Probabilistic Timed Automata

Simon Wimmer and Johannes Hölzl

March 17, 2025

Abstract

We present a formalization of probabilistic timed automata (PTA) for which we try to follow the formula "MDP + TA = PTA" as far as possible: our work starts from our existing formalizations of Markov decision processes (MDP) and timed automata (TA) and combines them modularly. We prove the fundamental result for probabilistic timed automata: the region construction that is known from timed automata carries over to the probabilistic setting. In particular, this allows us to prove that minimum and maximum reachability probabilities can be computed via a reduction to MDP model checking, including the case where one wants to disregard unrealizable behavior. Further information can be found in our ITP paper [2].

The definition of the PTA semantics can be found in Section 3.3, the region MDP is in Section 4.1, the bisimulation theorem is in Section 1, and the final theorems can be found in Section 7.4. The background theory we formalize is described in the seminal paper on PTA [1].

Contents

theory *PTA*
  imports *library/Lib*
begin

1 Bisimulation on a Relation

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

lemma *T-eq-rel-half*[*consumes 4*, *case-names prob sets cont*]:
  fixes R :: $'s \Rightarrow 't \Rightarrow \mathit{bool}$ **and** f :: $'s \Rightarrow 't$ **and** S :: $'s\ \mathit{set}$
  assumes *R-def*: $\bigwedge s\ t.\ R\ s\ t \longleftrightarrow (s \in S \land f\ s = t)$
  assumes *A*[*measurable*]: $A \in$ *sets* (*stream-space* (*count-space UNIV*))
    and *B*[*measurable*]: $B \in$ *sets* (*stream-space* (*count-space UNIV*))
    and *AB*: *rel-set-strong* (*stream-all2 R*) *A B* **and** *KL*: *rel-fun R* (*rel-pmf R*) *K L* **and** *xy*: *R x y*
  shows *MC-syntax.T K x A* = *MC-syntax.T L y B*
⟨*proof*⟩

no-notation *ccval* (‹{|-|}› [*100*])

hide-const *succ*

2 Additional Facts on Regions

declare *reset-set11*[*simp*] *reset-set1*[*simp*]

Defining the closest successor of a region. Only exists if at least one interval is upper-bounded.

abbreviation *is-upper-right* **where**
  is-upper-right $R \equiv (\forall\ t \geq 0.\ \forall\ u \in R.\ u \oplus t \in R)$

definition
  succ $\mathcal{R}$ $R \equiv$
  if is-upper-right R then R else
  (*THE* $R'.\ R' \neq R \land R' \in$ *Succ* $\mathcal{R}$ $R \land (\forall\ u \in R.\ \forall\ t \geq 0.\ (u \oplus t) \notin R \longrightarrow (\exists\ t' \leq t.\ (u \oplus t') \in R' \land 0 \leq$
$t')))$

lemma *region-continuous*:
  assumes *valid-region X k I r*
  defines *R*: $R \equiv$ *region X I r*
  assumes *between*: $0 \leq t1$ $t1 \leq t2$
  assumes *elem*: $u \in R$ $u \oplus t2 \in R$
  shows $u \oplus t1 \in R$
⟨*proof*⟩

lemma *upper-right-eq*:
  assumes *finite X valid-region X k I r*
  shows $(\forall\ x \in X.$ *isGreater* $(I\ x)) \longleftrightarrow$ *is-upper-right* (*region X I r*)
⟨*proof*⟩

lemma *bounded-region*:
  assumes *finite X valid-region X k I r*
  defines *R*: $R \equiv$ *region X I r*
  assumes $\neg$ *is-upper-right* R $u \in R$
  shows $u \oplus 1 \notin R$
⟨*proof*⟩

context *AlphaClosure-global*
begin

no-notation *Regions-Beta.part* (‹[-]-› [*61*,*61*] *61*)

lemma *succ-ex*:
  assumes $R \in \mathcal{R}$
  shows $succ\ \mathcal{R}\ R \in \mathcal{R}$ (**is** *?G1*) **and** $succ\ \mathcal{R}\ R \in Succ\ \mathcal{R}\ R$ (**is** *?G2*)
  and $\forall\ u \in R.\ \forall\ t \geq 0.\ (u \oplus t) \notin R \longrightarrow (\exists\ t' \leq t.\ (u \oplus t') \in succ\ \mathcal{R}\ R \wedge 0 \leq t')$ (**is** *?G3*)
⟨*proof*⟩

lemma *region-set′-closed*:
  fixes *d* :: *nat*
  assumes $R \in \mathcal{R}$ $d \geq 0$ $\forall x{\in}set\ r.\ d \leq k\ x$ $set\ r \subseteq X$
  shows $region\text{-}set'\ R\ r\ d \in \mathcal{R}$
⟨*proof*⟩

lemma *clock-set-cong*[*simp*]:
  assumes $\forall\ c \in set\ r.\ u\ c = d$
  shows $[r \rightarrow d]u = u$
⟨*proof*⟩

lemma *region-reset-not-Succ*:

  notes *regions-closed′-spec*[*intro*]
  assumes $R \in \mathcal{R}$ $set\ r \subseteq X$
  shows $region\text{-}set'\ R\ r\ 0 = R \vee region\text{-}set'\ R\ r\ 0 \notin Succ\ \mathcal{R}\ R$ (**is** $?R = R \vee$ -)
⟨*proof*⟩

end

3 Definition and Semantics

3.1 Syntactic Definition

We do not include:

- a labelling function, as we will assume that atomic propositions are simply sets of states
- a fixed set of locations or clocks, as we will implicitly derive it from the set of transitions
- start or end locations, as we will primarily study reachability

type-synonym
  ($'c$, $'t$, $'s$) *transition* = $'s$ ∗ ($'c$, $'t$) *cconstraint* ∗ ($'c$ *set* ∗ $'s$) *pmf*

type-synonym
  ($'c$, $'t$, $'s$) *pta* = ($'c$, $'t$, $'s$) *transition set* ∗ ($'c$, $'t$, $'s$) *invassn*

definition
  edges :: ($'c$, $'t$, $'s$) *transition* ⇒ ($'s$ ∗ ($'c$, $'t$) *cconstraint* ∗ ($'c$ *set* ∗ $'s$) *pmf* ∗ $'c$ *set* ∗ $'s$) *set*
where
  $edges \equiv \lambda\ (l,\ g,\ p).\ \{(l,\ g,\ p,\ X,\ l')\ |\ X\ l'.\ (X,\ l') \in set\text{-}pmf\ p\}$

definition
  $Edges\ A \equiv \bigcup\ \{edges\ t\ |\ t.\ t \in fst\ A\}$

definition
  trans-of :: ($'c$, $'t$, $'s$) *pta* ⇒ ($'c$, $'t$, $'s$) *transition set*
where

trans-of ≡ *fst*

definition
inv-of :: (′*c*, ′*time*, ′*s*) *pta* ⇒ (′*c*, ′*time*, ′*s*) *invassn*
where
inv-of ≡ *snd*

no-notation *transition* (‹- ⊢ - ⟶$^{-,-,-}$ -› [*61*,*61*,*61*,*61*,*61*,*61*] *61*)

abbreviation *transition* ::
(′*c*, ′*time*, ′*s*) *pta* ⇒ ′*s* ⇒ (′*c*, ′*time*) *cconstraint* ⇒ (′*c set* ∗ ′*s*) *pmf* ⇒ ′*c set* ⇒ ′*s* ⇒ *bool*
(‹- ⊢ - ⟶$^{-,-,-}$ -› [*61*,*61*,*61*,*61*,*61*,*61*] *61*) **where**
(*A* ⊢ *l* ⟶^{g,p,X} *l*′) ≡ (*l*, *g*, *p*, *X*, *l*′) ∈ *Edges A*

definition
locations :: (′*c*, ′*t*, ′*s*) *pta* ⇒ ′*s set*
where
locations A ≡ (*fst* ‘ *Edges A*) ∪ ((*snd o snd o snd o snd*) ‘ *Edges A*)

3.1.1 Collecting Information About Clocks

definition *collect-clkt* :: (′*c*, ′*t*::*time*, ′*s*) *transition set* ⇒ (′*c* ∗′*t*) *set*
where
collect-clkt S = ⋃ {*collect-clock-pairs* (*fst* (*snd t*)) | *t* . *t* ∈ *S*}

definition *collect-clki* :: (′*c*, ′*t* :: *time*, ′*s*) *invassn* ⇒ (′*c* ∗′*t*) *set*
where
collect-clki I = ⋃ {*collect-clock-pairs* (*I x*) | *x*. *True*}

definition *clkp-set* :: (′*c*, ′*t* :: *time*, ′*s*) *pta* ⇒ (′*c* ∗ ′*t*) *set*
where
clkp-set A = *collect-clki* (*inv-of A*) ∪ *collect-clkt* (*trans-of A*)

definition *collect-clkvt* :: (′*c*, ′*t* :: *time*, ′*s*) *pta* ⇒ ′*c set*
where
collect-clkvt A = ⋃ ((*fst o snd o snd o snd*) ‘ *Edges A*)

abbreviation *clocks* **where** *clocks A* ≡ *fst* ‘ *clkp-set A* ∪ *collect-clkvt A*

definition *valid-abstraction*
where
valid-abstraction A X k ≡
(∀ (*x*,*m*) ∈ *clkp-set A*. *m* ≤ *k x* ∧ *x* ∈ *X* ∧ *m* ∈ ℕ) ∧ *collect-clkvt A* ⊆ *X* ∧ *finite X*

lemma *valid-abstractionD*[*dest*]:
assumes *valid-abstraction A X k*
shows (∀ (*x*,*m*) ∈ *clkp-set A*. *m* ≤ *k x* ∧ *x* ∈ *X* ∧ *m* ∈ ℕ) *collect-clkvt A* ⊆ *X finite X*
⟨*proof*⟩

lemma *valid-abstractionI*[*intro*]:
assumes (∀ (*x*,*m*) ∈ *clkp-set A*. *m* ≤ *k x* ∧ *x* ∈ *X* ∧ *m* ∈ ℕ) *collect-clkvt A* ⊆ *X finite X*
shows *valid-abstraction A X k*
⟨*proof*⟩

3.2 Operational Semantics as an MDP

abbreviation (*input*) *clock-set-set* :: ′*c set* ⇒ ′*t*::*time* ⇒ (′*c*,′*t*) *cval* ⇒ (′*c*,′*t*) *cval*
(‹[-:=-]-› [*65*,*65*,*65*] *65*)
where
[*X*:=*t*]*u* ≡ *clock-set* (*SOME r*. *set r* = *X*) *t u*

term *region-set′*

abbreviation *region-set-set* :: *′c set* ⇒ *′t::time* ⇒ (*′c*,*′t*) *zone* ⇒ (*′c*,*′t*) *zone*
(‹[-::=-]-› [*65*,*65*,*65*] *65*)
where
[*X*::=*t*]*R* ≡ *region-set′ R* (*SOME r*. *set r* = *X*) *t*

no-notation *zone-set* (‹-$_{- \to\, 0}$› [*71*] *71*)

abbreviation *zone-set-set* :: (*′c*, *′t::time*) *zone* ⇒ *′c set* ⇒ (*′c*, *′t*) *zone*
(‹-$_{- \to\, 0}$› [*71*] *71*)
where
$Z_{X \to 0}$ ≡ *zone-set Z* (*SOME r*. *set r* = *X*)

abbreviation (*input*) *ccval* (‹{|-|}› [*100*]) **where** *ccval cc* ≡ {*v*. *v* ⊢ *cc*}

locale *Probabilistic-Timed-Automaton* =
fixes *A* :: (*′c*, *′t* :: *time*, *′s*) *pta*
assumes *admissible-targets*:
(*l*, *g*, *μ*) ∈ *trans-of A* ⟹ (*X*, *l′*) ∈ *μ* ⟹ $\{\!|g|\!\}_{X \to 0}$ ⊆ {|*inv-of A l′*|}
(*l*, *g*, *μ*) ∈ *trans-of A* ⟹ (*X*, *l′*) ∈ *μ* ⟹ *X* ⊆ *clocks A*
— Not necessarily what we want to have
begin

