Probabilistic Timed Automata

Simon Wimmer and Johannes Hölzl

June 17, 2024

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

lemma *T-eq-rel-half*[*consumes 4*, *case-names prob sets cont*]:
fixes $R :: 's \Rightarrow 't \Rightarrow bool$ **and** $f :: 's \Rightarrow 't$ **and** $S :: 's\ set$
assumes *R-def*: $\bigwedge s\ t.\ R\ s\ t \longleftrightarrow (s \in S \wedge f\ s = t)$
assumes *A*[*measurable*]: $A \in \mathit{sets}\ (\mathit{stream\text{-}space}\ (\mathit{count\text{-}space}\ \mathit{UNIV}))$
and *B*[*measurable*]: $B \in \mathit{sets}\ (\mathit{stream\text{-}space}\ (\mathit{count\text{-}space}\ \mathit{UNIV}))$
and *AB*: $\mathit{rel\text{-}set\text{-}strong}\ (\mathit{stream\text{-}all2}\ R)\ A\ B$ **and** *KL*: $\mathit{rel\text{-}fun}\ R\ (\mathit{rel\text{-}pmf}\ R)\ K\ L$ **and** *xy*: $R\ x\ y$
shows $\mathit{MC\text{-}syntax}.T\ K\ x\ A = \mathit{MC\text{-}syntax}.T\ L\ y\ B$
⟨*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**
$\mathit{is\text{-}upper\text{-}right}\ R \equiv (\forall\ t \geq 0.\ \forall\ u \in R.\ u \oplus t \in R)$

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

lemma *region-continuous*:
assumes *valid-region X k I r*
defines *R*: $R \equiv \mathit{region}\ X\ I\ r$
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 $(\forall\ x \in X.\ \mathit{isGreater}\ (I\ x)) \longleftrightarrow \mathit{is\text{-}upper\text{-}right}\ (\mathit{region}\ X\ I\ r)$
⟨*proof*⟩

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

context *AlphaClosure*

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

2.1 Justifying Timed Until vs *suntil*

lemma *guard-continuous*:
assumes $u \vdash g$ $u \oplus t \vdash g$ $0 \leq (t'::'t::time)$ $t' \leq t$
shows $u \oplus t' \vdash g$
⟨*proof*⟩

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
$edges :: ('c,\ 't,\ 's)\ transition \Rightarrow ('s * ('c,\ 't)\ cconstraint * ('c\ set * 's)\ pmf * 'c\ set * 's)\ set$
where

edges $\equiv \lambda\ (l,\ g,\ p).\ \{(l,\ g,\ p,\ X,\ l') \mid X\ l'.\ (X,\ l') \in$ *set-pmf p*$\}$

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

definition
trans-of :: $('c,\ 't,\ 's)$ *pta* $\Rightarrow$ $('c,\ 't,\ 's)$ *transition set*
where
trans-of $\equiv$ *fst*

definition
inv-of :: $('c,\ 'time,\ 's)$ *pta* $\Rightarrow$ $('c,\ 'time,\ 's)$ *invassn*
where
inv-of $\equiv$ *snd*

no-notation *transition* (- $\vdash$ - $\longrightarrow^{-,-,-}$ - $[61,61,61,61,61,61]\ 61$)

abbreviation *transition* ::
$('c,\ 'time,\ 's)$ *pta* $\Rightarrow$ $'s \Rightarrow ('c,\ 'time)$ *cconstraint* $\Rightarrow ('c\ set * 's)$ *pmf* $\Rightarrow$ $'c$ *set* $\Rightarrow 's \Rightarrow$ *bool*
(- $\vdash$ - $\longrightarrow^{-,-,-}$ - $[61,61,61,61,61,61]\ 61$) **where**
$(A \vdash l \longrightarrow^{g,p,X} l') \equiv (l,\ g,\ p,\ X,\ l') \in$ *Edges A*

definition
locations :: $('c,\ 't,\ 's)$ *pta* $\Rightarrow$ $'s$ *set*
where
locations A $\equiv$ (*fst* ‘ *Edges A*) $\cup$ ((*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* $\Rightarrow ('c * 't)$ *set*
where
collect-clkt S $= \bigcup\ \{$*collect-clock-pairs* (*fst* (*snd t*)) $\mid t\ .\ t \in S\}$

definition *collect-clki* :: $('c,\ 't :: time,\ 's)$ *invassn* $\Rightarrow ('c * 't)$ *set*
where
collect-clki I $= \bigcup\ \{$*collect-clock-pairs* $(I\ x) \mid x.$ *True*$\}$

definition *clkp-set* :: $('c,\ 't :: time,\ 's)$ *pta* $\Rightarrow ('c * 't)$ *set*
where
clkp-set A $=$ *collect-clki* (*inv-of A*) $\cup$ *collect-clkt* (*trans-of A*)

