*]:

assumes [*measurable*]: *Measurable.pred* (*stream-space M*) *P*

shows *sfirst P* $\in \text{measurable } (\text{stream-space } M)$ (*count-space UNIV*)

<proof>

lemma *measurable-epred* [*measurable*]: *epred* $\in \text{count-space UNIV} \rightarrow_M \text{count-space UNIV}$

<proof>

lemma *nn-integral-stretch*:

$f \in \text{borel} \rightarrow_M \text{borel} \implies c \neq 0 \implies (\int^{+x}. f (c * x) \partial \text{lborel}) = (1 / |c| :: \text{real}) * (\int^{+x}. f x \partial \text{lborel})$

<proof>

lemma *prod-sum-distrib*:

fixes $f g :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{comm-semiring-1}$

assumes *finite I* **shows** ($\bigwedge i. i \in I \implies \text{finite } (J i)$) $\implies (\prod i \in I. \sum j \in J i. f i j)$
 $= (\sum m \in \text{Pi } E \text{ } I \text{ } J. \prod i \in I. f i (m i))$

<proof>

lemma *prod-add-distrib*:

fixes $f g :: 'a \Rightarrow 'b :: \text{comm-semiring-1}$

assumes *finite I* **shows** ($\prod i \in I. f i + g i$) $= (\sum J \in \text{Pow } I. (\prod i \in J. f i) * (\prod i \in I - J. g i))$

<proof>

subclass (**in** *linordered-nonzero-semiring*) *ordered-semiring-0*

<proof>

lemma (**in** *linordered-nonzero-semiring*) *prod-nonneg*: ($\forall a \in A. 0 \leq f a$) $\implies 0 \leq \text{prod } f A$

<proof>

lemma (**in** *linordered-nonzero-semiring*) *prod-mono*:

$\forall i \in A. 0 \leq f i \wedge f i \leq g i \implies \text{prod } f A \leq \text{prod } g A$
 ⟨proof⟩

lemma (in *linordered-nonzero-semiring*) *prod-mono2*:
 assumes *finite* $J I \subseteq J \wedge i. i \in I \implies 0 \leq g i \wedge g i \leq f i (\wedge i. i \in J - I \implies 1 \leq f i)$
 shows $\text{prod } g I \leq \text{prod } f J$
 ⟨proof⟩

lemma (in *linordered-nonzero-semiring*) *prod-mono3*:
 assumes *finite* $J I \subseteq J \wedge i. i \in J \implies 0 \leq g i \wedge i. i \in I \implies g i \leq f i (\wedge i. i \in J - I \implies g i \leq 1)$
 shows $\text{prod } g J \leq \text{prod } f I$
 ⟨proof⟩

lemma (in *linordered-nonzero-semiring*) *one-le-prod*: $(\wedge i. i \in I \implies 1 \leq f i) \implies 1 \leq \text{prod } f I$
 ⟨proof⟩

lemma *sum-plus-one-le-prod-plus-one*:
 fixes $p :: 'a \Rightarrow 'b :: \text{linordered-nonzero-semiring}$
 assumes $\wedge i. i \in I \implies 0 \leq p i$
 shows $(\sum i \in I. p i) + 1 \leq (\prod i \in I. p i + 1)$
 ⟨proof⟩

lemma *summable-iff-convergent-prod*:
 fixes $p :: \text{nat} \Rightarrow \text{real}$ assumes $p: \wedge i. 0 \leq p i$
 shows $\text{summable } p \longleftrightarrow \text{convergent } (\lambda n. \prod i < n. p i + 1)$
 ⟨proof⟩

primrec *eexp* :: $\text{ereal} \Rightarrow \text{ennreal}$
 where
 $eexp \text{MInfty} = 0$
 $| eexp (\text{ereal } r) = \text{ennreal } (exp r)$
 $| eexp \text{PInfty} = \text{top}$

lemma
 shows *eexp-minus-infty[simp]*: $eexp (-\infty) = 0$
 and *eexp-infty[simp]*: $eexp \infty = \text{top}$
 ⟨proof⟩

lemma *eexp-0[simp]*: $eexp 0 = 1$
 ⟨proof⟩

lemma *eexp-inj[simp]*: $eexp x = eexp y \longleftrightarrow x = y$
 ⟨proof⟩

lemma *eexp-mono[simp]*: $eexp x \leq eexp y \longleftrightarrow x \leq y$
 ⟨proof⟩

lemma *eexp-strict-mono*[simp]: $eexp\ x < eexp\ y \longleftrightarrow x < y$
 ⟨proof⟩

lemma *eexp-eq-0-iff*[simp]: $eexp\ x = 0 \longleftrightarrow x = -\infty$
 ⟨proof⟩

lemma *eexp-surj*: $range\ eexp = UNIV$
 ⟨proof⟩

lemma *continuous-on-eexp'*: *continuous-on UNIV eexp*
 ⟨proof⟩

lemma *continuous-on-eexp*[continuous-intros]: *continuous-on A f \implies continuous-on A ($\lambda x. eexp\ (f\ x)$)*
 ⟨proof⟩

lemma *tendsto-eexp*[tendsto-intros]: $(f \longrightarrow x)\ F \implies ((\lambda x. eexp\ (f\ x)) \longrightarrow eexp\ x)\ F$
 ⟨proof⟩

lemma *measurable-eexp*[measurable]: $eexp \in borel \rightarrow_M borel$
 ⟨proof⟩

lemma *eexp-add*: $\neg ((x = \infty \wedge y = -\infty) \vee (x = -\infty \wedge y = \infty)) \implies eexp\ (x + y) = eexp\ x * eexp\ y$
 ⟨proof⟩

lemma *sum-Pinf*ty:
fixes $f :: 'a \Rightarrow ereal$
shows $sum\ f\ I = \infty \longleftrightarrow (finite\ I \wedge (\exists i \in I. f\ i = \infty))$
 ⟨proof⟩

lemma *sum-Minf*ty:
fixes $f :: 'a \Rightarrow ereal$
shows $sum\ f\ I = -\infty \longleftrightarrow (finite\ I \wedge \neg (\exists i \in I. f\ i = \infty) \wedge (\exists i \in I. f\ i = -\infty))$
 ⟨proof⟩

lemma *eexp-sum*: $\neg (\exists i \in I. \exists j \in I. f\ i = -\infty \wedge f\ j = \infty) \implies eexp\ (\sum i \in I. f\ i) = (\prod i \in I. eexp\ (f\ i))$
 ⟨proof⟩

lemma *eexp-suminf*:
assumes $wf\ f: \neg \{-\infty, \infty\} \subseteq range\ f$ **and** $f: summable\ f$
shows $(\lambda n. \prod i < n. eexp\ (f\ i)) \longrightarrow eexp\ (\sum i. f\ i)$
 ⟨proof⟩

lemma *continuous-onI-antimono*:
fixes $f :: 'a::linorder-topology \Rightarrow 'b::\{dense-order,linorder-topology\}$

assumes *open* ($f'A$)
and *mono*: $\bigwedge x y. x \in A \implies y \in A \implies x \leq y \implies f y \leq f x$
shows *continuous-on* $A f$
 \langle *proof* \rangle

lemma *minus-add-eq-ereal*: $\neg ((a = \infty \wedge b = -\infty) \vee (a = -\infty \wedge b = \infty)) \implies$
 $- (a + b :: \text{ereal}) = -a - b$
 \langle *proof* \rangle

lemma *setsum-negf-ereal*: $\neg \{-\infty, \infty\} \subseteq f'I \implies (\sum_{i \in I}. - f i) = - (\sum_{i \in I}. f i :: \text{ereal})$
 \langle *proof* \rangle

lemma *convergent-minus-iff-ereal*: *convergent* $(\lambda x. - f x :: \text{ereal}) \iff$ *convergent* f
 \langle *proof* \rangle

lemma *summable-minus-ereal*: $\neg \{-\infty, \infty\} \subseteq \text{range } f \implies$ *summable* $(\lambda n. f n) \implies$ *summable* $(\lambda n. - f n :: \text{ereal})$
 \langle *proof* \rangle

lemma (*in product-prob-space*) *product-nn-integral-component*:
assumes $f \in \text{borel-measurable } (M i) i \in I$
shows $\text{integral}^N (P i_M I M) (\lambda x. f (x i)) = \text{integral}^N (M i) f$
 \langle *proof* \rangle

lemma *ennreal-inverse-le[simp]*: *inverse* $x \leq$ *inverse* $y \iff y \leq (x :: \text{ennreal})$
 \langle *proof* \rangle

lemma *inverse-inverse-ennreal[simp]*: *inverse* (*inverse* $x :: \text{ennreal}$) = x
 \langle *proof* \rangle

lemma *range-inverse-ennreal*: *range* *inverse* = $(UNIV :: \text{ennreal set})$
 \langle *proof* \rangle

lemma *continuous-on-inverse-ennreal'*: *continuous-on* $(UNIV :: \text{ennreal set})$ *inverse*
 \langle *proof* \rangle

lemma *sums-minus-ereal*: $\neg \{-\infty, \infty\} \subseteq f' UNIV \implies (\lambda n. - f n :: \text{ereal})$ *sums* $x \implies f$ *sums* $- x$
 \langle *proof* \rangle

lemma *suminf-minus-ereal*: $\neg \{-\infty, \infty\} \subseteq f' UNIV \implies$ *summable* $f \implies (\sum n. - f n :: \text{ereal}) = - \text{suminf } f$
 \langle *proof* \rangle

end

3 Discrete-Time Markov Chain

theory *Discrete-Time-Markov-Chain*
imports *Markov-Models-Auxiliary*
begin

Markov chain with discrete time steps and discrete state space.

lemma *sstart-eq'*: $sstart \ \Omega \ (x \ \# \ xs) = \{\omega. \ shd \ \omega = x \wedge \ stl \ \omega \in \ sstart \ \Omega \ xs\}$
<proof>

lemma *measure-eq-stream-space-coinduct*[*consumes 1, case-names left right cont*]:
assumes $R \ N \ M$
assumes *R-1*: $\bigwedge N \ M. \ R \ N \ M \implies N \in \ space \ (prob\text{-}algebra \ (stream\text{-}space \ (count\text{-}space \ UNIV)))$
and *R-2*: $\bigwedge N \ M. \ R \ N \ M \implies M \in \ space \ (prob\text{-}algebra \ (stream\text{-}space \ (count\text{-}space \ UNIV)))$
and *cont*: $\bigwedge N \ M. \ R \ N \ M \implies \exists N' \ M' \ p. \ (\forall y \in \ set\text{-}pmf \ p. \ R \ (N' \ y) \ (M' \ y)) \wedge$
 $(\forall x. \ N' \ x \in \ space \ (prob\text{-}algebra \ (stream\text{-}space \ (count\text{-}space \ UNIV)))) \wedge (\forall x.$
 $M' \ x \in \ space \ (prob\text{-}algebra \ (stream\text{-}space \ (count\text{-}space \ UNIV)))) \wedge$
 $N = (measure\text{-}pmf \ p \gg (\lambda y. \ distr \ (N' \ y) \ (stream\text{-}space \ (count\text{-}space \ UNIV))$
 $((\#\#) \ y))) \wedge$
 $M = (measure\text{-}pmf \ p \gg (\lambda y. \ distr \ (M' \ y) \ (stream\text{-}space \ (count\text{-}space \ UNIV))$
 $((\#\#) \ y)))$
shows $N = M$
<proof>

3.1 Discrete Markov Kernel

locale *MC-syntax* =
fixes $K :: 's \Rightarrow 's \ pmf$
begin

abbreviation *acc* :: $('s \times 's) \ set \ where$
 $acc \equiv (SIGMA \ s:UNIV. \ K \ s)^*$

abbreviation *acc-on* :: $'s \ set \Rightarrow ('s \times 's) \ set \ where$
 $acc\text{-}on \ S \equiv (SIGMA \ s:UNIV. \ K \ s \cap S)^*$

lemma *countable-reachable*: $countable \ (acc \ \{\ s \})$
<proof>

lemma *countable-acc*: $countable \ X \implies countable \ (acc \ \{\ X \})$
<proof>

context
notes [[*inductive-internals*]]
begin

coinductive *enabled* **where**
 $enabled \ (shd \ \omega) \ (stl \ \omega) \implies shd \ \omega \in K \ s \implies enabled \ s \ \omega$

end

lemma *alw-enabled*: $enabled (shd \omega) (stl \omega) \implies alw (\lambda\omega. enabled (shd \omega) (stl \omega)) \omega$
<proof>

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

lemma *in-S* [*measurable (raw)*]: $x \in space S$
<proof>

inductive-simps *enabled-iff*: $enabled s \omega$

lemma *enabled-Stream*: $enabled x (y \#\#\omega) \longleftrightarrow y \in K x \wedge enabled y \omega$
<proof>

lemma *measurable-enabled*[*measurable*]:
 $Measurable.pred (stream-space (count-space UNIV)) (enabled s) (is Measurable.pred ?S -)$
<proof>

lemma *enabled-iff-snth*: $enabled s \omega \longleftrightarrow (\forall i. \omega !! i \in K ((s \#\#\omega) !! i))$
<proof>

primcorec *force-enabled where*

$force-enabled x \omega =$
 $(let y = if shd \omega \in K x then shd \omega else (SOME y. y \in K x) in y \#\# force-enabled y (stl \omega))$

lemma *force-enabled-in-set-pmf*[*simp, intro*]: $shd (force-enabled x \omega) \in K x$
<proof>

lemma *enabled-force-enabled*: $enabled x (force-enabled x \omega)$
<proof>

lemma *force-enabled*: $enabled x \omega \implies force-enabled x \omega = \omega$
<proof>

lemma *Ex-enabled*: $\exists \omega. enabled x \omega$
<proof>

lemma *measurable-force-enabled*: $force-enabled x \in measurable S S$
<proof>

abbreviation $D \equiv stream-space (\Pi_M s \in UNIV. K s)$

lemma *sets-D*: $sets D = sets (stream-space (\Pi_M s \in UNIV. count-space UNIV))$
<proof>

lemma *space-D*: $space\ D = space\ (stream-space\ (\prod_M\ s \in UNIV.\ count-space\ UNIV))$
 ⟨proof⟩

lemma *measurable-D-D*: $measurable\ D\ D =$
 $measurable\ (stream-space\ (\prod_M\ s \in UNIV.\ count-space\ UNIV))\ (stream-space\$
 $(\prod_M\ s \in UNIV.\ count-space\ UNIV))$
 ⟨proof⟩

primcorec *walk* :: $'s \Rightarrow ('s \Rightarrow 's)\ stream \Rightarrow 's\ stream$ **where**
 $shd\ (walk\ s\ \omega) = (if\ shd\ \omega\ s \in K\ s\ then\ shd\ \omega\ s\ else\ (SOME\ t.\ t \in K\ s))$
 $| stl\ (walk\ s\ \omega) = walk\ (if\ shd\ \omega\ s \in K\ s\ then\ shd\ \omega\ s\ else\ (SOME\ t.\ t \in K\ s))$
 $(stl\ \omega)$

lemma *enabled-walk*: $enabled\ s\ (walk\ s\ \omega)$
 ⟨proof⟩

lemma *measurable-walk[measurable]*: $walk\ s \in measurable\ D\ S$
 ⟨proof⟩

3.2 Trace Space for Discrete-Time Markov Chains

definition *T* :: $'s \Rightarrow 's\ stream\ measure$ **where**
 $T\ s = distr\ (stream-space\ (\prod_M\ s \in UNIV.\ K\ s))\ S\ (walk\ s)$

lemma *space-T[simp]*: $space\ (T\ s) = space\ S$
 ⟨proof⟩

lemma *sets-T[simp, measurable-cong]*: $sets\ (T\ s) = sets\ S$
 ⟨proof⟩

lemma *measurable-T1[simp]*: $measurable\ (T\ s)\ M = measurable\ S\ M$
 ⟨proof⟩

lemma *measurable-T2[simp]*: $measurable\ M\ (T\ s) = measurable\ M\ S$
 ⟨proof⟩

lemma *in-measurable-T1[measurable (raw)]*: $f \in measurable\ S\ M \implies f \in measurable\ (T\ s)\ M$
 ⟨proof⟩

lemma *in-measurable-T2[measurable (raw)]*: $f \in measurable\ M\ S \implies f \in measurable\ M\ (T\ s)$
 ⟨proof⟩

lemma *AE-T-enabled*: $AE\ \omega\ in\ T\ s.\ enabled\ s\ \omega$
 ⟨proof⟩

sublocale *T*: $prob-space\ T\ s\ for\ s$

<proof>

lemma *emeasure-T-const[simp]*: $\text{emeasure } (T \ s) \ (\text{space } S) = 1$
<proof>

lemma *nn-integral-T*:

assumes *f[measurable]*: $f \in \text{borel-measurable } S$

shows $(\int^{+X}. f \ X \ \partial T \ s) = (\int^{+t}. (\int^{+\omega}. f \ (t \ \#\# \ \omega) \ \partial T \ t) \ \partial K \ s)$

<proof>

lemma *nn-integral-T-gfp*:

fixes g

defines $l \equiv \lambda f \ \omega. g \ (\text{shd } \omega) \ (f \ (\text{stl } \omega))$

assumes *[measurable]*: $\text{case-prod } g \in \text{borel-measurable } (\text{count-space } UNIV \otimes_M \text{borel})$

assumes *cont-g[THEN inf-continuous-compose, order-continuous-intros]*: $\bigwedge s. \text{inf-continuous } (g \ s)$

assumes *int-g*: $\bigwedge f \ s. f \in \text{borel-measurable } S \implies (\int^{+\omega}. g \ s \ (f \ \omega) \ \partial T \ s) = g \ s$
 $(\int^{+\omega}. f \ \omega \ \partial T \ s)$

assumes *bnd-g*: $\bigwedge f \ s. g \ s \ f \leq b \ 0 \leq b \ b < \infty$

shows $(\int^{+\omega}. \text{gfp } l \ \omega \ \partial T \ s) = \text{gfp } (\lambda f \ s. \int^{+t}. g \ t \ (f \ t) \ \partial K \ s) \ s$

<proof>

lemma *nn-integral-T-lfp*:

fixes g

defines $l \equiv \lambda f \ \omega. g \ (\text{shd } \omega) \ (f \ (\text{stl } \omega))$

assumes *[measurable]*: $\text{case-prod } g \in \text{borel-measurable } (\text{count-space } UNIV \otimes_M \text{borel})$

assumes *cont-g[THEN sup-continuous-compose, order-continuous-intros]*: $\bigwedge s. \text{sup-continuous } (g \ s)$

assumes *int-g*: $\bigwedge f \ s. f \in \text{borel-measurable } S \implies (\int^{+\omega}. g \ s \ (f \ \omega) \ \partial T \ s) = g \ s$
 $(\int^{+\omega}. f \ \omega \ \partial T \ s)$

shows $(\int^{+\omega}. \text{lfp } l \ \omega \ \partial T \ s) = \text{lfp } (\lambda f \ s. \int^{+t}. g \ t \ (f \ t) \ \partial K \ s) \ s$

<proof>

lemma *emeasure-Collect-T*:

assumes *f[measurable]*: $\text{Measurable.pred } S \ P$

shows $\text{emeasure } (T \ s) \ \{x \in \text{space } (T \ s). P \ x\} = (\int^{+t}. \text{emeasure } (T \ t) \ \{x \in \text{space } (T \ t). P \ (t \ \#\# \ x)\} \ \partial K \ s)$

<proof>

lemma *AE-T-iff*:

assumes *[measurable]*: $\text{Measurable.pred } S \ P$

shows $(AE \ \omega \ \text{in } T \ x. P \ \omega) \longleftrightarrow (\forall y \in K \ x. AE \ \omega \ \text{in } T \ y. P \ (y \ \#\# \ \omega))$

<proof>

lemma *AE-T-alm*:

assumes *[measurable]*: $\text{Measurable.pred } S \ P$

assumes P : $\bigwedge s. (x, s) \in \text{acc} \implies AE \ \omega \ \text{in } T \ s. P \ \omega$

shows $AE \ \omega \text{ in } T \ x. \ alw \ P \ \omega$
 $\langle proof \rangle$

lemma *emeasure-suntil-disj*:

assumes $[measurable]: \text{Measurable.pred } S \ P$
assumes $*$: $\bigwedge t. AE \ \omega \text{ in } T \ t. \neg (P \sqcap (HLD \ X \sqcap \text{next } (HLD \ X \text{ until } P))) \ \omega$
shows $emeasure \ (T \ s) \ \{\omega \in space \ (T \ s). \ (HLD \ X \text{ until } P) \ \omega\} =$
 $\text{lfp } (\lambda F \ s. \ emeasure \ (T \ s) \ \{\omega \in space \ (T \ s). \ P \ \omega\} + (\int^+ t. F \ t \ * \ indicator \ X \ t$
 $\partial K \ s)) \ s$
 $\langle proof \rangle$

lemma *emeasure-HLD-next*:

assumes $[measurable]: \text{Measurable.pred } S \ P$
shows $emeasure \ (T \ s) \ \{\omega \in space \ (T \ s). \ (X \cdot P) \ \omega\} =$
 $(\int^+ x. \ emeasure \ (T \ x) \ \{\omega \in space \ (T \ x). \ P \ \omega\} \ * \ indicator \ X \ x \ \partial K \ s)$
 $\langle proof \rangle$

lemma *emeasure-HLD*:

$emeasure \ (T \ s) \ \{\omega \in space \ (T \ s). \ HLD \ X \ \omega\} = emeasure \ (K \ s) \ X$
 $\langle proof \rangle$

lemma *emeasure-suntil-HLD*:

assumes $[measurable]: \text{Measurable.pred } S \ P$
shows $emeasure \ (T \ s) \ \{x \in space \ (T \ s). \ (\text{not } (HLD \ \{t\}) \text{ until } (HLD \ \{t\} \ \text{aand} \ \text{next } P)) \ x\} =$
 $emeasure \ (T \ s) \ \{x \in space \ (T \ s). \ ev \ (HLD \ \{t\}) \ x\} \ * \ emeasure \ (T \ t) \ \{x \in space \ (T$
 $t). \ P \ x\}$
 $\langle proof \rangle$

lemma *AE-suntil*:

assumes $[measurable]: \text{Measurable.pred } S \ P$
shows $(AE \ x \ \text{in } T \ s. \ (\text{not } (HLD \ \{t\}) \text{ until } (HLD \ \{t\} \ \text{aand} \ \text{next } P)) \ x) \iff$
 $(AE \ x \ \text{in } T \ s. \ ev \ (HLD \ \{t\}) \ x) \wedge (AE \ x \ \text{in } T \ t. \ P \ x)$
 $\langle proof \rangle$

3.3 Fairness

definition *fair* :: $'s \Rightarrow 's \Rightarrow 's \ \text{stream} \Rightarrow \text{bool}$ **where**

$fair \ s \ t = alw \ (ev \ (HLD \ \{s\})) \ \text{impl} \ alw \ (ev \ (HLD \ \{s\} \ \text{aand} \ \text{next} \ (HLD \ \{t\})))$

lemma *AE-T-fair*:

assumes $t' \in K \ t$
shows $AE \ \omega \ \text{in } T \ s. \ fair \ t \ t' \ \omega$
 $\langle proof \rangle$

lemma *enabled-imp-trancl*:

assumes $alw \ (HLD \ B) \ \omega \ \text{enabled} \ s \ \omega$
shows $alw \ (HLD \ (\text{acc-on } B \ \text{“ } \{s\} \ \text{”})) \ \omega$
 $\langle proof \rangle$

lemma *AE-T-reachable*: $AE \omega$ in $T s$. $alw (HLD (acc \text{ “ } \{s\})) \omega$
 $\langle proof \rangle$

lemma *AE-T-all-fair*: $AE \omega$ in $T s$. $\forall (t,t') \in SIGMA t:UNIV$. $K t$. $fair t t' \omega$
 $\langle proof \rangle$

lemma *fair-imp*: **assumes** $fair t t' \omega$ $alw (ev (HLD \{t\})) \omega$ **shows** $alw (ev (HLD \{t'\})) \omega$
 $\langle proof \rangle$

lemma *AE-T-ev-HLD*:
assumes *exiting*: $\bigwedge t. (s, t) \in acc-on (-B) \implies \exists t' \in B. (t, t') \in acc$
assumes *fin*: $finite (acc-on (-B) \text{ “ } \{s\})$
shows $AE \omega$ in $T s$. $ev (HLD B) \omega$
 $\langle proof \rangle$

lemma *AE-T-ev-HLD'*:
assumes *exiting*: $\bigwedge s. s \notin X \implies \exists t \in X. (s, t) \in acc$
assumes *fin*: $finite (-X)$
shows $AE \omega$ in $T s$. $ev (HLD X) \omega$
 $\langle proof \rangle$

lemma *AE-T-max-sfirst*:
assumes [*measurable*]: $Measurable.pred S X$
assumes *AE*: $AE \omega$ in $T c$. $sfirst X (c \#\#\omega) < \infty$ **and** $0 < e$
shows $\exists N::nat$. $\mathcal{P}(\omega$ in $T c$. $N < sfirst X (c \#\#\omega)) < e$ (**is** $\exists N$. $?P N < e$)
 $\langle proof \rangle$

3.4 First Hitting Time

lemma *nn-integral-sfirst-finite'*:
assumes $s \notin H$
assumes [*simp*]: $finite (acc-on (-H) \text{ “ } \{s\})$
assumes *until*: $AE \omega$ in $T s$. $ev (HLD H) \omega$
shows $(\int^+ \omega. sfirst (HLD H) \omega \partial T s) \neq \infty$
 $\langle proof \rangle$

lemma *nn-integral-sfirst-finite*:
assumes [*simp*]: $finite (acc-on (-H) \text{ “ } \{s\})$
assumes *until*: $AE \omega$ in $T s$. $ev (HLD H) \omega$
shows $(\int^+ \omega. sfirst (HLD H) (s \#\#\omega) \partial T s) \neq \infty$
 $\langle proof \rangle$

lemma *prob-T*:
assumes P : $Measurable.pred S P$
shows $\mathcal{P}(\omega$ in $T s$. $P \omega) = (\int t. \mathcal{P}(\omega$ in $T t$. $P (t \#\#\omega)) \partial K s)$
 $\langle proof \rangle$

lemma *T-subprob[measurable]*: $T \in \text{measurable (measure-pmf } I) \text{ (subprob-algebra } S)$

<proof>

3.5 Markov chain with Initial Distribution

definition $T' :: 's \text{ pmf} \Rightarrow 's \text{ stream measure}$ **where**

$T' I = \text{bind } I (\lambda s. \text{distr } (T s) S ((\#\#) s))$

lemma *distr-Stream-subprob*:

$(\lambda s. \text{distr } (T s) S ((\#\#) s)) \in \text{measurable (measure-pmf } I) \text{ (subprob-algebra } S)$

<proof>

lemma *sets-T'*: $\text{sets } (T' I) = \text{sets } S$

<proof>

lemma *prob-space-T'*: $\text{prob-space } (T' I)$

<proof>

lemma *AE-T'*:

assumes *[measurable]*: $\text{Measurable.pred } S P$

shows $(AE x \text{ in } T' I. P x) \longleftrightarrow (\forall s \in I. AE x \text{ in } T s. P (s \#\# x))$

<proof>

lemma *emeasure-T'*:

assumes *[measurable]*: $X \in \text{sets } S$

shows $\text{emeasure } (T' I) X = (\int^+ s. \text{emeasure } (T s) \{\omega \in \text{space } S. s \#\# \omega \in X\} \partial I)$

<proof>

lemma *prob-T'*:

assumes *[measurable]*: $\text{Measurable.pred } S P$

shows $\mathcal{P}(x \text{ in } T' I. P x) = (\int s. \mathcal{P}(x \text{ in } T s. P (s \#\# x)) \partial I)$

<proof>

lemma *T-eq-T'*: $T s = T' (K s)$

<proof>

lemma *T-eq-bind*: $T s = (\text{measure-pmf } (K s) \gg (\lambda t. \text{distr } (T t) S ((\#\#) t)))$

<proof>

lemma *T-split*:

$T s = (T s \gg (\lambda \omega. \text{distr } (T ((s \#\# \omega) !! n)) S (\lambda \omega'. \text{stake } n \omega @- \omega')))$

<proof>

lemma *nn-integral-T-split*:

assumes *f[measurable]*: $f \in \text{borel-measurable } S$

shows $(\int^+ \omega. f \omega \partial T s) = (\int^+ \omega. (\int^+ \omega'. f (\text{stake } n \omega @- \omega') \partial T ((s \#\# \omega) !! n)) \partial T s)$

<proof>

lemma *emeasure-T-split:*

assumes $P[\text{measurable}]$: *Measurable.pred S P*

shows $\text{emeasure } (T \ s) \ \{\omega \in \text{space } (T \ s). \ P \ \omega\} =$

$(\int^+ \omega. \ \text{emeasure } (T \ ((s \ \#\# \ \omega) \ !! \ n)) \ \{\omega' \in \text{space } (T \ ((s \ \#\# \ \omega) \ !! \ n)). \ P \ (\text{stake } n \ \omega \ @- \ \omega')\} \ \partial T \ s)$