3.3 Syntactic Definition

definition *L* = *locations A*

definition $\mathcal{X}$ = *clocks A*

definition *S* ≡ {(*l*, *u*) . *l* ∈ *L* ∧ (∀ *x* ∈ $\mathcal{X}$. *u x* ≥ *0*) ∧ *u* ⊢ *inv-of A l*}

inductive-set
K :: (*′s* ∗ (*′c*, *′t*) *cval*) ⇒ (*′s* ∗ (*′c*, *′t*) *cval*) *pmf set* **for** *st* :: (*′s* ∗ (*′c*, *′t*) *cval*)
where
— Passage of time *delay*:
st ∈ *S* ⟹ *st* = (*l*, *u*) ⟹ *t* ≥ *0* ⟹ *u* ⊕ *t* ⊢ *inv-of A l* ⟹ *return-pmf* (*l*, *u* ⊕ *t*) ∈ *K st* |
— Discrete transitions *action*:
st ∈ *S* ⟹ *st* = (*l*, *u*) ⟹ (*l*, *g*, *μ*) ∈ *trans-of A* ⟹ *u* ⊢ *g*
⟹ *map-pmf* (λ (*X*, *l*). (*l*, ([*X* := *0*]*u*))) *μ* ∈ *K st* |
— Self loops – Note that this does not assume *st* ∈ *S loop*:
return-pmf st ∈ *K st*

declare *K*.*intros*[*intro*]

sublocale *MDP*: *Markov-Decision-Process K* ⟨*proof*⟩

end

4 Constructing the Corresponding Finite MDP on Regions

locale *Probabilistic-Timed-Automaton-Regions* =
Probabilistic-Timed-Automaton A + *Regions-global* $\mathcal{X}$
for *A* :: (*′c*, *t*, *′s*) *pta* +
— The following are necessary to obtain a *finite* MDP
assumes *finite*: *finite* $\mathcal{X}$ *finite L finite* (*trans-of A*)
assumes *not-trivial*: ∃ *l* ∈ *L*. ∃ *u* ∈ *V*. *u* ⊢ *inv-of A l*
assumes *valid*: *valid-abstraction A* $\mathcal{X}$ *k*
begin

lemmas $\mathit{finite}\text{-}\mathcal{R} = \mathit{finite}\text{-}\mathcal{R}[\mathit{OF\ finite}(1),\ \mathit{of}\ k,\ \mathit{folded}\ \mathcal{R}\text{-}\mathit{def}]$

4.1 Syntactic Definition

definition $\mathcal{S} \equiv \{(l,\ R)\ .\ l \in L \wedge R \in \mathcal{R} \wedge R \subseteq \{u.\ u \vdash \mathit{inv\text{-}of}\ A\ l\}\}$

lemma *S-alt-def*: $S = \{(l,\ u)\ .\ l \in L \wedge u \in V \wedge u \vdash \mathit{inv\text{-}of}\ A\ l\}$ ⟨*proof*⟩

Note how we relax the definition to allow more transitions in the first case. To obtain a more compact MDP the commented out version can be used an proved equivalent.

inductive-set
$\mathcal{K}$ :: $('s * ('c,\ t)\ \mathit{cval\ set}) \Rightarrow ('s * ('c,\ t)\ \mathit{cval\ set})\ \mathit{pmf\ set}$ **for** st :: $('s * ('c,\ t)\ \mathit{cval\ set})$
where

— Passage of time *delay*:
$st \in \mathcal{S} \Longrightarrow st = (l,R) \Longrightarrow R' \in \mathit{Succ}\ \mathcal{R}\ R \Longrightarrow R' \subseteq \{\!|\mathit{inv\text{-}of}\ A\ l|\!\} \Longrightarrow \mathit{return\text{-}pmf}\ (l,\ R') \in \mathcal{K}\ st$ |
— Discrete transitions *action*:
$st \in \mathcal{S} \Longrightarrow st = (l,\ R\ ) \Longrightarrow (l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A \Longrightarrow R \subseteq \{\!|g|\!\}$
$\Longrightarrow \mathit{map\text{-}pmf}\ (\lambda\ (X,\ l).\ (l,\ \mathit{region\text{-}set}'\ R\ (\mathit{SOME}\ r.\ \mathit{set}\ r = X)\ 0))\ \mu \in \mathcal{K}\ st$ |
— Self loops – Note that this does not assume $st \in \mathcal{S}$ *loop*:
$\mathit{return\text{-}pmf}\ st \in \mathcal{K}\ st$

lemmas [*intro*] = $\mathcal{K}$.*intros*

4.2 Many Closure Properties

lemma *transition-def*:
$(A \vdash l \longrightarrow^{g,\mu,X} l') = ((l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A \wedge (X,\ l') \in \mu)$
⟨*proof*⟩

lemma *transitionI*[*intro*]:
$A \vdash l \longrightarrow^{g,\mu,X} l'$ **if** $(l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A\ (X,\ l') \in \mu$
⟨*proof*⟩

lemma *transitionD*[*dest*]:
$(l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A\ (X,\ l') \in \mu$ **if** $A \vdash l \longrightarrow^{g,\mu,X} l'$
⟨*proof*⟩

lemma *bex-Edges*:
$(\exists\ x \in \mathit{Edges}\ A.\ P\ x) = (\exists\ l\ g\ \mu\ X\ l'.\ A \vdash l \longrightarrow^{g,\mu,X} l' \wedge P\ (l,\ g,\ \mu,\ X,\ l'))$
⟨*proof*⟩

lemma *L-trans*[*intro*]:
assumes $(l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A\ (X,\ l') \in \mu$
shows $l \in L\ l' \in L$
⟨*proof*⟩

lemma *transition-$\mathcal{X}$*:
$X \subseteq \mathcal{X}$ **if** $A \vdash l \longrightarrow^{g,\mu,X} l'$
⟨*proof*⟩

lemma *admissible-targets-alt*:
$A \vdash l \longrightarrow^{g,\mu,X} l' \Longrightarrow \{\!|g|\!\}_{X \to 0} \subseteq \{\!|\mathit{inv\text{-}of}\ A\ l'|\!\}$
$A \vdash l \longrightarrow^{g,\mu,X} l' \Longrightarrow X \subseteq \mathit{clocks}\ A$
⟨*proof*⟩

lemma *V-reset-closed*[*intro*]:
assumes $u \in V$

shows $[r \to (d{::}nat)]u \in V$
⟨*proof*⟩

lemmas *V-reset-closed′*[*intro*] = *V-reset-closed*[*of - - 0*, *simplified*]

lemma *regions-part-ex*[*intro*]:
assumes $u \in V$
shows $u \in [u]_{\mathcal{R}}$ $[u]_{\mathcal{R}} \in \mathcal{R}$
⟨*proof*⟩

lemma *rep-$\mathcal{R}$-ex*[*intro*]:
assumes $R \in \mathcal{R}$
shows $(SOME\ u.\ u \in R) \in R$
⟨*proof*⟩

lemma *V-nn-closed*[*intro*]:
$u \in V \Longrightarrow t \geq 0 \Longrightarrow u \oplus t \in V$
⟨*proof*⟩

lemma *K-S-closed*[*intro*]:
assumes $\mu \in K\ s$ $s' \in \mu$ $s \in S$
shows $s' \in S$
⟨*proof*⟩

lemma *S-V*[*intro*]:
$(l, u) \in S \Longrightarrow u \in V$
⟨*proof*⟩

lemma *L-V*[*intro*]:
$(l, u) \in S \Longrightarrow l \in L$
⟨*proof*⟩

lemma *$\mathcal{S}$-V*[*intro*]:
$(l, R) \in \mathcal{S} \Longrightarrow R \in \mathcal{R}$
⟨*proof*⟩

lemma *admissible-targets′*:
assumes $(l, g, \mu) \in trans\text{-}of\ A$ $(X, l') \in \mu$ $R \subseteq \{\!|g|\!\}$
shows $region\text{-}set'\ R\ (SOME\ r.\ set\ r = X)\ 0 \subseteq \{\!|inv\text{-}of\ A\ l'|\!\}$
⟨*proof*⟩

4.3 The Region Graph is a Finite MDP

lemma *$\mathcal{S}$-finite*:
finite $\mathcal{S}$
⟨*proof*⟩

lemma *$\mathcal{K}$-finite*:
finite $(\mathcal{K}\ st)$
⟨*proof*⟩

lemma *$\mathcal{R}$-not-empty*:
$\mathcal{R} \neq \{\}$
⟨*proof*⟩

lemma *$\mathcal{S}$-not-empty*:
$\mathcal{S} \neq \{\}$
⟨*proof*⟩

lemma *$\mathcal{K}$-$\mathcal{S}$-closed*:

assumes $s \in \mathcal{S}$
shows $(\bigcup D \in \mathcal{K}\ s.\ set\text{-}pmf\ D) \subseteq \mathcal{S}$
$\langle proof \rangle$

sublocale *R-G*: *Finite-Markov-Decision-Process* $\mathcal{K}$ $\mathcal{S}$
$\langle proof \rangle$

lemmas $\mathcal{K}$-$\mathcal{S}$-*closed'*[*intro*] = *R-G.set-pmf-closed*

5 Relating the MDPs

5.1 Translating From K to $\mathcal{K}$

lemma *ccompatible-inv*:
shows *ccompatible* $\mathcal{R}$ (*inv-of A l*)
$\langle proof \rangle$

lemma *ccompatible-guard*:
assumes $(l, g, \mu) \in$ *trans-of A*
shows *ccompatible* $\mathcal{R}$ *g*
$\langle proof \rangle$

lemmas *ccompatible-def* = *ccompatible-def*[*unfolded ccval-def*]

lemma *region-set'-eq*:
fixes X :: *'c set*
assumes $R \in \mathcal{R}$ $u \in R$
and $A \vdash l \longrightarrow^{g,\mu,X} l'$
shows
$[[X{:=}0]u]_{\mathcal{R}} = region\text{-}set'\ R\ (SOME\ r.\ set\ r = X)\ 0$ $[[X{:=}0]u]_{\mathcal{R}} \in \mathcal{R}$ $[X{:=}0]u \in [[X{:=}0]u]_{\mathcal{R}}$
$\langle proof \rangle$

lemma *regions-part-ex-reset*:
assumes $u \in V$
shows $[r \rightarrow (d{::}nat)]u \in [[r \rightarrow d]u]_{\mathcal{R}}$ $[[r \rightarrow d]u]_{\mathcal{R}} \in \mathcal{R}$
$\langle proof \rangle$

lemma *reset-sets-all-equiv*:
assumes $u \in V$ $u' \in [[r \rightarrow (d :: nat)]u]_{\mathcal{R}}$ $x \in set\ r$ $set\ r \subseteq \mathcal{X}$ $d \leq k\ x$
shows $u'\ x = d$
$\langle proof \rangle$

lemma *reset-eq*:
assumes $u \in V$ $([[r \rightarrow 0]u]_{\mathcal{R}}) = ([[r' \rightarrow 0]u]_{\mathcal{R}})$ $set\ r \subseteq \mathcal{X}$ $set\ r' \subseteq \mathcal{X}$
shows $[r \rightarrow 0]u = [r' \rightarrow 0]u$ $\langle proof \rangle$

lemma *admissible-targets-clocks*:
assumes $(l, g, \mu) \in$ *trans-of A* $(X, l') \in \mu$
shows $X \subseteq \mathcal{X}$ $set\ (SOME\ r.\ set\ r = X) \subseteq \mathcal{X}$
$\langle proof \rangle$