definition *collect-clkvt* :: $('c,\ 't :: time,\ 's)$ *pta* $\Rightarrow$ $'c$ *set*
where
collect-clkvt A $= \bigcup$ ((*fst o snd o snd o snd*) ‘ *Edges A*)

abbreviation *clocks* **where** *clocks A* $\equiv$ *fst* ‘ *clkp-set A* $\cup$ *collect-clkvt A*

definition *valid-abstraction*
where
valid-abstraction A X k $\equiv$
$(\forall\,(x,m) \in$ *clkp-set A*. $m \leq k\ x \wedge x \in X \wedge m \in \mathbb{N}) \wedge$ *collect-clkvt A* $\subseteq X \wedge$ *finite X*

lemma *valid-abstractionD*[*dest*]:
assumes *valid-abstraction A X k*
shows $(\forall\,(x,m) \in$ *clkp-set A*. $m \leq k\ x \wedge x \in X \wedge m \in \mathbb{N})$ *collect-clkvt A* $\subseteq X$ *finite X*
⟨*proof*⟩

lemma *valid-abstractionI*[*intro*]:
assumes $(\forall\,(x,m) \in$ *clkp-set A*. $m \leq k\ x \wedge x \in X \wedge m \in \mathbb{N})$ *collect-clkvt A* $\subseteq X$ *finite X*
shows *valid-abstraction A X k*
⟨*proof*⟩

3.2 Operational Semantics as an MDP

abbreviation (*input*) *clock-set-set* :: $'c\ set \Rightarrow {'t}::time \Rightarrow ('c,'t)\ cval \Rightarrow ('c,'t)\ cval$
([-:=-]- [*65,65,65*] *65*)
where
$[X{:=}t]u \equiv$ *clock-set* (*SOME r. set r* $= X$) $t\ u$

term *region-set′*

abbreviation *region-set-set* :: $'c\ set \Rightarrow {'t}::time \Rightarrow ('c,'t)\ zone \Rightarrow ('c,'t)\ zone$
([-::=-]- [*65,65,65*] *65*)
where
$[X{::=}t]R \equiv$ *region-set′* R (*SOME r. set r* $= X$) t

no-notation *zone-set* ($\text{-}_{\text{-}} \rightarrow {}_{0}$ [*71*] *71*)

abbreviation *zone-set-set* :: $('c,\ {'t}::time)\ zone \Rightarrow {'c}\ set \Rightarrow ('c,\ 't)\ zone$
($\text{-}_{\text{-}} \rightarrow {}_{0}$ [*71*] *71*)
where
$Z_{X \rightarrow 0} \equiv$ *zone-set* Z (*SOME r. set r* $= X$)

abbreviation (*input*) *ccval* ($\{\!|\text{-}|\!\}$ [*100*]) **where** *ccval cc* $\equiv \{v.\ v \vdash cc\}$

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

3.3 Syntactic Definition

definition $L =$ *locations* A

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

definition $S \equiv \{(l,\ u)\ .\ l \in L \wedge (\forall\ x \in \mathcal{X}.\ u\ x \geq 0) \wedge u \vdash$ *inv-of* $A\ l\}$

inductive-set
K :: $('s * ('c,\ 't)\ cval) \Rightarrow ('s * ('c,\ 't)\ cval)$ *pmf set* **for** *st* :: $('s * ('c,\ 't)\ cval)$
where
— Passage of time *delay*:
$st \in S \Longrightarrow st = (l,\ u) \Longrightarrow t \geq 0 \Longrightarrow u \oplus t \vdash$ *inv-of* $A\ l \Longrightarrow$ *return-pmf* $(l,\ u \oplus t) \in K\ st\ |$
— Discrete transitions *action*:
$st \in S \Longrightarrow st = (l,\ u) \Longrightarrow (l,\ g,\ \mu) \in$ *trans-of* $A \Longrightarrow u \vdash g$
$\Longrightarrow$ *map-pmf* $(\lambda\ (X,\ l).\ (l,\ ([X := 0]u)))\ \mu \in K\ st\ |$
— Self loops – Note that this does not assume $st \in S$ *loop*:
return-pmf $st \in K\ st$

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* $\mathcal{X}$
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*: $\exists\ l \in L.\ \exists\ u \in V.\ u \vdash$ *inv-of* $A\ l$
assumes *valid*: *valid-abstraction* $A\ \mathcal{X}\ k$
begin

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

4.1 Syntactic Definition

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

lemma *S-alt-def*: $S = \{(l,\ u)\ .\ l \in L \wedge u \in V \wedge u \vdash \textit{inv-of}\ A\ l\}$ ⟨*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) *cval set*) ⇒ ($'s$ * ($'c$, t) *cval set*) *pmf set* **for** *st* :: ($'s$ * ($'c$, t) *cval set*)
where