<proof>

lemma *prob-T-split:*

assumes $P[\text{measurable}]$: *Measurable.pred S P*

shows $\mathcal{P}(\omega \ \text{in } T \ s. \ P \ \omega) = (\int \omega. \ \mathcal{P}(\omega' \ \text{in } T \ ((s \ \#\# \ \omega) \ !! \ n)). \ P \ (\text{stake } n \ \omega \ @- \ \omega')) \ \partial T \ s)$

<proof>

lemma *enabled-imp-alw:*

$(\bigcup s \in X. \ \text{set-pmf } (K \ s)) \subseteq X \implies x \in X \implies \text{enabled } x \ \omega \implies \text{alw } (\text{HLD } X) \ \omega$

<proof>

lemma *alw-HLD-iff-sconst:*

$\text{alw } (\text{HLD } \{x\}) \ \omega \longleftrightarrow \omega = \text{sconst } x$

<proof>

lemma *enabled-iff-sconst:*

assumes $[\text{simp}]$: $\text{set-pmf } (K \ x) = \{x\}$ **shows** $\text{enabled } x \ \omega \longleftrightarrow \omega = \text{sconst } x$

<proof>

lemma *AE-sconst:*

assumes $[\text{simp}]$: $\text{set-pmf } (K \ x) = \{x\}$

shows $(\text{AE } \omega \ \text{in } T \ x. \ P \ \omega) \longleftrightarrow P \ (\text{sconst } x)$

<proof>

lemma *ev-eq-lfp:* $\text{ev } P = \text{lfp } (\lambda F \ \omega. \ P \ \omega \vee (\neg P \ \omega \wedge F \ (\text{stl } \omega)))$

<proof>

lemma *INF-eq-zero-iff-ennreal:* $((\prod i \in A. \ f \ i) = (0 :: \text{ennreal})) = (\forall x > 0. \ \exists i \in A. \ f \ i < x)$

<proof>

lemma *inf-continuous-cmul:*

fixes $c :: \text{ennreal}$

assumes f : *inf-continuous f* **and** c : $c < \top$

shows *inf-continuous* $(\lambda x. \ c * f \ x)$

<proof>

lemma *AE-T-ev-HLD-infinite:*

fixes $X :: 's \ \text{set}$ **and** $r :: \text{real}$

assumes $r < 1$

assumes r : $\bigwedge x. \ x \in X \implies \text{measure } (K \ x) \ X \leq r$

shows $AE \omega$ in $T x$. $ev (HLD (- X)) \omega$
 ⟨proof⟩

3.6 Trace space with Restriction

definition $rT x = restrict\text{-}space (T x) \{\omega. enabled\ x\ \omega\}$

lemma $space\text{-}rT: \omega \in space (rT x) \longleftrightarrow enabled\ x\ \omega$
 ⟨proof⟩

lemma $Collect\text{-}enabled\text{-}S[measurable]: Collect (enabled\ x) \in sets\ S$
 ⟨proof⟩

lemma $space\text{-}rT\text{-}in\text{-}S: space (rT x) \in sets\ S$
 ⟨proof⟩

lemma $sets\text{-}rT: A \in sets (rT x) \longleftrightarrow A \in sets\ S \wedge A \subseteq \{\omega. enabled\ x\ \omega\}$
 ⟨proof⟩

lemma $prob\text{-}space\text{-}rT: prob\text{-}space (rT x)$
 ⟨proof⟩

lemma $measurable\text{-}force\text{-}enabled2[measurable]: force\text{-}enabled\ x \in measurable\ S (rT\ x)$
 ⟨proof⟩

lemma $space\text{-}rT\text{-}not\text{-}empty[simp]: space (rT x) \neq \{\}$
 ⟨proof⟩

lemma $T\text{-}eq\text{-}bind': T x = do \{ y \leftarrow measure\text{-}pmf (K x) ; \omega \leftarrow T y ; return\ S (y\ \#\#\ \omega) \}$
 ⟨proof⟩

lemma $rT\text{-}eq\text{-}bind: rT x = do \{ y \leftarrow measure\text{-}pmf (K x) ; \omega \leftarrow rT y ; return (rT\ x)\ (y\ \#\#\ \omega) \}$
 ⟨proof⟩

lemma $snth\text{-}rT: (\lambda x. x !! n) \in measurable (rT x) (count\text{-}space (acc \text{“} \{x\}))$
 ⟨proof⟩

3.7 Bisimulation

lemma $T\text{-}coinduct[consumes\ 1, case\text{-}names\ prob\ sets\ cont]:$

assumes $R\ x\ M$

assumes $prob: \bigwedge x\ M. R\ x\ M \implies prob\text{-}space\ M$

and $sets: \bigwedge x\ M. R\ x\ M \implies sets\ M = sets\ S$

and $cont': \bigwedge x\ M. R\ x\ M \implies \exists M'. (\forall y \in K\ x. R\ y (M' y)) \wedge (\forall y. sets (M' y) = S \wedge prob\text{-}space (M' y)) \wedge$

$M = (measure\text{-}pmf (K x) \gg (\lambda y. distr (M' y) S ((\#\#\ y))))$

shows $T\ x = M$

<proof>

lemma *T-bisim*:

assumes $M: \bigwedge x. \text{prob-space } (M x) \wedge \bigwedge x. \text{sets } (M x) = \text{sets } S$

and $M\text{-eq}: \bigwedge x. M x = (\text{measure-pmf } (K x) \gg (\lambda s. \text{distr } (M s) S ((\#\#) s)))$

shows $T = M$

<proof>

lemma *T-subprob'[measurable]*: $T \in \text{measurable } (\text{count-space } UNIV) (\text{subprob-algebra } S)$

<proof>

lemma *T-subprob''[simp]*: $T a \in \text{space } (\text{subprob-algebra } S)$

<proof>

lemma *AE-not-suntil-coinduct* [*consumes 1, case-names $\psi \varphi$*]:

assumes $P s$

assumes $\psi: \bigwedge s. P s \implies s \notin \psi$

assumes $\varphi: \bigwedge s t. P s \implies s \in \varphi \implies t \in K s \implies P t$

shows $AE \omega \text{ in } T s. \text{not } (\text{HLD } \varphi \text{ until HLD } \psi) (s \#\# \omega)$

<proof>

lemma *AE-not-suntil-coinduct-strong* [*consumes 1, case-names $\psi \varphi$*]:

assumes $P s$

assumes $P\text{-}\psi: \bigwedge s. P s \implies s \notin \psi$

assumes $P\text{-}\varphi: \bigwedge s t. P s \implies s \in \varphi \implies t \in K s \implies P t \vee$

$(AE \omega \text{ in } T t. \text{not } (\text{HLD } \varphi \text{ until HLD } \psi) (t \#\# \omega))$

shows $AE \omega \text{ in } T s. \text{not } (\text{HLD } \varphi \text{ until HLD } \psi) (s \#\# \omega) (\text{is ?nuntil } s)$

<proof>

end

3.8 Reward Structure on Markov Chains

locale *MC-with-rewards* = *MC-syntax* K **for** $K :: 's \Rightarrow 's \text{ pmf} +$

fixes $\iota :: 's \Rightarrow 's \Rightarrow \text{ennreal}$ **and** $\rho :: 's \Rightarrow \text{ennreal}$

assumes $\iota\text{-nonneg}: \bigwedge s t. 0 \leq \iota s t$ **and** $\rho\text{-nonneg}: \bigwedge s. 0 \leq \rho s$

assumes *measurable- ι* [*measurable*]: $(\lambda(a, b). \iota a b) \in \text{borel-measurable } (\text{count-space } UNIV \otimes_M \text{count-space } UNIV)$

begin

definition *reward-until* :: $'s \text{ set} \Rightarrow 's \Rightarrow 's \text{ stream} \Rightarrow \text{ennreal}$ **where**

reward-until $X = \text{lfp } (\lambda F s \omega. \text{if } s \in X \text{ then } 0 \text{ else } \rho s + \iota s (\text{shd } \omega) + (F (\text{shd } \omega) (\text{stl } \omega)))$

lemma *measurable- ρ* [*measurable*]: $\rho \in \text{borel-measurable } (\text{count-space } UNIV)$

<proof>

lemma *measurable-reward-until*[*measurable (raw)*]:

assumes [*measurable*]: $f \in \text{measurable } M \text{ (count-space UNIV)}$
assumes [*measurable*]: $g \in \text{measurable } M \text{ } S$
shows $(\lambda x. \text{reward-until } X \text{ (} f \text{ } x \text{) (} g \text{ } x \text{)}) \in \text{borel-measurable } M$
 <proof>

lemma *continuous-reward-until*:

$\text{sup-continuous } (\lambda F \text{ } s \text{ } \omega. \text{if } s \in X \text{ then } 0 \text{ else } \varrho \text{ } s + \iota \text{ } s \text{ (shd } \omega \text{) + (} F \text{ (shd } \omega \text{) (stl } \omega \text{))})$
 <proof>

lemma

shows *reward-until-unfold*: $\text{reward-until } X \text{ } s \text{ } \omega =$
 $(\text{if } s \in X \text{ then } 0 \text{ else } \varrho \text{ } s + \iota \text{ } s \text{ (shd } \omega \text{) + reward-until } X \text{ (shd } \omega \text{) (stl } \omega \text{)})$
 (**is** *?unfold*)
 <proof>

lemma *reward-until-simps*[*simp*]:

shows $s \in X \implies \text{reward-until } X \text{ } s \text{ } \omega = 0$
and $s \notin X \implies \text{reward-until } X \text{ } s \text{ } \omega = \varrho \text{ } s + \iota \text{ } s \text{ (shd } \omega \text{) + reward-until } X \text{ (shd } \omega \text{) (stl } \omega \text{)}$
 <proof>

lemma *reward-until-SCons*[*simp*]:

$\text{reward-until } X \text{ } s \text{ (} t \text{ \#\# } \omega \text{) = (if } s \in X \text{ then } 0 \text{ else } \varrho \text{ } s + \iota \text{ } s \text{ } t \text{ + reward-until } X \text{ } t \text{ } \omega \text{)}$
 <proof>

lemma *nn-integral-reward-until-finite*:

assumes [*simp*]: $\text{finite (acc " } \{s\} \text{) (is finite (?R " } \{s\} \text{))}$
assumes ϱ : $\bigwedge t. (s, t) \in \text{acc-on } (-H) \implies \varrho \text{ } t < \infty$
assumes ι : $\bigwedge t \text{ } t'. (s, t) \in \text{acc-on } (-H) \implies t' \in K \text{ } t \implies \iota \text{ } t \text{ } t' < \infty$
assumes *ev*: $AE \text{ } \omega \text{ in } T \text{ } s. \text{ev (HLD } H \text{) } \omega$
shows $(\int^+ \omega. \text{reward-until } H \text{ } s \text{ } \omega \text{ } \partial T \text{ } s) \neq \infty$
 <proof>

end

3.9 Bisimulation on a relation

definition *rel-set-strong* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow \text{bool}$
where $\text{rel-set-strong } R \text{ } A \text{ } B \iff (\forall x \text{ } y. R \text{ } x \text{ } y \longrightarrow (x \in A \iff y \in B))$

lemma *T-eq-rel-half*[*consumes 4, case-names prob sets cont*]:

fixes $R :: 's \Rightarrow 't \Rightarrow \text{bool}$ **and** $f :: 's \Rightarrow 't$ **and** $S :: 's \text{ set}$
assumes *R-def*: $\bigwedge s \text{ } t. R \text{ } s \text{ } t \iff (s \in S \wedge f \text{ } s = t)$
assumes *A*[*measurable*]: $A \in \text{sets (stream-space (count-space UNIV))}$
and *B*[*measurable*]: $B \in \text{sets (stream-space (count-space UNIV))}$
and *AB*: $\text{rel-set-strong (stream-all2 } R \text{) } A \text{ } B$ **and** *KL*: $\text{rel-fun } R \text{ (rel-pmf } R \text{) } K$
 L **and** $xy: R \text{ } x \text{ } y$

shows $MC\text{-syntax}.T\ K\ x\ A = MC\text{-syntax}.T\ L\ y\ B$
 $\langle proof \rangle$

3.10 Product Construction

locale $MC\text{-pair} =$

$K1: MC\text{-syntax}\ K1 + K2: MC\text{-syntax}\ K2$ **for** $K1\ K2$

begin

definition $Kp \equiv \lambda(a, b). pair\text{-pmf}\ (K1\ a)\ (K2\ b)$

sublocale $MC\text{-syntax}\ Kp\ \langle proof \rangle$

definition

$zip_E\ a\ b \equiv \lambda(\omega1, \omega2). zip\ (K1.\text{force-enabled}\ a\ \omega1)\ (K2.\text{force-enabled}\ b\ \omega2)$

lemma $zip\text{-rT}[measurable]: (\lambda(\omega1, \omega2). zip\ \omega1\ \omega2) \in measurable\ (K1.\text{rT}\ x1\ \otimes_M\ K2.\text{rT}\ x2)\ S$
 $\langle proof \rangle$

lemma $measurable\text{-zip}_E[measurable]: zip_E\ a\ b \in measurable\ (K1.S\ \otimes_M\ K2.S)\ S$
 $\langle proof \rangle$

lemma $T\text{-eq-prod}: T = (\lambda(x1, x2). do\ \{\ \omega1 \leftarrow K1.T\ x1\ ;\ \omega2 \leftarrow K2.T\ x2\ ;\ return\ S\ (zip_E\ x1\ x2\ (\omega1, \omega2))\ \})$
 $(is\ _ =\ ?B)$
 $\langle proof \rangle$

lemma $nn\text{-integral-pT}:$

fixes f **assumes** $[measurable]: f \in borel\text{-measurable}\ S$

shows $(\int^{+\omega}. f\ \omega\ \partial T\ (x, y)) = (\int^{+\omega1}. \int^{+\omega2}. f\ (zip_E\ x\ y\ (\omega1, \omega2))\ \partial K2.T\ y\ \partial K1.T\ x)$
 $\langle proof \rangle$

lemma $prod\text{-eq-prob-T}:$

assumes $[measurable]: Measurable.pred\ K1.S\ P1\ Measurable.pred\ K2.S\ P2$

shows $\mathcal{P}(\omega\ in\ K1.T\ x1.\ P1\ \omega) * \mathcal{P}(\omega\ in\ K2.T\ x2.\ P2\ \omega) =$
 $\mathcal{P}(\omega\ in\ T\ (x1, x2).\ P1\ (smap\ fst\ \omega) \wedge P2\ (smap\ snd\ \omega))$
 $\langle proof \rangle$

end

end

3.11 Trace Space equal to Markov Chains

theory $Trace\text{-Space}\text{-Equals}\text{-Markov}\text{-Processes}$

imports $Discrete\text{-Time}\text{-Markov}\text{-Chain}$

begin

We can construct for each time-homogeneous discrete-time Markov chain a corresponding probability space using *Markov-Models.Discrete-Time-Markov-Chain*. The constructed probability space has the same probabilities.

```

locale Time-Homogeneous-Discrete-Markov-Process = M?: prob-space +
  fixes S :: 's set and X :: nat ⇒ 'a ⇒ 's
  assumes X [measurable]:  $\bigwedge t. X\ t \in \text{measurable } M \text{ (count-space UNIV)}$ 
  assumes S: countable S  $\bigwedge n. \text{AE } x \text{ in } M. X\ n\ x \in S$ 
  assumes MC:  $\bigwedge n\ s\ s'. \mathcal{P}(\omega \text{ in } M. \forall t \leq n. X\ t\ \omega = s\ t) \neq 0 \implies$ 
     $\mathcal{P}(\omega \text{ in } M. X\ (Suc\ n)\ \omega = s' \mid \forall t \leq n. X\ t\ \omega = s\ t) =$ 
     $\mathcal{P}(\omega \text{ in } M. X\ (Suc\ n)\ \omega = s' \mid X\ n\ \omega = s\ n)$ 
  assumes TH:  $\bigwedge n\ m\ s\ t. \mathcal{P}(\omega \text{ in } M. X\ n\ \omega = t) \neq 0 \implies \mathcal{P}(\omega \text{ in } M. X\ m\ \omega = t) \neq 0 \implies$ 
     $\mathcal{P}(\omega \text{ in } M. X\ (Suc\ n)\ \omega = s \mid X\ n\ \omega = t) = \mathcal{P}(\omega \text{ in } M. X\ (Suc\ m)\ \omega = s \mid X$ 
     $m\ \omega = t)$ 
begin

context
begin

interpretation pmf-as-measure  $\langle \text{proof} \rangle$ 

lift-definition I :: 's pmf is distr M (count-space UNIV) (X 0)
 $\langle \text{proof} \rangle$ 

lemma I-in-S:
  assumes pmf I s  $\neq 0$  shows s  $\in S$ 
 $\langle \text{proof} \rangle$ 

lift-definition K :: 's ⇒ 's pmf is
   $\lambda s. \text{with } (\lambda n. \mathcal{P}(\omega \text{ in } M. X\ n\ \omega = s) \neq 0)$ 
   $(\lambda n. \text{distr } (\text{uniform-measure } M \{ \omega \in \text{space } M. X\ n\ \omega = s \}) \text{ (count-space UNIV)}$ 
   $(X\ (Suc\ n)))$ 
   $(\text{uniform-measure } (\text{count-space UNIV}) \{s\})$ 
 $\langle \text{proof} \rangle$ 

lemma pmf-K:
  assumes n:  $0 < \mathcal{P}(\omega \text{ in } M. X\ n\ \omega = s)$ 
  shows pmf (K s) t  $= \mathcal{P}(\omega \text{ in } M. X\ (Suc\ n)\ \omega = t \mid X\ n\ \omega = s)$ 
 $\langle \text{proof} \rangle$ 

lemma pmf-K2:
   $(\bigwedge n. \mathcal{P}(\omega \text{ in } M. X\ n\ \omega = s) = 0) \implies \text{pmf } (K\ s)\ t = \text{indicator } \{t\}\ s$ 
 $\langle \text{proof} \rangle$ 

end

sublocale K: MC-syntax K  $\langle \text{proof} \rangle$ 

```

lemma *bind-I-K-eq-M*: $K.T' I = \text{distr } M K.S (\lambda\omega. \text{to-stream } (\lambda n. X n \omega))$ (is -
 $= ?D)$
 $\langle \text{proof} \rangle$

end

lemma (in *MC-syntax*) *is-THDTMC*:
fixes $I :: 's \text{ pmf}$
defines $U \equiv (\text{SIGMA } s:\text{UNIV}. K s)^* \text{ `` } I$
shows *Time-Homogeneous-Discrete-Markov-Process* $(T' I) U (\lambda n \omega. \omega !! n)$
 $\langle \text{proof} \rangle$

end

4 Classifying Markov Chain States

theory *Classifying-Markov-Chain-States*
imports
HOL-Computational-Algebra.Group-Closure
Discrete-Time-Markov-Chain
begin

lemma *eventually-mult-Gcd*:
fixes $S :: \text{nat set}$
assumes $S: \bigwedge s t. s \in S \implies t \in S \implies s + t \in S$
assumes $s: s \in S s > 0$
shows *eventually* $(\lambda m. m * \text{Gcd } S \in S)$ *sequentially*
 $\langle \text{proof} \rangle$

context *MC-syntax*
begin

4.1 Expected number of visits

definition $G s t = (\int^+ \omega. \text{scount } (\text{HLD } \{t\}) (s \#\#\ \omega) \partial T s)$

lemma *G-eq*: $G s t = (\int^+ \omega. \text{emeasure } (\text{count-space UNIV}) \{i. (s \#\#\ \omega) !! i = t\} \partial T s)$
 $\langle \text{proof} \rangle$

definition $p s t n = \mathcal{P}(\omega \text{ in } T s. (s \#\#\ \omega) !! n = t)$

definition $gf\text{-}G s t z = (\sum n. p s t n *_{\mathbb{R}} z \hat{\ } n)$

definition *convergence-G s t z* \longleftrightarrow *summable* $(\lambda n. p s t n * \text{norm } z \hat{\ } n)$

lemma *p-nonneg[simp]*: $0 \leq p x y n$
 $\langle \text{proof} \rangle$

lemma *p-le-1*: $p\ x\ y\ n \leq 1$
<proof>

lemma *p-x-x-0[simp]*: $p\ x\ x\ 0 = 1$
<proof>

lemma *p-0*: $p\ x\ y\ 0 = (\text{if } x = y \text{ then } 1 \text{ else } 0)$
<proof>

lemma *p-in-reachable*: **assumes** $(x, y) \notin (\text{SIGMA } x:\text{UNIV}. K\ x)^*$ **shows** $p\ x\ y\ n = 0$
<proof>

lemma *p-Suc*: $\text{ennreal } (p\ x\ y\ (\text{Suc } n)) = (\int^+ w. p\ w\ y\ n\ \partial K\ x)$
<proof>

lemma *p-Suc'*:
 $p\ x\ y\ (\text{Suc } n) = (\int x'. p\ x'\ y\ n\ \partial K\ x)$
<proof>

lemma *p-add*: $p\ x\ y\ (n + m) = (\int^+ w. p\ x\ w\ n * p\ w\ y\ m\ \partial \text{count-space UNIV})$
<proof>

lemma *prob-reachable-le*:
assumes *[simp]*: $m \leq n$
shows $p\ x\ y\ m * p\ y\ w\ (n - m) \leq p\ x\ w\ n$
<proof>

lemma *G-eq-suminf*: $G\ x\ y = (\sum i. \text{ennreal } (p\ x\ y\ i))$
<proof>

lemma *G-eq-real-suminf*:
convergence-G $x\ y\ (1::\text{real}) \implies G\ x\ y = \text{ennreal } (\sum i. p\ x\ y\ i)$
<proof>

lemma *convergence-norm-G*:
convergence-G $x\ y\ z \implies \text{summable } (\lambda n. p\ x\ y\ n * \text{norm } z ^ n)$
<proof>

lemma *convergence-G*:
convergence-G $x\ y\ (z::'a::\{\text{banach}, \text{real-normed-div-algebra}\}) \implies \text{summable } (\lambda n. p\ x\ y\ n *_{\mathbb{R}} z ^ n)$
<proof>

lemma *convergence-G-less-1*:
fixes $z :: - :: \{\text{banach}, \text{real-normed-field}\}$
assumes $z: \text{norm } z < 1$ **shows** *convergence-G* $x\ y\ z$
<proof>

lemma *lim-gf-G*: $((\lambda z. \text{ennreal } (gf-G \ x \ y \ z)) \longrightarrow G \ x \ y) \text{ (at-left } (1::\text{real}))$
 ⟨proof⟩

4.2 Reachability probability

definition $u \ x \ y \ n = \mathcal{P}(\omega \text{ in } T \ x. \text{ ev-at } (HLD \ \{y\}) \ n \ \omega)$

definition $U \ s \ t = \mathcal{P}(\omega \text{ in } T \ s. \text{ ev } (HLD \ \{t\}) \ \omega)$

definition $gf-U \ x \ y \ z = (\sum n. u \ x \ y \ n *_{\mathbb{R}} z \wedge Suc \ n)$

definition $f \ x \ y \ n = \mathcal{P}(\omega \text{ in } T \ x. \text{ ev-at } (HLD \ \{y\}) \ n \ (x \ \#\# \ \omega))$

definition $F \ s \ t = \mathcal{P}(\omega \text{ in } T \ s. \text{ ev } (HLD \ \{t\}) \ (s \ \#\# \ \omega))$

definition $gf-F \ x \ y \ z = (\sum n. f \ x \ y \ n * z \wedge n)$

lemma *f-Suc*: $x \neq y \implies f \ x \ y \ (Suc \ n) = u \ x \ y \ n$
 ⟨proof⟩

lemma *f-Suc-eq*: $f \ x \ x \ (Suc \ n) = 0$
 ⟨proof⟩

lemma *f-0*: $f \ x \ y \ 0 = (\text{if } x = y \text{ then } 1 \text{ else } 0)$
 ⟨proof⟩

lemma shows *u-nonneg*: $0 \leq u \ x \ y \ n$ **and** *u-le-1*: $u \ x \ y \ n \leq 1$
 ⟨proof⟩

lemma shows *f-nonneg*: $0 \leq f \ x \ y \ n$ **and** *f-le-1*: $f \ x \ y \ n \leq 1$
 ⟨proof⟩

lemma *U-nonneg[simp]*: $0 \leq U \ x \ y$
 ⟨proof⟩

lemma *U-le-1*: $U \ s \ t \leq 1$
 ⟨proof⟩

lemma *U-cases*: $U \ s \ s = 1 \vee U \ s \ s < 1$
 ⟨proof⟩

lemma *u-sums-U*: $u \ x \ y \ \text{sums } U \ x \ y$
 ⟨proof⟩

lemma *gf-U-eq-U*: $gf-U \ x \ y \ 1 = U \ x \ y$
 ⟨proof⟩

lemma *f-sums-F*: $f \ x \ y \ \text{sums } F \ x \ y$
 ⟨proof⟩

lemma *F-nonneg[simp]*: $0 \leq F x y$
<proof>

lemma *F-le-1*: $F x y \leq 1$
<proof>

lemma *gf-F-eq-F*: $gf-F x y 1 = F x y$
<proof>

lemma *gf-F-le-1*:
fixes $z :: real$
assumes $z: 0 \leq z \leq 1$
shows $gf-F x y z \leq 1$
<proof>

lemma *u-le-p*: $u x y n \leq p x y (Suc n)$
<proof>

lemma *f-le-p*: $f x y n \leq p x y n$
<proof>

lemma *convergence-norm-U*:
fixes $z :: - :: real-normed-div-algebra$
assumes $z: convergence-G x y z$
shows $summable (\lambda n. u x y n * norm z ^ Suc n)$
<proof>

lemma *convergence-norm-F*:
fixes $z :: - :: real-normed-div-algebra$
assumes $z: convergence-G x y z$
shows $summable (\lambda n. f x y n * norm z ^ n)$
<proof>

lemma *gf-G-nonneg*:
fixes $z :: real$
shows $0 \leq z \implies z < 1 \implies 0 \leq gf-G x y z$
<proof>

lemma *gf-F-nonneg*:
fixes $z :: real$
shows $0 \leq z \implies z < 1 \implies 0 \leq gf-F x y z$
<proof>

lemma *convergence-U*:
fixes $z :: - :: banach$
shows $convergence-G x y z \implies summable (\lambda n. u x y n * z ^ Suc n)$
<proof>

lemma *p-eq-sum-p-u*: $p\ x\ y\ (Suc\ n) = (\sum\ i \leq n. p\ y\ y\ (n - i) * u\ x\ y\ i)$
 ⟨proof⟩

lemma *p-eq-sum-p-f*: $p\ x\ y\ n = (\sum\ i \leq n. p\ y\ y\ (n - i) * f\ x\ y\ i)$
 ⟨proof⟩

lemma *gf-G-eq-gf-F*:
 assumes $z: norm\ z < 1$
 shows $gf\text{-}G\ x\ y\ z = gf\text{-}F\ x\ y\ z * gf\text{-}G\ y\ y\ z$
 ⟨proof⟩

lemma *gf-G-eq-gf-U*:
 fixes $z :: 'z :: \{banach, real\text{-}normed\text{-}field\}$
 assumes $z: convergence\text{-}G\ x\ x\ z$
 shows $gf\text{-}G\ x\ x\ z = 1 / (1 - gf\text{-}U\ x\ x\ z)$ $gf\text{-}U\ x\ x\ z \neq 1$
 ⟨proof⟩

lemma *gf-U*: $(gf\text{-}U\ x\ y \longrightarrow U\ x\ y)$ (*at-left 1*)
 ⟨proof⟩

lemma *gf-U-le-1*: assumes $z: 0 < z\ z < 1$ shows $gf\text{-}U\ x\ y\ z \leq (1::real)$
 ⟨proof⟩

lemma *gf-F*: $(gf\text{-}F\ x\ y \longrightarrow F\ x\ y)$ (*at-left 1*)
 ⟨proof⟩

lemma *U-bounded*: $0 \leq U\ x\ y\ U\ x\ y \leq 1$
 ⟨proof⟩

4.3 Recurrent states

definition *recurrent* :: $'s \Rightarrow bool$ **where**
 $recurrent\ s \longleftrightarrow (AE\ \omega\ in\ T\ s. ev\ (HLD\ \{s\})\ \omega)$

lemma *recurrent-iff-U-eq-1*: $recurrent\ s \longleftrightarrow U\ s\ s = 1$
 ⟨proof⟩

definition $H\ s\ t = \mathcal{P}(\omega\ in\ T\ s. alw\ (ev\ (HLD\ \{t\}))\ \omega)$

lemma *H-eq*:
 $recurrent\ s \longleftrightarrow H\ s\ s = 1$
 $\neg recurrent\ s \longleftrightarrow H\ s\ s = 0$
 $H\ s\ t = U\ s\ t * H\ t\ t$
 ⟨proof⟩

lemma *recurrent-iff-G-infinite*: $recurrent\ x \longleftrightarrow G\ x\ x = \infty$
 ⟨proof⟩

definition *communicating* :: $('s \times 's)$ **set where**

$$\text{communicating} = \text{acc} \cap \text{acc}^{-1}$$

definition *essential-class* :: 's set \Rightarrow bool **where**

essential-class $C \longleftrightarrow C \in \text{UNIV} // \text{communicating} \wedge \text{acc} \text{ `` } C \subseteq C$

lemma *accI-U*:

assumes $0 < U \ x \ y$ **shows** $(x, y) \in \text{acc}$

<proof>

lemma *accD-pos*:

assumes $(x, y) \in \text{acc}$

shows $\exists n. 0 < p \ x \ y \ n$

<proof>

lemma *accI-pos*: $0 < p \ x \ y \ n \Longrightarrow (x, y) \in \text{acc}$

<proof>

lemma *recurrent-iffI-communicating*:

assumes $(x, y) \in \text{communicating}$

shows $\text{recurrent } x \longleftrightarrow \text{recurrent } y$

<proof>

lemma *recurrent-acc*:

assumes $\text{recurrent } x \ (x, y) \in \text{acc}$

shows $U \ y \ x = 1 \ H \ y \ x = 1 \ \text{recurrent } y \ (x, y) \in \text{communicating}$