lemma
rel-pmf $(\lambda\ a\ b.\ f\ a = b)\ \mu\ (map\text{-}pmf\ f\ \mu)$
$\langle proof \rangle$

lemma *K-pmf-rel*:
defines $f \equiv \lambda\ (l, u).\ (l, [u]_{\mathcal{R}})$
shows *rel-pmf* $(\lambda\ (l, u)\ st.\ (l, [u]_{\mathcal{R}}) = st)\ \mu\ (map\text{-}pmf\ f\ \mu)$ $\langle proof \rangle$

lemma $\mathcal{K}$-*pmf-rel*:

assumes A: $\mu \in \mathcal{K}\ (l,\ R)$
defines $f \equiv \lambda\ (l,\ u).\ (l,\ \mathit{SOME}\ u.\ u \in R)$
shows $\mathit{rel\text{-}pmf}\ (\lambda\ (l,\ u)\ st.\ (l,\ \mathit{SOME}\ u.\ u \in R) = st)\ \mu\ (\mathit{map\text{-}pmf}\ f\ \mu)$ $\langle proof \rangle$

lemma *K-elem-abs-inj*:
assumes A: $\mu \in K\ (l,\ u)$
defines $f \equiv \lambda\ (l,\ u).\ (l,\ [u]_{\mathcal{R}})$
shows $\mathit{inj\text{-}on}\ f\ \mu$
$\langle proof \rangle$

lemma *K-elem-repr-inj*:
notes *alpha-interp.valid-regions-distinct-spec*[*intro*]
assumes A: $\mu \in \mathcal{K}\ (l,\ R)$
defines $f \equiv \lambda\ (l,\ R).\ (l,\ \mathit{SOME}\ u.\ u \in R)$
shows $\mathit{inj\text{-}on}\ f\ \mu$
$\langle proof \rangle$

lemma *K-elem-pmf-map-abs*:
assumes A: $\mu \in K\ (l,\ u)\ (l',\ u') \in \mu$
defines $f \equiv \lambda\ (l,\ u).\ (l,\ [u]_{\mathcal{R}})$
shows $\mathit{pmf}\ (\mathit{map\text{-}pmf}\ f\ \mu)\ (f\ (l',\ u')) = \mathit{pmf}\ \mu\ (l',\ u')$
$\langle proof \rangle$

lemma *K-elem-pmf-map-repr*:
assumes A: $\mu \in \mathcal{K}\ (l,\ R)\ (l',\ R') \in \mu$
defines $f \equiv \lambda\ (l,\ R).\ (l,\ \mathit{SOME}\ u.\ u \in R)$
shows $\mathit{pmf}\ (\mathit{map\text{-}pmf}\ f\ \mu)\ (f\ (l',\ R')) = \mathit{pmf}\ \mu\ (l',\ R')$
$\langle proof \rangle$

definition $\mathit{transp} :: ('s * ('c,\ t)\ \mathit{cval} \Rightarrow \mathit{bool}) \Rightarrow 's * ('c,\ t)\ \mathit{cval}\ \mathit{set} \Rightarrow \mathit{bool}$ **where**
$\mathit{transp}\ \varphi \equiv \lambda\ (l,\ R).\ \forall\ u \in R.\ \varphi\ (l,\ u)$

5.2 Translating Configurations

5.2.1 States

definition
$\mathit{abss} :: 's * ('c,\ t)\ \mathit{cval} \Rightarrow 's * ('c,\ t)\ \mathit{cval}\ \mathit{set}$
where
$\mathit{abss} \equiv \lambda\ (l,\ u).\ \mathit{if}\ u \in V\ \mathit{then}\ (l,\ [u]_{\mathcal{R}})\ \mathit{else}\ (l,\ -V)$

definition
$\mathit{reps} :: 's * ('c,\ t)\ \mathit{cval}\ \mathit{set} \Rightarrow 's * ('c,\ t)\ \mathit{cval}$
where
$\mathit{reps} \equiv \lambda\ (l,\ R).\ \mathit{if}\ R \in \mathcal{R}\ \mathit{then}\ (l,\ \mathit{SOME}\ u.\ u \in R)\ \mathit{else}\ (l,\ \lambda\text{-}.\ -1)$

lemma $\mathcal{S}$*-reps-S*[*intro*]:
assumes $s \in \mathcal{S}$
shows $\mathit{reps}\ s \in S$
$\langle proof \rangle$

lemma *S-abss-*$\mathcal{S}$[*intro*]:
assumes $s \in S$
shows $\mathit{abss}\ s \in \mathcal{S}$
$\langle proof \rangle$

lemma $\mathcal{S}$*-abss-reps*[*simp*]:
$s \in \mathcal{S} \Longrightarrow \mathit{abss}\ (\mathit{reps}\ s) = s$
$\langle proof \rangle$

lemma *map-pmf-abs-reps*:
  assumes $s \in \mathcal{S}$ $\mu \in \mathcal{K}$ s
  shows *map-pmf abss* (*map-pmf reps* μ) $= \mu$
⟨*proof*⟩

lemma *abss-reps-id*:
  notes *R-G.cfg-onD-state*[*simp del*]
  assumes $s' \in \mathcal{S}$ $s \in$ *set-pmf* (*action cfg*) *cfg* $\in$ *R-G.cfg-on* s'
  shows *abss* (*reps* s) $= s$
⟨*proof*⟩

lemma *abss-S*[*intro*]:
  assumes $(l, u) \in S$
  shows *abss* $(l, u) = (l, [u]_{\mathcal{R}})$
⟨*proof*⟩

lemma *reps-*$\mathcal{S}$[*intro*]:
  assumes $(l, R) \in \mathcal{S}$
  shows *reps* $(l, R) = (l,$ *SOME* $u.\ u \in R)$
⟨*proof*⟩

lemma *fst-abss*:
  fst (*abss st*) = *fst st* **for** *st*
  ⟨*proof*⟩

lemma *K-elem-abss-inj*:
  assumes *A*: $\mu \in K\ (l, u)\ (l, u) \in S$
  shows *inj-on abss* μ
⟨*proof*⟩

lemma $\mathcal{K}$*-elem-reps-inj*:
  assumes *A*: $\mu \in \mathcal{K}\ (l, R)\ (l, R) \in \mathcal{S}$
  shows *inj-on reps* μ
⟨*proof*⟩

lemma *P-elem-pmf-map-abss*:
  assumes *A*: $\mu \in K\ (l, u)\ (l, u) \in S\ s' \in \mu$
  shows *pmf* (*map-pmf abss* μ) (*abss* s') = *pmf* μ s'
⟨*proof*⟩

lemma $\mathcal{K}$*-elem-pmf-map-reps*:
  assumes *A*: $\mu \in \mathcal{K}\ (l, R)\ (l, R) \in \mathcal{S}\ (l', R') \in \mu$
  shows *pmf* (*map-pmf reps* μ) (*reps* (l', R')) = *pmf* μ (l', R')
⟨*proof*⟩

We need that $\mathcal{X}$ is non-trivial here

lemma *not-*$\mathcal{S}$*-reps*:
  $(l, R) \notin \mathcal{S} \Longrightarrow$ *reps* $(l, R) \notin S$
⟨*proof*⟩

lemma *neq-V-not-region*:
  $-V \notin \mathcal{R}$
⟨*proof*⟩

lemma $\mathcal{S}$*-abss-S*:
  abss $s \in \mathcal{S} \Longrightarrow s \in S$
  ⟨*proof*⟩

lemma *S-pred-stream-abss-*$\mathcal{S}$:

pred-stream (λ *s. s* $\in$ *S*) *xs* $\longleftrightarrow$ *pred-stream* (λ *s. s* $\in$ $\mathcal{S}$) (*smap abss xs*)
⟨*proof*⟩

sublocale *MDP*: *Markov-Decision-Process-Invariant K S* ⟨*proof*⟩

abbreviation (*input*) *valid-cfg* $\equiv$ *MDP.valid-cfg*

lemma *K-closed*:
$s \in S \Longrightarrow (\bigcup D{\in}K\ s.\ \textit{set-pmf}\ D) \subseteq S$
⟨*proof*⟩

5.2.2 Intermezzo

abbreviation *timed-bisim* (**infixr** ‹~› *60*) **where**
$s \sim s' \equiv \textit{abss}\ s = \textit{abss}\ s'$

lemma *bisim-loc-id*[*intro*]:
$(l, u) \sim (l', u') \Longrightarrow l = l'$
⟨*proof*⟩

lemma *bisim-val-id*[*intro*]:
$[u]_{\mathcal{R}} = [u']_{\mathcal{R}}$ **if** $u \in V\ (l, u) \sim (l', u')$
⟨*proof*⟩

lemma *bisim-symmetric*:
$(l, u) \sim (l', u') = (l', u') \sim (l, u)$
⟨*proof*⟩

lemma *bisim-val-id2*[*intro*]:
$u' \in V \Longrightarrow (l, u) \sim (l', u') \Longrightarrow [u]_{\mathcal{R}} = [u']_{\mathcal{R}}$
⟨*proof*⟩

lemma *K-bisim-unique*:
assumes $s \in S\ \mu \in K\ s\ x \in \mu\ x' \in \mu\ x \sim x'$
shows $x = x'$
⟨*proof*⟩

5.2.3 Predicates

definition *absp* **where**
absp $\varphi \equiv \varphi$ *o reps*

definition *repp* **where**
repp $\varphi \equiv \varphi$ *o absp*

5.2.4 Distributions

definition
abst :: $('s * ('c, t)\ \textit{cval})\ \textit{pmf} \Rightarrow ('s * ('c, t)\ \textit{cval set})\ \textit{pmf}$
where
abst = *map-pmf abss*

lemma *abss-*$\mathcal{S}$*D*:
assumes *abss s* $\in \mathcal{S}$
obtains *l u* **where** $s = (l, u)\ u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}$
⟨*proof*⟩

lemma *abss-*$\mathcal{S}$*D'*:
assumes *abss s* $\in \mathcal{S}$ *abss s* = (*l*, *R*)
obtains *u* **where** $s = (l, u)\ u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}\ R = [u]_{\mathcal{R}}$

⟨*proof*⟩

definition *infR* $R \equiv \lambda\ c.\ \textit{of-int}\ \lfloor(\textit{SOME}\ u.\ u \in R)\ c\rfloor$

term *let* $a = 3$ *in* b

definition *delayedR* $R\ u \equiv$
$u \oplus ($
$\quad \textit{let}\ I = (\textit{SOME}\ I.\ \exists\ r.\ \textit{valid-region}\ \mathcal{X}\ k\ I\ r \wedge R = \textit{region}\ \mathcal{X}\ I\ r);$
$\qquad m = 1 - \textit{Max}\ (\{\textit{frac}\ (u\ c) \mid c.\ c \in \mathcal{X} \wedge \textit{isIntv}\ (I\ c)\} \cup \{0\})$
$\quad \textit{in SOME}\ t.\ u \oplus t \in R \wedge t \geq m\ /\ 2$
$)$

lemma *delayedR-correct-aux-aux*:
fixes $c :: \textit{nat}$
fixes $a\ b :: \textit{real}$
assumes $c < a\ a < \textit{Suc}\ c\ b \geq 0\ a + b < \textit{Suc}\ c$
shows $\textit{frac}\ (a + b) = \textit{frac}\ a + b$