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

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

4.2 Many Closure Properties

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

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

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

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

lemma *L-trans*[*intro*]:
assumes $(l,\ g,\ \mu) \in \textit{trans-of}\ A\ (X,\ l') \in \mu$
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 $(\mathit{SOME}\ u.\ u \in R) \in R$
⟨*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 $(l,\ g,\ \mu) \in \mathit{trans\text{-}of}\ A$ $(X,\ l') \in \mu$ $R \subseteq \{\!|g|\!\}$
shows $\mathit{region\text{-}set'}\ R\ (\mathit{SOME}\ r.\ \mathit{set}\ r = X)\ 0 \subseteq \{\!|\mathit{inv\text{-}of}\ A\ l'|\!\}$
⟨*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*:
  $\mathcal{S} \neq \{\}$
$\langle proof \rangle$

lemma $\mathcal{K}$*-*$\mathcal{S}$*-closed*:
  assumes $s \in \mathcal{S}$
  shows $(\bigcup D{\in}\mathcal{K}\ s.\ \textit{set-pmf}\ D) \subseteq \mathcal{S}$
$\langle proof \rangle$

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 $(l, g, \mu) \in \textit{trans-of}\ A$
  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
    $[[X{:=}0]u]_{\mathcal{R}} = \textit{region-set}'\ R\ (\textit{SOME}\ r.\ \textit{set}\ r = X)\ 0\ [[X{:=}0]u]_{\mathcal{R}} \in \mathcal{R}\ [X{:=}0]u \in [[X{:=}0]u]_{\mathcal{R}}$
$\langle proof \rangle$

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

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

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

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

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

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

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

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 $f \equiv \lambda\ (l,\ R).\ (l,\ SOME\ u.\ u \in R)$
  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 $f \equiv \lambda\ (l,\ R).\ (l,\ SOME\ u.\ u \in R)$
  shows $\mathit{pmf}\ (\mathit{map\text{-}pmf}\ f\ \mu)\ (f\ (l',\ R')) = \mathit{pmf}\ \mu\ (l',\ R')$
$\langle proof \rangle$

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

5.2 Translating Configurations

5.2.1 States

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

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

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 *abss s* $\in \mathcal{S}$
⟨*proof*⟩

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

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

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

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

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

5.2.2 Intermezzo

abbreviation *timed-bisim* (**infixr** $\sim$ *60*) **where**
  $s \sim s' \equiv abss\ s = abss\ s'$

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

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

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

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

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

⟨*proof*⟩

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

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

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

lemma *delayedR-correct*:
fixes $I\ r$
defines $R' \equiv region\ \mathcal{X}\ I\ r$
assumes $u \in R\ R \in \mathcal{R}\ valid\text{-}region\ \mathcal{X}\ k\ I\ r\ \forall\ c \in \mathcal{X}.\ \neg\ isConst\ (I\ c)\ R' \in Succ\ \mathcal{R}\ R$
shows
$delayedR\ R'\ u \in R'$
$\exists\ t \geq 0.\ delayedR\ R'\ u = u \oplus t$
$\wedge\ t \geq (1 - Max\ (\{frac\ (u\ c)\ |\ c.\ c \in \mathcal{X} \wedge isIntv\ (I\ c)\} \cup \{0\}))\ /\ 2$
$\langle proof \rangle$

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

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

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

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 $u \oplus t \in R$
$\langle proof \rangle$

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

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

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 $abst\ (rept\ (reps\ s)\ \mu) = \mu$
$\langle proof \rangle$

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

lemma *theI'*:
assumes $P\ a$

and $\bigwedge x.\ P\ x \Longrightarrow x = a$
shows $P\ (\mathit{THE}\ x.\ P\ x) \wedge (\forall\ y.\ P\ y \longrightarrow y = (\mathit{THE}\ x.\ P\ x))$
$\langle proof \rangle$

lemma *cont-cfg-defined*:
fixes *cfg s*
assumes $cfg \in valid\text{-}cfg\ s \in abst\ (action\ cfg)$
defines $x \equiv \mathit{THE}\ x.\ abss\ x = s \wedge x \in action\ cfg$
shows $(abss\ x = s \wedge x \in action\ cfg) \wedge (\forall\ y.\ abss\ y = s \wedge y \in action\ cfg \longrightarrow y = x)$
$\langle proof \rangle$

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

5.2.5 Configuration

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

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*]:
$state\ cfg \in \mathcal{S} \Longrightarrow state\ (absc\ (repc\ cfg)) = state\ cfg$
$\langle proof \rangle$

