

Probabilistic Timed Automata

Simon Wimmer and Johannes Hölzl

December 14, 2021

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

1	Bisimulation on a Relation	3
2	Additional Facts on Regions	6
2.1	Justifying Timed Until vs <i>suntil</i>	10
3	Definition and Semantics	10
3.1	Syntactic Definition	10
3.1.1	Collecting Information About Clocks	11
3.2	Operational Semantics as an MDP	12
3.3	Syntactic Definition	12
4	Constructing the Corresponding Finite MDP on Regions	13
4.1	Syntactic Definition	13
4.2	Many Closure Properties	13
4.3	The Region Graph is a Finite MDP	15
5	Relating the MDPs	16
5.1	Translating From \mathcal{K} to \mathcal{K}	16
5.2	Translating Configurations	20
5.2.1	States	20
5.2.2	Intermezzo	22
5.2.3	Predicates	23
5.2.4	Distributions	23
5.2.5	Configuration	31
5.3	Equalities Between Measures of Trace Spaces	42
6	Classifying Regions for Divergence	46
6.1	Pairwise	46
6.2	Regions	47
6.3	Unbounded and Zero Regions	48

7	Reachability	49
7.1	Definitions	49
7.2	Easier Result on All Configurations	50
7.3	Divergent Adversaries	51
7.4	Main Result	81

```

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 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  $\Rightarrow$  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  $\in$  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 -

```

```

  interpret K: MC-syntax K by unfold-locales

```

```

  interpret L: MC-syntax L by unfold-locales

```

```

  have x  $\in$  S using  $\langle R x y \rangle$  by (auto simp: R-def)

```

```

  define g where g t = (SOME s. R s t) for t

```

```

  have measurable-g: g  $\in$  count-space UNIV  $\rightarrow_M$  count-space UNIV by auto

```

```

  have g: R i j  $\Longrightarrow$  R (g j) j for i j

```

```

    unfolding g-def by (rule someI)

```

```

  have K-subset: x  $\in$  S  $\Longrightarrow$  K x  $\subseteq$  S for x

```

```

    using KL[THEN rel-funD, of x f x, THEN rel-pmf-imp-rel-set] by (auto simp: rel-set-def R-def)

```

```

  have in-S: AE  $\omega$  in K.T x.  $\omega \in$  streams S

```

```

    using K.AE-T-enabled

```

```

proof eventually-elim

```

```

  case (elim  $\omega$ ) with  $\langle x \in S \rangle$  show ?case

```

```

    apply (coinduction arbitrary: x  $\omega$ )

```

```

    subgoal for x  $\omega$  using K-subset by (cases  $\omega$ ) (auto simp: K.enabled-Stream)

```

```

    done

```

```

qed

```

```

  have L-eq: L y = map-pmf f (K x) if xy: R x y for x y

```

```

proof -

```

```

  have rel-pmf ( $\lambda x y. x = y$ ) (map-pmf f (K x)) (L y)

```

```

    using KL[THEN rel-funD, OF xy] by (auto intro: pmf.rel-mono-strong simp: R-def pmf.rel-map)

```

```

  then show ?thesis unfolding pmf.rel-eq by simp

```

```

qed

```

```

  let ?D =  $\lambda x. \text{distr } (K.T x) K.S (\text{smap } f)$ 

```

```

  have prob-space-D: ?D x  $\in$  space (prob-algebra K.S) for x

```

```

    by (auto simp: space-prob-algebra K.T.prob-space-distr)

```

```

  have D-eq-D: ?D x = ?D x' if R x y R x' y for x x' y

```

```

proof (rule stream-space-eq-sstart)

```

```

  define A where A = K.acc “ {x, x'}

```

```

  have x-A: x  $\in$  A x'  $\in$  A by (auto simp: A-def)

```

```

  let ? $\Omega$  = f ‘ A

```

```

  show countable ? $\Omega$ 

```

```

    unfolding A-def by (intro countable-image K.countable-acc) auto

```

```

  show prob-space (?D x) prob-space (?D x') by (auto intro!: K.T.prob-space-distr)

```

```

  show sets (?D x) = sets L.S sets (?D x') = sets L.S by auto

```

```

  have AE-streams: AE x in ?D x''. x  $\in$  streams ? $\Omega$  if x''  $\in$  A for x''