⟨*proof*⟩

lemma *delayedR-correct-aux*:
fixes $I\ r$
defines $R \equiv \textit{region}\ \mathcal{X}\ I\ r$
assumes $u \in R\ \textit{valid-region}\ \mathcal{X}\ k\ I\ r\ \forall\ c \in \mathcal{X}.\ \neg\ \textit{isConst}\ (I\ c)$
$\forall\ c \in \mathcal{X}.\ \textit{isIntv}\ (I\ c) \longrightarrow (u \oplus t)\ c < \textit{intv-const}\ (I\ c) + 1$
$t \geq 0$
shows $u \oplus t \in R$ ⟨*proof*⟩

lemma *delayedR-correct-aux'*:
fixes $I\ r$
defines $R \equiv \textit{region}\ \mathcal{X}\ I\ r$
assumes $u \oplus t1 \in R\ \textit{valid-region}\ \mathcal{X}\ k\ I\ r\ \forall\ c \in \mathcal{X}.\ \neg\ \textit{isConst}\ (I\ c)$
$\forall\ c \in \mathcal{X}.\ \textit{isIntv}\ (I\ c) \longrightarrow (u \oplus t2)\ c < \textit{intv-const}\ (I\ c) + 1$
$t1 \leq t2$
shows $u \oplus t2 \in R$
⟨*proof*⟩

lemma *valid-regions-intv-distinct*:
$\textit{valid-region}\ X\ k\ I\ r \Longrightarrow \textit{valid-region}\ X\ k\ I'\ r' \Longrightarrow u \in \textit{region}\ X\ I\ r \Longrightarrow u \in \textit{region}\ X\ I'\ r'$
$\Longrightarrow x \in X \Longrightarrow I\ x = I'\ x$
⟨*proof*⟩

lemma *delayedR-correct*:
fixes $I\ r$
defines $R' \equiv \textit{region}\ \mathcal{X}\ I\ r$
assumes $u \in R\ R \in \mathcal{R}\ \textit{valid-region}\ \mathcal{X}\ k\ I\ r\ \forall\ c \in \mathcal{X}.\ \neg\ \textit{isConst}\ (I\ c)\ R' \in \textit{Succ}\ \mathcal{R}\ R$
shows
$\textit{delayedR}\ R'\ u \in R'$

$\exists\ t \geq 0.\ delayedR\ R'\ u = u \oplus t$
$\quad \wedge\ t \geq (1 - Max\ (\{frac\ (u\ c) \mid c.\ c \in \mathcal{X} \wedge isIntv\ (I\ c)\} \cup \{0\}))\ /\ 2$
⟨*proof*⟩

definition
rept :: $'s * ('c, t)\ cval \Rightarrow ('s * ('c, t)\ cval\ set)\ pmf \Rightarrow ('s * ('c, t)\ cval)\ pmf$
where
$rept\ s\ \mu\text{-}abs \equiv let\ (l, u) = s\ in$
$\quad if\ (\exists\ R'.\ (l, u) \in S \wedge \mu\text{-}abs = return\text{-}pmf\ (l, R') \wedge$
$\quad\quad (([u]_{\mathcal{R}} = R' \wedge (\forall\ c \in \mathcal{X}.\ u\ c > k\ c))))$
$\quad then\ return\text{-}pmf\ (l, u \oplus 0.5)$
$\quad else\ if$
$\quad\ (\exists\ R'.\ (l, u) \in S \wedge \mu\text{-}abs = return\text{-}pmf\ (l, R') \wedge R' \in Succ\ \mathcal{R}\ ([u]_{\mathcal{R}}) \wedge [u]_{\mathcal{R}} \neq R'$
$\quad\quad \wedge (\forall\ u \in R'.\ \forall\ c \in \mathcal{X}.\ \nexists\ d.\ d \leq k\ c \wedge\ u\ c = real\ d))$
$\quad then\ return\text{-}pmf\ (l, delayedR\ (SOME\ R'.\ \mu\text{-}abs = return\text{-}pmf\ (l, R'))\ u)$
$\quad else\ SOME\ \mu.\ \mu \in K\ s \wedge abst\ \mu = \mu\text{-}abs$

lemma $\mathcal{S}$-*L*:
$l \in L$ **if** $(l, R) \in \mathcal{S}$
⟨*proof*⟩

lemma $\mathcal{S}$-*inv*:
$(l, R) \in \mathcal{S} \Longrightarrow R \subseteq \{\!|inv\text{-}of\ A\ l|\!\}$
⟨*proof*⟩

lemma *upper-right-closed*:
assumes $\forall\ c{\in}\mathcal{X}.\ real\ (k\ c) < u\ c\ \ u \in R\ \ R \in \mathcal{R}\ \ t \geq 0$
shows $u \oplus t \in R$
⟨*proof*⟩

lemma *S-I*[*intro*]:
$(l, u) \in S$ **if** $l \in L\ \ u \in V\ \ u \vdash inv\text{-}of\ A\ l$
⟨*proof*⟩

lemma *rept-ex*:
assumes $\mu \in \mathcal{K}\ (abss\ s)$
shows $rept\ s\ \mu \in K\ s \wedge abst\ (rept\ s\ \mu) = \mu$ ⟨*proof*⟩

lemmas *rept-K*[*intro*] = *rept-ex*[*THEN conjunct1*]
lemmas *abst-rept-id*[*simp*] = *rept-ex*[*THEN conjunct2*]

lemma *abst-rept2*:
assumes $\mu \in \mathcal{K}\ s\ \ s \in \mathcal{S}$
shows $abst\ (rept\ (reps\ s)\ \mu) = \mu$
⟨*proof*⟩

lemma *rept-K2*:
assumes $\mu \in \mathcal{K}\ s\ \ s \in \mathcal{S}$
shows $rept\ (reps\ s)\ \mu \in K\ (reps\ s)$
⟨*proof*⟩

lemma *theI'*:
assumes $P\ a$
and $\bigwedge x.\ P\ x \Longrightarrow x = a$
shows $P\ (THE\ x.\ P\ x) \wedge (\forall\ y.\ P\ y \longrightarrow y = (THE\ x.\ P\ x))$
⟨*proof*⟩

lemma *cont-cfg-defined*:
fixes *cfg s*

assumes $cfg \in valid\text{-}cfg$ $s \in abst\ (action\ cfg)$
defines $x \equiv THE\ x.\ abss\ x = s \wedge x \in action\ cfg$
shows $(abss\ x = s \wedge x \in action\ cfg) \wedge (\forall\ y.\ abss\ y = s \wedge y \in action\ cfg \longrightarrow y = x)$
⟨*proof*⟩

definition
$absc' :: ('s * ('c,\ t)\ cval)\ cfg \Rightarrow ('s * ('c,\ t)\ cval\ set)\ cfg$
where
$absc'\ cfg = cfg\text{-}corec$
$(abss\ (state\ cfg))$
$(abst\ o\ action)$
$(\lambda\ cfg\ s.\ cont\ cfg\ (THE\ x.\ abss\ x = s \wedge x \in action\ cfg))\ cfg$

5.2.5 Configuration

definition
$absc :: ('s * ('c,\ t)\ cval)\ cfg \Rightarrow ('s * ('c,\ t)\ cval\ set)\ cfg$
where
$absc\ cfg = cfg\text{-}corec$
$(abss\ (state\ cfg))$
$(abst\ o\ action)$
$(\lambda\ cfg\ s.\ cont\ cfg\ (THE\ x.\ abss\ x = s \wedge x \in action\ cfg))\ cfg$

definition
$repcs :: 's * ('c,\ t)\ cval \Rightarrow ('s * ('c,\ t)\ cval\ set)\ cfg \Rightarrow ('s * ('c,\ t)\ cval)\ cfg$
where
$repcs\ s\ cfg = cfg\text{-}corec$
s
$(\lambda\ (s,\ cfg).\ rept\ s\ (action\ cfg))$
$(\lambda\ (s,\ cfg)\ s'.\ (s',\ cont\ cfg\ (abss\ s')))\ (s,\ cfg)$

definition
$repc\ cfg = repcs\ (reps\ (state\ cfg))\ cfg$

lemma $\mathcal{S}$*-state-absc-repc*[*simp*]:
$state\ cfg \in \mathcal{S} \Longrightarrow state\ (absc\ (repc\ cfg)) = state\ cfg$
⟨*proof*⟩

lemma *action-repc*:
$action\ (repc\ cfg) = rept\ (reps\ (state\ cfg))\ (action\ cfg)$
⟨*proof*⟩

lemma *action-absc*:
$action\ (absc\ cfg) = abst\ (action\ cfg)$
⟨*proof*⟩

lemma *action-absc'*:
$action\ (absc\ cfg) = map\text{-}pmf\ abss\ (action\ cfg)$
⟨*proof*⟩

lemma
notes *R-G.cfg-onD-state*[*simp del*]
assumes $state\ cfg \in \mathcal{S}$ $s' \in set\text{-}pmf\ (action\ (repc\ cfg))$ $cfg \in R\text{-}G.cfg\text{-}on\ (state\ cfg)$
shows $cont\ (repc\ cfg)\ s' = repcs\ s'\ (cont\ cfg\ (abss\ s'))$
⟨*proof*⟩

lemma *cont-repcs1*:
notes *R-G.cfg-onD-state*[*simp del*]
assumes $abss\ s \in \mathcal{S}$ $s' \in set\text{-}pmf\ (action\ (repcs\ s\ cfg))$ $cfg \in R\text{-}G.cfg\text{-}on\ (abss\ s)$
shows $cont\ (repcs\ s\ cfg)\ s' = repcs\ s'\ (cont\ cfg\ (abss\ s'))$

$\langle proof \rangle$

lemma *cont-absc-1*:
notes *MDP.cfg-onD-state*[*simp del*]
assumes $cfg \in valid\text{-}cfg$ $s' \in set\text{-}pmf$ $(action\ cfg)$
shows $cont\ (absc\ cfg)\ (abss\ s') = absc\ (cont\ cfg\ s')$
$\langle proof \rangle$

lemma *state-repc*:
$state\ (repc\ cfg) = reps\ (state\ cfg)$
$\langle proof \rangle$

lemma *abss-reps-id'*:
notes *R-G.cfg-onD-state*[*simp del*]
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $s \in set\text{-}pmf$ $(action\ cfg)$
shows $abss\ (reps\ s) = s$
$\langle proof \rangle$

lemma *valid-cfg-coinduct*[*coinduct set*: *valid-cfg*]:
assumes $P\ cfg$
assumes $\bigwedge cfg.\ P\ cfg \Longrightarrow state\ cfg \in S$
assumes $\bigwedge cfg.\ P\ cfg \Longrightarrow action\ cfg \in K\ (state\ cfg)$
assumes $\bigwedge cfg\ t.\ P\ cfg \Longrightarrow t \in action\ cfg \Longrightarrow P\ (cont\ cfg\ t)$
shows $cfg \in valid\text{-}cfg$
$\langle proof \rangle$

lemma *state-repcD*[*simp*]:
assumes $cfg \in R\text{-}G.cfg\text{-}on\ s$
shows $state\ (repc\ cfg) = reps\ s$
$\langle proof \rangle$

lemma *ccompatible-subs*[*intro*]:
assumes $ccompatible\ \mathcal{R}\ g$ $R \in \mathcal{R}$ $u \in R$ $u \vdash g$
shows $R \subseteq \{u.\ u \vdash g\}$
$\langle proof \rangle$

lemma *action-abscD*[*dest*]:
$cfg \in MDP.cfg\text{-}on\ s \Longrightarrow action\ (absc\ cfg) \in \mathcal{K}\ (abss\ s)$
$\langle proof \rangle$

lemma *repcs-valid*[*intro*]:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$
shows $repcs\ s\ cfg \in valid\text{-}cfg$
$\langle proof \rangle$

lemma *repc-valid*[*intro*]:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$
shows $repc\ cfg \in valid\text{-}cfg$
$\langle proof \rangle$

lemma *action-abst-repcs*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$
shows $abst\ (action\ (repcs\ s\ cfg)) = action\ cfg$
$\langle proof \rangle$

lemma *action-abst-repc*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$
shows $abst\ (action\ (repc\ cfg)) = action\ cfg$
$\langle proof \rangle$