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

lemma *action-absc*:
$action\ (absc\ cfg) = abst\ (action\ cfg)$
$\langle proof \rangle$

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

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 $cont\ (repcs\ s\ cfg)\ s' = repcs\ s'\ (cont\ cfg\ (abss\ s'))$
⟨*proof*⟩

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 $cont\ (absc\ cfg)\ (abss\ s') = absc\ (cont\ cfg\ s')$
⟨*proof*⟩

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

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 $abss\ (reps\ s) = s$
⟨*proof*⟩

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 $cfg \in valid\text{-}cfg$
⟨*proof*⟩

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

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

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

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

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

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*:
  $state\ (absc\ cfg) = abss\ (state\ cfg)$
$\langle proof \rangle$

lemma *state-repcs*[*simp*]:
  $state\ (repcs\ s\ cfg) = s$
$\langle proof \rangle$

lemma *repcs-bisim*:
  notes *R-G.cfg-onD-state*[*simp del*]
  assumes $cfg \in R\text{-}G.valid\text{-}cfg\ x \in S\ x \sim x'\ abss\ x = state\ cfg$
  shows $absc\ (repcs\ x\ cfg) = absc\ (repcs\ x'\ cfg)$
$\langle proof \rangle$

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 $cfg \in R\text{-}G.valid\text{-}cfg\ abss\ s = state\ cfg$
  shows $absc\ (repcs\ s\ cfg) = cfg$ $\langle proof \rangle$

lemma *absc-repc-id*:
  notes *R-G.cfg-onD-state*[*simp del*]
  assumes $cfg \in R\text{-}G.valid\text{-}cfg$
  shows $absc\ (repc\ cfg) = cfg$ $\langle proof \rangle$

lemma *K-cfg-map-absc*:
  $cfg \in valid\text{-}cfg \Longrightarrow K\text{-}cfg\ (absc\ cfg) = map\text{-}pmf\ absc\ (K\text{-}cfg\ cfg)$
$\langle proof \rangle$

lemma *smap-comp*:
  $(smap\ f\ o\ smap\ g) = smap\ (f\ o\ g)$
$\langle proof \rangle$

lemma *state-abscD*[*simp*]:
  assumes $cfg \in MDP.cfg\text{-}on\ s$
  shows $state\ (absc\ cfg) = abss\ s$
$\langle proof \rangle$

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

lemma *absc-valid*[*intro*]:
assumes $cfg \in valid\text{-}cfg$
shows $absc\ cfg \in R\text{-}G.valid\text{-}cfg$
⟨*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
$emeasure\ (R\text{-}G.T\ cfg')\ X = emeasure\ (MDP.T\ cfg)\ Y$
⟨*proof*⟩

lemma *path-measure-eq-repcs1-new*:
fixes *cfg s*
defines $cfg' \equiv repcs\ s\ cfg$
assumes *s*: $abss\ s = state\ cfg$
assumes *valid*: $cfg \in R\text{-}G.valid\text{-}cfg$
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
$emeasure\ (R\text{-}G.T\ cfg)\ X = emeasure\ (MDP.T\ cfg')\ Y$
⟨*proof*⟩

lemma *region-compatible-suntil1*:
assumes $(holds\ (\lambda x.\ \varphi\ (reps\ x))\ suntil\ holds\ (\lambda x.\ \psi\ (reps\ x)))\ (smap\ abss\ x)$
and $pred\text{-}stream\ (\lambda\ s.\ \varphi\ (reps\ (abss\ s)) \longrightarrow \varphi\ s)\ x$
and $pred\text{-}stream\ (\lambda\ s.\ \psi\ (reps\ (abss\ s)) \longrightarrow \psi\ s)\ x$
shows $(holds\ \varphi\ suntil\ holds\ \psi)\ x$ ⟨*proof*⟩

lemma *region-compatible-suntil2*:
assumes $(holds\ \varphi\ suntil\ holds\ \psi)\ x$
and $pred\text{-}stream\ (\lambda\ s.\ \varphi\ s \longrightarrow \varphi\ (reps\ (abss\ s)))\ x$
and $pred\text{-}stream\ (\lambda\ s.\ \psi\ s \longrightarrow \psi\ (reps\ (abss\ s)))\ x$
shows $(holds\ (\lambda x.\ \varphi\ (reps\ x))\ suntil\ holds\ (\lambda x.\ \psi\ (reps\ x)))\ (smap\ abss\ x)$ ⟨*proof*⟩

lemma *region-compatible-suntil*:
assumes $pred\text{-}stream\ (\lambda\ s.\ \varphi\ (reps\ (abss\ s)) \longleftrightarrow \varphi\ s)\ x$
and $pred\text{-}stream\ (\lambda\ s.\ \psi\ (reps\ (abss\ s)) \longleftrightarrow \psi\ s)\ x$
shows $(holds\ (\lambda x.\ \varphi\ (reps\ x))\ suntil\ holds\ (\lambda x.\ \psi\ (reps\ x)))\ (smap\ abss\ x)$
$\longleftrightarrow (holds\ \varphi\ suntil\ holds\ \psi)\ x$ ⟨*proof*⟩