<proof>

lemma *equiv-communicating*: *equiv UNIV communicating*

<proof>

lemma *recurrent-class*:

assumes $\text{recurrent } x$

shows $\text{acc} \text{ `` } \{x\} = \text{communicating} \text{ `` } \{x\}$

<proof>

lemma *irreducible-recurrent-class*:

assumes $\text{recurrent } x$ **shows** $\text{acc} \text{ `` } \{x\} \in \text{UNIV} // \text{communicating}$

<proof>

lemma *essential-classI*:

assumes $C: C \in \text{UNIV} // \text{communicating}$

assumes $\text{eq: } \bigwedge x \ y. x \in C \Longrightarrow (x, y) \in \text{acc} \Longrightarrow y \in C$

shows *essential-class* C

<proof>

lemma *essential-recurrent-class*:

assumes $\text{recurrent } x$ **shows** *essential-class* $(\text{communicating} \text{ `` } \{x\})$

<proof>

lemma *essential-classD2*:

essential-class $C \implies x \in C \implies (x, y) \in \text{acc} \implies y \in C$

<proof>

lemma *essential-classD3*:

essential-class $C \implies x \in C \implies y \in C \implies (x, y) \in \text{communicating}$

<proof>

lemma *AE-acc*:

shows *AE* ω in $T x$. $\forall m. (x, (x \#\# \omega) !! m) \in \text{acc}$

<proof>

lemma *finite-essential-class-imp-recurrent*:

assumes C : *essential-class* C *finite* C **and** $x: x \in C$

shows *recurrent* x

<proof>

lemma *irreducibleD*:

$C \in \text{UNIV} // \text{communicating} \implies a \in C \implies b \in C \implies (a, b) \in \text{communicating}$

<proof>

lemma *irreducibleD2*:

$C \in \text{UNIV} // \text{communicating} \implies a \in C \implies (a, b) \in \text{communicating} \implies b \in C$

<proof>

lemma *essential-class-iff-recurrent*:

finite $C \implies C \in \text{UNIV} // \text{communicating} \implies \text{essential-class } C \longleftrightarrow (\forall x \in C. \text{recurrent } x)$

<proof>

definition $U' x y = (\int^{+\omega}. e\text{Suc } (s\text{first } (HLD \{y\}) \omega) \partial T x)$

lemma *U'-neq-zero[simp]*: $U' x y \neq 0$

<proof>

definition $gf\text{-}U' x y z = (\sum n. u x y n * \text{Suc } n * z \wedge n)$

definition *pos-recurrent* $x \longleftrightarrow \text{recurrent } x \wedge U' x x \neq \infty$

lemma *summable-gf-U'*:

assumes z : *norm* $z < 1$

shows *summable* $(\lambda n. u x y n * \text{Suc } n * z \wedge n)$

<proof>

lemma *gf-U'-nonneg[simp]*: $0 < z \implies z < 1 \implies 0 \leq gf\text{-}U' x y z$

<proof>

lemma *DERIV-gf-U*:

fixes $z :: \text{real}$ **assumes** $z: 0 < z < 1$
shows $\text{DERIV } (gf-U \ x \ y) \ z :> gf-U' \ x \ y \ z$
 $\langle \text{proof} \rangle$

lemma *sfirst-finiteI-recurrent*:
 $\text{recurrent } x \implies (x, y) \in \text{acc} \implies \text{AE } \omega \text{ in } T \ x. \text{ sfirst } (\text{HLD } \{y\}) \ \omega < \infty$
 $\langle \text{proof} \rangle$

lemma *U'-eq-suminf*:
assumes $x: \text{recurrent } x \ (x, y) \in \text{acc}$
shows $U' \ x \ y = (\sum i. \text{ennreal } (u \ x \ y \ i * \text{Suc } i))$
 $\langle \text{proof} \rangle$

lemma *gf-U'-tendsto-U'*:
assumes $x: \text{recurrent } x \ (x, y) \in \text{acc}$
shows $((\lambda z. \text{ennreal } (gf-U' \ x \ y \ z)) \longrightarrow U' \ x \ y) \ (\text{at-left } 1)$
 $\langle \text{proof} \rangle$

lemma *one-le-integral-t*:
assumes $x: \text{recurrent } x$ **shows** $1 \leq U' \ x \ x$
 $\langle \text{proof} \rangle$

lemma *gf-U'-pos*:
fixes $z :: \text{real}$
assumes $z: 0 < z < 1$ **and** $U \ x \ y \neq 0$
shows $0 < gf-U' \ x \ y \ z$
 $\langle \text{proof} \rangle$

lemma *inverse-gf-U'-tendsto*:
assumes $\text{recurrent } y$
shows $((\lambda x. - 1 / - gf-U' \ y \ y \ x) \longrightarrow \text{enn2real } (1 / U' \ y \ y)) \ (\text{at-left } (1::\text{real}))$
 $\langle \text{proof} \rangle$

lemma *gf-G-pos*:
fixes $z :: \text{real}$
assumes $z: 0 < z < 1$ **and** $*: (x, y) \in \text{acc}$
shows $0 < gf-G \ x \ y \ z$
 $\langle \text{proof} \rangle$

lemma *pos-recurrentI-communicating*:
assumes $y: \text{pos-recurrent } y$ **and** $x: (y, x) \in \text{communicating}$
shows $\text{pos-recurrent } x$
 $\langle \text{proof} \rangle$

lemma *pos-recurrent-iffI-communicating*:
 $(y, x) \in \text{communicating} \implies \text{pos-recurrent } y \iff \text{pos-recurrent } x$
 $\langle \text{proof} \rangle$

lemma *U-le-F*: $U \ x \ y \leq F \ x \ y$

<proof>

lemma not-empty-irreducible: $C \in UNIV // \text{communicating} \implies C \neq \{\}$
<proof>

4.4 Stationary distribution

definition stat :: 's set \Rightarrow 's measure where
 $\text{stat } C = \text{point-measure } UNIV (\lambda x. \text{indicator } C \ x / U' \ x \ x)$

lemma sets-stat[simp]: $\text{sets } (\text{stat } C) = \text{sets } (\text{count-space } UNIV)$
<proof>

lemma space-stat[simp]: $\text{space } (\text{stat } C) = UNIV$
<proof>

lemma stat-subprob:
assumes C : essential-class C **and** countable C **and** pos: $\forall c \in C. \text{pos-recurrent } c$
shows $\text{emeasure } (\text{stat } C) \ C \leq 1$
<proof>

lemma emeasure-stat-not-C:
assumes $y \notin C$
shows $\text{emeasure } (\text{stat } C) \ \{y\} = 0$
<proof>

definition stationary-distribution :: 's pmf \Rightarrow bool where
 $\text{stationary-distribution } N \longleftrightarrow N = \text{bind-pmf } N \ K$

lemma stationary-distributionI:
assumes $le: \bigwedge y. (\int x. \text{pmf } (K \ x) \ y \ \partial \text{measure-pmf } N) \leq \text{pmf } N \ y$
shows stationary-distribution N
<proof>

lemma stationary-distribution-iterate:
assumes N : stationary-distribution N
shows $\text{ennreal } (\text{pmf } N \ y) = (\int^+ x. p \ x \ y \ n \ \partial N)$
<proof>

lemma stationary-distribution-iterate':
assumes stationary-distribution N
shows $\text{measure } N \ \{y\} = (\int x. p \ x \ y \ n \ \partial N)$
<proof>

lemma stationary-distributionD:
assumes C : essential-class C countable C
assumes N : stationary-distribution $N \subseteq C$
shows $\forall x \in C. \text{pos-recurrent } x \ \text{measure-pmf } N = \text{stat } C$
<proof>

lemma *measure-point-measure-singleton*:

$x \in A \implies \text{measure } (\text{point-measure } A \ X) \ \{x\} = \text{enn2real } (X \ x)$
(proof)

lemma *stationary-distribution-imp-int-t*:

assumes C : *essential-class* C *countable* C *stationary-distribution* $N \ N \subseteq C$
assumes x : $x \in C$ **shows** $U' \ x \ x = 1 / \text{ennreal } (\text{pmf } N \ x)$
(proof)

definition *period-set* $x = \{i. \ 0 < i \wedge 0 < p \ x \ x \ i\}$

definition *period* $C = (\text{SOME } d. \ \forall x \in C. \ d = \text{Gcd } (\text{period-set } x))$

lemma *Gcd-period-set-invariant*:

assumes c : $(x, y) \in \text{communicating}$
shows $\text{Gcd } (\text{period-set } x) = \text{Gcd } (\text{period-set } y)$
(proof)

lemma *period-eq*:

assumes $C \in \text{UNIV} // \text{communicating}$ $x \in C$
shows $\text{period } C = \text{Gcd } (\text{period-set } x)$
(proof)

definition *aperiodic* $C \longleftrightarrow C \in \text{UNIV} // \text{communicating} \wedge \text{period } C = 1$

definition *not-ephemeral* $C \longleftrightarrow C \in \text{UNIV} // \text{communicating} \wedge \neg (\exists x. \ C = \{x\} \wedge p \ x \ x \ 1 = 0)$

lemma *not-ephemeralD*:

assumes C : *not-ephemeral* C $x \in C$
shows $\exists n > 0. \ 0 < p \ x \ x \ n$
(proof)

lemma *not-ephemeralD-pos-period*:

assumes C : *not-ephemeral* C
shows $0 < \text{period } C$
(proof)

lemma *period-posD*:

assumes C : $C \in \text{UNIV} // \text{communicating}$ **and** $0 < \text{period } C$ $x \in C$
shows $\exists n > 0. \ 0 < p \ x \ x \ n$
(proof)

lemma *not-ephemeralD-pos-period'*:

assumes C : $C \in \text{UNIV} // \text{communicating}$
shows *not-ephemeral* $C \longleftrightarrow 0 < \text{period } C$
(proof)

lemma eventually-periodic:
assumes $C: C \in UNIV // \text{communicating } 0 < \text{period } C$ $x \in C$
shows *eventually* $(\lambda m. 0 < p \ x \ x \ (m * \text{period } C))$ *sequentially*
 $\langle \text{proof} \rangle$

lemma aperiodic-eventually-recurrent:
 $\text{aperiodic } C \longleftrightarrow C \in UNIV // \text{communicating} \wedge (\forall x \in C. \text{eventually } (\lambda m. 0 < p \ x \ x \ m))$ *sequentially*
 $\langle \text{proof} \rangle$

lemma stationary-distributionD-emeasure:
assumes $N: \text{stationary-distribution } N$
shows $\text{emeasure } N \ A = (\int^+ s. \text{emeasure } (K \ s) \ A \ \partial N)$
 $\langle \text{proof} \rangle$

lemma communicatingD1:
 $C \in UNIV // \text{communicating} \implies (a, b) \in \text{communicating} \implies a \in C \implies b \in C$
 $\langle \text{proof} \rangle$

lemma communicatingD2:
 $C \in UNIV // \text{communicating} \implies (a, b) \in \text{communicating} \implies b \in C \implies a \in C$
 $\langle \text{proof} \rangle$

lemma acc-iff: $(x, y) \in \text{acc} \longleftrightarrow (\exists n. 0 < p \ x \ y \ n)$
 $\langle \text{proof} \rangle$

lemma communicating-iff: $(x, y) \in \text{communicating} \longleftrightarrow (\exists n. 0 < p \ x \ y \ n) \wedge (\exists n. 0 < p \ y \ x \ n)$
 $\langle \text{proof} \rangle$

end

context $MC\text{-pair}$
begin

lemma p-eq-p1-p2:
 $p \ (x1, x2) \ (y1, y2) \ n = K1.p \ x1 \ y1 \ n * K2.p \ x2 \ y2 \ n$
 $\langle \text{proof} \rangle$

lemma P-accD:
assumes $((x1, x2), (y1, y2)) \in \text{acc}$ **shows** $(x1, y1) \in K1.\text{acc} \ (x2, y2) \in K2.\text{acc}$
 $\langle \text{proof} \rangle$

lemma aperiodicI-pair:
assumes $C1: K1.\text{aperiodic } C1$ **and** $C2: K2.\text{aperiodic } C2$

```

shows aperiodic ( $C1 \times C2$ )
  ⟨proof⟩

lemma stationary-distributionI-pair:
  assumes  $N1: K1.stationary-distribution\ N1$ 
  assumes  $N2: K2.stationary-distribution\ N2$ 
  shows stationary-distribution (pair-pmf  $N1\ N2$ )
  ⟨proof⟩

end

context MC-syntax
begin

lemma stationary-distribution-imp-limit:
  assumes  $C: aperiodic\ C\ essential-class\ C\ countable\ C$  and  $N: stationary-distribution\ N\ N \subseteq C$ 
  assumes [simp]:  $y \in C$ 
  shows  $(\lambda n. \int x. |p\ y\ x\ n - pmf\ N\ x| \partial count-space\ C) \longrightarrow 0$ 
  (is ? $L \longrightarrow 0$ )
  ⟨proof⟩

lemma stationary-distribution-imp-p-limit:
  assumes  $aperiodic\ C\ essential-class\ C$  and [simp]: countable  $C$ 
  assumes  $N: stationary-distribution\ N\ N \subseteq C$ 
  assumes [simp]:  $x \in C\ y \in C$ 
  shows  $p\ x\ y \longrightarrow pmf\ N\ y$ 
  ⟨proof⟩

end

lemma (in MC-syntax) essential-classI2:
  assumes  $X \neq \{\}$ 
  assumes  $accI: \bigwedge x\ y. x \in X \implies y \in X \implies (x, y) \in acc$ 
  assumes  $ED: \bigwedge x\ y. x \in X \implies y \in set-pmf\ (K\ x) \implies y \in X$ 
  shows essential-class  $X$ 
  ⟨proof⟩

end

## 5 Markov Decision Processes

theory Markov-Decision-Process
  imports Discrete-Time-Markov-Chain
begin

definition some-elem  $s = (SOME\ x. x \in s)$ 

lemma some-elem-ne:  $s \neq \{\} \implies some-elem\ s \in s$ 

```

<proof>

5.1 Configurations

We want to construct a *non-free* codatatype $'s\ cfg = Cfg\ (state: 's)\ (action: 's\ pmf)\ (cont: 's \Rightarrow 's\ cfg)$. with the restriction $state\ (cont\ cfg\ s) = s$

hide-const $cont$

codatatype $'s\ scheduler = Scheduler\ (action-sch: 's\ pmf)\ (cont-sch: 's \Rightarrow 's\ scheduler)$

lemma $equivp\ rel\ prod: equivp\ R \Longrightarrow equivp\ Q \Longrightarrow equivp\ (rel\ prod\ R\ Q)$

<proof>

coinductive $eq\ scheduler :: 's\ scheduler \Rightarrow 's\ scheduler \Rightarrow bool$

where

$\bigwedge D. action\ sch\ sc1 = D \Longrightarrow action\ sch\ sc2 = D \Longrightarrow$
 $(\forall s \in D. eq\ scheduler\ (cont\ sch\ sc1\ s)\ (cont\ sch\ sc2\ s)) \Longrightarrow eq\ scheduler\ sc1\ sc2$

lemma $eq\ scheduler\ refl[intro]: eq\ scheduler\ sc\ sc$

<proof>

quotient-type $'s\ cfg = 's \times 's\ scheduler / rel\ prod\ (=)\ eq\ scheduler$

<proof>

lift-definition $state :: 's\ cfg \Rightarrow 's\ is\ fst$

<proof>

lift-definition $action :: 's\ cfg \Rightarrow 's\ pmf\ is\ \lambda(s, sc). action\ sch\ sc$

<proof>

lift-definition $cont :: 's\ cfg \Rightarrow 's \Rightarrow 's\ cfg\ is$

$\lambda(s, sc)\ t. if\ t \in action\ sch\ sc\ then\ (t, cont\ sch\ sc\ t)\ else$
 $(t, cont\ sch\ sc\ (some\ elem\ (action\ sch\ sc)))$

<proof>

lift-definition $Cfg :: 's \Rightarrow 's\ pmf \Rightarrow ('s \Rightarrow 's\ cfg) \Rightarrow 's\ cfg\ is$

$\lambda s\ D\ c. (s, Scheduler\ D\ (\lambda t. snd\ (c\ t)))$

<proof>

lift-definition $cfg\ corec :: 's \Rightarrow ('a \Rightarrow 's\ pmf) \Rightarrow ('a \Rightarrow 's \Rightarrow 'a) \Rightarrow 'a \Rightarrow 's\ cfg$
is

$\lambda s\ D\ C\ x. (s, corec\ scheduler\ D\ (\lambda x\ s. Inr\ (C\ x\ s))\ x)$ *<proof>*

lemma $state\ cont[simp]: state\ (cont\ cfg\ s) = s$

<proof>

lemma $state\ Cfg[simp]: state\ (Cfg\ s\ d'\ c') = s$

<proof>

lemma *action-Cfg[simp]*: $\text{action } (\text{Cfg } s \ d' \ c') = d'$

<proof>

lemma *cont-Cfg[simp]*: $t \in \text{set-pmf } d' \implies \text{state } (c' \ t) = t \implies \text{cont } (\text{Cfg } s \ d' \ c')$

$t = c' \ t$
<proof>

lemma *state-cfg-corec[simp]*: $\text{state } (\text{cfg-corec } s \ d \ c \ x) = s$

<proof>

lemma *action-cfg-corec[simp]*: $\text{action } (\text{cfg-corec } s \ d \ c \ x) = d \ x$

<proof>

lemma *cont-cfg-corec[simp]*: $t \in \text{set-pmf } (d \ x) \implies \text{cont } (\text{cfg-corec } s \ d \ c \ x) \ t = \text{cfg-corec } t \ d \ c \ (c \ x \ t)$

<proof>

lemma *cfg-coinduct[consumes 1, case-names state action cont, coinduct pred]*:

$X \ c \ d \implies (\bigwedge c \ d. X \ c \ d \implies \text{state } c = \text{state } d) \implies (\bigwedge c \ d. X \ c \ d \implies \text{action } c = \text{action } d) \implies$

$(\bigwedge c \ d \ t. X \ c \ d \implies t \in \text{set-pmf } (\text{action } c) \implies X \ (\text{cont } c \ t) \ (\text{cont } d \ t)) \implies c = d$

<proof>

coinductive *rel-cfg* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \ \text{cfg} \Rightarrow 'b \ \text{cfg} \Rightarrow \text{bool}$ **for** $P :: 'a \Rightarrow 'b \Rightarrow \text{bool}$

where

$P \ (\text{state } \text{cfg1}) \ (\text{state } \text{cfg2}) \implies$

$\text{rel-pmf } (\lambda s \ t. \text{rel-cfg } P \ (\text{cont } \text{cfg1 } s) \ (\text{cont } \text{cfg2 } t)) \ (\text{action } \text{cfg1}) \ (\text{action } \text{cfg2})$

\implies

$\text{rel-cfg } P \ \text{cfg1} \ \text{cfg2}$

lemma *rel-cfg-state*: $\text{rel-cfg } P \ \text{cfg1} \ \text{cfg2} \implies P \ (\text{state } \text{cfg1}) \ (\text{state } \text{cfg2})$

<proof>

lemma *rel-cfg-cont*:

$\text{rel-cfg } P \ \text{cfg1} \ \text{cfg2} \implies$

$\text{rel-pmf } (\lambda s \ t. \text{rel-cfg } P \ (\text{cont } \text{cfg1 } s) \ (\text{cont } \text{cfg2 } t)) \ (\text{action } \text{cfg1}) \ (\text{action } \text{cfg2})$

<proof>

lemma *rel-cfg-action*:

assumes $P: \text{rel-cfg } P \ \text{cfg1} \ \text{cfg2}$ **shows** $\text{rel-pmf } P \ (\text{action } \text{cfg1}) \ (\text{action } \text{cfg2})$

<proof>

lemma *rel-cfg-eq*: $\text{rel-cfg } (=) \ \text{cfg1} \ \text{cfg2} \longleftrightarrow \text{cfg1} = \text{cfg2}$

<proof>

5.2 Configuration with Memoryless Scheduler

definition $memoryless-on\ f\ s = cfg-corec\ s\ f\ (\lambda\ t.\ t)\ s$

lemma

shows $state-memoryless-on[simp]: state\ (memoryless-on\ f\ s) = s$
and $action-memoryless-on[simp]: action\ (memoryless-on\ f\ s) = f\ s$
and $cont-memoryless-on[simp]: t \in (f\ s) \implies cont\ (memoryless-on\ f\ s)\ t =$
 $memoryless-on\ f\ t$
 $\langle proof \rangle$

definition $K-cfg :: 's\ cfg \Rightarrow 's\ cfg\ pmf$ **where**

$K-cfg\ cfg = map-pmf\ (cont\ cfg)\ (action\ cfg)$

lemma $set-K-cfg: set-pmf\ (K-cfg\ cfg) = cont\ cfg\ `set-pmf\ (action\ cfg)$

$\langle proof \rangle$

lemma $nn-integral-K-cfg: (\int^+ cfg.\ f\ cfg\ \partial K-cfg\ cfg) = (\int^+ s.\ f\ (cont\ cfg\ s)\ \partial action$
 $cfg)$

$\langle proof \rangle$

5.3 MDP Kernel and Induced Configurations

locale $Markov-Decision-Process =$

fixes $K :: 's \Rightarrow 's\ pmf\ set$

assumes $K-wf: \bigwedge s.\ K\ s \neq \{\}$

begin

definition $E = (SIGMA\ s:UNIV.\ \bigcup D \in K\ s.\ set-pmf\ D)$

coinductive $cfg-onp :: 's \Rightarrow 's\ cfg \Rightarrow bool$ **where**

$\bigwedge s.\ state\ cfg = s \implies action\ cfg \in K\ s \implies (\bigwedge t.\ t \in action\ cfg \implies cfg-onp\ t$
 $(cont\ cfg\ t)) \implies$
 $cfg-onp\ s\ cfg$

definition $cfg-on\ s = \{cfg.\ cfg-onp\ s\ cfg\}$

lemma

shows $cfg-onD-action[intro, simp]: cfg \in cfg-on\ s \implies action\ cfg \in K\ s$
and $cfg-onD-cont[intro, simp]: cfg \in cfg-on\ s \implies t \in action\ cfg \implies cont\ cfg\ t$
 $\in cfg-on\ t$
and $cfg-onD-state[simp]: cfg \in cfg-on\ s \implies state\ cfg = s$
and $cfg-onI: state\ cfg = s \implies action\ cfg \in K\ s \implies (\bigwedge t.\ t \in action\ cfg \implies$
 $cont\ cfg\ t \in cfg-on\ t) \implies cfg \in cfg-on\ s$
 $\langle proof \rangle$

lemma $cfg-on-coinduct[coinduct\ set: cfg-on]:$

assumes $P\ s\ cfg$

assumes $\bigwedge cfg\ s.\ P\ s\ cfg \implies state\ cfg = s$

assumes $\bigwedge cfg\ s.\ P\ s\ cfg \implies action\ cfg \in K\ s$

assumes $\bigwedge \text{cfg } s \ t. P \ s \ \text{cfg} \implies t \in \text{action } \text{cfg} \implies P \ t \ (\text{cont } \text{cfg } t)$
shows $\text{cfg} \in \text{cfg-on } s$
 $\langle \text{proof} \rangle$

lemma *memoryless-on-cfg-onI*:

assumes $\bigwedge s. f \ s \in K \ s$
shows *memoryless-on* $f \ s \in \text{cfg-on } s$
 $\langle \text{proof} \rangle$

lemma *cfg-of-cfg-onI*:

$D \in K \ s \implies (\bigwedge t. t \in D \implies c \ t \in \text{cfg-on } t) \implies \text{Cfg } s \ D \ c \in \text{cfg-on } s$
 $\langle \text{proof} \rangle$

definition *arb-act* $s = (\text{SOME } D. D \in K \ s)$

lemma *arb-actI[simp]*: $\text{arb-act } s \in K \ s$
 $\langle \text{proof} \rangle$

lemma *cfg-on-not-empty[intro, simp]*: $\text{cfg-on } s \neq \{\}$
 $\langle \text{proof} \rangle$

sublocale *MC*: *MC-syntax* $K\text{-cfg}$ $\langle \text{proof} \rangle$

abbreviation *St* :: 's *stream measure where*
 $St \equiv \text{stream-space } (\text{count-space } UNIV)$

5.4 Trace Space

definition $T \ \text{cfg} = \text{distr } (MC.T \ \text{cfg}) \ St \ (\text{smap } \text{state})$

sublocale *T*: *prob-space* $T \ \text{cfg}$ **for** cfg
 $\langle \text{proof} \rangle$

lemma *space-T[simp]*: $\text{space } (T \ \text{cfg}) = \text{space } St$
 $\langle \text{proof} \rangle$

lemma *sets-T[simp]*: $\text{sets } (T \ \text{cfg}) = \text{sets } St$
 $\langle \text{proof} \rangle$

lemma *measurable-T1[simp]*: $\text{measurable } (T \ \text{cfg}) \ N = \text{measurable } St \ N$
 $\langle \text{proof} \rangle$

lemma *measurable-T2[simp]*: $\text{measurable } N \ (T \ \text{cfg}) = \text{measurable } N \ St$
 $\langle \text{proof} \rangle$

lemma *nn-integral-T*:

assumes [*measurable*]: $f \in \text{borel-measurable } St$
shows $(\int^+ X. f \ X \ \partial T \ \text{cfg}) = (\int^+ \text{cfg}'. (\int^+ x. f \ (\text{state } \text{cfg}' \ \#\# \ x) \ \partial T \ \text{cfg}^{\wedge}) \ \partial K\text{-cfg } \text{cfg})$

<proof>

lemma *T-eq*:

$T \text{ cfg} = (\text{measure-pmf } (K\text{-cfg } \text{cfg}) \gg (\lambda \text{cfg}'. \text{distr } (T \text{ cfg}') \text{ St } (\lambda \omega. \text{state } \text{cfg}' \text{ ## } \omega)))$
<proof>

lemma *T-memoryless-on*: $T (\text{memoryless-on } ct \ s) = MC\text{-syntax}.T \ ct \ s$
<proof>

lemma *nn-integral-T-lfp*:

assumes [*measurable*]: $\text{case-prod } g \in \text{borel-measurable } (\text{count-space } UNIV \otimes_M \text{borel})$

assumes *cont-g*: $\bigwedge s. \text{sup-continuous } (g \ s)$

assumes *int-g*: $\bigwedge f \text{cfg}. f \in \text{borel-measurable } (\text{stream-space } (\text{count-space } UNIV))$

\implies

$(\int^{+\omega}. g (\text{state } \text{cfg}) (f \ \omega) \ \partial T \ \text{cfg}) = g (\text{state } \text{cfg}) (\int^{+\omega}. f \ \omega \ \partial T \ \text{cfg})$

shows $(\int^{+\omega}. \text{lfp } (\lambda f \ \omega. g (\text{shd } \omega) (f (\text{stl } \omega))) \ \omega \ \partial T \ \text{cfg}) =$

$\text{lfp } (\lambda f \ \text{cfg}. \int^{+t}. g (\text{state } t) (f \ t) \ \partial K\text{-cfg } \text{cfg}) \ \text{cfg}$

<proof>

lemma *emeasure-Collect-T*:

assumes [*measurable*]: $\text{Measurable.pred } \text{St } P$

shows $\text{emeasure } (T \ \text{cfg}) \ \{x \in \text{space } \text{St}. P \ x\} =$

$(\int^{+ \text{cfg}'}. \text{emeasure } (T \ \text{cfg}') \ \{x \in \text{space } \text{St}. P (\text{state } \text{cfg}' \ \text{## } x)\} \ \partial K\text{-cfg } \text{cfg})$

<proof>

definition *E-sup* :: $'s \Rightarrow ('s \ \text{stream} \Rightarrow \text{ennreal}) \Rightarrow \text{ennreal}$

where

$E\text{-sup } s \ f = (\bigsqcup \text{cfg} \in \text{cfg-on } s. \int^{+x}. f \ x \ \partial T \ \text{cfg})$

lemma *E-sup-const*: $0 \leq c \implies E\text{-sup } s \ (\lambda \cdot. c) = c$
<proof>

lemma *E-sup-mult-right*:

assumes [*measurable*]: $f \in \text{borel-measurable } \text{St}$ **and** [*simp*]: $0 \leq c$

shows $E\text{-sup } s \ (\lambda x. c * f \ x) = c * E\text{-sup } s \ f$

<proof>

lemma *E-sup-mono*:

$(\bigwedge \omega. f \ \omega \leq g \ \omega) \implies E\text{-sup } s \ f \leq E\text{-sup } s \ g$

<proof>

lemma *E-sup-add*:

assumes [*measurable*]: $f \in \text{borel-measurable } \text{St}$ $g \in \text{borel-measurable } \text{St}$

shows $E\text{-sup } s \ (\lambda x. f \ x + g \ x) \leq E\text{-sup } s \ f + E\text{-sup } s \ g$

<proof>

lemma *E-sup-add-left*:

assumes $[measurable]: f \in \text{borel-measurable } St$
shows $E\text{-sup } s (\lambda x. f x + c) = E\text{-sup } s f + c$
 $\langle \text{proof} \rangle$

lemma $E\text{-sup-add-right}$:
 $f \in \text{borel-measurable } St \implies E\text{-sup } s (\lambda x. c + f x) = c + E\text{-sup } s f$
 $\langle \text{proof} \rangle$

lemma $E\text{-sup-SUP}$:
assumes $[measurable]: \bigwedge i. f i \in \text{borel-measurable } St$ **and** $[simp]: \text{incseq } f$
shows $E\text{-sup } s (\lambda x. \bigsqcup i. f i x) = (\bigsqcup i. E\text{-sup } s (f i))$
 $\langle \text{proof} \rangle$

lemma $E\text{-sup-iterate}$:
assumes $[measurable]: f \in \text{borel-measurable } St$
shows $E\text{-sup } s f = (\bigsqcup D \in K s. \int^+ t. E\text{-sup } t (\lambda \omega. f (t \#\# \omega)) \partial \text{measure-pmf } D)$
 $\langle \text{proof} \rangle$

lemma $E\text{-sup-bot}$: $E\text{-sup } s \perp = 0$
 $\langle \text{proof} \rangle$

lemma $E\text{-sup-lfp}$:
fixes g
defines $l \equiv \lambda f \omega. g (\text{shd } \omega) (f (\text{stl } \omega))$
assumes $\text{measurable-g}[measurable]: \text{case-prod } g \in \text{borel-measurable } (\text{count-space } UNIV \otimes_M \text{borel})$
assumes $\text{cont-g}: \bigwedge s. \text{sup-continuous } (g s)$
assumes $\text{int-g}: \bigwedge f \text{cfg}. f \in \text{borel-measurable } St \implies$
 $(\int^+ \omega. g (\text{state } \text{cfg}) (f \omega) \partial T \text{cfg}) = g (\text{state } \text{cfg}) (\text{integral}^N (T \text{cfg}) f)$
shows $(\lambda s. E\text{-sup } s (lfp l)) = lfp (\lambda f s. \bigsqcup D \in K s. \int^+ t. g t (f t) \partial \text{measure-pmf } D)$
 $\langle \text{proof} \rangle$

definition $P\text{-sup } s P = (\bigsqcup \text{cfg} \in \text{cfg-on } s. \text{emeasure } (T \text{cfg}) \{x \in \text{space } St. P x\})$

lemma $P\text{-sup-eq-E-sup}$:
assumes $[measurable]: \text{Measurable.pred } St P$
shows $P\text{-sup } s P = E\text{-sup } s (\text{indicator } \{x \in \text{space } St. P x\})$
 $\langle \text{proof} \rangle$

lemma $P\text{-sup-True}[simp]$: $P\text{-sup } t (\lambda \omega. \text{True}) = 1$
 $\langle \text{proof} \rangle$

lemma $P\text{-sup-False}[simp]$: $P\text{-sup } t (\lambda \omega. \text{False}) = 0$
 $\langle \text{proof} \rangle$

lemma $P\text{-sup-SUP}$:
fixes $P :: \text{nat} \Rightarrow 's \text{ stream} \Rightarrow \text{bool}$

assumes *mono P and P[measurable]*: $\bigwedge i. \text{Measurable.pred St } (P i)$
shows $P\text{-sup } s (\lambda x. \exists i. P i x) = (\bigsqcup i. P\text{-sup } s (P i))$
 $\langle \text{proof} \rangle$

lemma *P-sup-lfp*:

assumes *Q: sup-continuous Q*

assumes *f: f ∈ measurable St M*

assumes *Q-m: $\bigwedge P. \text{Measurable.pred M } P \implies \text{Measurable.pred M } (Q P)$*

shows $P\text{-sup } s (\lambda x. \text{lfp } Q (f x)) = (\bigsqcup i. P\text{-sup } s (\lambda x. (Q \overset{\sim}{\sim} i) \perp (f x)))$

$\langle \text{proof} \rangle$

lemma *P-sup-iterate*:

assumes *[measurable]: Measurable.pred St P*

shows $P\text{-sup } s P = (\bigsqcup D \in K s. \int^+ t. P\text{-sup } t (\lambda \omega. P (t \#\#\ \omega)) \partial \text{measure-pmf } D)$

$\langle \text{proof} \rangle$

definition $E\text{-inf } s f = (\prod \text{cfg} \in \text{cfg-on } s. \int^+ x. f x \partial T \text{cfg})$

lemma *E-inf-const*: $0 \leq c \implies E\text{-inf } s (\lambda \cdot. c) = c$

$\langle \text{proof} \rangle$

lemma *E-inf-mono*:

$(\bigwedge \omega. f \omega \leq g \omega) \implies E\text{-inf } s f \leq E\text{-inf } s g$

$\langle \text{proof} \rangle$

lemma *E-inf-iterate*:

assumes *[measurable]: f ∈ borel-measurable St*

shows $E\text{-inf } s f = (\prod D \in K s. \int^+ t. E\text{-inf } t (\lambda \omega. f (t \#\#\ \omega)) \partial \text{measure-pmf } D)$

$\langle \text{proof} \rangle$

lemma *emeasure-T-const[simp]*: $\text{emeasure } (T s) (\text{space St}) = 1$

$\langle \text{proof} \rangle$

lemma *E-inf-greatest*:

$(\bigwedge \text{cfg}. \text{cfg} \in \text{cfg-on } s \implies x \leq (\int^+ x. f x \partial T \text{cfg})) \implies x \leq E\text{-inf } s f$

$\langle \text{proof} \rangle$

lemma *E-inf-lower2*:

$\text{cfg} \in \text{cfg-on } s \implies (\int^+ x. f x \partial T \text{cfg}) \leq x \implies E\text{-inf } s f \leq x$

$\langle \text{proof} \rangle$

Maybe the following statement can be generalized to infinite $K s$.

lemma *E-inf-lfp*:

fixes *g*

defines $l \equiv \lambda f \omega. g (\text{shd } \omega) (f (\text{stl } \omega))$

assumes *measurable-g[measurable]: case-prod g ∈ borel-measurable (count-space UNIV \otimes_M borel)*

assumes *cont-g: $\bigwedge s. \text{sup-continuous } (g s)$*

assumes *int-g*: $\bigwedge f \text{ cfg. } f \in \text{borel-measurable } St \implies$
 $(\int^+ \omega. g (\text{state } \text{cfg}) (f \ \omega) \ \partial T \ \text{cfg}) = g (\text{state } \text{cfg}) (\text{integral}^N (T \ \text{cfg}) f)$
assumes *K-finite*: $\bigwedge s. \text{finite } (K \ s)$
shows $(\lambda s. E\text{-inf } s (\text{lfp } l)) = \text{lfp } (\lambda f s. \bigcap D \in K \ s. \int^+ t. g \ t (f \ t) \ \partial \text{measure-pmf } D)$
 $\langle \text{proof} \rangle$

definition $P\text{-inf } s \ P = (\bigcap \text{cfg} \in \text{cfg-on } s. \text{emeasure } (T \ \text{cfg}) \ \{x \in \text{space } St. \ P \ x\})$

lemma *P-inf-eq-E-inf*:
assumes [*measurable*]: $\text{Measurable.pred } St \ P$
shows $P\text{-inf } s \ P = E\text{-inf } s (\text{indicator } \{x \in \text{space } St. \ P \ x\})$
 $\langle \text{proof} \rangle$

lemma *P-inf-True[simp]*: $P\text{-inf } t (\lambda \omega. \ \text{True}) = 1$
 $\langle \text{proof} \rangle$

lemma *P-inf-False[simp]*: $P\text{-inf } t (\lambda \omega. \ \text{False}) = 0$
 $\langle \text{proof} \rangle$

lemma *P-inf-INF*:
fixes $P :: \text{nat} \Rightarrow 's \ \text{stream} \Rightarrow \text{bool}$
assumes *decseq P* **and** $P[\text{measurable}] : \bigwedge i. \text{Measurable.pred } St (P \ i)$
shows $P\text{-inf } s (\lambda x. \ \forall i. \ P \ i \ x) = (\bigcap i. \ P\text{-inf } s (P \ i))$
 $\langle \text{proof} \rangle$

lemma *P-inf-gfp*:
assumes *Q*: $\text{inf-continuous } Q$
assumes *f*: $f \in \text{measurable } St \ M$
assumes *Q-m*: $\bigwedge P. \text{Measurable.pred } M \ P \implies \text{Measurable.pred } M (Q \ P)$
shows $P\text{-inf } s (\lambda x. \ \text{gfp } Q (f \ x)) = (\bigcap i. \ P\text{-inf } s (\lambda x. (Q \ \sim i) \ \top (f \ x)))$
 $\langle \text{proof} \rangle$

lemma *P-inf-iterate*:
assumes [*measurable*]: $\text{Measurable.pred } St \ P$
shows $P\text{-inf } s \ P = (\bigcap D \in K \ s. \int^+ t. \ P\text{-inf } t (\lambda \omega. \ P (t \ \#\# \ \omega)) \ \partial \text{measure-pmf } D)$
 $\langle \text{proof} \rangle$

end

5.5 Finite MDPs

locale *Finite-Markov-Decision-Process* = *Markov-Decision-Process K* **for** $K :: 's \Rightarrow 's \ \text{pmf set} +$
fixes $S :: 's \ \text{set}$
assumes *S-not-empty*: $S \neq \{\}$
assumes *S-finite*: $\text{finite } S$
assumes *K-closed*: $\bigwedge s. \ s \in S \implies (\bigcup D \in K \ s. \ \text{set-pmf } D) \subseteq S$
assumes *K-finite*: $\bigwedge s. \ s \in S \implies \text{finite } (K \ s)$

begin

lemma *action-closed*: $s \in S \implies cfg \in cfg\text{-on } s \implies t \in action\ cfg \implies t \in S$
<proof>

lemma *set-pmf-closed*: $s \in S \implies D \in K\ s \implies t \in D \implies t \in S$
<proof>

lemma *Pi-closed*: $ct \in Pi\ S\ K \implies s \in S \implies t \in ct\ s \implies t \in S$
<proof>

lemma *E-closed*: $s \in S \implies (s, t) \in E \implies t \in S$
<proof>

lemma *set-pmf-finite*: $s \in S \implies D \in K\ s \implies finite\ D$
<proof>

definition *valid-cfg* = $(\bigcup_{s \in S}. cfg\text{-on } s)$

lemma *valid-cfgI*: $s \in S \implies cfg \in cfg\text{-on } s \implies cfg \in valid\text{-cfg}$
<proof>

lemma *valid-cfgD*: $cfg \in valid\text{-cfg} \implies cfg \in cfg\text{-on } (state\ cfg)$
<proof>

lemma

shows *valid-cfg-state-in-S*: $cfg \in valid\text{-cfg} \implies state\ cfg \in S$

and *valid-cfg-action*: $cfg \in valid\text{-cfg} \implies s \in action\ cfg \implies s \in S$

and *valid-cfg-cont*: $cfg \in valid\text{-cfg} \implies s \in action\ cfg \implies cont\ cfg\ s \in valid\text{-cfg}$

<proof>

lemma *valid-K-cfg[intro]*: $cfg \in valid\text{-cfg} \implies cfg' \in K\text{-cfg } cfg \implies cfg' \in valid\text{-cfg}$
<proof>

definition *simple ct* = *memoryless-on* ($\lambda s. if\ s \in S\ then\ ct\ s\ else\ arb\text{-act } s$)

lemma *simple-cfg-on[simp]*: $ct \in Pi\ S\ K \implies simple\ ct\ s \in cfg\text{-on } s$
<proof>

lemma *simple-valid-cfg[simp]*: $ct \in Pi\ S\ K \implies s \in S \implies simple\ ct\ s \in valid\text{-cfg}$
<proof>

lemma *cont-simple[simp]*: $s \in S \implies t \in set\text{-pmf } (ct\ s) \implies cont\ (simple\ ct\ s)\ t = simple\ ct\ t$
<proof>

lemma *state-simple[simp]*: $state\ (simple\ ct\ s) = s$
<proof>

lemma *action-simple*[simp]: $s \in S \implies \text{action (simple ct s) = ct s}$
 ⟨proof⟩

lemma *simple-valid-cfg-iff*: $ct \in \text{Pi } S \ K \implies \text{simple ct s} \in \text{valid-cfg} \longleftrightarrow s \in S$
 ⟨proof⟩

end

end

theory *MDP-Reachability-Problem*
imports *Markov-Decision-Process*
begin

inductive-set *directed-towards* :: 'a set \Rightarrow ('a \times 'a) set \Rightarrow 'a set **for** $A \ r$ **where**
start: $\bigwedge x. x \in A \implies x \in \text{directed-towards } A \ r$
step: $\bigwedge x \ y. y \in \text{directed-towards } A \ r \implies (x, y) \in r \implies x \in \text{directed-towards } A \ r$

hide-fact (**open**) *start step*

lemma *directed-towards-mono*:

assumes $s \in \text{directed-towards } A \ F \ F \subseteq G$ **shows** $s \in \text{directed-towards } A \ G$
 ⟨proof⟩

lemma *directed-eq-rtrancl*: $x \in \text{directed-towards } A \ r \longleftrightarrow (\exists a \in A. (x, a) \in r^*)$
 ⟨proof⟩

lemma *directed-eq-rtrancl-Image*: $\text{directed-towards } A \ r = (r^*)^{-1} \text{ `` } A$
 ⟨proof⟩

locale *Reachability-Problem* = *Finite-Markov-Decision-Process* $K \ S$ **for** $K :: 's \Rightarrow$
 's pmf set **and** $S +$
fixes $S1 \ S2 :: 's \text{ set}$
assumes $S1: S1 \subseteq S$ **and** $S2: S2 \subseteq S$ **and** $S1-S2: S1 \cap S2 = \{\}$
begin

lemma [*measurable*]:

$S \in \text{sets (count-space UNIV)}$ $S1 \in \text{sets (count-space UNIV)}$ $S2 \in \text{sets (count-space UNIV)}$
 ⟨proof⟩

definition

$v = (\lambda \text{cfg} \in \text{valid-cfg}. \text{emeasure (T cfg) } \{x \in \text{space St. (HLD } S1 \text{ until HLD } S2) \text{ (state cfg \#\# x)}\})$

lemma *v-eq*: $\text{cfg} \in \text{valid-cfg} \implies$

$v \text{ cfg} = \text{emeasure (T cfg) } \{x \in \text{space St. (HLD } S1 \text{ until HLD } S2) \text{ (state cfg \#\# x)}\}$
 ⟨proof⟩

lemma *real-v*: $cfg \in \text{valid-cfg} \implies \text{enn2real } (v \text{ cfg}) = \mathcal{P}(\omega \text{ in } T \text{ cfg. (HLD S1 suntil HLD S2) (state cfg \#\# \omega)})$

<proof>

lemma *v-le-1*: $cfg \in \text{valid-cfg} \implies v \text{ cfg} \leq 1$

<proof>

lemma *v-neg-Pinf[simp]*: $cfg \in \text{valid-cfg} \implies v \text{ cfg} \neq \text{top}$

<proof>

lemma *v-1-AE*: $cfg \in \text{valid-cfg} \implies v \text{ cfg} = 1 \iff (\text{AE } \omega \text{ in } T \text{ cfg. (HLD S1 suntil HLD S2) (state cfg \#\# \omega)})$

<proof>

lemma *v-0-AE*: $cfg \in \text{valid-cfg} \implies v \text{ cfg} = 0 \iff (\text{AE } x \text{ in } T \text{ cfg. not (HLD S1 suntil HLD S2) (state cfg \#\# x)})$

<proof>

lemma *v-S2[simp]*: $cfg \in \text{valid-cfg} \implies \text{state cfg} \in S2 \implies v \text{ cfg} = 1$

<proof>

lemma *v-nS12[simp]*: $cfg \in \text{valid-cfg} \implies \text{state cfg} \notin S1 \implies \text{state cfg} \notin S2 \implies v \text{ cfg} = 0$

<proof>

lemma *v-nS[simp]*: $cfg \notin \text{valid-cfg} \implies v \text{ cfg} = \text{undefined}$

<proof>

lemma *v-S1*:

assumes *cfg[simp, intro]*: $cfg \in \text{valid-cfg}$ **and** *cfg-S1[simp]*: $\text{state cfg} \in S1$

shows $v \text{ cfg} = (\int^+ s. v (\text{cont } cfg \ s) \ \partial \text{action } cfg)$

<proof>

lemma *real-v-integrable*:

integrable (action cfg) (λs. enn2real (v (cont cfg s)))

<proof>

lemma *real-v-integral-eq*:

assumes *cfg[simp]*: $cfg \in \text{valid-cfg}$

shows $\text{enn2real } (\int^+ s. v (\text{cont } cfg \ s) \ \partial \text{action } cfg) = \int s. \text{enn2real } (v (\text{cont } cfg \ s) \ \partial \text{action } cfg)$

<proof>

lemma *v-eq-0-coinduct[consumes 3, case-names valid nS2 cont]*:

assumes *: $P \text{ cfg}$

assumes *valid*: $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{cfg} \in \text{valid-cfg}$

assumes *nS2*: $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{state cfg} \notin S2$

assumes *cont*: $\bigwedge \text{cfg } \text{cfg}'. P \text{ cfg} \implies \text{state cfg} \in S1 \implies \text{cfg}' \in K\text{-cfg } \text{cfg} \implies P \text{ cfg}' \vee v \text{ cfg}' = 0$

shows $v \text{ cfg} = 0$
 ⟨proof⟩

definition $p = (\lambda s \in S. P\text{-sup } s (\lambda \omega. (HLD \ S1 \ \text{suntil} \ HLD \ S2) (s \ \#\# \ \omega)))$

lemma $p\text{-eq-SUP-}v: s \in S \implies p \ s = \bigsqcup (v \ \text{' } \ \text{cfg-on } s)$
 ⟨proof⟩

lemma $v\text{-le-}p: \text{cfg} \in \text{valid-cfg} \implies v \ \text{cfg} \leq p \ (\text{state } \text{cfg})$
 ⟨proof⟩

lemma $p\text{-eq-0-imp}: \text{cfg} \in \text{valid-cfg} \implies p \ (\text{state } \text{cfg}) = 0 \implies v \ \text{cfg} = 0$
 ⟨proof⟩

lemma $p\text{-eq-0-iff}: s \in S \implies p \ s = 0 \iff (\forall \text{cfg} \in \text{cfg-on } s. v \ \text{cfg} = 0)$
 ⟨proof⟩

lemma $p\text{-le-}1: s \in S \implies p \ s \leq 1$
 ⟨proof⟩

lemma $p\text{-undefined}[simp]: s \notin S \implies p \ s = \text{undefined}$
 ⟨proof⟩

lemma $p\text{-not-inf}[simp]: s \in S \implies p \ s \neq \text{top}$
 ⟨proof⟩

lemma $p\text{-}S1: s \in S1 \implies p \ s = (\bigsqcup D \in K \ s. \int^+ t. p \ t \ \partial \text{measure-pmf } D)$
 ⟨proof⟩

lemma $p\text{-}S2[simp]: s \in S2 \implies p \ s = 1$
 ⟨proof⟩

lemma $p\text{-}nS12: s \in S \implies s \notin S1 \implies s \notin S2 \implies p \ s = 0$
 ⟨proof⟩

lemma $p\text{-pos}$:

assumes $(s, t) \in (SIGMA \ s:S1. \bigcup D \in K \ s. \text{set-pmf } D)^* \ t \in S2$ **shows** $0 < p \ s$
 ⟨proof⟩

definition $F\text{-sup} :: ('s \Rightarrow \text{ennreal}) \Rightarrow 's \Rightarrow \text{ennreal}$ **where**

$F\text{-sup } f = (\lambda s \in S. \text{if } s \in S2 \text{ then } 1 \text{ else if } s \in S1 \text{ then } SUP \ D \in K \ s. \int^+ t. f \ t \ \partial \text{measure-pmf } D \text{ else } 0)$

lemma $F\text{-sup-cong}: (\bigwedge s. s \in S \implies f \ s = g \ s) \implies F\text{-sup } f \ s = F\text{-sup } g \ s$
 ⟨proof⟩

lemma $\text{continuous-}F\text{-sup}: \text{sup-continuous } F\text{-sup}$
 ⟨proof⟩

lemma *mono-F-sup*: *mono F-sup*

<proof>

lemma *lfp-F-sup-iterate*: $lfp\ F-sup = (SUP\ i.\ (F-sup\ \sim\ i)\ (\lambda x \in S.\ 0))$

<proof>

lemma *p-eq-lfp-F-sup*: $p = lfp\ F-sup$

<proof>

definition $S_e = \{s \in S.\ p\ s = 0\}$

lemma $S_e: S_e \subseteq S$

<proof>

lemma $v-S_e$: $cfg \in valid-cfg \implies state\ cfg \in S_e \implies v\ cfg = 0$

<proof>

lemma S_e-nS2 : $S_e \cap S2 = \{\}$

<proof>

lemma S_e-E1 : $s \in S_e \cap S1 \implies (s, t) \in E \implies t \in S_e$

<proof>

lemma S_e-E2 : $s \in S1 \implies (\bigwedge t.\ (s, t) \in E \implies t \in S_e) \implies s \in S_e$

<proof>

lemma $S_e-E-iff$: $s \in S1 \implies s \in S_e \iff (\forall t.\ (s, t) \in E \implies t \in S_e)$

<proof>

definition $S_r = S - (S_e \cup S2)$

lemma $S_r: S_r \subseteq S$

<proof>

lemma S_r-S1 : $S_r \subseteq S1$

<proof>

lemma S_r-eq : $S_r = S1 - S_e$

<proof>

lemma $v-neq-0-imp$: $cfg \in valid-cfg \implies v\ cfg \neq 0 \implies state\ cfg \in S_r \cup S2$

<proof>

lemma $valid-cfg-action-in-K$: $cfg \in valid-cfg \implies action\ cfg \in K\ (state\ cfg)$

<proof>

lemma $K-cfg-E$: $cfg \in valid-cfg \implies cfg' \in K-cfg\ cfg \implies (state\ cfg, state\ cfg') \in E$

<proof>

lemma *S_r-directed-towards-S2*:

assumes *s*: $s \in S_r$

shows $s \in \text{directed-towards } S2 \{(s, t) \mid s \ t. s \in S_r \wedge (s, t) \in E\}$ (**is** $s \in ?D$)

<proof>

definition *proper ct* $\longleftrightarrow ct \in Pi_E S K \wedge (\forall s \in S_r. v(\text{simple } ct \ s) > 0)$

lemma *S_r-nS2*: $s \in S_r \implies s \notin S2$

<proof>

lemma *properD1*: *proper ct* $\implies ct \in Pi_E S K$

<proof>

lemma *proper-eq*:

assumes *ct[simp, intro]*: $ct \in Pi_E S K$

shows *proper ct* $\longleftrightarrow S_r \subseteq \text{directed-towards } S2$ (*SIGMA* $s:S_r. ct \ s$)

(**is** $- \longleftrightarrow - \subseteq ?D$)

<proof>

lemma *exists-proper*:

obtains *ct where proper ct*

<proof>

definition *l-desc X ct l s* \longleftrightarrow

$s \in \text{directed-towards } S2$ (*SIGMA* $s : X. \{l \ s\}$) \wedge

$v(\text{simple } ct \ s) \leq v(\text{simple } ct \ (l \ s)) \wedge$

$l \ s \in \text{maximal } (\lambda s. v(\text{simple } ct \ s)) \ (ct \ s)$

lemma *exists-l-desc*:

assumes *ct: proper ct*

shows $\exists l \in S_r \rightarrow S_r \cup S2. \forall s \in S_r. l\text{-desc } S_r \ ct \ l \ s$

<proof>

lemma *F-v-memoryless*:

obtains *ct where* $ct \in Pi_E S K \ v \circ \text{simple } ct = F\text{-sup } (v \circ \text{simple } ct)$

<proof>

lemma *p-v-memoryless*:

obtains *ct where* $ct \in Pi_E S K \ p = v \circ \text{simple } ct$

<proof>

definition $n = (\lambda s \in S. P\text{-inf } s \ (\lambda \omega. (\text{HLD } S1 \ \text{suntil } \text{HLD } S2) \ (s \ \#\# \ \omega)))$

lemma *n-eq-INF-v*: $s \in S \implies n \ s = (\bigcap \text{cfg} \in \text{cfg-on } s. v \ \text{cfg})$

<proof>

lemma *n-le-v*: $s \in S \implies \text{cfg} \in \text{cfg-on } s \implies n \ s \leq v \ \text{cfg}$

<proof>

lemma *n-eq-1-imp*: $s \in S \implies \text{cfg} \in \text{cfg-on } s \implies n \ s = 1 \implies v \ \text{cfg} = 1$
<proof>

lemma *n-eq-1-iff*: $s \in S \implies n \ s = 1 \iff (\forall \text{cfg} \in \text{cfg-on } s. v \ \text{cfg} = 1)$
<proof>

lemma *n-le-1*: $s \in S \implies n \ s \leq 1$
<proof>

lemma *n-undefined[simp]*: $s \notin S \implies n \ s = \text{undefined}$
<proof>

lemma *n-eq-0*: $s \in S \implies \text{cfg} \in \text{cfg-on } s \implies v \ \text{cfg} = 0 \implies n \ s = 0$
<proof>

lemma *n-not-inf[simp]*: $s \in S \implies n \ s \neq \text{top}$
<proof>

lemma *n-S1*: $s \in S1 \implies n \ s = (\prod D \in K \ s. \int^+ t. n \ t \ \partial \text{measure-pmf } D)$
<proof>

lemma *n-S2[simp]*: $s \in S2 \implies n \ s = 1$
<proof>

lemma *n-nS12*: $s \in S \implies s \notin S1 \implies s \notin S2 \implies n \ s = 0$
<proof>

lemma *n-pos*:

assumes $P \ s \ s \in S1 \ \text{wf } R$

assumes *cont*: $\bigwedge s \ D. P \ s \implies s \in S1 \implies D \in K \ s \implies \exists w \in D. ((w, s) \in R \wedge w \in S1 \wedge P \ w) \vee 0 < n \ w$

shows $0 < n \ s$

<proof>

definition *F-inf* :: $('s \Rightarrow \text{ennreal}) \Rightarrow ('s \Rightarrow \text{ennreal})$ **where**

$F\text{-inf } f = (\lambda s \in S. \text{if } s \in S2 \text{ then } 1 \text{ else if } s \in S1 \text{ then } (\prod D \in K \ s. \int^+ t. f \ t \ \partial \text{measure-pmf } D) \text{ else } 0)$

lemma *F-inf-n*: $F\text{-inf } n = n$
<proof>

lemma *F-inf-nS[simp]*: $s \notin S \implies F\text{-inf } f \ s = \text{undefined}$
<proof>

lemma *mono-F-inf*: *mono* $F\text{-inf}$
<proof>

lemma *S1-nS2*: $s \in S1 \implies s \notin S2$
 <proof>

lemma *n-eq-lfp-F-inf*: $n = \text{lfp } F\text{-inf}$
 <proof>

lemma *real-n*: $s \in S \implies \text{ennreal } (\text{enn2real } (n \ s)) = n \ s$
 <proof>

lemma *real-p*: $s \in S \implies \text{ennreal } (\text{enn2real } (p \ s)) = p \ s$
 <proof>

lemma *p-ub*:

fixes x

assumes $s \in S$

assumes *solution*: $\bigwedge s \ D. s \in S1 \implies D \in K \ s \implies (\sum t \in S. \text{pmf } D \ t * x \ t) \leq x \ s$

assumes *solution-0*: $\bigwedge s. s \in S \implies p \ s = 0 \implies x \ s = 0$

assumes *solution-S2*: $\bigwedge s. s \in S2 \implies x \ s = 1$

shows $\text{enn2real } (p \ s) \leq x \ s$ (**is** $?y \ s \leq -$)

<proof>

lemma *n-lb*:

fixes x

assumes $s \in S$

assumes *solution*: $\bigwedge s \ D. s \in S1 \implies D \in K \ s \implies x \ s \leq (\sum t \in S. \text{pmf } D \ t * x \ t)$

assumes *solution-n0*: $\bigwedge s. s \in S \implies n \ s = 0 \implies x \ s = 0$

assumes *solution-S2*: $\bigwedge s. s \in S2 \implies x \ s = 1$

shows $x \ s \leq \text{enn2real } (n \ s)$ (**is** $- \leq ?y \ s$)

<proof>

end

end

6 Discrete-time Markov Processes

In this file we construct discrete-time Markov processes, e.g. with arbitrary state spaces.