lemma *state-absc*:
state (*absc cfg*) = *abss* (*state cfg*)
⟨*proof*⟩

lemma *state-repcs*[*simp*]:
state (*repcs s cfg*) = *s*
⟨*proof*⟩

lemma *repcs-bisim*:
notes *R-G.cfg-onD-state*[*simp del*]
assumes *cfg* ∈ *R-G.valid-cfg* *x* ∈ *S* *x* ~ *x′* *abss x* = *state cfg*
shows *absc* (*repcs x cfg*) = *absc* (*repcs x′ cfg*)
⟨*proof*⟩

named-theorems *R-G-I*

lemmas *R-G.valid-cfg-state-in-S*[*R-G-I*] *R-G.valid-cfgD*[*R-G-I*] *R-G.valid-cfg-action*

lemma *absc-repcs-id*:
notes *R-G.cfg-onD-state*[*simp del*]
assumes *cfg* ∈ *R-G.valid-cfg* *abss s* = *state cfg*
shows *absc* (*repcs s cfg*) = *cfg* ⟨*proof*⟩

lemma *absc-repc-id*:
notes *R-G.cfg-onD-state*[*simp del*]
assumes *cfg* ∈ *R-G.valid-cfg*
shows *absc* (*repc cfg*) = *cfg* ⟨*proof*⟩

lemma *K-cfg-map-absc*:
cfg ∈ *valid-cfg* ⟹ *K-cfg* (*absc cfg*) = *map-pmf absc* (*K-cfg cfg*)
⟨*proof*⟩

lemma *smap-comp*:
(*smap f o smap g*) = *smap* (*f o g*)
⟨*proof*⟩

lemma *state-abscD*[*simp*]:
assumes *cfg* ∈ *MDP.cfg-on s*
shows *state* (*absc cfg*) = *abss s*
⟨*proof*⟩

lemma *R-G-valid-cfg-coinduct*[*coinduct set*: *valid-cfg*]:
assumes *P cfg*
assumes $\bigwedge$*cfg*. *P cfg* ⟹ *state cfg* ∈ $\mathcal{S}$
assumes $\bigwedge$*cfg*. *P cfg* ⟹ *action cfg* ∈ $\mathcal{K}$ (*state cfg*)
assumes $\bigwedge$*cfg t*. *P cfg* ⟹ *t* ∈ *action cfg* ⟹ *P* (*cont cfg t*)
shows *cfg* ∈ *R-G.valid-cfg*
⟨*proof*⟩

lemma *absc-valid*[*intro*]:
assumes *cfg* ∈ *valid-cfg*
shows *absc cfg* ∈ *R-G.valid-cfg*
⟨*proof*⟩

lemma *K-cfg-set-absc*:

assumes $cfg \in valid\text{-}cfg$ $cfg' \in K\text{-}cfg\ cfg$
shows $absc\ cfg' \in K\text{-}cfg\ (absc\ cfg)$
⟨*proof*⟩

lemma *abst-action-repcs*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$
shows $abst\ (action\ (repcs\ s\ cfg)) = action\ cfg$
⟨*proof*⟩

lemma *abst-action-repc*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$
shows $abst\ (action\ (repc\ cfg)) = action\ cfg$
⟨*proof*⟩

lemma *K-elem-abss-inj'*:
assumes $\mu \in K\ s$
and $s \in S$
shows $inj\text{-}on\ abss\ (set\text{-}pmf\ \mu)$
⟨*proof*⟩

lemma *K-cfg-rept-aux*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$ $x \in rept\ s\ (action\ cfg)$
defines $t \equiv \lambda\ cfg'.\ THE\ s'.\ s' \in rept\ s\ (action\ cfg) \wedge s' \sim x$
shows $t\ cfg' = x$
⟨*proof*⟩

lemma *K-cfg-rept-action*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$ $cfg' \in set\text{-}pmf\ (K\text{-}cfg\ cfg)$
shows $abss\ (THE\ s'.\ s' \in rept\ s\ (action\ cfg) \wedge abss\ s' = state\ cfg') = state\ cfg'$
⟨*proof*⟩

lemma *K-cfg-map-repcs*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $abss\ s = state\ cfg$
defines $repc' \equiv (\lambda\ cfg'.\ repcs\ (THE\ s'.\ s' \in rept\ s\ (action\ cfg) \wedge abss\ s' = state\ cfg')\ cfg')$
shows $K\text{-}cfg\ (repcs\ s\ cfg) = map\text{-}pmf\ repc'\ (K\text{-}cfg\ cfg)$
⟨*proof*⟩

lemma *K-cfg-map-repc*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$
defines
$repc'\ cfg' \equiv repcs\ (THE\ s.\ s \in rept\ (reps\ (state\ cfg))\ (action\ cfg) \wedge abss\ s = state\ cfg')\ cfg'$
shows
$K\text{-}cfg\ (repc\ cfg) = map\text{-}pmf\ repc'\ (K\text{-}cfg\ cfg)$
⟨*proof*⟩

lemma *R-G-K-cfg-valid-cfgD*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$ $cfg' \in K\text{-}cfg\ cfg$
shows $cfg' = cont\ cfg\ (state\ cfg')$ $state\ cfg' \in action\ cfg$
⟨*proof*⟩

lemma *K-cfg-valid-cfgD*:
assumes $cfg \in valid\text{-}cfg$ $cfg' \in K\text{-}cfg\ cfg$
shows $cfg' = cont\ cfg\ (state\ cfg')$ $state\ cfg' \in action\ cfg$
⟨*proof*⟩

lemma *absc-bisim-abss*:
assumes $absc\ x = absc\ x'$
shows $state\ x \sim state\ x'$
⟨*proof*⟩

lemma *K-cfg-bisim-unique*:
assumes $cfg \in valid\text{-}cfg$ **and** $x \in K\text{-}cfg\ cfg\ x' \in K\text{-}cfg\ cfg$ **and** $state\ x \sim state\ x'$
shows $x = x'$
⟨*proof*⟩

lemma *absc-distr-self*:
$MDP.MC.T\ (absc\ cfg) = distr\ (MDP.MC.T\ cfg)\ MDP.MC.S\ (smap\ absc)$ **if** $cfg \in valid\text{-}cfg$
⟨*proof*⟩

lemma *R-G-trace-space-distr-eq*:
assumes $cfg \in R\text{-}G.valid\text{-}cfg\ abss\ s = state\ cfg$
shows $MDP.MC.T\ cfg = distr\ (MDP.MC.T\ (repcs\ s\ cfg))\ MDP.MC.S\ (smap\ absc)$
⟨*proof*⟩

lemma *repc-inj-on-K-cfg*:
assumes $cfg \in R\text{-}G.cfg\text{-}on\ s\ s \in \mathcal{S}$
shows $inj\text{-}on\ repc\ (set\text{-}pmf\ (K\text{-}cfg\ cfg))$
⟨*proof*⟩

lemma *smap-absc-iff*:
assumes $\bigwedge x\ y.\ x \in X \Longrightarrow smap\ abss\ x = smap\ abss\ y \Longrightarrow y \in X$
shows $(smap\ state\ xs \in X) = (smap\ (\lambda z.\ abss\ (state\ z))\ xs \in smap\ abss\ `\ X)$
⟨*proof*⟩

lemma *valid-abss-reps*[*simp*]:
assumes $cfg \in R\text{-}G.valid\text{-}cfg$
shows $abss\ (reps\ (state\ cfg)) = state\ cfg$
⟨*proof*⟩

lemma *in-space-UNIV*: $x \in space\ (count\text{-}space\ UNIV)$
⟨*proof*⟩

lemma *S-reps-$\mathcal{S}$-aux*:
$reps\ (l,\ R) \in S \Longrightarrow (l,\ R) \in \mathcal{S}$
⟨*proof*⟩

lemma *S-reps-$\mathcal{S}$*[*intro*]:
$reps\ s \in S \Longrightarrow s \in \mathcal{S}$
⟨*proof*⟩

lemma *absc-valid-cfg-eq*:
$absc\ `\ valid\text{-}cfg = R\text{-}G.valid\text{-}cfg$
⟨*proof*⟩

lemma *action-repcs*:
$action\ (repcs\ (l,\ u)\ cfg) = rept\ (l,\ u)\ (action\ cfg)$
⟨*proof*⟩

5.3 Equalities Between Measures of Trace Spaces

lemma *path-measure-eq-absc1-new*:
fixes $cfg\ s$
defines $cfg' \equiv absc\ cfg$
assumes *valid*: $cfg \in valid\text{-}cfg$
assumes X[*measurable*]: $X \in R\text{-}G.St$ **and** Y[*measurable*]: $Y \in MDP.St$
assumes P: $AE\ x\ in\ (R\text{-}G.T\ cfg').\ P\ x$ **and** Q: $AE\ x\ in\ (MDP.T\ cfg).\ Q\ x$
assumes P'[*measurable*]: $Measurable.pred\ R\text{-}G.St\ P$
and Q'[*measurable*]: $Measurable.pred\ MDP.St\ Q$
assumes *X-Y-closed*: $\bigwedge x\ y.\ P\ x \Longrightarrow smap\ abss\ y = x \Longrightarrow x \in X \Longrightarrow y \in Y \wedge Q\ y$

assumes *Y-X-closed*: $\bigwedge$ *x y*. *Q y* $\Longrightarrow$ *smap abss y* = *x* $\Longrightarrow$ *y* $\in$ *Y* $\Longrightarrow$ *x* $\in$ *X* $\wedge$ *P x*
shows
emeasure (*R-G.T cfg*′) *X* = *emeasure* (*MDP.T cfg*) *Y*
⟨*proof*⟩

lemma *path-measure-eq-repcs1-new*:
fixes *cfg s*
defines *cfg*′ ≡ *repcs s cfg*
assumes *s*: *abss s* = *state cfg*
assumes *valid*: *cfg* $\in$ *R-G.valid-cfg*
assumes *X*[*measurable*]: *X* $\in$ *R-G.St* **and** *Y*[*measurable*]: *Y* $\in$ *MDP.St*
assumes *P*: *AE x in* (*R-G.T cfg*). *P x* **and** *Q*: *AE x in* (*MDP.T cfg*′). *Q x*
assumes *P*′[*measurable*]: *Measurable.pred R-G.St P*
and *Q*′[*measurable*]: *Measurable.pred MDP.St Q*
assumes *X-Y-closed*: $\bigwedge$ *x y*. *P x* $\Longrightarrow$ *smap abss y* = *x* $\Longrightarrow$ *x* $\in$ *X* $\Longrightarrow$ *y* $\in$ *Y* $\wedge$ *Q y*
assumes *Y-X-closed*: $\bigwedge$ *x y*. *Q y* $\Longrightarrow$ *smap abss y* = *x* $\Longrightarrow$ *y* $\in$ *Y* $\Longrightarrow$ *x* $\in$ *X* $\wedge$ *P x*
shows
emeasure (*R-G.T cfg*) *X* = *emeasure* (*MDP.T cfg*′) *Y*
⟨*proof*⟩

lemma *region-compatible-suntil1*:
assumes (*holds* (λ*x*. φ (*reps x*)) *suntil holds* (λ*x*. ψ (*reps x*))) (*smap abss x*)
and *pred-stream* (λ *s*. φ (*reps* (*abss s*)) $\longrightarrow$ φ *s*) *x*
and *pred-stream* (λ *s*. ψ (*reps* (*abss s*)) $\longrightarrow$ ψ *s*) *x*
shows (*holds* φ *suntil holds* ψ) *x* ⟨*proof*⟩

lemma *region-compatible-suntil2*:
assumes (*holds* φ *suntil holds* ψ) *x*
and *pred-stream* (λ *s*. φ *s* $\longrightarrow$ φ (*reps* (*abss s*))) *x*
and *pred-stream* (λ *s*. ψ *s* $\longrightarrow$ ψ (*reps* (*abss s*))) *x*
shows (*holds* (λ*x*. φ (*reps x*)) *suntil holds* (λ*x*. ψ (*reps x*))) (*smap abss x*) ⟨*proof*⟩