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

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

lemma *path-measure-eq-repcs''-new*:
notes *in-space-UNIV*[*measurable*]
fixes $cfg\ \varphi\ \psi\ s$
defines $cfg' \equiv repcs\ s\ cfg$
defines $\varphi' \equiv absp\ \varphi$ **and** $\psi' \equiv absp\ \psi$
assumes *s*: $abss\ s = state\ cfg$

assumes *valid*: $cfg \in R\text{-}G.valid\text{-}cfg$
assumes *valid'*: $cfg' \in valid\text{-}cfg$
assumes *equiv-φ*: $\bigwedge x.\ pred\text{-}stream\ (\lambda\ s.\ s \in S)\ x$
$\Longrightarrow pred\text{-}stream\ (\lambda\ s.\ \varphi\ (reps\ (abss\ s)) \longleftrightarrow \varphi\ s)\ (state\ cfg'\ \#\#\ x)$
and *equiv-ψ*: $\bigwedge x.\ pred\text{-}stream\ (\lambda\ s.\ s \in S)\ x$
$\Longrightarrow pred\text{-}stream\ (\lambda\ s.\ \psi\ (reps\ (abss\ s)) \longleftrightarrow \psi\ s)\ (state\ cfg'\ \#\#\ x)$
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-region* $X\ k\ I\ r\ c \in X\ u\ c = d\ d \leq k\ c$
  shows $I\ c = $ *Const* d
⟨*proof*⟩

lemma *not-Greater-bounded*:
  assumes $I\ x \neq$ *Greater* $(k\ x)\ x \in X$ *valid-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-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-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-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-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-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-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-in-X* A
    for $k\ v\ n$ *not-in-X* **and** A :: $('c,\ t,\ 's)$ *pta* +
  fixes $\varphi\ \psi$ :: $('s * ('c,\ t)\ cval) \Rightarrow bool$ **fixes** s

assumes φ: $\bigwedge x\ y.\ x \in S \Longrightarrow x \sim y \Longrightarrow \varphi\ x \longleftrightarrow \varphi\ y$
assumes ψ: $\bigwedge x\ y.\ x \in S \Longrightarrow x \sim y \Longrightarrow \psi\ x \longleftrightarrow \psi\ y$
assumes $s[intro,\ simp]$: $s \in S$
begin

definition $\varphi' \equiv absp\ \varphi$
definition $\psi' \equiv absp\ \psi$
definition $s' \equiv abss\ s$

lemma $s\text{-}s'\text{-}cfg\text{-}on[intro]$:
assumes $cfg \in MDP.cfg\text{-}on\ s$
shows $absc\ cfg \in R\text{-}G.cfg\text{-}on\ s'$
$\langle proof \rangle$

lemma $s'\text{-}\mathcal{S}[simp,\ intro]$:
$s' \in \mathcal{S}$
$\langle proof \rangle$

lemma $s'\text{-}s\text{-}cfg\text{-}on[intro]$:
assumes $cfg \in R\text{-}G.cfg\text{-}on\ s'$
shows $repcs\ s\ cfg \in MDP.cfg\text{-}on\ s$
$\langle proof \rangle$

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 $pred\text{-}stream\ (\lambda s.\ s \in S)\ xs$
and $[intro]$: $x \in S$
shows $pred\text{-}stream\ (\lambda s.\ \varphi\ (reps\ (abss\ s)) = \varphi\ s)\ (x\ \#\#\ xs)$
$\langle proof \rangle$

lemma $\varphi\text{-}stream'$:
$pred\text{-}stream\ (\lambda s.\ \varphi\ (reps\ (abss\ s)) = \varphi\ s)\ (x\ \#\#\ xs)$ **if** $pred\text{-}stream\ (\lambda s.\ s \in S)\ xs\ x \in S$
$\langle proof \rangle$

lemma $\psi\text{-}stream'$:
$pred\text{-}stream\ (\lambda s.\ \psi\ (reps\ (abss\ s)) = \psi\ s)\ (x\ \#\#\ xs)$ **if** $pred\text{-}stream\ (\lambda s.\ s \in S)\ xs\ x \in S$
$\langle proof \rangle$

lemmas $\varphi\text{-}stream = compatible\text{-}stream[of\ \varphi,\ OF\ \varphi]$
lemmas $\psi\text{-}stream = compatible\text{-}stream[of\ \psi,\ OF\ \psi]$