theory *Discrete-Time-Markov-Process*

imports *Markov-Models-Auxiliary*

begin

lemma *measure-eqI-PiM-sequence*:

fixes $M :: \text{nat} \Rightarrow 'a \ \text{measure}$

assumes *[simp]*: $\text{sets } P = \text{PiM } UNIV \ M \ \text{sets } Q = \text{PiM } UNIV \ M$

assumes *eq*: $\bigwedge A \ n. (\bigwedge i. A \ i \in \text{sets } (M \ i)) \implies$

$P \ (\text{prod-emb } UNIV \ M \ \{..n\}) \ (Pi_E \ \{..n\} \ A) = Q \ (\text{prod-emb } UNIV \ M \ \{..n\})$

($Pi_E \{..n\} A$)
assumes A : *finite-measure* P
shows $P = Q$
 $\langle proof \rangle$

lemma *distr-cong-simp*:

$M = K \implies sets\ N = sets\ L \implies (\bigwedge x. x \in space\ M = simp \implies f\ x = g\ x) \implies$
 $distr\ M\ N\ f = distr\ K\ L\ g$
 $\langle proof \rangle$

6.1 Constructing Discrete-Time Markov Processes

locale *discrete-Markov-process* =

fixes $M :: 'a\ measure$ **and** $K :: 'a \Rightarrow 'a\ measure$
assumes $K[measurable]$: $K \in M \rightarrow_M\ prob\ algebra\ M$
begin

lemma *space-K*: $x \in space\ M \implies space\ (K\ x) = space\ M$
 $\langle proof \rangle$

lemma *sets-K[measurable-cong]*: $x \in space\ M \implies sets\ (K\ x) = sets\ M$
 $\langle proof \rangle$

lemma *prob-space-K*: $x \in space\ M \implies prob\ space\ (K\ x)$
 $\langle proof \rangle$

definition $K' :: 'a \Rightarrow nat \Rightarrow (nat \Rightarrow 'a) \Rightarrow 'a\ measure$
where
 $K'\ x\ n'\ \omega' = K\ (case\ nat\ x\ \omega'\ n')$

lemma *IT-K'*:

assumes x : $x \in space\ M$ **shows** *Ionescu-Tulcea* $(K'\ x)\ (\lambda-. M)$
 $\langle proof \rangle$

definition *lim-sequence* :: $'a \Rightarrow (nat \Rightarrow 'a)\ measure$

where

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

lemma

assumes x : $x \in space\ M$
shows *space-lim-sequence*: $space\ (lim\ sequence\ x) = space\ (\prod_M\ i \in UNIV.\ M)$
and *sets-lim-sequence[measurable-cong]*: $sets\ (lim\ sequence\ x) = sets\ (\prod_M\ i \in UNIV.\ M)$
and *emeasure-lim-sequence-emb*: $\bigwedge J\ X.\ finite\ J \implies X \in sets\ (\prod_M\ j \in J.\ M)$
 \implies
 $emeasure\ (lim\ sequence\ x)\ (prod\ emb\ UNIV\ (\lambda-. M)\ J\ X) =$
 $emeasure\ (Ionescu\ Tulcea.\ CI\ (K'\ x)\ (\lambda-. M)\ J)\ X$
and *emeasure-lim-sequence-emb-I0o*: $\bigwedge n\ X.\ X \in sets\ (\prod_M\ i \in \{0..<n\}.\ M)$

\implies

$\text{emeasure } (\text{lim-sequence } x) (\text{prod-emb } UNIV (\lambda-. M) \{0..<n\} X) =$
 $\text{emeasure } (\text{Ionescu-Tulcea.C } (K' x) (\lambda-. M) 0 n (\lambda x. \text{undefined})) X$

$\langle \text{proof} \rangle$

lemma *lim-sequence[measurable]*: $\text{lim-sequence} \in M \rightarrow_M \text{prob-algebra } (\prod_M i \in UNIV. M)$

$\langle \text{proof} \rangle$

lemma *step-C*:

assumes $x: x \in \text{space } M$

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

$K x \ggg (\lambda y. \text{Ionescu-Tulcea.C } (K' x) (\lambda-. M) 1 n (\text{case-nat } y (\lambda-. \text{undefined})))$

$\langle \text{proof} \rangle$

lemma *lim-sequence-eq*:

assumes $x: x \in \text{space } M$

shows $\text{lim-sequence } x = \text{bind } (K x) (\lambda y. \text{distr } (\text{lim-sequence } y) (\prod_M j \in UNIV. M) (\text{case-nat } y))$

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

$\langle \text{proof} \rangle$

lemma *AE-lim-sequence*:

assumes $x[\text{simp}]$: $x \in \text{space } M$ **and** $P[\text{measurable}]$: $\text{Measurable.pred } (\prod_M i \in UNIV. M) P$

shows $(\text{AE } \omega \text{ in } \text{lim-sequence } x. P \omega) \longleftrightarrow (\text{AE } y \text{ in } K x. \text{AE } \omega \text{ in } \text{lim-sequence } y. P (\text{case-nat } y \omega))$

$\langle \text{proof} \rangle$

definition *lim-stream* :: $'a \Rightarrow 'a \text{ stream measure}$

where

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

lemma *space-lim-stream*: $\text{space } (\text{lim-stream } x) = \text{streams } (\text{space } M)$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *lim-stream[measurable]*: $\text{lim-stream} \in M \rightarrow_M \text{prob-algebra } (\text{stream-space } M)$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *prob-space-lim-stream*: $x \in \text{space } M \implies \text{prob-space } (\text{lim-stream } x)$

<proof>

lemma *lim-stream-eq*:

assumes $x: x \in \text{space } M$

shows $\text{lim-stream } x = \text{do } \{ y \leftarrow K \ x; \omega \leftarrow \text{lim-stream } y; \text{return } (\text{stream-space } M) \ (y \ \#\# \ \omega) \}$

<proof>

lemma *AE-lim-stream*:

assumes $x[\text{simp}]: x \in \text{space } M$ **and** $P[\text{measurable}]: \text{Measurable.pred } (\text{stream-space } M) \ P$

shows $(\text{AE } \omega \text{ in } \text{lim-stream } x. \ P \ \omega) \longleftrightarrow (\text{AE } y \text{ in } K \ x. \ \text{AE } \omega \text{ in } \text{lim-stream } y. \ P \ (y \ \#\# \ \omega))$

<proof>

lemma *emeasure-lim-stream*:

assumes $x[\text{measurable}, \text{simp}]: x \in \text{space } M$ **and** $A[\text{measurable}, \text{simp}]: A \in \text{sets } (\text{stream-space } M)$

shows $\text{lim-stream } x \ A = (\int^+ y. \ \text{emeasure } (\text{lim-stream } y) \ (((\#\#) \ y) -' \ A \cap \text{space } (\text{stream-space } M)) \ \partial K \ x)$

<proof>

lemma *lim-stream-eq-coinduct*[*case-names in-space step*]:

fixes $R :: 'a \Rightarrow 'a \text{ stream measure} \Rightarrow \text{bool}$

assumes $x: R \ x \ B \ x \in \text{space } M$

assumes $R: \bigwedge x \ B. \ R \ x \ B \implies \exists B' \in M \rightarrow_M \text{prob-algebra } (\text{stream-space } M).$

$(\text{AE } y \text{ in } K \ x. \ R \ y \ (B' \ y) \vee \text{lim-stream } y = B' \ y) \wedge$

$B = \text{do } \{ y \leftarrow K \ x; \omega \leftarrow B' \ y; \text{return } (\text{stream-space } M) \ (y \ \#\# \ \omega) \}$

shows $\text{lim-stream } x = B$

<proof>

lemma *prob-space-lim-sequence*: $x \in \text{space } M \implies \text{prob-space } (\text{lim-sequence } x)$

<proof>

end

6.2 Strong Markov Property for Discrete-Time Markov Processes

The filtration adopted to streams, i.e. to the n -th projection.

definition *stream-filtration* :: $'a \text{ measure} \Rightarrow \text{enat} \Rightarrow 'a \text{ stream measure}$

where $\text{stream-filtration } M \ n = (\text{SUP } i \in \{i :: \text{nat}. \ i \leq n\}. \ \text{vimage-algebra } (\text{streams } (\text{space } M)) \ (\lambda \omega. \ \omega \ !! \ i) \ M)$

lemma *measurable-stream-filtration1*: $\text{enat } i \leq n \implies (\lambda \omega. \ \omega \ !! \ i) \in \text{stream-filtration } M \ n \rightarrow_M \ M$

<proof>

lemma *measurable-stream-filtration2*:

$f \in \text{space } N \rightarrow \text{streams } (\text{space } M) \implies (\bigwedge i. \text{enat } i \leq n \implies (\lambda x. f x !! i) \in N \rightarrow_M M) \implies f \in N \rightarrow_M \text{stream-filtration } M n$
 ⟨proof⟩

lemma *space-stream-filtration*: $\text{space } (\text{stream-filtration } M n) = \text{space } (\text{stream-space } M)$
 ⟨proof⟩

lemma *sets-stream-filtration-le-stream-space*: $\text{sets } (\text{stream-filtration } M n) \subseteq \text{sets } (\text{stream-space } M)$
 ⟨proof⟩

interpretation *stream-filtration*: $\text{filtration space } (\text{stream-space } M) \text{ stream-filtration } M$
 ⟨proof⟩

lemma *measurable-stopping-time-stream*:
 $\text{stopping-time } (\text{stream-filtration } M) T \implies T \in \text{stream-space } M \rightarrow_M \text{count-space } UNIV$
 ⟨proof⟩

lemma *measurable-stopping-time-All-eq-0*:
assumes $T: \text{stopping-time } (\text{stream-filtration } M) T$
shows $\{x \in \text{space } M. \forall \omega \in \text{streams } (\text{space } M). T (x \#\#\ \omega) = 0\} \in \text{sets } M$
 ⟨proof⟩

lemma *stopping-time-0*:
assumes $T: \text{stopping-time } (\text{stream-filtration } M) T$
and $x: x \in \text{space } M$ **and** $\omega: \omega \in \text{streams } (\text{space } M) T (x \#\#\ \omega) > 0$
and $\omega': \omega' \in \text{streams } (\text{space } M)$
shows $T (x \#\#\ \omega') > 0$
 ⟨proof⟩

lemma *stopping-time-epred-SCons*:
assumes $T: \text{stopping-time } (\text{stream-filtration } M) T$
and $x: x \in \text{space } M$ **and** $\omega: \omega \in \text{streams } (\text{space } M) T (x \#\#\ \omega) > 0$
shows $\text{stopping-time } (\text{stream-filtration } M) (\lambda \omega. \text{epred } (T (x \#\#\ \omega)))$
 ⟨proof⟩

context *discrete-Markov-process*
begin

lemma *lim-stream-strong-Markov*:
assumes $x: x \in \text{space } M$ **and** $T: \text{stopping-time } (\text{stream-filtration } M) T$
shows $\text{lim-stream } x =$
 $\text{lim-stream } x \gg (\lambda \omega. \text{case } T \ \omega \ \text{of}$
 $\text{enat } i \Rightarrow \text{distr } (\text{lim-stream } (\omega !! i)) (\text{stream-space } M) (\lambda \omega'. \text{stake } (\text{Suc } i) \ \omega$
 $@- \ \omega')$
 $| \ \infty \Rightarrow \text{return } (\text{stream-space } M) \ \omega)$

(is - = ?L T x)
 ⟨proof⟩

end

end

7 Continuous-time Markov chains

theory *Continuous-Time-Markov-Chain*
imports *Discrete-Time-Markov-Process Discrete-Time-Markov-Chain*
begin

7.1 Trace Operations: relate ('a × real) stream and real ⇒ 'a

partial-function (*tailrec*) *trace-at* :: 'a ⇒ (real × 'a) stream ⇒ real ⇒ 'a
where

trace-at s ω j = (case ω of (t', s')##ω ⇒ if t' ≤ j then *trace-at* s' ω j else s)

lemma *trace-at-simp[simp]*: *trace-at* s ((t', s')##ω) j = (if t' ≤ j then *trace-at* s' ω j else s)
 ⟨proof⟩

lemma *trace-at-eq*:

trace-at s ω j = (case *sfirst* (λx. j < fst (shd x)) ω of ∞ ⇒ undefined | enat i ⇒ (s ## *smap* snd ω) !! i)
 ⟨proof⟩

lemma *trace-at-shift*: *trace-at* s (*smap* (λ(t, s'). (t + t', s')) ω) t = *trace-at* s ω (t - t')
 ⟨proof⟩

primcorec *merge-at* :: (real × 'a) stream ⇒ real ⇒ (real × 'a) stream ⇒ (real × 'a) stream

where

merge-at ω j ω' = (case ω of (t, s) ## ω ⇒ if t ≤ j then (t, s)##*merge-at* ω j ω' else ω')

lemma *merge-at-simp[simp]*: *merge-at* (x##ω) j ω' = (if fst x ≤ j then x##*merge-at* ω j ω' else ω')
 ⟨proof⟩

7.2 Exponential Distribution

definition *exponential* :: real ⇒ real measure

where

exponential l = *density lborel* (*exponential-density* l)

lemma *space-exponential*: *space* (*exponential* l) = UNIV

<proof>

lemma *sets-exponential[measurable-cong]*: *sets (exponential l) = sets borel*
<proof>

lemma *prob-space-exponential*: $0 < l \implies \text{prob-space (exponential l)}$
<proof>

lemma *AE-exponential*: $0 < l \implies \text{AE } x \text{ in exponential l. } 0 < x$
<proof>

lemma *emeasure-exponential-Ioi-cutoff*:
assumes $0 < l$
shows $\text{emeasure (exponential l) } \{x < ..\} = \text{exp } (- (\text{max } 0 \ x) * l)$
<proof>

lemma *emeasure-exponential-Ioi*:
 $0 < l \implies 0 \leq x \implies \text{emeasure (exponential l) } \{x < ..\} = \text{exp } (- x * l)$
<proof>

lemma *exponential-eq-stretch*:
assumes $0 < l$
shows $\text{exponential l} = \text{distr (exponential 1) borel } (\lambda x. (1/l) * x)$
<proof>

lemma *uniform-measure-exponential*:
assumes $0 < l \ 0 \leq t$
shows $\text{uniform-measure (exponential l) } \{t < ..\} = \text{distr (exponential l) borel } ((+)$
 $t) \text{ (is ?L = ?R)}$
<proof>

lemma *emeasure-PiM-exponential-Ioi-finite*:
assumes $J \subseteq I \text{ finite } J \wedge i. i \in I \implies 0 < R \ i \ 0 \leq x$
shows $\text{emeasure } (\prod_M i \in I. \text{exponential } (R \ i)) \text{ (prod-emb } I \ (\lambda i. \text{exponential } (R$
 $i)) \ J \ (\prod_E j \in J. \{x < ..\})) = \text{exp } (- x * (\sum i \in J. R \ i))$
<proof>

lemma *emeasure-PiM-exponential-Ioi-sequence*:
assumes $R: \text{summable } R \wedge i. 0 < R \ i \ 0 \leq x$
shows $\text{emeasure } (\prod_M i \in UNIV. \text{exponential } (R \ i)) \ (\prod i \in UNIV. \{x < ..\}) = \text{exp}$
 $(- x * \text{suminf } R)$
<proof>

lemma *emeasure-PiM-exponential-Ioi-countable*:
assumes $R: J \subseteq I \text{ countable } J \wedge i. i \in I \implies 0 < R \ i \ 0 \leq x$ **and** *finite: integrable*
(count-space J) R
shows $\text{emeasure } (\prod_M i \in I. \text{exponential } (R \ i)) \text{ (prod-emb } I \ (\lambda i. \text{exponential } (R$
 $i)) \ J \ (\prod_E j \in J. \{x < ..\})) =$
 $\text{exp } (- x * (\text{LINT } i | \text{count-space } J. R \ i))$

<proof>

lemma *AE-PiM-exponential-suminf-infty*:

fixes $R :: \text{nat} \Rightarrow \text{real}$

assumes $R: \bigwedge n. 0 < R\ n$ **and** *finite*: $(\sum n. \text{ennreal } (1 / R\ n)) = \text{top}$

shows *AE* ω *in* $\Pi_M n \in \text{UNIV}. \text{exponential } (R\ n). (\sum n. \text{ereal } (\omega\ n)) = \infty$

<proof>

7.3 Transition Rates

locale *transition-rates* =

fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{real}$

assumes *R-nonneg[simp]*: $\bigwedge x\ y. 0 \leq R\ x\ y$

assumes *R-diagonal-0[simp]*: $\bigwedge x. R\ x\ x = 0$

assumes *finite-weight*: $\bigwedge x. (\int^+ y. R\ x\ y\ \partial \text{count-space UNIV}) < \infty$

assumes *positive-weight*: $\bigwedge x. 0 < (\int^+ y. R\ x\ y\ \partial \text{count-space UNIV})$

begin

abbreviation $S :: (\text{real} \times 'a)$ *measure*

where $S \equiv (\text{borel} \otimes_M \text{count-space UNIV})$

abbreviation $T :: (\text{real} \times 'a)$ *stream measure*

where $T \equiv \text{stream-space } S$

abbreviation $I :: 'a \Rightarrow 'a$ *set*

where $I\ x \equiv \{y. 0 < R\ x\ y\}$

lemma *I-countable*: *countable* $(I\ x)$

<proof>

definition *escape-rate* :: $'a \Rightarrow \text{real}$ **where**

escape-rate $x = \int y. R\ x\ y\ \partial \text{count-space UNIV}$

lemma *ennreal-escape-rate*: $\text{ennreal } (\text{escape-rate } x) = (\int^+ y. R\ x\ y\ \partial \text{count-space UNIV})$

<proof>

lemma *escape-rate-pos*: $0 < \text{escape-rate } x$

<proof>

lemma *nonneg-escape-rate[simp]*: $0 \leq \text{escape-rate } x$

<proof>

lemma *prob-space-exponential-escape-rate*: *prob-space* $(\text{exponential } (\text{escape-rate } x))$

<proof>

lemma *measurable-escape-rate[measurable]*: $\text{escape-rate} \in \text{count-space UNIV} \rightarrow_M \text{borel}$

<proof>

lemma *measurable-exponential-escape-rate*[*measurable*]: $(\lambda x. \text{exponential} (\text{escape-rate } x)) \in \text{count-space UNIV} \rightarrow_M \text{prob-algebra borel}$
 ⟨*proof*⟩

interpretation *pmf-as-function* ⟨*proof*⟩

lift-definition *J* :: $'a \Rightarrow 'a$ **pmf is** $\lambda x y. R x y / \text{escape-rate } x$
 ⟨*proof*⟩

lemma *set-pmf-J*: $\text{set-pmf} (J x) = I x$
 ⟨*proof*⟩

interpretation *exp-esc*: *pair-prob-space distr* $(\text{exponential} (\text{escape-rate } x)) \text{ borel}$
 $((+) t) J x$ **for** x
 ⟨*proof*⟩

7.4 Continuous-time Kernel

definition *K* :: $(\text{real} \times 'a) \Rightarrow (\text{real} \times 'a)$ **measure where**
 $K = (\lambda(t, x). (\text{distr} (\text{exponential} (\text{escape-rate } x)) \text{ borel} ((+) t)) \otimes_M J x)$

interpretation *K*: *discrete-Markov-process borel* $\otimes_M \text{count-space UNIV } K$
 ⟨*proof*⟩

interpretation *DTMC*: *MC-syntax J* ⟨*proof*⟩

lemma *in-space-S*[*simp*]: $x \in \text{space } S$
 ⟨*proof*⟩

lemma *in-space-T*[*simp*]: $x \in \text{space } T$
 ⟨*proof*⟩

lemma *in-space-lim-stream*: $\omega \in \text{space} (K.\text{lim-stream } x)$
 ⟨*proof*⟩

lemma *prob-space-K-lim*: *prob-space* $(K.\text{lim-stream } x)$
 ⟨*proof*⟩

definition *select-first* :: $'a \Rightarrow ('a \Rightarrow \text{real}) \Rightarrow 'a \Rightarrow \text{bool}$
where $\text{select-first } x p y = (y \in I x \wedge (\forall y' \in I x - \{y\}. p y < p y'))$

lemma *select-firstD1*: $\text{select-first } x p y \Longrightarrow y \in I x$
 ⟨*proof*⟩

lemma *select-first-unique*:

assumes $y: \text{select-first } x p y1 \text{ select-first } x p y2$ **shows** $y1 = y2$
 ⟨*proof*⟩

lemma *The-select-first[simp]*: $\text{select-first } x \ p \ y \implies \text{The } (\text{select-first } x \ p) = y$
 ⟨proof⟩

lemma *select-first-INF*:
 $\text{select-first } x \ p \ y \implies (\text{INF } x \in I \ x. \ p \ x) = p \ y$
 ⟨proof⟩

lemma *measurable-select-first[measurable]*:
 $(\lambda p. \text{select-first } x \ p \ y) \in (\prod_M y \in I \ x. \ \text{borel}) \rightarrow_M \text{count-space UNIV}$
 ⟨proof⟩

lemma *measurable-THE-select-first[measurable]*:
 $(\lambda p. \text{The } (\text{select-first } x \ p)) \in (\prod_M y \in I \ x. \ \text{borel}) \rightarrow_M \text{count-space UNIV}$
 ⟨proof⟩

lemma *sets-S-eq*: $S = \text{sigma-sets UNIV } \{ \{t \ ..\} \times A \mid t \ A. \ A \subseteq - \ I \ x \ \vee (\exists s \in I \ x. \ A = \{s\}) \}$
 ⟨proof⟩

7.5 Kernel equals Parallel Choice

abbreviation *PAR* :: 'a \Rightarrow ('a \Rightarrow real) measure

where

$\text{PAR } x \equiv (\prod_M y \in I \ x. \ \text{exponential } (R \ x \ y))$

lemma *PAR-least*:

assumes $y: y \in I \ x$

shows $\text{PAR } x \ \{p \in \text{space } (\text{PAR } x). \ t \leq p \ y \wedge \text{select-first } x \ p \ y\} =$

$\text{emeasure } (\text{exponential } (\text{escape-rate } x)) \ \{t \ ..\} \ * \ \text{ennreal } (\text{pmf } (J \ x) \ y)$

⟨proof⟩

lemma *AE-PAR-least*: $\text{AE } p \ \text{in } \text{PAR } x. \ \exists y \in I \ x. \ \text{select-first } x \ p \ y$

⟨proof⟩

lemma *K-alt*: $K \ (t, x) = \text{distr } (\prod_M y \in I \ x. \ \text{exponential } (R \ x \ y)) \ S \ (\lambda p. \ (t + (\text{INF } y \in I \ x. \ p \ y), \ \text{The } (\text{select-first } x \ p))) \ (\text{is } - = ?R)$

⟨proof⟩

lemma *AE-K*: $\text{AE } y \ \text{in } K \ x. \ \text{fst } x < \text{fst } y \wedge \text{snd } y \in J \ (\text{snd } x)$

⟨proof⟩

lemma *AE-lim-stream*:

$\text{AE } \omega \ \text{in } K.\text{lim-stream } x. \ \forall i. \ \text{snd } ((x \ \#\# \ \omega) \ !! \ i) \in \text{DTMC}.\text{acc}^{\{\text{snd } x\}} \wedge \text{snd } (\omega \ !! \ i) \in J \ (\text{snd } ((x \ \#\# \ \omega) \ !! \ i)) \wedge \text{fst } ((x \ \#\# \ \omega) \ !! \ i) < \text{fst } (\omega \ !! \ i)$

$(\text{is } \text{AE } \omega \ \text{in } K.\text{lim-stream } x. \ \forall i. \ ?P \ \omega \ i)$

⟨proof⟩

lemma *measurable-merge-at[measurable]*: $(\lambda(\omega, \omega'). \ \text{merge-at } \omega \ j \ \omega') \in (T \otimes_M T) \rightarrow_M T$

<proof>

lemma *measurable-trace-at*[*measurable*]: $(\lambda(s, \omega). \text{trace-at } s \ \omega \ j) \in (\text{count-space } UNIV \otimes_M T) \rightarrow_M \text{count-space } UNIV$
<proof>

lemma *measurable-trace-at'*: $(\lambda((s, j), \omega). \text{trace-at } s \ \omega \ j) \in ((\text{count-space } UNIV \otimes_M \text{borel}) \otimes_M T) \rightarrow_M \text{count-space } UNIV$
<proof>

lemma *K-time-split*:

assumes $t \leq j$ **and** [*measurable*]: $f \in S \rightarrow_M \text{borel}$

shows $(\int^+ x. f \ x * \text{indicator } \{j <..\} (fst \ x) \ \partial K \ (t, s)) = (\int^+ x. f \ x \ \partial K \ (j, s)) * \text{exponential } (\text{escape-rate } s) \ \{j - t <..\}$

<proof>

lemma *K-in-space*[*simp*]: $K \ x \in \text{space } (\text{prob-algebra } S)$
<proof>

lemma *L-in-space*[*simp*]: $K.\text{lim-stream } x \in \text{space } (\text{prob-algebra } T)$
<proof>

7.6 Markov Chain Property

lemma *lim-time-split*:

$t \leq j \implies K.\text{lim-stream } (t, s) = \text{do } \{ \omega \leftarrow K.\text{lim-stream } (t, s) ; \omega' \leftarrow K.\text{lim-stream } (j, \text{trace-at } s \ \omega \ j) ; \text{return } T \ (\text{merge-at } \omega \ j \ \omega') \}$

(**is** $- \implies - = ?DO \ t \ s$)

<proof>

lemma *K-eq*: $K \ (t, s) = \text{distr } (\text{exponential } (\text{escape-rate } s) \otimes_M J \ s) \ S \ (\lambda(t', s). (t + t', s))$
<proof>

lemma *K-shift*: $K \ (t + t', s) = \text{distr } (K \ (t, s)) \ S \ (\lambda(t, s). (t + t', s))$
<proof>

lemma *K-not-empty*: $\text{space } (K \ x) \neq \{\}$
<proof>

lemma *lim-stream-not-empty*: $\text{space } (K.\text{lim-stream } x) \neq \{\}$
<proof>

lemma *lim-shift*: — Generalize to bijective function on $K.\text{lim-stream}$ invariant on K

$K.\text{lim-stream } (t + t', s) = \text{distr } (K.\text{lim-stream } (t, s)) \ T \ (\text{smap } (\lambda(t, s). (t + t', s)))$

(**is** $- = ?D \ t \ s$)

<proof>

lemma *lim-0*: $K.\text{lim-stream } (t, s) = \text{distr } (K.\text{lim-stream } (0, s)) \ T \ (\text{smap } (\lambda(t', s). (t' + t, s)))$
 ⟨proof⟩

7.7 Explosion time

definition *explosion* :: $(\text{real} \times 'a) \text{ stream} \Rightarrow \text{ereal}$
 where $\text{explosion } \omega = (\text{SUP } i. \text{ereal } (\text{fst } (\omega \ !! \ i)))$

lemma *ball-less-Suc-eq*: $(\forall i < \text{Suc } n. P \ i) \longleftrightarrow (P \ 0 \wedge (\forall i < n. P \ (\text{Suc } i)))$
 ⟨proof⟩

lemma *lim-stream-timediff-eq-exponential-1*:
 $\text{distr } (K.\text{lim-stream } ts) \ (\text{PiM } \text{UNIV } (\lambda-. \text{borel}))$
 $(\lambda \omega \ i. \text{escape-rate } (\text{snd } ((ts \ ## \ \omega) \ !! \ i)) * (\text{fst } (\omega \ !! \ i) - \text{fst } ((ts \ ## \ \omega) \ !! \ i))) =$
 $\text{PiM } \text{UNIV } (\lambda-. \text{exponential } 1)$
 (is ?D = ?P)
 ⟨proof⟩

lemma *AE-explosion-infity*:
 assumes *bdd*: *bdd-above* (range escape-rate)
 shows *AE* ω in $K.\text{lim-stream } x. \text{explosion } \omega = \infty$
 ⟨proof⟩

7.8 Transition probability p_t

context
begin

declare $[[\text{inductive-internals} = \text{true}]]$

inductive *trace-in* :: $'a \text{ set} \Rightarrow \text{real} \Rightarrow 'a \Rightarrow (\text{real} \times 'a) \text{ stream} \Rightarrow \text{bool}$ for $S \ t$
where

$t < t' \Longrightarrow s \in S \Longrightarrow \text{trace-in } S \ t \ s \ ((t', s') \ ## \ \omega)$
 $| t \geq t' \Longrightarrow \text{trace-in } S \ t \ s' \ \omega \Longrightarrow \text{trace-in } S \ t \ s \ ((t', s') \ ## \ \omega)$

end

lemma *trace-in-simps[simp]*:
 $\text{trace-in } ss \ t \ s \ (x \ ## \ \omega) = (\text{if } t < \text{fst } x \ \text{then } s \in ss \ \text{else } \text{trace-in } ss \ t \ (\text{snd } x) \ \omega)$
 ⟨proof⟩

lemma *trace-in-eq-lfp*:
 $\text{trace-in } ss \ t = \text{lfp } (\lambda F \ s. \lambda(t', s') \ ## \ \omega \Rightarrow \text{if } t < t' \ \text{then } s \in ss \ \text{else } F \ s' \ \omega)$
 ⟨proof⟩

lemma *trace-in-shiftD*: $\text{trace-in } ss \ t \ s \ \omega \Longrightarrow \text{trace-in } ss \ (t + t') \ s \ (\text{smap } (\lambda(t, s'). (t + t', s')) \ \omega)$
 ⟨proof⟩

lemma *trace-in-shift[simp]*: $\text{trace-in ss } t \ s \ (\text{smap } (\lambda(t, s'). (t + t', s')) \ \omega) \longleftrightarrow \text{trace-in ss } (t - t') \ s \ \omega$
 ⟨proof⟩

lemma *measurable-trace-in'*:

Measurable.pred (borel \otimes_M count-space UNIV \otimes_M T) ($\lambda(t, s, \omega). \text{trace-in ss } t \ s \ \omega$)
 (is ?M ($\lambda(t, s, \omega). \text{trace-in ss } t \ s \ \omega$))
 ⟨proof⟩

lemma *measurable-trace-in[measurable (raw)]*:

assumes [*measurable*]: $f \in M \rightarrow_M \text{borel } g \in M \rightarrow_M \text{count-space UNIV } h \in M \rightarrow_M T$
shows *Measurable.pred M ($\lambda x. \text{trace-in ss } (f \ x) \ (g \ x) \ (h \ x)$)*
 ⟨proof⟩

definition $p :: 'a \Rightarrow 'a \Rightarrow \text{real} \Rightarrow \text{real}$

where $p \ s \ s' \ t = \mathcal{P}(\omega \text{ in } K.\text{lim-stream } (0, s). \text{trace-in } \{s^\frown\} \ t \ s \ \omega)$

lemma *p[measurable]*: $(\lambda(s, t). p \ s \ s' \ t) \in (\text{count-space UNIV } \otimes_M \text{borel}) \rightarrow_M \text{borel}$
 ⟨proof⟩

lemma *p-nonpos*: **assumes** $t \leq 0$ **shows** $p \ s \ s' \ t = \text{of-bool } (s = s')$
 ⟨proof⟩

lemma *p-0*: $p \ s \ s' \ 0 = \text{of-bool } (s = s')$
 ⟨proof⟩

lemma *in-sets-T[measurable (raw)]*: *Measurable.pred T P $\implies \{\omega. P \ \omega\} \in \text{sets } T$*
 ⟨proof⟩

lemma *distr-id'*: *sets M = sets N $\implies \text{distr M N } (\lambda x. x) = M$*
 ⟨proof⟩

lemma *p-nonneg[simp]*: $0 \leq p \ s \ s' \ t$
 ⟨proof⟩

lemma *p-le-1[simp]*: $p \ s \ s' \ t \leq 1$
 ⟨proof⟩

lemma *p-eq*:

assumes $0 \leq t$
shows $p \ s \ s'' \ t = (\text{of-bool } (s = s'')) + (\text{LINT } u:\{0..t\} | \text{borel. escape-rate } s * \text{exp } (\text{escape-rate } s * u) * (\text{LINT } s' | J \ s. p \ s' \ s'' \ u))) / \text{exp } (t * \text{escape-rate } s)$
 ⟨proof⟩

lemma *continuous-on-p*: *continuous-on A (p s s')*

<proof>

lemma *p-vector-derivative*: — Backward equation

assumes $0 \leq t$

shows $(p \ s \ s' \ \text{has-vector-derivative} \ (LINT \ s'' | \text{count-space} \ UNIV. \ R \ s \ s'' * p \ s'' \ s' \ t) - \text{escape-rate} \ s * p \ s \ s' \ t)$

(at t within $\{0..\}$)

(is $(- \ \text{has-vector-derivative} \ ?A) \ -)$)

<proof>

coinductive *wf-times* :: $real \Rightarrow (real \times 'a) \ \text{stream} \Rightarrow bool$

where

$t < t' \Longrightarrow \text{wf-times} \ t' \ \omega \Longrightarrow \text{wf-times} \ t \ ((t', s') \ \#\# \ \omega)$

lemma *wf-times-simp[simp]*: $\text{wf-times} \ t \ (x \ \#\# \ \omega) \longleftrightarrow t < \text{fst} \ x \wedge \text{wf-times} \ (\text{fst} \ x) \ \omega$

<proof>

lemma *trace-in-merge-at*:

assumes ω' : $\text{wf-times} \ t' \ \omega'$

shows $\text{trace-in} \ ss \ t \ x \ (\text{merge-at} \ \omega \ t' \ \omega') \longleftrightarrow$

(if $t < t'$ then $\text{trace-in} \ ss \ t \ x \ \omega$ else $\exists y. \text{trace-in} \ \{y\} \ t' \ x \ \omega \wedge \text{trace-in} \ ss \ t \ y \ \omega'$)

(is $?merge \longleftrightarrow ?cases$)

<proof>

lemma *AE-lim-wf-times*: $AE \ \omega \ \text{in} \ K.\text{lim-stream} \ (t, s). \ \text{wf-times} \ t \ \omega$

<proof>

lemma *wf-times-shiftD*: $\text{wf-times} \ t' \ (\text{smap} \ (\lambda(t', y). \ (t' + t, y)) \ \omega) \Longrightarrow \text{wf-times} \ (t' - t) \ \omega$

<proof>

lemma *wf-times-shift[simp]*: $\text{wf-times} \ t' \ (\text{smap} \ (\lambda(t', y). \ (t' + t, y)) \ \omega) = \text{wf-times} \ (t' - t) \ \omega$

<proof>

lemma *trace-in-unique*: $\text{trace-in} \ \{y1\} \ t \ x \ \omega \Longrightarrow \text{trace-in} \ \{y2\} \ t \ x \ \omega \Longrightarrow y1 = y2$

<proof>

lemma *trace-at-eq*: $\text{trace-in} \ \{z\} \ t \ x \ \omega \Longrightarrow \text{trace-at} \ x \ \omega \ t = z$

<proof>

lemma *AE-lim-acc*: $AE \ \omega \ \text{in} \ K.\text{lim-stream} \ (t, x). \ \forall t \ z. \ \text{trace-in} \ \{z\} \ t \ x \ \omega \longrightarrow (x, z) \in \text{DTMC.}acc$

<proof>

lemma *p-add*:

assumes $0 \leq t \ 0 \leq t'$

shows $p \ x \ y \ (t + t') = (LINT \ z | \text{count-space} \ (\text{DTMC.}acc \ \{\{x\}\}). \ p \ x \ z \ t * p \ z \ y)$


```

t')
⟨proof⟩

end

end

theory Markov-Models
imports
  Markov-Models-Auxiliary
  Discrete-Time-Markov-Chain
  Trace-Space-Equals-Markov-Processes
  Classifying-Markov-Chain-States
  Markov-Decision-Process
  MDP-Reachability-Problem
  Discrete-Time-Markov-Process
  Continuous-Time-Markov-Chain
begin

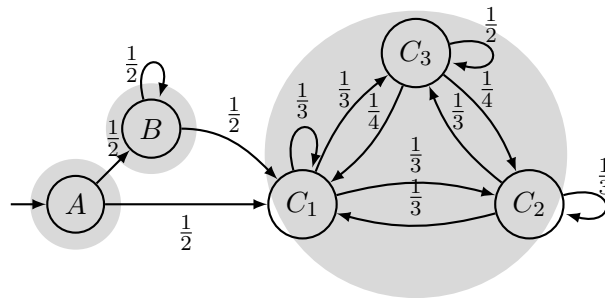
end

theory Example-A
imports ../Classifying-Markov-Chain-States
begin

```

8 Example A

We formalize the following Markov chain:



First we define the state space as its own type:

```
datatype state = A | B | C1 | C2 | C3
```

Now the state space is *UNIV* :: *state set*

```
lemma UNIV-state: UNIV = {A, B, C1, C2, C3}
⟨proof⟩
```

```
instance state :: finite
⟨proof⟩
```

The transition function τ is easily defined using the case statement, this allows us to give a sparse specification as all θ cases are collected at the end.

definition $\tau :: \text{state} \Rightarrow \text{state} \Rightarrow \text{real}$ **where**

```

 $\tau \ s \ t = (\text{case } (s, t) \text{ of}$ 
   $(A, B) \Rightarrow 1 / 2 \mid (A, C1) \Rightarrow 1 / 2$ 
   $\mid (B, B) \Rightarrow 1 / 2 \mid (B, C1) \Rightarrow 1 / 2$ 
   $\mid (C1, C1) \Rightarrow 1 / 3 \mid (C1, C2) \Rightarrow 1 / 3 \mid (C1, C3) \Rightarrow 1 / 3$ 
   $\mid (C2, C1) \Rightarrow 1 / 3 \mid (C2, C2) \Rightarrow 1 / 3 \mid (C2, C3) \Rightarrow 1 / 3$ 
   $\mid (C3, C1) \Rightarrow 1 / 4 \mid (C3, C2) \Rightarrow 1 / 4 \mid (C3, C3) \Rightarrow 1 / 2$ 
   $\mid - \Rightarrow 0)$ 

```

lift-definition $K :: \text{state} \Rightarrow \text{state}$ **pmf is** τ

$\langle \text{proof} \rangle$

We use the *finite-pmf*-locale which introduces the point measure $\tau.M$, and provides us with the necessary simplifier setup.

interpretation $A: \text{MC-syntax } K$ $\langle \text{proof} \rangle$

8.1 The essential class $\{C1, C2, C3\}$

context

begin

interpretation pmf-as-function $\langle \text{proof} \rangle$

lemma $A\text{-E-eq}$:

```

 $\text{set-pmf } (K \ x) = (\text{case } x \text{ of } A \Rightarrow \{B, C1\} \mid B \Rightarrow \{B, C1\} \mid - \Rightarrow \{C1, C2, C3\})$ 
 $\langle \text{proof} \rangle$ 

```

lemma $A\text{-essential}$: $A.\text{essential-class } \{C1, C2, C3\}$

$\langle \text{proof} \rangle$

lemma $A\text{-aperiodic}$: $A.\text{aperiodic } \{C1, C2, C3\}$

$\langle \text{proof} \rangle$

8.2 The stationary distribution n

Similar to τ we introduce n using the *finite-pmf*-locale.

lift-definition $n :: \text{state}$ **pmf is** $\lambda C1 \Rightarrow 0.3 \mid C2 \Rightarrow 0.3 \mid C3 \Rightarrow 0.4 \mid - \Rightarrow 0$

$\langle \text{proof} \rangle$

lemma $\text{stationary-distribution-N}$: $A.\text{stationary-distribution } n$

$\langle \text{proof} \rangle$

lemma exclusive-N[simp] : $\text{set-pmf } n = \{C1, C2, C3\}$

$\langle \text{proof} \rangle$

end

lemma *n-is-limit*:

assumes $x: x \in \{C1, C2, C3\}$ **and** $y: y \in \{C1, C2, C3\}$

shows $(A.p\ x\ y) \longrightarrow pmf\ n\ y$

<proof>

lemma *C-is-pos-recurrent*: $x \in \{C1, C2, C3\} \implies A.pos-recurrent\ x$

<proof>

lemma *C-recurrence-time*:

assumes $x: x \in \{C1, C2, C3\}$

shows $A.U'\ x\ x = 1 / pmf\ n\ x$

<proof>

end

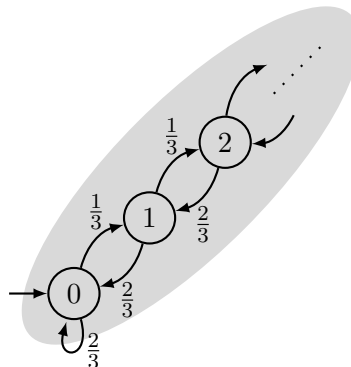
theory *Example-B*

imports *../Classifying-Markov-Chain-States*

begin

9 Example B

We now formalize the following Markov chain:



As state space we have the set of natural numbers, the transition function τ has three cases:

definition $K :: nat \Rightarrow nat\ pmf$ **where**

$K\ x = map-pmf\ (\lambda True \Rightarrow x + 1 \mid False \Rightarrow x - 1)\ (bernoulli-pmf\ (1/3))$

For the special case when $x = 0$ we have $x - 1 = 0$ and hence $\tau\ (0::'b)\ (0::'c) = (2::'a) / (3::'a)$.

We pack this transition function into a discrete Markov kernel.

We call the locale of the Markov chain B , hence all constants and theorems from this Markov chain get a B prefix.

interpretation $B: MC-syntax\ K$ *<proof>*

9.1 Enabled, accessible and communicating states

For each step the predecessor and the successor are enabled (in the $0::'a$ case, the predecessor is again $0::'a$). Hence every state is accessible from everywhere and every states is communicating with each other state. Finally we know that the state space is an essential class.

lemma *B-E-eq*: $set\text{-}pmf\ (K\ x) = \{x - 1, x + 1\}$
<proof>

lemma *B-E-Suc*: $Suc\ x \in set\text{-}pmf\ (K\ x)\ x \in set\text{-}pmf\ (K\ (Suc\ x))$
<proof>

lemma *B-accessible[intro]*: $(i, j) \in B.\text{acc}$
<proof>

lemma *B-communicating[intro]*: $(i, j) \in B.\text{communicating}$
<proof>

lemma *B-essential*: $B.\text{essential-class}\ UNIV$
<proof>

9.2 B is aperiodic

lemma *B-aperiodic*: $B.\text{aperiodic}\ UNIV$
<proof>

9.3 The stationary distribution N

abbreviation $N :: nat\ pmf$ where
 $N \equiv geometric\text{-}pmf\ (1 / 2)$

lemma *stationary-distribution-N*: $B.\text{stationary-distribution}\ N$
<proof>

9.4 Limit behavior and recurrence times

lemma *limit*: $(B.p\ i\ j) \longrightarrow (1/2) \hat{\sim} Suc\ j$
<proof>

lemma *pos-recurrent*: $B.\text{pos-recurrent}\ i$
<proof>

lemma *recurrence-time*: $B.U'\ i\ i = 2 \hat{\sim} Suc\ i$
<proof>

end

theory *PCTL*
imports

../Discrete-Time-Markov-Chain
 Gauss–Jordan–Elim–Fun. Gauss-Jordan-Elim-Fun
 HOL–Library.While-Combinator
 HOL–Library.Monad-Syntax
begin

10 Adapt Gauss-Jordan elimination to DTMCs

locale *Finite-DTMC* =
fixes $K :: 's \Rightarrow 's \text{ pmf}$ **and** $S :: 's \text{ set}$ **and** $\varrho :: 's \Rightarrow \text{real}$ **and** $\iota :: 's \Rightarrow 's \Rightarrow \text{real}$
assumes $\iota\text{-nonneg[simp]}: \bigwedge s t. 0 \leq \iota s t$ **and** $\varrho\text{-nonneg[simp]}: \bigwedge s. 0 \leq \varrho s$
assumes $\text{measurable-}\iota: (\lambda(a, b). \iota a b) \in \text{borel-measurable (count-space UNIV)}$
 $\otimes_M \text{count-space UNIV}$
assumes $\text{finite-}S\text{[simp]}: \text{finite } S$ **and** $S\text{-not-empty}: S \neq \{\}$
assumes $E\text{-closed}: (\bigcup_{s \in S}. \text{set-pmf } (K s)) \subseteq S$
begin

lemma $\text{measurable-}\iota'$ [*measurable (raw)*]:
 $f \in \text{measurable } M \text{ (count-space UNIV)} \implies g \in \text{measurable } M \text{ (count-space UNIV)} \implies$
 $(\lambda x. \iota (f x) (g x)) \in \text{borel-measurable } M$
<proof>

lemma $\text{measurable-}\varrho$ [*measurable*]: $\varrho \in \text{borel-measurable (count-space UNIV)}$
<proof>

sublocale $R?$: *MC-with-rewards* $K \iota \varrho$
<proof>

lemma *single-l*:
fixes s **and** $x :: \text{real}$ **assumes** $s \in S$
shows $(\sum_{s' \in S}. (\text{if } s' = s \text{ then } 1 \text{ else } 0) * l s') = x \longleftrightarrow l s = x$
<proof>

definition $\text{order} = (\text{SOME } f. \text{bij-betw } f \{.. < \text{card } S\} S)$

lemma
shows $\text{bij-order[simp]}: \text{bij-betw } \text{order} \{.. < \text{card } S\} S$
and $\text{inj-order[simp]}: \text{inj-on } \text{order} \{.. < \text{card } S\}$
and $\text{image-order[simp]}: \text{order} ' \{.. < \text{card } S\} = S$
and $\text{order-}S\text{[simp, intro]}: \bigwedge i. i < \text{card } S \implies \text{order } i \in S$
<proof>

lemma *order-Ex*:
assumes $s \in S$ **obtains** i **where** $i < \text{card } S$ $s = \text{order } i$
<proof>

definition $\text{iorder} = \text{the-inv-into } \{.. < \text{card } S\} \text{order}$

lemma *bij-iorder*: *bij-betw iorder S {..*card S*}*
 ⟨*proof*⟩

lemma *iorder-image-eq*: *iorder ' S = {..*card S*}*
and *inj-iorder*: *inj-on iorder S*
 ⟨*proof*⟩

lemma *order-iorder*: $\bigwedge s. s \in S \implies \text{order } (iorder\ s) = s$
 ⟨*proof*⟩

definition *gauss-jordan'* :: (*'s* \Rightarrow *'s* \Rightarrow *real*) \Rightarrow (*'s* \Rightarrow *real*) \Rightarrow (*'s* \Rightarrow *real*) *option*
where

gauss-jordan' *M a* = *do* {
let *M'* = ($\lambda i\ j. \text{if } j = \text{card } S \text{ then } a \text{ (order } i) \text{ else } M \text{ (order } i) \text{ (order } j)$);
sol \leftarrow *gauss-jordan* *M'* (*card S*);
Some ($\lambda i. \text{sol } (iorder\ i) \text{ (card } S)$)
 }

lemma *gauss-jordan'-correct*:
assumes *gauss-jordan'* *M a* = *Some f*
shows $\forall s \in S. (\sum s' \in S. M\ s\ s' * f\ s') = a\ s$
 ⟨*proof*⟩

lemma *gauss-jordan'-complete*:
assumes *exists*: $\forall s \in S. (\sum s' \in S. M\ s\ s' * x\ s') = a\ s$
assumes *unique*: $\bigwedge y. \forall s \in S. (\sum s' \in S. M\ s\ s' * y\ s') = a\ s \implies \forall s \in S. y\ s = x\ s$
shows $\exists y. \text{gauss-jordan}'\ M\ a = \text{Some } y$
 ⟨*proof*⟩

end

11 pCTL model checking

11.1 Syntax

datatype *realrel* = *LessEqual* | *Less* | *Greater* | *GreaterEqual* | *Equal*

datatype *'s sform* = *true*
 | *Label 's set*
 | *Neg 's sform*
 | *And 's sform 's sform*
 | *Prob realrel real 's pform*
 | *Exp realrel real 's eform*
and *'s pform* = *X 's sform*
 | *U nat 's sform 's sform*
 | *UInfinity 's sform 's sform (U[∞])*
and *'s eform* = *Cumm nat (C[≤])*
 | *State nat (I⁼)*
 | *Future 's sform*

primrec *bound-until* **where**

bound-until 0 $\varphi \psi = \psi$
| *bound-until* (*Suc* *n*) $\varphi \psi = \psi$ or (φ and *next* (*bound-until* *n* $\varphi \psi$))

lemma *measurable-bound-until*[*measurable*]:

assumes [*measurable*]: *Measurable.pred* (*stream-space* *M*) φ *Measurable.pred* (*stream-space* *M*) ψ

shows *Measurable.pred* (*stream-space* *M*) (*bound-until* *n* $\varphi \psi$)
⟨*proof*⟩

11.2 Semantics

primrec *inrealrel* :: *realrel* \Rightarrow 'a \Rightarrow ('a::*linorder*) \Rightarrow *bool* **where**

inrealrel *LessEqual* *r q* $\longleftrightarrow q \leq r$ |
inrealrel *Less* *r q* $\longleftrightarrow q < r$ |
inrealrel *Greater* *r q* $\longleftrightarrow q > r$ |
inrealrel *GreaterEqual* *r q* $\longleftrightarrow q \geq r$ |
inrealrel *Equal* *r q* $\longleftrightarrow q = r$

context *Finite-DTMC*

begin

abbreviation *prob* *s P* \equiv *measure* (*T s*) {*x* ∈ *space* (*T s*). *P x*}

abbreviation *E s* \equiv *set-pmf* (*K s*)

primrec *svalid* :: 's *sform* \Rightarrow 's *set*

and *pvalid* :: 's *pform* \Rightarrow 's *stream* \Rightarrow *bool*

and *reward* :: 's *eform* \Rightarrow 's *stream* \Rightarrow *ennreal* **where**

svalid *true* = *S* |
svalid (*Label* *L*) = {*s* ∈ *S*. *s* ∈ *L*} |
svalid (*Neg* *F*) = *S* - *svalid* *F* |
svalid (*And* *F1 F2*) = *svalid* *F1* ∩ *svalid* *F2* |
svalid (*Prob* *rel r F*) = {*s* ∈ *S*. *inrealrel* *rel r* $\mathcal{P}(\omega$ in *T s*. *pvalid* *F* (*s* ## ω)) } |
svalid (*Exp* *rel r F*) = {*s* ∈ *S*. *inrealrel* *rel* (*ennreal* *r*) (\int^+ ω . *reward* *F* (*s* ## ω) ∂ *T s*) } |

pvalid (*X F*) = *next* (*HLD* (*svalid* *F*)) |

pvalid (*U* *k F1 F2*) = *bound-until* *k* (*HLD* (*svalid* *F1*)) (*HLD* (*svalid* *F2*)) |

pvalid (*U*[∞] *F1 F2*) = *HLD* (*svalid* *F1*) *suntil* *HLD* (*svalid* *F2*) |

reward (*C*[≤] *k*) = ($\lambda\omega$. ($\sum_{i < k} \rho(\omega !! i) + \iota(\omega !! i)(\omega !! (\text{Suc } i))$)) |

reward (*I*⁼ *k*) = ($\lambda\omega$. $\rho(\omega !! k)$) |

reward (*Future* *F*) = ($\lambda\omega$. *if ev* (*HLD* (*svalid* *F*)) ω then *reward-until* (*svalid* *F*) (*shd* ω) (*stl* ω) else ∞)

lemma *svalid-subset-S*: *svalid* *F* \subseteq *S*

⟨*proof*⟩

lemma *finite-svalid*[simp, intro]: *finite* (*svalid* F)
 ⟨*proof*⟩

lemma *svalid-sets*[measurable]: *svalid* $F \in \text{sets}$ (*count-space* S)
 ⟨*proof*⟩

lemma *pvalid-sets*[measurable]: *Measurable.pred* $R.S$ (*pvalid* F)
 ⟨*proof*⟩

lemma *reward-measurable*[measurable]: *reward* $F \in \text{borel-measurable}$ $R.S$
 ⟨*proof*⟩

11.3 Implementation of *Sat*

11.3.1 *Prob0*

definition *Prob0* **where**

Prob0 $\Phi \Psi = S - \text{while } (\lambda R. \exists s \in \Phi. R \cap E s \neq \{\} \wedge s \notin R) (\lambda R. R \cup \{s \in \Phi. R \cap E s \neq \{\}\}) \Psi$

lemma *Prob0-subset-S*: *Prob0* $\Phi \Psi \subseteq S$
 ⟨*proof*⟩

lemma *Prob0-iff-reachable*:

assumes $\Phi \subseteq S \Psi \subseteq S$

shows *Prob0* $\Phi \Psi = \{s \in S. ((\text{SIGMA } x:\Phi. E x)^* \{s\}) \cap \Psi = \{\}\}$ (**is** $- = ?U$)

⟨*proof*⟩

lemma *Prob0-iff*:

assumes $\Phi \subseteq S \Psi \subseteq S$

shows *Prob0* $\Phi \Psi = \{s \in S. \text{AE } \omega \text{ in } T s. \neg (\text{HLD } \Phi \text{ suntil } \text{HLD } \Psi) (s \#\#\ \omega)\}$
 (**is** $- = ?U$)

⟨*proof*⟩

lemma *E-rtrancl-closed*:

assumes $s \in S (s, t) \in (\text{SIGMA } x:A. B x)^* \wedge x. x \in A \implies B x \subseteq E x$ **shows** $t \in S$

⟨*proof*⟩

11.3.2 *Prob1*

definition *Prob1* **where**

Prob1 $Y \Phi \Psi = \text{Prob0 } (\Phi - \Psi) Y$

lemma *Prob1-iff*:

assumes $\Phi \subseteq S \Psi \subseteq S$

shows *Prob1* (*Prob0* $\Phi \Psi$) $\Phi \Psi = \{s \in S. \text{AE } \omega \text{ in } T s. (\text{HLD } \Phi \text{ suntil } \text{HLD } \Psi) (s \#\#\ \omega)\}$

(**is** *Prob1* $?P0 - - = \{s \in S. ?pU s\}$)

⟨*proof*⟩

11.3.3 *ProbU, ExpCumm, and ExpState*

abbreviation $\tau s t \equiv pmf (K s) t$

fun *ProbU* :: 's \Rightarrow nat \Rightarrow 's set \Rightarrow 's set \Rightarrow real **where**
ProbU q 0 S1 S2 = (if q \in S2 then 1 else 0) |
ProbU q (Suc k) S1 S2 =
 (if q \in S1 - S2 then ($\sum q' \in S. \tau q q' * ProbU q' k S1 S2$)
 else if q \in S2 then 1 else 0)

fun *ExpCumm* :: 's \Rightarrow nat \Rightarrow ennreal **where**
ExpCumm s 0 = 0 |
ExpCumm s (Suc k) = $\varrho s + (\sum s' \in S. \tau s s' * (\iota s s' + ExpCumm s' k))$

fun *ExpState* :: 's \Rightarrow nat \Rightarrow ennreal **where**
ExpState s 0 = ϱs |
ExpState s (Suc k) = ($\sum s' \in S. \tau s s' * ExpState s' k$)

11.3.4 *LES*

definition *LES* :: 's set \Rightarrow 's \Rightarrow 's \Rightarrow real **where**
LES F r c =
 (if r \in F then (if c = r then 1 else 0)
 else (if c = r then $\tau r c - 1$ else $\tau r c$))

11.3.5 *ProbUinfy, compute unbounded until*

definition *ProbUinfy* :: 's set \Rightarrow 's set \Rightarrow ('s \Rightarrow real) option **where**
ProbUinfy S1 S2 = gauss-jordan' (*LES* (Prob0 S1 S2 \cup S2))
 ($\lambda i. \text{if } i \in S2 \text{ then } 1 \text{ else } 0$)

11.3.6 *ExpFuture, compute unbounded reward*

definition *ExpFuture* :: 's set \Rightarrow ('s \Rightarrow ennreal) option **where**
ExpFuture F = do {
 let N = Prob0 S F ;
 let Y = Prob1 N S F ;
 sol \leftarrow gauss-jordan' (*LES* (S - Y \cup F))
 ($\lambda i. \text{if } i \in Y \wedge i \notin F \text{ then } -\varrho i - (\sum s' \in S. \tau i s' * \iota i s') \text{ else } 0$) ;
 Some ($\lambda s. \text{if } s \in Y \text{ then ennreal (sol s) else } \infty$)
 }

11.3.7 *Sat*

fun *Sat* :: 's sform \Rightarrow 's set option **where**
Sat true = Some S |
Sat (Label L) = Some {s \in S. s \in L} |
Sat (Neg F) = do { F \leftarrow Sat F ; Some (S - F) } |
Sat (And F1 F2) = do { F1 \leftarrow Sat F1 ; F2 \leftarrow Sat F2 ; Some (F1 \cap F2) } |

$Sat (Prob\ rel\ r\ (X\ F)) = do\ \{ F \leftarrow Sat\ F ; Some\ \{ q \in S.\ inrealrel\ rel\ r\ (\sum\ q' \in F.\ \tau\ q\ q') \} \} |$
 $Sat (Prob\ rel\ r\ (U\ k\ F1\ F2)) = do\ \{ F1 \leftarrow Sat\ F1 ; F2 \leftarrow Sat\ F2 ; Some\ \{ q \in S.\ inrealrel\ rel\ r\ (ProbU\ q\ k\ F1\ F2) \} \} |$
 $Sat (Prob\ rel\ r\ (U^\infty\ F1\ F2)) = do\ \{ F1 \leftarrow Sat\ F1 ; F2 \leftarrow Sat\ F2 ; P \leftarrow ProbUinfty\ F1\ F2 ; Some\ \{ q \in S.\ inrealrel\ rel\ r\ (P\ q) \} \} |$

$Sat (Exp\ rel\ r\ (Cumm\ k)) = Some\ \{ s \in S.\ inrealrel\ rel\ r\ (ExpCumm\ s\ k) \} |$
 $Sat (Exp\ rel\ r\ (State\ k)) = Some\ \{ s \in S.\ inrealrel\ rel\ r\ (ExpState\ s\ k) \} |$
 $Sat (Exp\ rel\ r\ (Future\ F)) = do\ \{ F \leftarrow Sat\ F ; E \leftarrow ExpFuture\ F ; Some\ \{ q \in S.\ inrealrel\ rel\ (ennreal\ r)\ (E\ q) \} \}$

lemma *prob-sum*:

$s \in S \implies Measurable.pred\ R.S\ P \implies \mathcal{P}(\omega\ in\ T\ s.\ P\ \omega) = (\sum\ t \in S.\ \tau\ s\ t * \mathcal{P}(\omega\ in\ T\ t.\ P\ (t\ \#\#\ \omega)))$
<proof>

lemma *nn-integral-eq-sum*:

$s \in S \implies f \in borel-measurable\ R.S \implies (\int^{+x}.\ f\ x\ \partial T\ s) = (\sum\ t \in S.\ \tau\ s\ t * (\int^{+x}.\ f\ (t\ \#\#\ x)\ \partial T\ t))$
<proof>

lemma *T-space[simp]*: $measure\ (T\ s)\ (space\ R.S) = 1$
<proof>

lemma *emeasure-T-space[simp]*: $emeasure\ (T\ s)\ (space\ R.S) = 1$
<proof>

lemma *τ -distr[simp]*: $s \in S \implies (\sum\ t \in S.\ \tau\ s\ t) = 1$
<proof>

lemma *ProbU*:

$q \in S \implies ProbU\ q\ k\ (svalid\ F1)\ (svalid\ F2) = \mathcal{P}(\omega\ in\ T\ q.\ pvalid\ (U\ k\ F1\ F2)\ (q\ \#\#\ \omega))$
<proof>

lemma *Prob0-imp-not-Psi*:

assumes $\Phi \subseteq S\ \Psi \subseteq S\ s \in Prob0\ \Phi\ \Psi$ **shows** $s \notin \Psi$
<proof>

lemma *Psi-imp-not-Prob0*:

assumes $\Phi \subseteq S\ \Psi \subseteq S$ **shows** $s \in \Psi \implies s \notin Prob0\ \Phi\ \Psi$
<proof>

11.3.8 Finite expected reward

abbreviation $s0 \equiv SOME\ s.\ s \in S$

lemma *s0-in-S*: $s0 \in S$

<proof>

lemma *nn-integral-reward-finite*:

assumes $s \in S$

assumes *until*: $AE \omega \text{ in } T s. (HLD S \text{ until } HLD (svalid F)) (s \#\#\omega)$

shows $(\int^+ \omega. \text{reward } (Future F) (s \#\#\omega) \partial T s) \neq \infty$

<proof>

lemma *unique*:

assumes *in-S*: $\Phi \subseteq S \Psi \subseteq S N \subseteq S Prob0 \Phi \Psi \subseteq N \Psi \subseteq N$

assumes *l1*: $\bigwedge s. s \in S \implies s \notin N \implies l1 s - c s = (\sum s' \in S. \tau s s' * l1 s')$

assumes *l2*: $\bigwedge s. s \in S \implies s \notin N \implies l2 s - c s = (\sum s' \in S. \tau s s' * l2 s')$

assumes *eq*: $\bigwedge s. s \in N \implies l1 s = l2 s$

shows $\forall s \in S. l1 s = l2 s$

<proof>

lemma *uniqueness-of-ProbU*:

assumes *sol*:

$\forall s \in S. (\sum s' \in S. LES (Prob0 (svalid F1) (svalid F2)) \cup svalid F2) s s' * l s' =$
(if $s \in svalid F2$ *then* 1 *else* 0 *)*

shows $\forall s \in S. l s = \mathcal{P}(\omega \text{ in } T s. pvalid (U^\infty F1 F2) (s \#\#\omega))$

<proof>

lemma *infinite-reward*:

fixes $s F$

defines $N \equiv Prob0 S (svalid F) (\text{is } - \equiv Prob0 S ?F)$

defines $Y \equiv Prob1 N S (svalid F)$

assumes $s: s \in S s \notin Y$

shows $(\int^+ \omega. \text{reward } (Future F) (s \#\#\omega) \partial T s) = \infty$

<proof>

11.3.9 The expected reward implies a unique LES

lemma *existence-of-ExpFuture*:

fixes $s F$

assumes *N-def*: $N \equiv Prob0 S (svalid F) (\text{is } - \equiv Prob0 S ?F)$

assumes *Y-def*: $Y \equiv Prob1 N S (svalid F)$

assumes $s: s \in S s \notin Y$

shows $enn2real (\int^+ \omega. \text{reward } (Future F) (s \#\#\omega) \partial T s) - (\varrho s + (\sum s' \in S. \tau$
 $s s' * \iota s s')) =$

$(\sum s' \in S. \tau s s' * enn2real (\int^+ \omega. \text{reward } (Future F) (s' \#\#\omega) \partial T s'))$

<proof>

lemma *uniqueness-of-ExpFuture*:

fixes F

assumes *N-def*: $N \equiv Prob0 S (svalid F) (\text{is } - \equiv Prob0 S ?F)$

assumes *Y-def*: $Y \equiv Prob1 N S (svalid F)$

assumes *const-def*: $const \equiv \lambda s. \text{if } s \in Y \wedge s \notin \text{svalid } F \text{ then } - \rho s - (\sum s' \in S. \tau s s' * \iota s s') \text{ else } 0$
assumes *sol*: $\bigwedge s. s \in S \implies (\sum s' \in S. LES (S - Y \cup ?F) s s' * \iota s s') = const s$
shows $\forall s \in S. \iota s = \text{enn2real } (\int^+ \omega. \text{reward } (Future F) (s \#\#\ \omega) \partial T s)$
(is $\forall s \in S. \iota s = \text{enn2real } (\int^+ \omega. ?R (s \#\#\ \omega) \partial T s)$
 <proof>

11.4 Soundness of Sat

theorem *Sat-sound*:

$Sat F \neq None \implies Sat F = Some (\text{svalid } F)$
 <proof>

11.5 Completeness of Sat

theorem *Sat-complete*:

$Sat F \neq None$
 <proof>

11.6 Completeness and Soundness Sat

corollary *Sat*: $Sat \Phi = Some (\text{svalid } \Phi)$
 <proof>

end

end

12 Probabilistic Guarded Command Language (pGCL)

theory *PGCL*

imports *../Markov-Decision-Process*
begin

12.1 Syntax

datatype $'s \text{ pgcl} =$

$Skip$
 $| Abort$
 $| Assign 's \Rightarrow 's$
 $| Seq 's \text{ pgcl } 's \text{ pgcl}$
 $| Par 's \text{ pgcl } 's \text{ pgcl}$
 $| If 's \Rightarrow bool 's \text{ pgcl } 's \text{ pgcl}$
 $| Prob bool \text{ pmf } 's \text{ pgcl } 's \text{ pgcl}$
 $| While 's \Rightarrow bool 's \text{ pgcl}$

12.2 Denotational Semantics

primrec $wp :: 's \text{ pgcl} \Rightarrow ('s \Rightarrow \text{ennreal}) \Rightarrow ('s \Rightarrow \text{ennreal})$ **where**

$wp\ Skip\ f = f$
 $| wp\ Abort\ f = (\lambda-. 0)$
 $| wp\ (Assign\ u)\ f = f \circ u$
 $| wp\ (Seq\ c_1\ c_2)\ f = wp\ c_1\ (wp\ c_2\ f)$
 $| wp\ (If\ b\ c_1\ c_2)\ f = (\lambda s. if\ b\ s\ then\ wp\ c_1\ f\ s\ else\ wp\ c_2\ f\ s)$
 $| wp\ (Par\ c_1\ c_2)\ f = wp\ c_1\ f \sqcap wp\ c_2\ f$
 $| wp\ (Prob\ p\ c_1\ c_2)\ f = (\lambda s. pmf\ p\ True * wp\ c_1\ f\ s + pmf\ p\ False * wp\ c_2\ f\ s)$
 $| wp\ (While\ b\ c)\ f = lfp\ (\lambda X\ s. if\ b\ s\ then\ wp\ c\ X\ s\ else\ f\ s)$

lemma *wp-mono*: *mono* (wp c)
 ⟨proof⟩

abbreviation *det* :: 's pgcl \Rightarrow 's \Rightarrow ('s pgcl \times 's) pmf set ($\ll -, - \gg$) **where**
det c s \equiv {return-pmf (c, s)}

12.3 Operational Semantics

fun *step* :: ('s pgcl \times 's) \Rightarrow ('s pgcl \times 's) pmf set **where**
 $step\ (Skip,\ s) = \ll Skip,\ s \gg$
 $| step\ (Abort,\ s) = \ll Abort,\ s \gg$
 $| step\ (Assign\ u,\ s) = \ll Skip,\ u\ s \gg$
 $| step\ (Seq\ c_1\ c_2,\ s) = (map-pmf\ (\lambda(p1',\ s'). (if\ p1' = Skip\ then\ c_2\ else\ Seq\ p1'\ c_2,\ s'))\ 'step\ (c_1,\ s))$
 $| step\ (If\ b\ c_1\ c_2,\ s) = (if\ b\ s\ then\ \ll c_1,\ s \gg\ else\ \ll c_2,\ s \gg)$
 $| step\ (Par\ c_1\ c_2,\ s) = \ll c_1,\ s \gg \cup \ll c_2,\ s \gg$
 $| step\ (Prob\ p\ c_1\ c_2,\ s) = \{map-pmf\ (\lambda b. if\ b\ then\ (c_1,\ s)\ else\ (c_2,\ s))\ p\}$
 $| step\ (While\ b\ c,\ s) = (if\ b\ s\ then\ \ll Seq\ c\ (While\ b\ c),\ s \gg\ else\ \ll Skip,\ s \gg)$

lemma *step-finite*: *finite* (step x)
 ⟨proof⟩

lemma *step-non-empty*: *step* x \neq {}
 ⟨proof⟩

interpretation *step*: *Markov-Decision-Process* *step*
 ⟨proof⟩

definition *rF* :: ('s \Rightarrow ennreal) \Rightarrow (('s pgcl \times 's) stream \Rightarrow ennreal) \Rightarrow ('s pgcl \times 's) stream \Rightarrow ennreal **where**
 $rF\ f\ F\ \omega = (if\ fst\ (shd\ \omega) = Skip\ then\ f\ (snd\ (shd\ \omega))\ else\ F\ (stl\ \omega))$

abbreviation *r* :: ('s \Rightarrow ennreal) \Rightarrow ('s pgcl \times 's) stream \Rightarrow ennreal **where**
r f \equiv lfp (rF f)

lemma *continuous-rF*: *sup-continuous* (rF f)
 ⟨proof⟩

lemma *mono-rF*: *mono* (rF f)
 ⟨proof⟩

lemma *r-unfold*: $r f \omega = (\text{if fst } (\text{shd } \omega) = \text{Skip then } f (\text{snd } (\text{shd } \omega)) \text{ else } r f (\text{stl } \omega))$
 ⟨proof⟩

lemma *mono-r*: $F \leq G \implies r F \omega \leq r G \omega$
 ⟨proof⟩

lemma *measurable-rF*:
assumes $F[\text{measurable}]$: $F \in \text{borel-measurable step.St}$
shows $rF f F \in \text{borel-measurable step.St}$
 ⟨proof⟩

lemma *measurable-r[measurable]*: $r f \in \text{borel-measurable step.St}$
 ⟨proof⟩

lemma *mono-r'*: $\text{mono } (\lambda F s. \prod_{D \in \text{step } s}. \int^+ t. (\text{if fst } t = \text{Skip then } f (\text{snd } t) \text{ else } F t) \partial \text{measure-pmf } D)$
 ⟨proof⟩

lemma *E-inf-r*:
 $\text{step.E-inf } s (r f) =$
 $\text{lfp } (\lambda F s. \prod_{D \in \text{step } s}. \int^+ t. (\text{if fst } t = \text{Skip then } f (\text{snd } t) \text{ else } F t) \partial \text{measure-pmf } D) s$
 ⟨proof⟩

lemma *E-inf-r-unfold*:
 $\text{step.E-inf } s (r f) = (\prod_{D \in \text{step } s}. \int^+ t. (\text{if fst } t = \text{Skip then } f (\text{snd } t) \text{ else } \text{step.E-inf } t (r f)) \partial \text{measure-pmf } D)$
 ⟨proof⟩

lemma *E-inf-r-induct[consumes 1, case-names step]*:
assumes $P s y$
assumes *: $\bigwedge F s y. P s y \implies$
 $(\bigwedge s y. P s y \implies F s \leq y) \implies (\bigwedge s. F s \leq \text{step.E-inf } s (r f)) \implies$
 $(\prod_{D \in \text{step } s}. \int^+ t. (\text{if fst } t = \text{Skip then } f (\text{snd } t) \text{ else } F t) \partial \text{measure-pmf } D) \leq$
 y
shows $\text{step.E-inf } s (r f) \leq y$
 ⟨proof⟩

lemma *E-inf-Skip*: $\text{step.E-inf } (\text{Skip}, s) (r f) = f s$
 ⟨proof⟩

lemma *E-inf-Seq*:
assumes $[\text{simp}]$: $\bigwedge x. 0 \leq f x$
shows $\text{step.E-inf } (\text{Seq } a b, s) (r f) = \text{step.E-inf } (a, s) (r (\lambda s. \text{step.E-inf } (b, s) (r f)))$
 ⟨proof⟩

lemma *E-inf-While*:

step.E-inf (While g c, s) (r f) =
lfp (λF s. if g s then step.E-inf (c, s) (r F) else f s) s
 ⟨*proof*⟩

12.4 Equate Both Semantics

lemma *E-inf-r-eq-wp*: *step.E-inf (c, s) (r f) = wp c f s*
 ⟨*proof*⟩

end

13 Formalization of the Crowds-Protocol

theory *Crowds-Protocol*

imports *../Discrete-Time-Markov-Chain*

begin

lemma *cond-prob-nonneg[simp]*: $0 \leq \text{cond-prob } M A B$
 ⟨*proof*⟩

lemma (in *MC-syntax*) *emeasure-suntil-geometric*:

assumes [*measurable*]: *Measurable.pred S P*

assumes $s \in X$ **and** $*[simp]$: $0 \leq p \leq r$

assumes r : $\bigwedge s. s \in X \implies \text{emeasure } (T s) \{\omega \in \text{space } (T s). P \omega\} = \text{ennreal } r$

assumes p : $\bigwedge s. s \in X \implies \text{emeasure } (K s) X = \text{ennreal } p \text{ } p < 1$

assumes $\bigwedge t. AE \omega \text{ in } T t. \neg (P \sqcap (HLD X \sqcap \text{next } (HLD X \text{ until } P))) \omega$

shows $\text{emeasure } (T s) \{\omega \in \text{space } (T s). (HLD X \text{ until } P) \omega\} = r / (1 - p)$

⟨*proof*⟩

13.1 Definition of the Crowds-Protocol

datatype *'a state* = *Start | Init 'a | Mix 'a | End*

lemma *inj-Mix[simp]*: *inj-on Mix A*
 ⟨*proof*⟩

lemma *inj-Init[simp]*: *inj-on Init A*
 ⟨*proof*⟩

lemma *distinct-state-image[simp]*:

$\text{Start} \notin \text{Mix } 'A \text{ } \text{Init } j \notin \text{Mix } 'A \text{ } \text{End} \notin \text{Mix } 'A \text{ } \text{Mix } j \in \text{Mix } 'A \iff j \in A$

$\text{Start} \notin \text{Init } 'A \text{ } \text{Mix } j \notin \text{Init } 'A \text{ } \text{End} \notin \text{Init } 'A \text{ } \text{Init } j \in \text{Init } 'A \iff j \in A$

⟨*proof*⟩

lemma *Init-cut-Mix[simp]*:

$\text{Init } 'H \cap \text{Mix } 'J = \{\}$

⟨*proof*⟩

abbreviation $J\text{ondo } B \equiv \text{Init}'B \cup \text{Mix}'B$

locale $\text{Crowds-Protocol} =$

fixes $J :: 'a \text{ set}$ **and** $C :: 'a \text{ set}$ **and** $p\text{-}f :: \text{real}$ **and** $p\text{-}i :: 'a \Rightarrow \text{real}$

assumes $J\text{-not-empty}: J \neq \{\}$ **and** $\text{finite-}J[\text{simp}]: \text{finite } J$

assumes $C\text{-smaller}: C \subset J$ **and** $C\text{-non-empty}: C \neq \{\}$

assumes $p\text{-}f: 0 < p\text{-}f \text{ and } p\text{-}f < 1$

assumes $p\text{-}i\text{-nonneg}[\text{simp}]: \bigwedge j. j \in J \Rightarrow 0 \leq p\text{-}i j$

assumes $p\text{-}i\text{-distr}: (\sum_{j \in J}. p\text{-}i j) = 1$

assumes $p\text{-}i\text{-}C: \bigwedge j. j \in C \Rightarrow p\text{-}i j = 0$

begin

abbreviation $H :: 'a \text{ set}$ **where**

$H \equiv J - C$

definition $p\text{-}j = 1 / \text{card } J$

lemma $p\text{-}f\text{-nonneg}[\text{simp}]: 0 \leq p\text{-}f \text{ and } p\text{-}f \leq 1$
 $\langle \text{proof} \rangle$

lemma $p\text{-}j\text{-nonneg}[\text{simp}]: 0 \leq p\text{-}j$
 $\langle \text{proof} \rangle$

definition $p\text{-}H = \text{card } H / \text{card } J$

lemma $p\text{-}H\text{-nonneg}[\text{simp}]: 0 \leq p\text{-}H \text{ and } p\text{-}H \leq 1$
 $\langle \text{proof} \rangle$

definition $\text{next-prob} :: 'a \text{ state} \Rightarrow 'a \text{ state} \Rightarrow \text{real}$ **where**

$\text{next-prob } s \ t = (\text{case } (s, t) \text{ of } (\text{Start}, \text{Init } j) \Rightarrow \text{if } j \in H \text{ then } p\text{-}i j \text{ else } 0$
 $\quad | (\text{Init } j, \text{Mix } j') \Rightarrow \text{if } j' \in J \text{ then } p\text{-}j \text{ else } 0$
 $\quad | (\text{Mix } j, \text{Mix } j') \Rightarrow \text{if } j' \in J \text{ then } p\text{-}f * p\text{-}j \text{ else } 0$
 $\quad | (\text{Mix } j, \text{End}) \Rightarrow 1 - p\text{-}f$
 $\quad | (\text{End}, \text{End}) \Rightarrow 1$
 $\quad | - \Rightarrow 0)$

definition $N \ s = \text{embed-pmf } (\text{next-prob } s)$

interpretation $\text{MC-syntax } N \ \langle \text{proof} \rangle$

abbreviation $\mathfrak{P} \equiv T \ \text{Start}$

abbreviation $E \ s \equiv \text{set-pmf } (N \ s)$

lemma $\text{finite-}C[\text{simp}]: \text{finite } C$
 $\langle \text{proof} \rangle$

lemma $\text{sum-}p\text{-}i\text{-}C[\text{simp}]: \text{sum } p\text{-}i \ C = 0$
 $\langle \text{proof} \rangle$

lemma *sum-p-i-H[simp]*: $\text{sum } p\text{-}i\ H = 1$
<proof>

lemma *possible-jondo*:
obtains j **where** $j \in J\ j \notin C\ p\text{-}i\ j \neq 0$
<proof>

lemma *C-le-J[simp]*: $\text{card } C < \text{card } J$
<proof>

lemma *p-H*: $0 < p\text{-}H\ p\text{-}H < 1$
<proof>

lemma *p-H-p-f-pos*: $0 < p\text{-}H * p\text{-}f$
<proof>

lemma *p-H-p-f-less-1*: $p\text{-}H * p\text{-}f < 1$
<proof>

lemma *p-j-pos*: $0 < p\text{-}j$
<proof>

lemma *H-compl*: $1 - p\text{-}H = \text{real } (\text{card } C) / \text{real } (\text{card } J)$
<proof>

lemma *H-compl2*: $1 - p\text{-}H = \text{card } C * p\text{-}j$
<proof>

lemma *H-eq2*: $\text{card } H * p\text{-}j = p\text{-}H$
<proof>

lemma *pmf-next-pmf[simp]*: $\text{pmf } (N\ s)\ t = \text{next-prob } s\ t$
<proof>

lemma *next-prob-Start[simp]*: $\text{next-prob } \text{Start } (\text{Init } j) = (\text{if } j \in H \text{ then } p\text{-}i\ j \text{ else } 0)$
<proof>

lemma *next-prob-to-Init[simp]*: $j \in H \implies \text{next-prob } s\ (\text{Init } j) =$
 $(\text{case } s \text{ of } \text{Start} \Rightarrow p\text{-}i\ j \mid _ \Rightarrow 0)$
<proof>

lemma *next-prob-to-Mix[simp]*: $j \in J \implies \text{next-prob } s\ (\text{Mix } j) =$
 $(\text{case } s \text{ of } \text{Init } j \Rightarrow p\text{-}j \mid \text{Mix } j \Rightarrow p\text{-}f * p\text{-}j \mid _ \Rightarrow 0)$
<proof>

lemma *next-prob-to-End[simp]*: $\text{next-prob } s\ \text{End} =$
 $(\text{case } s \text{ of } \text{Mix } j \Rightarrow 1 - p\text{-}f \mid \text{End} \Rightarrow 1 \mid _ \Rightarrow 0)$

<proof>

lemma *next-prob-from-End[simp]*: $\text{next-prob } \text{End } s = 0 \longleftrightarrow s \neq \text{End}$
<proof>

lemma *next-prob-Mix-MixI*: $\exists j. s = \text{Mix } j \implies \exists j \in J. s' = \text{Mix } j \implies \text{next-prob } s$
 $s' = p\text{-f } * p\text{-j}$
<proof>

lemma *E-Start*: $E \text{ Start} = \{ \text{Init } j \mid j. j \in H \wedge p\text{-i } j \neq 0 \}$
<proof>

lemma *E-Init*: $E (\text{Init } j) = \{ \text{Mix } j \mid j. j \in J \}$
<proof>

lemma *E-Mix*: $E (\text{Mix } j) = \{ \text{Mix } j \mid j. j \in J \} \cup \{ \text{End} \}$
<proof>

lemma *E-End*: $E \text{ End} = \{ \text{End} \}$
<proof>

lemma *enabled-End*:
 $\text{enabled } \text{End } \omega \longleftrightarrow \omega = \text{sconst } \text{End}$
<proof>

lemma *AE-End*: $(AE \ \omega \text{ in } T \ \text{End}. P \ \omega) \longleftrightarrow P (\text{sconst } \text{End})$
<proof>

lemma *emeasure-Init-eq-Mix*:
assumes *[measurable]*: $\text{Measurable.pred } S \ P$
assumes *AE-End*: $AE \ x \text{ in } T \ \text{End}. \neg P (\text{End } \#\# \ x)$
shows $\text{emeasure } (T (\text{Init } j)) \{x \in \text{space } (T (\text{Init } j)). P \ x\} =$
 $\text{emeasure } (T (\text{Mix } j)) \{x \in \text{space } (T (\text{Mix } j)). P \ x\} / p\text{-f}$
<proof>

What is the probability that the server sees a specific jondo (including the initiator) as sender.

definition *visit* :: 'a set \Rightarrow 'a set \Rightarrow 'a state stream \Rightarrow bool **where**
 $\text{visit } I \ L = \text{Init}'(I \cap H) \cdot (\text{HLD } (\text{Mix}'J) \ \text{suntil } (\text{Mix}'(L \cap J) \cdot \text{HLD } \{ \text{End} \}))$

lemma *visit-unique1*:
 $\text{visit } I1 \ L1 \ \omega \implies \text{visit } I2 \ L2 \ \omega \implies I1 \cap I2 \neq \{ \}$
<proof>

lemma *visit-unique2*:
assumes $\text{visit } I1 \ L1 \ \omega \ \text{visit } I2 \ L2 \ \omega$
shows $L1 \cap L2 \neq \{ \}$
<proof>

lemma *visit-imp-in-H*: $\text{visit } \{i\} J \omega \implies i \in H$
 ⟨proof⟩

lemma *emeasure-visit*:

assumes $I: I \subseteq H$ and $L: L \subseteq J$

shows $\text{emeasure } \mathfrak{P} \{ \omega \in \text{space } \mathfrak{P}. \text{visit } I L \omega \} = (\sum_{i \in I} p\text{-}i \ i) * (\text{card } L * p\text{-}j)$
 ⟨proof⟩

lemma *measurable-visit*[*measurable*]: *Measurable.pred S (visit I L)*
 ⟨proof⟩

lemma *AE-visit*: $AE \ \omega \ \text{in } \mathfrak{P}. \text{visit } H J \ \omega$
 ⟨proof⟩

13.2 Server gets no information

lemma *server-view1*: $j \in J \implies \mathcal{P}(\omega \ \text{in } \mathfrak{P}. \text{visit } H \ \{j\} \ \omega) = p\text{-}j$
 ⟨proof⟩

lemma *server-view-indep*:

$L \subseteq J \implies I \subseteq H \implies \mathcal{P}(\omega \ \text{in } \mathfrak{P}. \text{visit } I L \ \omega) = \mathcal{P}(\omega \ \text{in } \mathfrak{P}. \text{visit } H L \ \omega) * \mathcal{P}(\omega \ \text{in } \mathfrak{P}. \text{visit } I J \ \omega)$
 ⟨proof⟩

lemma *server-view*: $\mathcal{P}(\omega \ \text{in } \mathfrak{P}. \exists j \in H. \text{visit } \{j\} \ \{j\} \ \omega) = p\text{-}j$
 ⟨proof⟩

13.3 Probability that collaborators gain information

definition *hit-C* = *Init' H · ev (HLD (Mix' C))*

definition *before-C B* = *(HLD (Jondo H)) suntil ((Jondo (B ∩ H)) · HLD (Mix' C))*

lemma *measurable-hit-C*[*measurable*]: *Measurable.pred S hit-C*
 ⟨proof⟩

lemma *measurable-before-C*[*measurable*]: *Measurable.pred S (before-C B)*
 ⟨proof⟩

lemma *before-C*:

assumes ω : *enabled Start* ω

shows $\text{before-C } B \ \omega \longleftrightarrow$

$((\text{Init}' H \cdot (\text{HLD } (\text{Mix}' H) \ \text{suntil } (\text{Mix}' (B \cap H)) \cdot \text{HLD } (\text{Mix}' C)))) \ \text{or} \ (\text{Init}' (B \cap H) \cdot \text{HLD } (\text{Mix}' C)) \ \omega$

⟨proof⟩

lemma *before-C-unique*:

assumes ω : *before-C I1* ω *before-C I2* ω **shows** $I1 \cap I2 \neq \{\}$

⟨proof⟩

lemma *hit-C-imp-before-C*:
assumes *enabled Start* ω *hit-C* ω **shows** *before-C* $H \omega$
 \langle *proof* \rangle

lemma *before-C-single*:
assumes *before-C* $I \omega$ **shows** $\exists i \in I \cap H. \text{before-C } \{i\} \omega$
 \langle *proof* \rangle

lemma *before-C-imp-in-H*: *before-C* $\{i\} \omega \implies i \in H$
 \langle *proof* \rangle

13.4 The probability that the sender hits a collaborator

lemma *Pr-hit-C*: $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \text{hit-C } \omega) = (1 - p-H) / (1 - p-H * p-f)$
 \langle *proof* \rangle

lemma *before-C-imp-hit-C*:
assumes *enabled Start* ω *before-C* $B \omega$
shows *hit-C* ω
 \langle *proof* \rangle

lemma *negE*: $\neg P \implies P \implies \text{False}$
 \langle *proof* \rangle

lemma *Pr-visit-before-C*:
assumes $L: L \subseteq H$ **and** $I: I \subseteq H$
shows $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \text{visit } I J \omega \wedge \text{before-C } L \omega \mid \text{hit-C } \omega) =$
 $(\sum i \in I. p-i i) * \text{card } L * p-j * p-f + (\sum i \in I \cap L. p-i i) * (1 - p-H * p-f)$
 \langle *proof* \rangle

lemma *Pr-visit-eq-before-C*:
 $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \exists j \in H. \text{visit } \{j\} J \omega \wedge \text{before-C } \{j\} \omega \mid \text{hit-C } \omega) = 1 - (p-H - p-j) * p-f$
 \langle *proof* \rangle

lemma *probably-innocent*:
assumes *approx*: $1 / (2 * (p-H - p-j)) \leq p-f$ **and** $p-H \neq p-j$
shows $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \exists j \in H. \text{visit } \{j\} J \omega \wedge \text{before-C } \{j\} \omega \mid \text{hit-C } \omega) \leq 1 / 2$
 \langle *proof* \rangle

lemma *Pr-before-C*:
assumes $L: L \subseteq H$
shows $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \text{before-C } L \omega \mid \text{hit-C } \omega) =$
 $\text{card } L * p-j * p-f + (\sum l \in L. p-i l) * (1 - p-H * p-f)$
 \langle *proof* \rangle

lemma *P-visit*:
assumes $I: I \subseteq H$

shows $\mathcal{P}(\omega \text{ in } \mathfrak{F}. \text{ visit } I J \omega \mid \text{ hit-C } \omega) = (\sum_{i \in I}. p_i)$
 ⟨proof⟩