lemma *region-compatible-suntil*:
assumes *pred-stream* (λ *s*. φ (*reps* (*abss s*)) $\longleftrightarrow$ φ *s*) *x*
and *pred-stream* (λ *s*. ψ (*reps* (*abss s*)) $\longleftrightarrow$ ψ *s*) *x*
shows (*holds* (λ*x*. φ (*reps x*)) *suntil holds* (λ*x*. ψ (*reps x*))) (*smap abss x*)
$\longleftrightarrow$ (*holds* φ *suntil holds* ψ) *x* ⟨*proof*⟩

lemma *reps-abss-S*:
assumes *reps* (*abss s*) $\in$ *S*
shows *s* $\in$ *S*
⟨*proof*⟩

lemma *measurable-sset*[*measurable* (*raw*)]:
assumes *f*[*measurable*]: *f* $\in$ *N* $\rightarrow_M$ *stream-space M* **and** *P*[*measurable*]: *Measurable.pred M P*
shows *Measurable.pred N* (λ*x*. $\forall$ *s*$\in$*sset* (*f x*). *P s*)
⟨*proof*⟩

lemma *path-measure-eq-repcs*′′*-new*:
notes *in-space-UNIV*[*measurable*]
fixes *cfg* φ ψ *s*
defines *cfg*′ ≡ *repcs s cfg*
defines φ′ ≡ *absp* φ **and** ψ′ ≡ *absp* ψ
assumes *s*: *abss s* = *state cfg*
assumes *valid*: *cfg* $\in$ *R-G.valid-cfg*
assumes *valid*′: *cfg*′ $\in$ *valid-cfg*
assumes *equiv-*φ: $\bigwedge$ *x*. *pred-stream* (λ *s*. *s* $\in$ *S*) *x*
$\Longrightarrow$ *pred-stream* (λ *s*. φ (*reps* (*abss s*)) $\longleftrightarrow$ φ *s*) (*state cfg*′ ## *x*)
and *equiv-*ψ: $\bigwedge$ *x*. *pred-stream* (λ *s*. *s* $\in$ *S*) *x*
$\Longrightarrow$ *pred-stream* (λ *s*. ψ (*reps* (*abss s*)) $\longleftrightarrow$ ψ *s*) (*state cfg*′ ## *x*)

shows
$emeasure\ (R\text{-}G.T\ cfg)\ \{x{\in}space\ R\text{-}G.St.\ (holds\ \varphi'\ suntil\ holds\ \psi')\ (state\ cfg\ \#\#\ x)\} =$
$emeasure\ (MDP.T\ cfg')\ \{x{\in}space\ MDP.St.\ (holds\ \varphi\ \ suntil\ holds\ \psi)\ \ (state\ cfg'\ \#\#\ x)\}$
⟨*proof*⟩

end

end
theory *PTA-Reachability*
imports *PTA*
begin

6 Classifying Regions for Divergence

6.1 Pairwise

coinductive *pairwise* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ stream \Rightarrow bool$ **for** P **where**
$P\ a\ b \Longrightarrow pairwise\ P\ (b\ \#\#\ xs) \Longrightarrow pairwise\ P\ (a\ \#\#\ b\ \#\#\ xs)$

lemma *pairwise-Suc*:
$pairwise\ P\ xs \Longrightarrow P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i))$
⟨*proof*⟩

lemma *Suc-pairwise*:
$\forall\ i.\ P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i)) \Longrightarrow pairwise\ P\ xs$
⟨*proof*⟩

lemma *pairwise-iff*:
$pairwise\ P\ xs \longleftrightarrow (\forall\ i.\ P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i)))$
⟨*proof*⟩

lemma *pairwise-stlD*:
$pairwise\ P\ xs \Longrightarrow pairwise\ P\ (stl\ xs)$
⟨*proof*⟩

lemma *pairwise-pairD*:
$pairwise\ P\ xs \Longrightarrow P\ (shd\ xs)\ (shd\ (stl\ xs))$
⟨*proof*⟩

lemma *pairwise-mp*:
assumes $pairwise\ P\ xs$ **and** *lift*: $\bigwedge x\ y.\ x \in sset\ xs \Longrightarrow y \in sset\ xs \Longrightarrow P\ x\ y \Longrightarrow Q\ x\ y$
shows $pairwise\ Q\ xs$ ⟨*proof*⟩

lemma *pairwise-sdropD*:
$pairwise\ P\ (sdrop\ i\ xs)$ **if** $pairwise\ P\ xs$
⟨*proof*⟩

6.2 Regions

lemma *gt-GreaterD*:
assumes $u \in region\ X\ I\ r\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ u\ c > k\ c$
shows $I\ c = Greater\ (k\ c)$
⟨*proof*⟩

lemma *const-ConstD*:
assumes $u \in region\ X\ I\ r\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ u\ c = d\ d \le k\ c$
shows $I\ c = Const\ d$
⟨*proof*⟩

lemma *not-Greater-bounded*:

assumes $I\ x \neq Greater\ (k\ x)\ x \in X\ valid\text{-}region\ X\ k\ I\ r\ u \in region\ X\ I\ r$
shows $u\ x \leq k\ x$
⟨*proof*⟩

lemma *Greater-closed*:
fixes $t :: real$
assumes $u \in region\ X\ I\ r\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ I\ c = Greater\ (k\ c)\ t > k\ c$
shows $u(c := t) \in region\ X\ I\ r$
⟨*proof*⟩

lemma *Greater-unbounded-aux*:
assumes $finite\ X\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ I\ c = Greater\ (k\ c)$
shows $\exists\ u \in region\ X\ I\ r.\ u\ c > t$
⟨*proof*⟩

6.3 Unbounded and Zero Regions

definition $unbounded\ x\ R \equiv \forall\ t.\ \exists\ u \in R.\ u\ x > t$

definition $zero\ x\ R \equiv \forall\ u \in R.\ u\ x = 0$

lemma *Greater-unbounded*:
assumes $finite\ X\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ I\ c = Greater\ (k\ c)$
shows $unbounded\ c\ (region\ X\ I\ r)$
⟨*proof*⟩

lemma *unbounded-Greater*:
assumes $valid\text{-}region\ X\ k\ I\ r\ c \in X\ unbounded\ c\ (region\ X\ I\ r)$
shows $I\ c = Greater\ (k\ c)$
⟨*proof*⟩

lemma *Const-zero*:
assumes $c \in X\ I\ c = Const\ 0$
shows $zero\ c\ (region\ X\ I\ r)$
⟨*proof*⟩

lemma *zero-Const*:
assumes $finite\ X\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ zero\ c\ (region\ X\ I\ r)$
shows $I\ c = Const\ 0$
⟨*proof*⟩

lemma *zero-all*:
assumes $finite\ X\ valid\text{-}region\ X\ k\ I\ r\ c \in X\ u \in region\ X\ I\ r\ u\ c = 0$
shows $zero\ c\ (region\ X\ I\ r)$
⟨*proof*⟩

7 Reachability

7.1 Definitions

locale *Probabilistic-Timed-Automaton-Regions-Reachability* =
Probabilistic-Timed-Automaton-Regions $k\ v\ n\ not\text{-}in\text{-}X\ A$
for $k\ v\ n\ not\text{-}in\text{-}X$ **and** $A :: ('c,\ t,\ 's)\ pta$ +
fixes $\varphi\ \psi :: ('s * ('c,\ t)\ cval) \Rightarrow bool$ **fixes** s
assumes φ: $\bigwedge x\ y.\ x \in S \Longrightarrow timed\text{-}bisim\ x\ y \Longrightarrow \varphi\ x \longleftrightarrow \varphi\ y$
assumes ψ: $\bigwedge x\ y.\ x \in S \Longrightarrow timed\text{-}bisim\ x\ y \Longrightarrow \psi\ x \longleftrightarrow \psi\ y$
assumes s[*intro*, *simp*]: $s \in S$
begin

definition $\varphi' \equiv absp\ \varphi$

definition $\psi' \equiv$ *absp* ψ
definition $s' \equiv$ *abss* s

lemma *s-s′-cfg-on*[*intro*]:
  assumes *cfg* $\in$ *MDP.cfg-on s*
  shows *absc cfg* $\in$ *R-G.cfg-on s′*
⟨*proof*⟩

lemma *s′-*$\mathcal{S}$[*simp*, *intro*]:
  $s' \in \mathcal{S}$
  ⟨*proof*⟩

lemma *s′-s-cfg-on*[*intro*]:
  assumes *cfg* $\in$ *R-G.cfg-on s′*
  shows *repcs s cfg* $\in$ *MDP.cfg-on s*
⟨*proof*⟩

lemma (**in** *Probabilistic-Timed-Automaton-Regions*) *compatible-stream*:
  assumes φ: $\bigwedge x\ y.\ x \in S \Longrightarrow x \sim y \Longrightarrow \varphi\ x \longleftrightarrow \varphi\ y$
  assumes *pred-stream* ($\lambda s.\ s \in S$) *xs*
    and [*intro*]: $x \in S$
   shows *pred-stream* ($\lambda s.\ \varphi$ (*reps* (*abss s*)) $= \varphi\ s$) (x ## *xs*)
⟨*proof*⟩

lemma φ*-stream′*:
  pred-stream ($\lambda s.\ \varphi$ (*reps* (*abss s*)) $= \varphi\ s$) (x ## *xs*) **if** *pred-stream* ($\lambda s.\ s \in S$) *xs* $x \in S$
  ⟨*proof*⟩

lemma ψ*-stream′*:
  pred-stream ($\lambda s.\ \psi$ (*reps* (*abss s*)) $= \psi\ s$) (x ## *xs*) **if** *pred-stream* ($\lambda s.\ s \in S$) *xs* $x \in S$
  ⟨*proof*⟩

lemmas φ*-stream* = *compatible-stream*[*of* φ, *OF* φ]
lemmas ψ*-stream* = *compatible-stream*[*of* ψ, *OF* ψ]

7.2 Easier Result on All Configurations

lemma *suntil-reps*:
  assumes
    $\forall s \in$ *sset* (*smap abss y*). $s \in \mathcal{S}$
    (*holds* φ' *suntil holds* ψ') (s' ## *smap abss y*)
  shows (*holds* φ *suntil holds* ψ) (s ## y)
  ⟨*proof*⟩

lemma *suntil-abss*:
  assumes
    $\forall s \in$ *sset y*. $s \in S$
    (*holds* φ *suntil holds* ψ) (s ## y)
  shows
    (*holds* φ' *suntil holds* ψ') (s' ## *smap abss y*)
  ⟨*proof*⟩

theorem *P-sup-sunitl-eq*:
  notes [*measurable*] = *in-space-UNIV* **and** [*iff*] = *pred-stream-iff*
  shows
    (*MDP.P-sup s* (λx. (*holds* φ *suntil holds* ψ) (s ## x)))
   = (*R-G.P-sup s′* (λx. (*holds* φ' *suntil holds* ψ') (s' ## x)))
  ⟨*proof*⟩

end

7.3 Divergent Adversaries

context *Probabilistic-Timed-Automaton*
begin

definition *elapsed* $u\ u' \equiv Max\ (\{u'\ c - u\ c \mid c.\ c \in \mathcal{X}\} \cup \{0\})$

definition *eq-elapsed* $u\ u' \equiv elapsed\ u\ u' > 0 \longrightarrow (\forall\ c \in \mathcal{X}.\ u'\ c - u\ c = elapsed\ u\ u')$

fun *dur* :: ('*c*, *t*) *cval stream* ⇒ *nat* ⇒ *t* **where**
dur - $0 = 0$ |
dur (*x* ## *y* ## *xs*) (*Suc i*) = *elapsed x y* + *dur* (*y* ## *xs*) *i*