7.2 Easier Result on All Configurations

lemma *suntil-reps*:
assumes
$\forall s \in sset\ (smap\ abss\ y).\ s \in \mathcal{S}$
$(holds\ \varphi'\ suntil\ holds\ \psi')\ (s'\ \#\#\ smap\ abss\ y)$
shows $(holds\ \varphi\ suntil\ holds\ \psi)\ (s\ \#\#\ y)$
$\langle proof \rangle$

lemma *suntil-abss*:
assumes
$\forall s \in sset\ y.\ s \in S$
$(holds\ \varphi\ suntil\ holds\ \psi)\ (s\ \#\#\ y)$
shows
$(holds\ \varphi'\ suntil\ holds\ \psi')\ (s'\ \#\#\ smap\ abss\ y)$
$\langle proof \rangle$

theorem *P-sup-sunitl-eq*:

notes [*measurable*] = *in-space-UNIV* **and** [*iff*] = *pred-stream-iff*
shows
$(MDP.P\text{-}sup\ s\ \ (\lambda x.\ (holds\ \varphi\ suntil\ holds\ \psi)\ \ \ (s\ \ \#\#\ x)))$
$= (R\text{-}G.P\text{-}sup\ s'\ (\lambda x.\ (holds\ \varphi'\ suntil\ holds\ \psi')\ (s'\ \#\#\ x)))$
⟨*proof*⟩

end

7.3 Divergent Adversaries

context *Probabilistic-Timed-Automaton*
begin

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

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

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

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

definition $div\text{-}cfg\ cfg \equiv AE\ \omega\ in\ MDP.MC.T\ cfg.\ divergent\ (smap\ (snd\ o\ state)\ \omega)$

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

definition $R\text{-}G\text{-}div\text{-}cfg\ cfg \equiv AE\ \omega\ in\ MDP.MC.T\ cfg.\ \mathcal{R}\text{-}div\ (smap\ (snd\ o\ state)\ \omega)$

end

context *Probabilistic-Timed-Automaton-Regions*
begin

definition $cfg\text{-}on\text{-}div\ st \equiv MDP.cfg\text{-}on\ st \cap \{cfg.\ div\text{-}cfg\ cfg\}$

definition $R\text{-}G\text{-}cfg\text{-}on\text{-}div\ st \equiv R\text{-}G.cfg\text{-}on\ st \cap \{cfg.\ R\text{-}G\text{-}div\text{-}cfg\ cfg\}$

lemma *measurable-$\mathcal{R}$-div*[*measurable*]: $Measurable.pred\ MDP.MC.S\ \mathcal{R}\text{-}div$
⟨*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 $\mathit{eq\text{-}elapsed}\ u\ u'\ c \in \mathcal{X}$
  shows $\mathit{elapsed}\ u\ u' \geq u'\ c - u\ c$
  ⟨proof⟩

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

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

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

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

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

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

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

abbreviation *stream-trans* ≡ *pairwise trans*

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

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

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

lemma *trans-eq-elapsed*:

assumes *trans u u′ u* $\in$ *V*
shows *eq-elapsed u u′*
⟨*proof*⟩

lemma *pairwise-trans-eq-elapsed*:
assumes *stream-trans xs pred-stream* (λ *u. u* $\in$ *V*) *xs*
shows *pairwise eq-elapsed xs*
⟨*proof*⟩

lemma *not-reset-dur*:
assumes $\forall k>i.\ k \leq j \longrightarrow \neg$ *zero c* ($[xs\ !!\ k]_{\mathcal{R}}$) $j \geq i$ $c \in \mathcal{X}$ *stream-trans xs*
$\forall$ *i. eq-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 $\forall j\geq i.\ \neg$ *zero c* ($[xs\ !!\ j]_{\mathcal{R}}$) $j \geq i$ $c \in \mathcal{X}$ *stream-trans xs*
$\forall$ *i. eq-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 $\forall j\geq i.\ \neg$ *zero c* ($[xs\ !!\ j]_{\mathcal{R}}$) $j \geq i$ $c \in \mathcal{X}$ *stream-trans xs*
$\forall$ *i. eq-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* $\neq$ *u′*) $\longrightarrow$ *u′* $=$ *u* $\oplus$ *0.5*) $\wedge$
(($\exists$ $c \in \mathcal{X}$. *u c* $=$ *0* $\wedge$ *u′ c* $>$ *0* $\wedge$ ($\forall c\in\mathcal{X}.\ \nexists d.\ d \leq k\ c \wedge u'\ c = real\ d$))
$\longrightarrow$ *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 $u \in V$ $\forall c\in\mathcal{X}.\ \nexists d.\ d \leq k\ c \wedge u\ c = real\ d$

shows $\forall\, c{\in}\mathcal{X}.\ \forall\, u \in [u]_{\mathcal{R}}.\ \nexists\, d.\ d \le k\ c \wedge u\ c = \mathit{real}\ d$
$\langle \mathit{proof} \rangle$