13.5 Probability space of hitting a collaborator

definition $hC = \text{uniform-measure } \mathfrak{F} \{ \omega \in \text{space } \mathfrak{F}. \text{ hit-C } \omega \}$

lemma *emeasure-hit-C-not-0*: $\text{emeasure } \mathfrak{F} \{ \omega \in \text{space } \mathfrak{F}. \text{ hit-C } \omega \} \neq 0$
 ⟨proof⟩

lemma *measurable-hC*[*measurable (raw)*]:
 $A \in \text{sets } S \implies A \in \text{sets } hC$
 $f \in \text{measurable } M S \implies f \in \text{measurable } M hC$
 $g \in \text{measurable } S M \implies g \in \text{measurable } hC M$
 $A \cap \text{space } S \in \text{sets } S \implies A \cap \text{space } hC \in \text{sets } S$
 ⟨proof⟩

lemma *vimage-Int-space-C*[*simp*]:
 $f^{-1} \{x\} \cap \text{space } hC = \{ \omega \in \text{space } S. f \omega = x \}$
 ⟨proof⟩

sublocale hC : *information-space* hC 2
 ⟨proof⟩

abbreviation
mutual-information-Pow-CP ($\mathcal{I}'(- ; -')$) **where**
 $\mathcal{I}(X ; Y) \equiv hC.\text{mutual-information } 2$ (*count-space* (X' space hC)) (*count-space* (Y' space hC)) $X Y$

lemma *simple-functionI*:
assumes *finite* (*range* f)
assumes [*measurable*]: $\bigwedge x. \{ \omega \in \text{space } S. f \omega = x \} \in \text{sets } S$
shows *simple-function* $hC f$
 ⟨proof⟩

13.6 Estimate the information to the collaborators

lemma *measure-hC*[*simp*]:
assumes A [*measurable*]: $A \in \text{sets } S$
shows $\text{measure } hC A = \mathcal{P}(\omega \text{ in } \mathfrak{F}. \omega \in A \mid \text{ hit-C } \omega)$
 ⟨proof⟩

13.6.1 Setup random variables for mutual information

definition *first-J* $\omega = (\text{THE } i. \text{ visit } \{i\} J \omega)$

lemma *first-J-eq*:
 $\text{visit } \{i\} J \omega \implies \text{first-J } \omega = i$
 ⟨proof⟩

lemma *AE-first-J*:

AE ω in \mathfrak{P} . *visit* $\{i\}$ $J \omega \longleftrightarrow \text{first-}J \omega = i$
<proof>

lemma *measurbale-first-J[measurable]*: *first-J* \in *measurable S* (*count-space UNIV*)

<proof>

definition *last-H* $\omega = (\text{THE } i. \text{before-}C \{i\} \omega)$

lemma *measurbale-last-H[measurable]*: *last-H* \in *measurable S* (*count-space UNIV*)

<proof>

lemma *last-H-eq*:

before-C $\{i\} \omega \implies \text{last-}H \omega = i$
<proof>

lemma *last-H*:

assumes *enabled Start* ω *hit-C* ω

shows *before-C* $\{\text{last-}H \omega\} \omega$ *last-H* $\omega \in H$

<proof>

lemma *AE-last-H*:

AE ω in \mathfrak{P} . *hit-C* $\omega \longrightarrow \text{before-}C \{i\} \omega \longleftrightarrow \text{last-}H \omega = i$
<proof>

lemma *information-flow*:

defines $h \equiv \text{real } (\text{card } H)$

assumes *init-uniform*: $\bigwedge i. i \in H \implies p\text{-}i \ i = 1 / h$

shows $\mathcal{I}(\text{first-}J ; \text{last-}H) \leq (1 - (h - 1) * p\text{-}j * p\text{-}f) * \log 2 \ h$

<proof>

end

end

14 Formalizing the IPv4-address allocation in ZeroConf

theory *Zeroconf-Analysis*

imports *../Discrete-Time-Markov-Chain*

begin

declare *UNIV-bool[simp]*

14.1 Definition of a ZeroConf allocation run

datatype *zc-state* = *start*

| *probe nat*

| *ok*
| *error*

lemma *inj-probe*: *inj-on probe X*
⟨*proof*⟩

Countability of *zc-state* simplifies measurability of functions on *zc-state*.

instance *zc-state* :: *countable*
⟨*proof*⟩

locale *Zeroconf-Analysis* =
 fixes *N* :: *nat* **and** *p q r e* :: *real*
 assumes *p*: $0 < p < 1$ **and** *q*: $0 < q < 1$
 assumes *r[simp]*: $0 \leq r$ **and** *e[simp]*: $0 \leq e$
begin

lemma *p-bounds[simp]*: $0 \leq p < 1$
⟨*proof*⟩

lemma *q-bounds[simp]*: $0 < q < 1$
⟨*proof*⟩

abbreviation *states* **where**
 states \equiv *probe* ‘ $\{.. N\} \cup \{start, ok, error\}$ ’

primrec τ :: *zc-state* \Rightarrow *zc-state* *pmf* **where**
 τ *start* = *map-pmf* ($\lambda True \Rightarrow probe\ 0 \mid False \Rightarrow ok$) (*bernoulli-pmf* *q*)
 τ (*probe* *n*) = *map-pmf* ($\lambda True \Rightarrow (if\ n < N\ then\ probe\ (Suc\ n)\ else\ error) \mid$
 $False \Rightarrow start$) (*bernoulli-pmf* *p*)
 τ *ok* = *return-pmf* *ok*
 τ *error* = *return-pmf* *error*

primrec ρ :: *zc-state* \Rightarrow *zc-state* \Rightarrow *real* **where**
 ρ *start* = ($\lambda-. 0$) (*probe* $0 := r$, *ok* $:= r * (N + 1)$)
 ρ (*probe* *n*) = (*if* $n < N$ *then* ($\lambda-. 0$) (*probe* (*Suc* *n*) $:= r$) *else* ($\lambda-. 0$) (*error* $:=$
 e))
 ρ *ok* = ($\lambda-. 0$) (*ok* $:= 0$)
 ρ *error* = ($\lambda-. 0$) (*error* $:= 0$)

lemma *ρ-nonneg[simp]*: $0 \leq \rho\ s\ t$
⟨*proof*⟩

sublocale *MC-with-rewards* $\tau\ \rho\ \lambda s. 0$
⟨*proof*⟩

14.2 The allocation run is a rewarded DTMC

abbreviation *E* *s* \equiv *set-pmf* ($\tau\ s$)

lemma *enabled-ok*: $\text{enabled ok } \omega \longleftrightarrow \omega = \text{sconst ok}$
 ⟨proof⟩

lemma *finite-E*[*intro, simp*]: $\text{finite } (E s)$
 ⟨proof⟩

lemma *E-closed*: $s \in \text{states} \implies E s \subseteq \text{states}$
 ⟨proof⟩

lemma *enabled-error*: $\text{enabled error } \omega \longleftrightarrow \omega = \text{sconst error}$
 ⟨proof⟩

lemma *pos-neg-q-pn*: $0 < 1 - q * (1 - p \hat{\text{Suc}} N)$
 ⟨proof⟩

lemma *to-error*: **assumes** $n \leq N$ **shows** $(\text{probe } n, \text{error}) \in \text{acc}$
 ⟨proof⟩

14.3 Probability of a erroneous allocation

definition *P-err* $s = \mathcal{P}(\omega \text{ in } T s. \text{ev } (HLD \{\text{error}\}) (s \#\#\omega))$

lemma *P-err*:

defines $p\text{-start} == (q * p \hat{\text{Suc}} N) / (1 - q * (1 - p \hat{\text{Suc}} N))$

defines $p\text{-probe} == (\lambda n. p \hat{\text{Suc}} (N - n) + (1 - p \hat{\text{Suc}} (N - n)) * p\text{-start})$

assumes $s: s \in \text{states} - \{\text{ok}, \text{error}\}$

shows $P\text{-err } s = (\text{case } s \text{ of } \text{ok} \Rightarrow 0 \mid \text{error} \Rightarrow 1 \mid \text{probe } n \Rightarrow p\text{-probe } n \mid \text{start} \Rightarrow p\text{-start})$

(**is** ... = ? $E s$)

⟨proof⟩

lemma *P-err-start*: $P\text{-err start} = (q * p \hat{\text{Suc}} N) / (1 - q * (1 - p \hat{\text{Suc}} N))$
 ⟨proof⟩

14.4 An allocation run terminates almost surely

lemma *states-closed*:

assumes $s \in \text{states}$

assumes $(s, t) \in \text{acc-on } (- \{\text{error}, \text{ok}\})$

shows $t \in \text{states}$

⟨proof⟩

lemma *finite-reached*:

assumes $s: s \in \text{states}$ **shows** $\text{finite } (\text{acc-on } (- \{\text{error}, \text{ok}\}) \text{ `` } \{s\})$

⟨proof⟩

lemma *AE-reaches-error-or-ok*:

assumes $s: s \in \text{states}$

shows $AE \omega \text{ in } T s. \text{ev } (HLD \{\text{error}, \text{ok}\}) \omega$

⟨proof⟩

14.5 Expected runtime of an allocation run

definition $R\ s = (\int^+ \omega. \text{reward-until } \{\text{error}, \text{ok}\} \ s \ \omega \ \partial T\ s)$

definition $R'\ s = \text{enn2real } (R\ s)$

lemma $R\text{-iter}: s \neq \text{error} \implies s \neq \text{ok} \implies R\ s = (\int^+ t. \text{ennreal } (\varrho\ s\ t) + R\ t\ \partial\tau\ s)$
<proof>

lemma $R\text{-finite}$:
assumes $s: s \in \text{states}$
shows $R\ s \neq \infty$
<proof>

lemma $R\text{-less-top}: s \in \text{states} \implies R\ s < \text{top}$
<proof>

lemma $R'\text{-iter}$: **assumes** $s: s \in \text{states}$ $s \neq \text{error}$ $s \neq \text{ok}$ **shows** $R'\ s = (\int^+ t. \varrho\ s\ t + R'\ t\ \partial\tau\ s)$
<proof>

lemma cost-from-start :
 $R'\ \text{start} =$
 $(q * (r + p \widehat{\text{Suc}}\ N * e + r * p * (1 - p \widehat{N}) / (1 - p)) + (1 - q) * (r * \text{Suc}\ N)) /$
 $(1 - q + q * p \widehat{\text{Suc}}\ N)$
<proof>

end

interpretation ZC : $\text{Zeroconf-Analysis } 2\ 16 / 65024 :: \text{real } 0.01\ 0.002\ 3600$
<proof>

lemma $ZC.P\text{-err}\ \text{start} \leq 1 / 10^{12}$
<proof>

lemma $ZC.R'\ \text{start} \leq 0.007$
<proof>

end

15 Formalization of the Gossip-Broadcast

theory Gossip-Broadcast
imports $\dots / \text{Discrete-Time-Markov-Chain}$
begin

lemma inj-on-upd-PiE :

assumes $i \notin I$ **shows** *inj-on* $(\lambda(x,f). f(i := x)) (M \times (\prod_{E \ i \in I}. A \ i))$
 $\langle proof \rangle$

lemma *sum-folded-product*:

fixes $I :: 'i \text{ set}$ **and** $f :: 's \Rightarrow 'i \Rightarrow 'a :: \{semiring-0, comm-monoid-mult\}$
assumes $finite \ I \ \wedge \ i. \ i \in I \implies finite \ (S \ i)$
shows $(\sum_{x \in P^{i_E} \ I \ S}. \prod_{i \in I}. f \ (x \ i) \ i) = (\prod_{i \in I}. \sum_{s \in S} i. f \ s \ i)$
 $\langle proof \rangle$

15.1 Definition of the Gossip-Broadcast

datatype $state = listening \mid sending \mid sleeping$

type-synonym $sys\text{-}state = (nat \times nat) \Rightarrow state$

lemma *state-UNIV*: $UNIV = \{listening, sending, sleeping\}$
 $\langle proof \rangle$

locale *gossip-broadcast* =

fixes $size :: nat$ **and** $p :: real$
assumes $size: 0 < size$
assumes $p: 0 < p \ p < 1$

begin

interpretation *pmf-as-function* $\langle proof \rangle$

definition *states* :: $sys\text{-}state \text{ set}$ **where**

$states = (\{.. < size\} \times \{.. < size\}) \rightarrow_E \{listening, sending, sleeping\}$

definition *start* :: $sys\text{-}state$ **where**

$start = (\lambda x \in \{.. < size\} \times \{.. < size\}. listening)((0, 0) := sending)$

definition *neighbour-sending* **where**

$neighbour\text{-}sending \ s = (\lambda(x,y).$
 $(x > 0 \wedge s \ (x - 1, y) = sending) \vee$
 $(x < size \wedge s \ (x + 1, y) = sending) \vee$
 $(y > 0 \wedge s \ (x, y - 1) = sending) \vee$
 $(y < size \wedge s \ (x, y + 1) = sending))$

definition *node-trans* :: $sys\text{-}state \Rightarrow (nat \times nat) \Rightarrow state \Rightarrow state \Rightarrow real$ **where**

$node\text{-}trans \ g \ x \ s = (case \ s \ of$
 $listening \Rightarrow (if \ neighbour\text{-}sending \ g \ x$
 $then \ (\lambda-.0) \ (sending := p, sleeping := 1 - p)$
 $else \ (\lambda-.0) \ (listening := 1))$
 $\mid sending \Rightarrow (\lambda-.0) \ (sleeping := 1)$
 $\mid sleeping \Rightarrow (\lambda-.0) \ (sleeping := 1))$

lemma *node-trans-sum-eq-1* [*simp*]:

$node\text{-}trans \ g \ x \ s' \ listening + (node\text{-}trans \ g \ x \ s' \ sending + node\text{-}trans \ g \ x \ s')$

sleeping) = 1
 ⟨proof⟩

lemma *node-trans-nonneg[simp]*: $0 \leq \text{node-trans } s \ i \ j$
 ⟨proof⟩

lift-definition *proto-trans* :: *sys-state* \Rightarrow *sys-state pmf* **is**
 $\lambda s \ s'. \text{ if } s' \in \text{states then } (\prod x \in \{.. < \text{size}\} \times \{.. < \text{size}\}. \text{node-trans } s \ x \ (s \ x) \ (s' \ x))$
 else 0
 ⟨proof⟩

end

15.2 The Gossip-Broadcast forms a DTMC

sublocale *gossip-broadcast* \subseteq *MC-syntax proto-trans* ⟨proof⟩

end

16 Certification of Reachability Problems on MDPs

theory *MDP-RP-Certification*

imports

../MDP-Reachability-Problem

HOL-Library.IArray

HOL-Library.Code-Target-Numeral

begin

context *Reachability-Problem*

begin

lemma *p-ub'*:

fixes *x*

assumes 1: $s \in S \wedge s \ D. \ s \in S1 \Longrightarrow D \in K \ s \Longrightarrow (\sum t \in S. \text{pmf } D \ t \ * \ x \ t) \leq x \ s$

assumes 2: $\bigwedge s. \ s \in S1 \Longrightarrow x \ s \neq 0 \Longrightarrow (\exists t \in S2. (s, t) \in (\text{SIGMA } s: S1. \bigcup D \in K \ s. \text{set-pmf } D)^*)$

assumes 3: $\bigwedge s. \ s \in S - S1 - S2 \Longrightarrow x \ s = 0$

assumes 4: $\bigwedge s. \ s \in S2 \Longrightarrow x \ s = 1$

shows *enn2real* (*p s*) $\leq x \ s$

⟨proof⟩

lemma *n-lb'*:

fixes *x*

assumes *wf R*

assumes 1: $s \in S \wedge s \ D. \ s \in S1 \Longrightarrow D \in K \ s \Longrightarrow x \ s \leq (\sum t \in S. \text{pmf } D \ t \ * \ x \ t)$

assumes 2: $\bigwedge s \ D. \ s \in S1 \Longrightarrow D \in K \ s \Longrightarrow x \ s \neq 0 \Longrightarrow \exists t \in D. ((t, s) \in R \wedge t \in S1 \wedge x \ t \neq 0) \vee t \in S2$

assumes 3: $\bigwedge s. \ s \in S - S1 - S2 \Longrightarrow x \ s = 0$

assumes 4: $\bigwedge s. \ s \in S2 \Longrightarrow x \ s = 1$

shows $x s \leq \text{enn2real } (n s)$
 $\langle \text{proof} \rangle$

end

no-notation Stream.snth (**infixl** !! 100) — we use !! for IArray

16.1 Computable representation

record $\text{mdp-reachability-problem} =$
 $\text{state-count} :: \text{nat}$
 $\text{distrs} :: (\text{nat} \times \text{rat}) \text{ list list iarray}$
 $\text{states1} :: \text{bool iarray}$
 $\text{states2} :: \text{bool iarray}$

record $'a \text{ RP-sub-cert} =$
 $\text{solution} :: \text{rat iarray}$
 $\text{witness} :: ('a \times \text{nat}) \text{ iarray}$

record $\text{RP-cert} =$
 $\text{pos-cert} :: (\text{nat} \times \text{nat}) \text{ RP-sub-cert}$
 $\text{neg-cert} :: \text{nat list RP-sub-cert}$

definition $\text{sparse-mult } sx y = \text{sum-list } (\text{map } (\lambda(n, x). x * y !! n) sx)$

primrec lookup where
 $\text{lookup } d [] x = d$
 $| \text{lookup } d (y\#\text{ys}) x = (\text{if } \text{fst } y = x \text{ then } \text{snd } y \text{ else } \text{lookup } d \text{ ys } x)$

lemma $\text{lookup-eq-map-of: lookup } d \text{ xs } x = (\text{case map-of xs } x \text{ of Some } x \Rightarrow x \mid \text{None} \Rightarrow d)$
 $\langle \text{proof} \rangle$

lemma lookup-in-set:
 $\text{distinct } (\text{map } \text{fst } xs) \Longrightarrow x \in \text{set } xs \Longrightarrow \text{lookup } d \text{ xs } (\text{fst } x) = \text{snd } x$
 $\langle \text{proof} \rangle$

lemma $\text{lookup-not-in-set:}$
 $x \notin \text{fst } \text{' set } xs \Longrightarrow \text{lookup } d \text{ xs } x = d$
 $\langle \text{proof} \rangle$

lemma lookup-nonneg:
 $(\bigwedge x v. (x, v) \in \text{set } xs \Longrightarrow 0 \leq v) \Longrightarrow (0 :: 'a :: \text{ordered-comm-monoid-add}) \leq \text{lookup } 0 \text{ xs } x$
 $\langle \text{proof} \rangle$

lemma $\text{sparse-mult-eq-sum-lookup:}$
fixes $xs :: (\text{nat} \times 'a :: \text{comm-semiring-1}) \text{ list}$
assumes $\text{list-all } (\lambda(n, x). n < M) \text{ xs } \text{distinct } (\text{map } \text{fst } xs)$

shows $\text{sparse-mult } xs \ y = (\sum i < M. \text{lookup } 0 \ xs \ i * y \ ! \ i)$
 ⟨proof⟩

lemma *sum-list-eq-sum-lookup*:

fixes $xs :: (\text{nat} \times 'a :: \text{comm-semiring-1}) \text{ list}$
assumes $\text{list-all } (\lambda(n, x). n < M) \ xs \ \text{distinct } (\text{map } \text{fst } xs)$
shows $\text{sum-list } (\text{map } \text{snd } xs) = (\sum i < M. \text{lookup } 0 \ xs \ i)$
 ⟨proof⟩

definition

$\text{valid-mdp-rp } mdp \longleftrightarrow$
 $0 < \text{state-count } mdp \wedge$
 $IArray.length (\text{distrs } mdp) = \text{state-count } mdp \wedge$
 $IArray.length (\text{states1 } mdp) = \text{state-count } mdp \wedge$
 $IArray.length (\text{states2 } mdp) = \text{state-count } mdp \wedge$
 $(\forall i < \text{state-count } mdp. \neg (\text{states1 } mdp \ ! \ i \wedge \text{states2 } mdp \ ! \ i) \wedge$
 $\text{list-all } (\lambda ds. \text{distinct } (\text{map } \text{fst } ds) \wedge \text{list-all } (\lambda(n, x). 0 \leq x \wedge n < \text{state-count}$
 $mdp) \ ds \wedge$
 $\text{sum-list } (\text{map } \text{snd } ds) = 1) (\text{distrs } mdp \ ! \ i) \wedge$
 $\neg \text{List.null } (\text{distrs } mdp \ ! \ i))$

definition

$\text{valid-sub-cert } mdp \ c \ \text{ord } \text{check} \longleftrightarrow$
 $IArray.length (\text{witness } c) = \text{state-count } mdp \wedge$
 $IArray.length (\text{solution } c) = \text{state-count } mdp \wedge$
 $(\forall i < \text{state-count } mdp.$
 $\text{if } \text{states2 } mdp \ ! \ i \text{ then } \text{solution } c \ ! \ i = 1$
 $\text{else if } \text{states1 } mdp \ ! \ i \text{ then } 0 \leq \text{solution } c \ ! \ i \wedge$
 $(\text{list-all } (\lambda ds. \text{ord } (\text{sparse-mult } ds \ (\text{solution } c)) (\text{solution } c \ ! \ i)) (\text{distrs } mdp$
 $\ ! \ i)) \wedge$
 $(0 < \text{solution } c \ ! \ i \longrightarrow \text{check } (\text{distrs } mdp \ ! \ i) (\text{witness } c \ ! \ i))$
 $\text{else } \text{solution } c \ ! \ i = 0)$

definition

$\text{valid-pos-cert } mdp \ c \longleftrightarrow$
 $\text{valid-sub-cert } mdp \ c \ (\leq)$
 $(\lambda D ((j, a), n). j < \text{state-count } mdp \wedge \text{snd } (\text{witness } c \ ! \ j) < n \wedge 0 < \text{solution}$
 $c \ ! \ j \wedge$
 $a < \text{length } D \wedge \text{lookup } 0 \ (D \ ! \ a) \ j \neq 0)$

definition

$\text{valid-neg-cert } mdp \ c \longleftrightarrow$
 $\text{valid-sub-cert } mdp \ c \ (\geq)$
 $(\lambda D (J, n). \text{list-all2 } (\lambda j \ d. j < \text{state-count } mdp \wedge \text{snd } (\text{witness } c \ ! \ j) < n \wedge$
 $\text{lookup } 0 \ d \ j \neq 0 \wedge 0 < \text{solution } c \ ! \ j) \ J \ D)$

definition

$\text{valid-cert } mdp \ c \longleftrightarrow \text{valid-pos-cert } mdp \ (\text{pos-cert } c) \wedge \text{valid-neg-cert } mdp \ (\text{neg-cert}$
 $c)$

lemma *valid-mdp-rpD-length*:

assumes *valid-mdp-rp* *mdp*

shows $0 < \text{state-count } mdp \text{ IArray.length } (\text{distrs } mdp) = \text{state-count } mdp$

$\text{IArray.length } (\text{states1 } mdp) = \text{state-count } mdp \text{ IArray.length } (\text{states2 } mdp) = \text{state-count } mdp$

<proof>

lemma *valid-mdp-rpD*:

assumes *valid-mdp-rp* *mdp* $i < \text{state-count } mdp$

shows $\neg (\text{states1 } mdp !! i \wedge \text{states2 } mdp !! i)$

and $\bigwedge ds \ n \ x. ds \in \text{set } (\text{distrs } mdp !! i) \implies (n, x) \in \text{set } ds \implies n < \text{state-count } mdp$

and $\bigwedge ds \ n \ x. ds \in \text{set } (\text{distrs } mdp !! i) \implies (n, x) \in \text{set } ds \implies 0 \leq x$

and $\bigwedge ds. ds \in \text{set } (\text{distrs } mdp !! i) \implies \text{sum-list } (\text{map } \text{snd } ds) = 1$

and $\bigwedge ds. ds \in \text{set } (\text{distrs } mdp !! i) \implies \text{distinct } (\text{map } \text{fst } ds)$

and $\text{distrs } mdp !! i \neq []$

<proof>

lemma *valid-mdp-rp-sparse-mult*:

assumes *valid-mdp-rp* *mdp* $i < \text{state-count } mdp$ $ds \in \text{set } (\text{distrs } mdp !! i)$

shows $\text{sparse-mult } ds \ y = (\sum i < \text{state-count } mdp. \text{lookup } 0 \ ds \ i * y !! i)$

<proof>

lemma *valid-sub-certD*:

assumes *valid-mdp-rp* *mdp* *valid-sub-cert* *mdp* *c* *ord* *check* $i < \text{state-count } mdp$

shows $\neg \text{states1 } mdp !! i \implies \neg \text{states2 } mdp !! i \implies \text{solution } c !! i = 0$

and $\text{states2 } mdp !! i \implies \text{solution } c !! i = 1$

and $\text{states1 } mdp !! i \implies 0 \leq \text{solution } c !! i$

and $\bigwedge ds. \text{states1 } mdp !! i \implies ds \in \text{set } (\text{distrs } mdp !! i) \implies \text{ord } (\text{sparse-mult } ds \ (\text{solution } c)) \ (\text{solution } c !! i)$

and $\bigwedge ds. \text{states1 } mdp !! i \implies 0 < \text{solution } c !! i \longrightarrow \text{check } (\text{distrs } mdp !! i) \ (\text{witness } c !! i)$

<proof>

lemma *valid-pos-certD*:

assumes *valid-mdp-rp* *mdp* *valid-pos-cert* *mdp* *c* $i < \text{state-count } mdp$ $\text{states1 } mdp !! i$

$0 < \text{solution } c !! i$ *witness* $c !! i = ((j, a), n)$

shows $\text{snd } (\text{witness } c !! j) < n \wedge j < \text{state-count } mdp \wedge a < \text{length } (\text{distrs } mdp !! i) \wedge$

$\text{lookup } 0 \ ((\text{distrs } mdp !! i) ! a) \ j \neq 0 \wedge 0 < \text{solution } c !! j$

<proof>

lemma *valid-neg-certD*:

assumes *valid-mdp-rp* *mdp* *valid-neg-cert* *mdp* *c* $i < \text{state-count } mdp$ $\text{states1 } mdp !! i$

$0 < \text{solution } c !! i$ *witness* $c !! i = (js, n)$

shows $\text{list-all2 } (\lambda j \ ds. j < \text{state-count } mdp \wedge \text{snd } (\text{witness } c !! j) < n \wedge \text{lookup}$

$0 \text{ ds } j \neq 0 \wedge 0 < \text{solution } c \text{ !! } j \text{ } js \text{ (distrs } mdp \text{ !! } i)$
 $\langle \text{proof} \rangle$

context

fixes $mdp \ c$
assumes rp : *valid-mdp-rp* mdp
assumes $cert$: *valid-cert* $mdp \ c$

begin

interpretation *pmf-as-function* $\langle \text{proof} \rangle$

abbreviation $S \equiv \{.. < \text{state-count } mdp\}$

abbreviation $S1 \equiv \{i. i < \text{state-count } mdp \wedge (\text{states1 } mdp) \text{ !! } i\}$

abbreviation $S2 \equiv \{i. i < \text{state-count } mdp \wedge (\text{states2 } mdp) \text{ !! } i\}$

lift-definition $K :: \text{nat} \Rightarrow \text{nat pmf set is}$

$\lambda i. \text{if } i < \text{state-count } mdp \text{ then}$

$\{ (\lambda j. \text{of-rat } (\text{lookup } 0 \ D \ j) :: \text{real}) \mid D. D \in \text{set } (\text{distrs } mdp \text{ !! } i) \}$
 $\text{else } \{ \text{indicator } \{0\} \}$

$\langle \text{proof} \rangle$

interpretation *MDP: Reachability-Problem* $K \ S \ S1 \ S2$

$\langle \text{proof} \rangle$

definition $P\text{-max } s = \text{enn2real } (MDP.p \ s)$

definition $P\text{-min } s = \text{enn2real } (MDP.n \ s)$

lemma

assumes $i < \text{state-count } mdp$

shows $P\text{-max}$: $P\text{-max } i \leq \text{real-of-rat } (\text{solution } (\text{pos-cert } c) \text{ !! } i)$ (**is** $?max$)

and $P\text{-min}$: $P\text{-min } i \geq \text{real-of-rat } (\text{solution } (\text{neg-cert } c) \text{ !! } i)$ (**is** $?min$)

$\langle \text{proof} \rangle$

end

end

References

- [1] J. Hölzl. Formalising semantics for expected running time of probabilistic programs. In J. C. Blanchette and S. Merz, editors, *Interactive Theorem Proving (ITP 2016)*, pages 475–482. Springer, 2016.
- [2] J. Hölzl. Markov processes in isabelle/hol. In Y. Bertot and V. Vafeiadis, editors, *Certified Programs and Proofs (CPP 2017)*. ACM, 2017.
- [3] J. Hölzl and T. Nipkow. Interactive verification of Markov chains: Two distributed protocol case studies. In U. Fahrenberg, A. Legay, and

C. Thrane, editors, *Quantities in Formal Methods (QFM 2012)*, volume 103 of *EPTCS*. arXiv, 2012.

- [4] J. Hölzl and T. Nipkow. Verifying pCTL model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012)*, LNCS, 2012.
- [5] H. Johannes. Markov chains and markov decision processes in Isabelle/HOL. *Journal of Automated Reasoning*, 2017.