definition *divergent* $\omega \equiv \forall\ t.\ \exists\ n.\ dur\ \omega\ n > t$

definition *div-cfg cfg* ≡ *AE* ω *in MDP.MC.T cfg. divergent* (*smap* (*snd o state*) ω)

definition $\mathcal{R}$-*div* ω ≡
$\forall x \in \mathcal{X}.\ (\forall\ i.\ (\exists\ j \geq i.\ zero\ x\ (\omega\ !!\ j)) \wedge (\exists\ j \geq i.\ \neg\ zero\ x\ (\omega\ !!\ j)))$
$\vee\ (\exists\ i.\ \forall\ j \geq i.\ unbounded\ x\ (\omega\ !!\ j))$

definition *R-G-div-cfg cfg* ≡ *AE* ω *in MDP.MC.T cfg.* $\mathcal{R}$-*div* (*smap* (*snd o state*) ω)

end

context *Probabilistic-Timed-Automaton-Regions*
begin

definition *cfg-on-div st* ≡ *MDP.cfg-on st* ∩ {*cfg. div-cfg cfg*}

definition *R-G-cfg-on-div st* ≡ *R-G.cfg-on st* ∩ {*cfg. R-G-div-cfg cfg*}

lemma *measurable-*$\mathcal{R}$*-div*[*measurable*]: *Measurable.pred MDP.MC.S* $\mathcal{R}$-*div*
⟨*proof*⟩

lemma *elapsed-ge0*[*simp*]: $elapsed\ x\ y \geq 0$
⟨*proof*⟩

lemma *dur-pos*:
$dur\ xs\ i \geq 0$
⟨*proof*⟩

lemma *dur-mono*:
$i \leq j \Longrightarrow dur\ xs\ i \leq dur\ xs\ j$
⟨*proof*⟩

lemma *dur-monoD*:
assumes $dur\ xs\ i < dur\ xs\ j$
shows $i < j$ ⟨*proof*⟩

lemma *elapsed-0D*:
assumes $c \in \mathcal{X}$ $elapsed\ u\ u' \leq 0$
shows $u'\ c - u\ c \leq 0$
⟨*proof*⟩

lemma *elapsed-ge*:
assumes *eq-elapsed* $u\ u'$ $c \in \mathcal{X}$
shows $elapsed\ u\ u' \geq u'\ c - u\ c$
⟨*proof*⟩

lemma *elapsed-eq*:
  assumes $eq\text{-}elapsed\ u\ u'\ c \in \mathcal{X}\ u'\ c - u\ c \geq 0$
  shows $elapsed\ u\ u' = u'\ c - u\ c$
  ⟨*proof*⟩

lemma *dur-shift*:
  $dur\ \omega\ (i + j) = dur\ \omega\ i + dur\ (sdrop\ i\ \omega)\ j$
  ⟨*proof*⟩

lemma *dur-zero*:
  assumes
    $\forall\ i.\ xs\ !!\ i \in \omega\ !!\ i\ \forall\ j \leq i.\ zero\ x\ (\omega\ !!\ j)\ x \in \mathcal{X}$
    $\forall\ i.\ eq\text{-}elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)$
  shows $dur\ xs\ i = 0$ ⟨*proof*⟩

lemma *dur-zero-tail*:
  assumes $\forall\ i.\ xs\ !!\ i \in \omega\ !!\ i\ \forall\ k \geq i.\ k \leq j \longrightarrow zero\ x\ (\omega\ !!\ k)\ x \in \mathcal{X}\ j \geq i$
    $\forall\ i.\ eq\text{-}elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)$
  shows $dur\ xs\ j = dur\ xs\ i$
⟨*proof*⟩

lemma *elapsed-ge-pos*:
  fixes $u :: ('c,\ t)\ cval$
  assumes $eq\text{-}elapsed\ u\ u'\ c \in \mathcal{X}\ u \in V\ u' \in V$
  shows $elapsed\ u\ u' \leq u'\ c$
⟨*proof*⟩

lemma *dur-Suc*:
  $dur\ xs\ (Suc\ i) - dur\ xs\ i = elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)$
  ⟨*proof*⟩

inductive *trans* **where**
  succ: $t \geq 0 \Longrightarrow u' = u \oplus t \Longrightarrow trans\ u\ u'$ |
  reset: $set\ l \subseteq \mathcal{X} \Longrightarrow u' = clock\text{-}set\ l\ 0\ u \Longrightarrow trans\ u\ u'$ |
  id: $u = u' \Longrightarrow trans\ u\ u'$

abbreviation $stream\text{-}trans \equiv pairwise\ trans$

lemma *K-cfg-trans*:
  assumes $cfg \in MDP.cfg\text{-}on\ (l,\ R)\ cfg' \in K\text{-}cfg\ cfg\ state\ cfg' = (l',\ R')$
  shows $trans\ R\ R'$
⟨*proof*⟩

lemma *enabled-stream-trans*:
  assumes $cfg \in valid\text{-}cfg\ MDP.MC.enabled\ cfg\ xs$
  shows $stream\text{-}trans\ (smap\ (snd\ o\ state)\ xs)$
  ⟨*proof*⟩

lemma *stream-trans-trans*:
  assumes $stream\text{-}trans\ xs$
  shows $trans\ (xs\ !!\ i)\ (stl\ xs\ !!\ i)$
⟨*proof*⟩

lemma *trans-eq-elapsed*:
  assumes $trans\ u\ u'\ u \in V$
  shows $eq\text{-}elapsed\ u\ u'$
⟨*proof*⟩

lemma *pairwise-trans-eq-elapsed*:
  assumes $stream\text{-}trans\ xs\ pred\text{-}stream\ (\lambda\ u.\ u \in V)\ xs$

shows *pairwise eq-elapsed xs*
⟨*proof*⟩

lemma *not-reset-dur*:
assumes $\forall k>i.\ k \le j \longrightarrow \neg\ zero\ c\ ([xs\ !!\ k]_{\mathcal{R}})\ j \ge i\ c \in \mathcal{X}\ stream\text{-}trans\ xs$
$\forall\ i.\ eq\text{-}elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)\ \forall\ i.\ xs\ !!\ i \in V$
shows $dur\ xs\ j - dur\ xs\ i = (xs\ !!\ j)\ c - (xs\ !!\ i)\ c$
⟨*proof*⟩

lemma *not-reset-dur′*:
assumes $\forall j \ge i.\ \neg\ zero\ c\ ([xs\ !!\ j]_{\mathcal{R}})\ j \ge i\ c \in \mathcal{X}\ stream\text{-}trans\ xs$
$\forall\ i.\ eq\text{-}elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)\ \forall\ j.\ xs\ !!\ j \in V$
shows $dur\ xs\ j - dur\ xs\ i = (xs\ !!\ j)\ c - (xs\ !!\ i)\ c$
⟨*proof*⟩

lemma *not-reset-unbounded*:
assumes $\forall j \ge i.\ \neg\ zero\ c\ ([xs\ !!\ j]_{\mathcal{R}})\ j \ge i\ c \in \mathcal{X}\ stream\text{-}trans\ xs$
$\forall\ i.\ eq\text{-}elapsed\ (xs\ !!\ i)\ (xs\ !!\ Suc\ i)\ \forall\ j.\ xs\ !!\ j \in V$
$unbounded\ c\ ([xs\ !!\ i]_{\mathcal{R}})$
shows $unbounded\ c\ ([xs\ !!\ j]_{\mathcal{R}})$
⟨*proof*⟩

lemma *gt-unboundedD*:
assumes $u \in R$
and $R \in \mathcal{R}$
and $c \in \mathcal{X}$
and $real\ (k\ c) < u\ c$
shows $unbounded\ c\ R$
⟨*proof*⟩

definition $trans' :: ('c,\ t)\ cval \Rightarrow ('c,\ t)\ cval \Rightarrow bool$ **where**
$trans'\ u\ u' \equiv$
$((\forall\ c \in \mathcal{X}.\ u\ c > k\ c \wedge u'\ c > k\ c \wedge u \ne u') \longrightarrow u' = u \oplus 0.5) \wedge$
$((\exists\ c \in \mathcal{X}.\ u\ c = 0 \wedge u'\ c > 0 \wedge (\forall c{\in}\mathcal{X}.\ \nexists d.\ d \le k\ c \wedge u'\ c = real\ d))$
$\longrightarrow u' = delayedR\ ([u']_{\mathcal{R}})\ u)$

lemma *zeroI*:
assumes $c \in \mathcal{X}\ u \in V\ u\ c = 0$
shows $zero\ c\ ([u]_{\mathcal{R}})$
⟨*proof*⟩

lemma *zeroD*:
$u\ x = 0$ **if** $zero\ x\ ([u]_{\mathcal{R}})\ u \in V$
⟨*proof*⟩

lemma *not-zeroD*:
assumes $\neg\ zero\ x\ ([u]_{\mathcal{R}})\ u \in V\ x \in \mathcal{X}$
shows $u\ x > 0$
⟨*proof*⟩

lemma *not-const-intv*:
assumes $u \in V\ \forall c{\in}\mathcal{X}.\ \nexists d.\ d \le k\ c \wedge u\ c = real\ d$
shows $\forall c{\in}\mathcal{X}.\ \forall u \in [u]_{\mathcal{R}}.\ \nexists d.\ d \le k\ c \wedge u\ c = real\ d$
⟨*proof*⟩

lemma *K-cfg-trans′*:
assumes $repcs\ (l,\ u)\ cfg \in MDP.cfg\text{-}on\ (l,\ u)\ cfg' \in K\text{-}cfg\ (repcs\ (l,\ u)\ cfg)$
$state\ cfg' = (l',\ u')\ (l,\ u) \in S\ cfg \in R\text{-}G.valid\text{-}cfg\ abss\ (l,\ u) = state\ cfg$

shows *trans′ u u′*
⟨*proof*⟩

coinductive *enabled-repcs* **where**
enabled-repcs (*shd xs*) (*stl xs*) $\Longrightarrow$ *shd xs* = *repcs st′ cfg′* $\Longrightarrow$ *st′* ∈ *rept st* (*action cfg*)
$\Longrightarrow$ *abss st′* = *state cfg′*
$\Longrightarrow$ *cfg′* ∈ *R-G.valid-cfg*
$\Longrightarrow$ *enabled-repcs* (*repcs st cfg*) *xs*

lemma *K-cfg-rept-in*:
assumes *cfg* ∈ *R-G.valid-cfg*
and *abss st* = *state cfg*
and *cfg′* ∈ *K-cfg cfg*
shows (*THE s′*. *s′* ∈ *set-pmf* (*rept st* (*action cfg*)) ∧ *abss s′* = *state cfg′*)
∈ *set-pmf* (*rept st* (*action cfg*))
⟨*proof*⟩

lemma *enabled-repcsI*:
assumes *cfg* ∈ *R-G.valid-cfg abss st* = *state cfg MDP.MC.enabled* (*repcs st cfg*) *xs*
shows *enabled-repcs* (*repcs st cfg*) *xs* ⟨*proof*⟩

lemma *repcs-eq-rept*:
rept st (*action cfg*) = *rept st′′* (*action cfg′′*) **if** *repcs st cfg* = *repcs st′′ cfg′′*
⟨*proof*⟩

lemma *enabled-stream-trans′*:
assumes *cfg* ∈ *R-G.valid-cfg abss st* = *state cfg MDP.MC.enabled* (*repcs st cfg*) *xs*
shows *pairwise trans′* (*smap* (*snd o state*) *xs*)
⟨*proof*⟩

lemma *divergent-$\mathcal{R}$-divergent*:
assumes *in-S*: *pred-stream* (λ *u*. *u* ∈ *V*) *xs*
and *div*: *divergent xs*
and *trans*: *stream-trans xs*
shows $\mathcal{R}$-*div* (*smap* (λ *u*. $[u]_{\mathcal{R}}$) *xs*) (**is** $\mathcal{R}$-*div ?ω*)
⟨*proof*⟩

lemma (**in** −)
fixes *f* :: *nat* ⇒ *real*
assumes ∀ *i*. *f i* ≥ *0* ∀ *i*. ∃ *j* ≥ *i*. *f j* > *d d* > *0*
shows ∃ *n*. (∑ *i* ≤ *n*. *f i*) > *t*
⟨*proof*⟩

lemma *dur-ev-exceedsI*:
assumes ∀ *i*. ∃ *j* ≥ *i*. *dur xs j* − *dur xs i* ≥ *d* **and** *d* > *0*
obtains *i* **where** *dur xs i* > *t*
⟨*proof*⟩

lemma *not-reset-mono*:
assumes *stream-trans xs shd xs c1* ≥ *shd xs c2 stream-all* (λ *u*. *u* ∈ *V*) *xs c2* ∈ $\mathcal{X}$
shows (*holds* (λ *u*. *u c1* ≥ *u c2*) *until holds* (λ *u*. *u c1* = *0*)) *xs* ⟨*proof*⟩

lemma $\mathcal{R}$-*divergent-divergent-aux*:
fixes *xs* :: (′*c*, *t*) *cval stream*
assumes *stream-trans xs stream-all* (λ *u*. *u* ∈ *V*) *xs*
(*xs* !! *i*) *c1* = *0* ∃ *k* > *i*. *k* ≤ *j* ∧ (*xs* !! *k*) *c2* = *0*
∀ *k* > *i*. *k* ≤ *j* ⟶ (*xs* !! *k*) *c1* ≠ *0*
c1 ∈ $\mathcal{X}$ *c2* ∈ $\mathcal{X}$
shows (*xs* !! *j*) *c1* ≥ (*xs* !! *j*) *c2*