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

coinductive *enabled-repcs* **where**
$\mathit{enabled\text{-}repcs}\ (\mathit{shd}\ \mathit{xs})\ (\mathit{stl}\ \mathit{xs}) \Longrightarrow \mathit{shd}\ \mathit{xs} = \mathit{repcs}\ \mathit{st}'\ \mathit{cfg}' \Longrightarrow \mathit{st}' \in \mathit{rept}\ \mathit{st}\ (\mathit{action}\ \mathit{cfg})$
$\Longrightarrow \mathit{abss}\ \mathit{st}' = \mathit{state}\ \mathit{cfg}'$
$\Longrightarrow \mathit{cfg}' \in \mathit{R\text{-}G.valid\text{-}cfg}$
$\Longrightarrow \mathit{enabled\text{-}repcs}\ (\mathit{repcs}\ \mathit{st}\ \mathit{cfg})\ \mathit{xs}$

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

lemma *enabled-repcsI*:
assumes $\mathit{cfg} \in \mathit{R\text{-}G.valid\text{-}cfg}\ \mathit{abss}\ \mathit{st} = \mathit{state}\ \mathit{cfg}\ \mathit{MDP.MC.enabled}\ (\mathit{repcs}\ \mathit{st}\ \mathit{cfg})\ \mathit{xs}$
shows $\mathit{enabled\text{-}repcs}\ (\mathit{repcs}\ \mathit{st}\ \mathit{cfg})\ \mathit{xs}$ $\langle \mathit{proof} \rangle$

lemma *repcs-eq-rept*:
$\mathit{rept}\ \mathit{st}\ (\mathit{action}\ \mathit{cfg}) = \mathit{rept}\ \mathit{st}''\ (\mathit{action}\ \mathit{cfg}'')$ **if** $\mathit{repcs}\ \mathit{st}\ \mathit{cfg} = \mathit{repcs}\ \mathit{st}''\ \mathit{cfg}''$
$\langle \mathit{proof} \rangle$

lemma *enabled-stream-trans'*:
assumes $\mathit{cfg} \in \mathit{R\text{-}G.valid\text{-}cfg}\ \mathit{abss}\ \mathit{st} = \mathit{state}\ \mathit{cfg}\ \mathit{MDP.MC.enabled}\ (\mathit{repcs}\ \mathit{st}\ \mathit{cfg})\ \mathit{xs}$
shows $\mathit{pairwise}\ \mathit{trans}'\ (\mathit{smap}\ (\mathit{snd}\ o\ \mathit{state})\ \mathit{xs})$
$\langle \mathit{proof} \rangle$

lemma *divergent-$\mathcal{R}$-divergent*:
assumes *in-S*: $\mathit{pred\text{-}stream}\ (\lambda\ u.\ u \in V)\ \mathit{xs}$
and *div*: $\mathit{divergent}\ \mathit{xs}$
and *trans*: $\mathit{stream\text{-}trans}\ \mathit{xs}$
shows $\mathcal{R}\text{-}\mathit{div}\ (\mathit{smap}\ (\lambda\ u.\ [u]_{\mathcal{R}})\ \mathit{xs})$ (**is** $\mathcal{R}\text{-}\mathit{div}\ ?\omega$)
$\langle \mathit{proof} \rangle$

lemma (**in** $-$)
fixes $f :: \mathit{nat} \Rightarrow \mathit{real}$
assumes $\forall\ i.\ f\, i \ge 0\ \forall\ i.\ \exists\ j \ge i.\ f\, j > d\ d > 0$
shows $\exists\ n.\ (\sum\ i \le n.\ f\ i) > t$
$\langle \mathit{proof} \rangle$

lemma *dur-ev-exceedsI*:
assumes $\forall\ i.\ \exists\ j \ge i.\ \mathit{dur}\ \mathit{xs}\ j - \mathit{dur}\ \mathit{xs}\ i \ge d$ **and** $d > 0$
obtains i **where** $\mathit{dur}\ \mathit{xs}\ i > t$
$\langle \mathit{proof} \rangle$

lemma *not-reset-mono*:
assumes $\mathit{stream\text{-}trans}\ \mathit{xs}\ \mathit{shd}\ \mathit{xs}\ c1 \ge \mathit{shd}\ \mathit{xs}\ c2\ \mathit{stream\text{-}all}\ (\lambda\ u.\ u \in V)\ \mathit{xs}\ c2 \in \mathcal{X}$
shows $(\mathit{holds}\ (\lambda\ u.\ u\ c1 \ge u\ c2)\ \mathit{until}\ \mathit{holds}\ (\lambda\ u.\ u\ c1 = 0))\ \mathit{xs}$ $\langle \mathit{proof} \rangle$