$\langle proof \rangle$

lemma *unbounded-all*:
 assumes $R \in \mathcal{R}$ $u \in R$ *unbounded* x R $x \in \mathcal{X}$
 shows $u\ x > k\ x$
$\langle proof \rangle$

lemma *trans-not-delay-mono*:
 $u'\ c \leq u\ c$ **if** *trans* $u\ u'$ $u \in V$ $x \in \mathcal{X}$ $u'\ x = 0$ $c \in \mathcal{X}$
 $\langle proof \rangle$

lemma *dur-reset*:
 assumes *pairwise eq-elapsed xs pred-stream* $(\lambda\ u.\ u \in V)$ *xs zero* x $([xs\ !!\ Suc\ i]_{\mathcal{R}})$ $x \in \mathcal{X}$
 shows *dur xs* $(Suc\ i)$ $-$ *dur xs* $i = 0$
$\langle proof \rangle$

lemma *resets-mono-0'*:
 assumes *pairwise eq-elapsed xs stream-all* $(\lambda\ u.\ u \in V)$ *xs stream-trans xs*
 $\forall\ j \leq i.\ zero\ x\ ([xs\ !!\ j]_{\mathcal{R}})$ $x \in \mathcal{X}$ $c \in \mathcal{X}$
 shows $(xs\ !!\ i)\ c = (xs\ !!\ 0)\ c \vee (xs\ !!\ i)\ c = 0$
$\langle proof \rangle$

lemma *resets-mono'*:
 assumes *pairwise eq-elapsed xs pred-stream* $(\lambda\ u.\ u \in V)$ *xs stream-trans xs*
 $\forall\ k \geq i.\ k \leq j \longrightarrow zero\ x\ ([xs\ !!\ k]_{\mathcal{R}})$ $x \in \mathcal{X}$ $c \in \mathcal{X}$ $i \leq j$
 shows $(xs\ !!\ j)\ c = (xs\ !!\ i)\ c \vee (xs\ !!\ j)\ c = 0$ $\langle proof \rangle$

lemma *resets-mono*:
 assumes *pairwise eq-elapsed xs pred-stream* $(\lambda\ u.\ u \in V)$ *xs stream-trans xs*
 $\forall\ k \geq i.\ k \leq j \longrightarrow zero\ x\ ([xs\ !!\ k]_{\mathcal{R}})$ $x \in \mathcal{X}$ $c \in \mathcal{X}$ $i \leq j$
 shows $(xs\ !!\ j)\ c \leq (xs\ !!\ i)\ c$ $\langle proof \rangle$

lemma *$\mathcal{R}$-divergent-divergent-aux2*:
 fixes $M :: (nat \Rightarrow bool)\ set$
 assumes $\forall\ i.\ \forall\ P \in M.\ \exists\ j \geq i.\ P\ j$ $M \neq \{\}$ *finite* M
 shows $\forall i.\exists j{\geq}i.\exists k{>}j.\exists\ P \in M.\ P\ j \wedge P\ k \wedge (\forall\ m < k.\ j < m \longrightarrow \neg\ P\ m)$
 $\wedge\ (\forall\ Q \in M.\ \exists\ m \leq k.\ j < m \wedge Q\ m)$
$\langle proof \rangle$

lemma *$\mathcal{R}$-divergent-divergent*:
 assumes *in-S*: *pred-stream* $(\lambda\ u.\ u \in V)$ *xs*
 and *div*: *$\mathcal{R}$-div* $(smap\ (\lambda\ u.\ [u]_{\mathcal{R}})\ xs)$
 and *trans*: *stream-trans xs*
 and *trans'*: *pairwise trans' xs*
 and *unbounded-not-const*:
 $\forall u.\ (\forall c{\in}\mathcal{X}.\ real\ (k\ c) < u\ c) \longrightarrow \neg\ ev\ (alw\ (\lambda xs.\ shd\ xs = u))\ xs$
 shows *divergent xs*
 $\langle proof \rangle$

lemma *cfg-on-div-absc*:
 notes *in-space-UNIV*[*measurable*]
 assumes $cfg \in$ *cfg-on-div st* $st \in S$
 shows *absc cfg* $\in$ *R-G-cfg-on-div* $(abss\ st)$
$\langle proof \rangle$

definition
 alternating cfg $=$ $(AE\ \omega\ in\ MDP.MC.T\ cfg.$
 $alw\ (ev\ (HLD\ \{cfg.\ \forall\, cfg' \in K\text{-}cfg\ cfg.\ fst\ (state\ cfg') = fst\ (state\ cfg)\}))\ \omega)$

lemma *K-cfg-same-loc-iff*:
 $(\forall\, cfg'{\in}\ K\text{-}cfg\ cfg.\ fst\ (state\ cfg') = fst\ (state\ cfg))$

$\longleftrightarrow$ (∀ *cfg'*∈ *K-cfg* (*absc cfg*). *fst* (*state cfg'*) = *fst* (*state* (*absc cfg*)))
if *cfg* ∈ *valid-cfg*
⟨*proof*⟩

lemma (**in** −) *stream-all2-flip*:
stream-all2 (λ*a b*. *R b a*) *xs ys* = *stream-all2 R ys xs*
⟨*proof*⟩

lemma *AE-alw-ev-same-loc-iff*:
assumes *cfg* ∈ *valid-cfg*
shows *alternating cfg* ⟷ *alternating* (*absc cfg*)
⟨*proof*⟩

lemma *AE-alw-ev-same-loc-iff'*:
assumes *cfg* ∈ *R-G.cfg-on* (*abss st*) *st* ∈ *S*
shows *alternating cfg* ⟷ *alternating* (*repcs st cfg*)
⟨*proof*⟩

lemma (**in** −) *cval-add-non-id*:
False **if** *b* ⊕ *d* = *b d* > *0* **for** *d* :: *real*
⟨*proof*⟩

lemma *repcs-unbounded-AE-non-loop-end-strong*:
assumes *cfg* ∈ *R-G.cfg-on* (*abss st*) *st* ∈ *S*
and *alternating cfg*
shows *AE* ω *in MDP.MC.T* (*repcs st cfg*).
(∀ *u* :: (*'c* ⇒ *real*). (∀ *c* ∈ $\mathcal{X}$. *u c* > *real* (*k c*)) ⟶
¬ (*ev* (*alw* (λ *xs*. *shd xs* = *u*))) (*smap* (*snd o state*) ω)) (**is** *AE* ω *in ?M*. *?P* ω)
⟨*proof*⟩

lemma *cfg-on-div-repcs-strong*:
notes *in-space-UNIV*[*measurable*]
assumes *cfg* ∈ *R-G-cfg-on-div* (*abss st*) *st* ∈ *S* **and** *alternating cfg*
shows *repcs st cfg* ∈ *cfg-on-div st*
⟨*proof*⟩

lemma *repcs-unbounded-AE-non-loop-end*:
assumes *cfg* ∈ *R-G.cfg-on* (*abss st*) *st* ∈ *S*
shows *AE* ω *in MDP.MC.T* (*repcs st cfg*).
(∀ *s* :: (*'s* × (*'c* ⇒ *real*)). (∀ *c* ∈ $\mathcal{X}$. *snd s c* > *k c*) ⟶
¬ (*ev* (*alw* (λ *xs*. *shd xs* = *s*))) (*smap state* ω)) (**is** *AE* ω *in ?M*. *?P* ω)
⟨*proof*⟩

end

7.4 Main Result

context *Probabilistic-Timed-Automaton-Regions-Reachability*
begin

lemma *R-G-cfg-on-valid*:
cfg ∈ *R-G.valid-cfg* **if** *cfg* ∈ *R-G-cfg-on-div s'*
⟨*proof*⟩

lemma *cfg-on-valid*:
cfg ∈ *valid-cfg* **if** *cfg* ∈ *cfg-on-div s*
⟨*proof*⟩

abbreviation *path-measure P cfg* ≡ *emeasure* (*MDP.T cfg*) {*x*∈*space MDP.St*. *P x*}
abbreviation *R-G-path-measure P cfg* ≡ *emeasure* (*R-G.T cfg*) {*x*∈*space R-G.St*. *P x*}

abbreviation *progressive st* ≡ *cfg-on-div st* ∩ {*cfg. alternating cfg*}
abbreviation *R-G-progressive st* ≡ *R-G-cfg-on-div st* ∩ {*cfg. alternating cfg*}

Summary of our results on divergent configurations:

lemma *absc-valid-cfg-eq*:
  absc ‘ *progressive s* = *R-G-progressive s′*
  ⟨*proof*⟩

Main theorem:

theorem *Min-Max-reachability*:
  notes *in-space-UNIV*[*measurable*] **and** [*iff*] = *pred-stream-iff*
  shows
  $(\bigsqcup cfg \in progressive\ s.\ path\text{-}measure\ (\lambda\ x.\ (holds\ \varphi\ suntil\ holds\ \psi)\ (s\ \#\#\ x))\ cfg)$
  $= (\bigsqcup cfg \in R\text{-}G\text{-}progressive\ s'.\ R\text{-}G\text{-}path\text{-}measure\ (\lambda\ x.\ (holds\ \varphi'\ suntil\ holds\ \psi')\ (s'\ \#\#\ x))\ cfg)$
  $\wedge (\bigsqcap cfg \in progressive\ s.\ path\text{-}measure\ (\lambda\ x.\ (holds\ \varphi\ suntil\ holds\ \psi)\ (s\ \#\#\ x))\ cfg)$
  $= (\bigsqcap cfg \in R\text{-}G\text{-}progressive\ s'.\ R\text{-}G\text{-}path\text{-}measure\ (\lambda\ x.\ (holds\ \varphi'\ suntil\ holds\ \psi')\ (s'\ \#\#\ x))\ cfg)$
⟨*proof*⟩

end

end

References

[1] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Th. Comp. Sci.*, 282(1).

[2] S. Wimmer and J. Hölzl. MDP + TA = PTA: Probabilistic timed automata, formalized. In J. Avigad and A. Mahboubi, editors, *ITP 2018, Proceedings*, Lecture Notes in Computer Science. Springer, 2018.