lemma *$\mathcal{R}$-divergent-divergent-aux*:

fixes xs :: $('c,\ t)$ *cval stream*
assumes *stream-trans xs stream-all* $(\lambda\ u.\ u \in V)$ *xs*
$(xs\ !!\ i)\ c1 = 0\ \exists\ k > i.\ k \le j \wedge (xs\ !!\ k)\ c2 = 0$
$\forall\ k > i.\ k \le j \longrightarrow (xs\ !!\ k)\ c1 \ne 0$
$c1 \in \mathcal{X}\ c2 \in \mathcal{X}$
shows $(xs\ !!\ j)\ c1 \ge (xs\ !!\ j)\ c2$
⟨*proof*⟩

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

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

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* $i = 0$
⟨*proof*⟩

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

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

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

lemma $\mathcal{R}$*-divergent-divergent-aux2*:
fixes M :: $(nat \Rightarrow bool)$ *set*
assumes $\forall\ i.\ \forall\ P \in M.\ \exists\ j \ge i.\ P\ j\ M \ne \{\}$ *finite* M
shows $\forall i.\exists j{\ge}i.\exists k{>}j.\exists\ P \in M.\ P\ j \wedge P\ k \wedge (\forall\ m < k.\ j < m \longrightarrow \neg\ P\ m)$
$\wedge (\forall\ Q \in M.\ \exists\ m \le k.\ j < m \wedge Q\ m)$
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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

lemma *K-cfg-same-loc-iff*:
($\forall$ *cfg'*$\in$ *K-cfg cfg. fst* (*state cfg'*) = *fst* (*state cfg*))
$\longleftrightarrow$ ($\forall$ *cfg'*$\in$ *K-cfg* (*absc cfg*). *fst* (*state cfg'*) = *fst* (*state* (*absc cfg*)))
if *cfg* $\in$ *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* $\in$ *valid-cfg*
shows *alternating cfg* $\longleftrightarrow$ *alternating* (*absc cfg*)
⟨*proof*⟩

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

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

lemma *repcs-unbounded-AE-non-loop-end-strong*:
assumes *cfg* $\in$ *R-G.cfg-on* (*abss st*) *st* $\in$ *S*
and *alternating cfg*
shows *AE* ω *in MDP.MC.T* (*repcs st cfg*).
($\forall$ *u* :: (*'c* $\Rightarrow$ *real*). ($\forall$ *c* $\in$ $\mathcal{X}$. *u c* > *real* (*k c*)) $\longrightarrow$
$\neg$ (*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* $\in$ *R-G-cfg-on-div* (*abss st*) *st* $\in$ *S* **and** *alternating cfg*
shows *repcs st cfg* $\in$ *cfg-on-div st*
⟨*proof*⟩

lemma *repcs-unbounded-AE-non-loop-end*:
assumes *cfg* $\in$ *R-G.cfg-on* (*abss st*) *st* $\in$ *S*
shows *AE* ω *in MDP.MC.T* (*repcs st cfg*).
($\forall$ *s* :: (*'s* $\times$ (*'c* $\Rightarrow$ *real*)). ($\forall$ *c* $\in$ $\mathcal{X}$. *snd s c* > *k c*) $\longrightarrow$
$\neg$ (*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 $\in$ *R-G.valid-cfg* **if** *cfg* $\in$ *R-G-cfg-on-div s'*
⟨*proof*⟩

lemma *cfg-on-valid*:
cfg $\in$ *valid-cfg* **if** *cfg* $\in$ *cfg-on-div s*
$\langle proof \rangle$

abbreviation *path-measure P cfg* $\equiv$ *emeasure* (*MDP.T cfg*) $\{x{\in}space\ MDP.St.\ P\ x\}$
abbreviation *R-G-path-measure P cfg* $\equiv$ *emeasure* (*R-G.T cfg*) $\{x{\in}space\ R\text{-}G.St.\ P\ x\}$
abbreviation *progressive st* $\equiv$ *cfg-on-div st* $\cap$ $\{cfg.\ alternating\ cfg\}$
abbreviation *R-G-progressive st* $\equiv$ *R-G-cfg-on-div st* $\cap$ $\{cfg.\ alternating\ cfg\}$

Summary of our results on divergent configurations:

lemma *absc-valid-cfg-eq*:
absc ' *progressive s* = *R-G-progressive s'*
$\langle proof \rangle$

Main theorem:

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

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.