```

```

apply (simp add: space-stream-space streams-sets AE-distr-iff)
using K.AE-T-reachable[of x'] unfolding alw-HLD-iff-streams
proof eventually-elim
  fix s assume s ∈ streams (K.acc “ {x'”)
  moreover have K.acc “ {x'”) ⊆ A
    using ⟨x' ∈ A⟩ by (auto simp: A-def Image-def intro: rtrancl-trans)
  ultimately show smap f s ∈ streams (f ‘ A)
    by (auto intro: smap-streams)
qed
with x-A show AE x in ?D x'. x ∈ streams ?Ω AE x in ?D x. x ∈ streams ?Ω
  by auto
from ⟨x ∈ A⟩ ⟨x' ∈ A⟩ that show ?D x (sstart (f ‘ A) xs) = ?D x' (sstart (f ‘ A) xs) for xs
proof (induction xs arbitrary: x x' y)
  case Nil
  moreover have ?D x (streams (f ‘ A)) = 1 if x ∈ A for x
    using AE-streams[of x] that
    by (intro prob-space.emeasure-eq-1-AE[OF K.T.prob-space-distr]) (auto simp: streams-sets)
  ultimately show ?case by simp
next
  case (Cons z zs x x' y)
  have rel-pmf (R OO R-1-1) (K x) (K x')
    using KL[THEN rel-funD, OF Cons(4)] KL[THEN rel-funD, OF Cons(5)]
    unfolding pmf.rel-compp pmf.rel-flip by auto
  then obtain p :: ('s × 's) pmf where p: ∧ a b. (a, b) ∈ p ⇒ (R OO R-1-1) a b and
    eq: map-pmf fst p = K x map-pmf snd p = K x'
    by (auto simp: pmf.in-rel)
  let ?S = stream-space (count-space UNIV)
  have *: (##) y -‘ smap f -‘ sstart (f ‘ A) (z # zs) = (if f y = z then smap f -‘ sstart (f ‘ A) zs else
  {} ) for y z zs
    by auto
  have **: ?D x (sstart (f ‘ A) (z # zs)) = (∫+ y'. (if f y' = z then ?D y' (sstart (f ‘ A) zs) else 0) ∂K x)
for x
  apply (simp add: emeasure-distr)
  apply (subst K.T-eq-bind)
  apply (subst emeasure-bind[where N=?S])
    apply simp
    apply (rule measurable-distr2[where M=?S])
    apply measurable
  apply (intro nn-integral-cong-AE AE-pmfI)
  apply (auto simp add: emeasure-distr)
  apply (simp-all add: * space-stream-space)
  done
  have fst-A: fst ab ∈ A if ab ∈ p for ab
  proof -
    have fst ab ∈ K x using ⟨ab ∈ p⟩ set-map-pmf [of fst p] by (auto simp: eq)
    with ⟨x ∈ A⟩ show fst ab ∈ A
    by (auto simp: A-def intro: rtrancl.rtrancl-into-rtrancl)
  qed
  have snd-A: snd ab ∈ A if ab ∈ p for ab
  proof -
    have snd ab ∈ K x' using ⟨ab ∈ p⟩ set-map-pmf [of snd p] by (auto simp: eq)
    with ⟨x' ∈ A⟩ show snd ab ∈ A
    by (auto simp: A-def intro: rtrancl.rtrancl-into-rtrancl)
  qed
  show ?case
    unfolding ** eq[symmetric] nn-integral-map-pmf
    apply (intro nn-integral-cong-AE AE-pmfI)
    subgoal for ab using p[of fst ab snd ab] by (auto simp: R-def intro!: Cons(1) fst-A snd-A)
    done
  qed
qed

```

```

have L-eq-D: L.T y = ?D x
  using ⟨R x y⟩
proof (coinduction arbitrary: x y rule: L.T-coinduct)
  case (cont x y)
  then have Kx-Ly: rel-pmf R (K x) (L y)
    by (rule KL[THEN rel-funD])
  then have *: y' ∈ L y ⇒ ∃ x' ∈ K x. R x' y' for y'
    by (auto dest!: rel-pmf-imp-rel-set simp: rel-set-def)
  have **: y' ∈ L y ⇒ R (g y') y' for y'
    using *[of y'] unfolding g-def by (auto intro: someI)

have D-SCons-eq-D-D: distr (K.T i) K.S (λx. z ## smap f x) = distr (?D i) K.S (λx. z ## x) for i z
  by (subst distr-distr) (auto simp: comp-def)
have D-eq-D-gi: ?D i = ?D (g (f i)) if i: i ∈ K x for i
proof -
  obtain j where j ∈ L y R i j f i = j
    using Kx-Ly i by (force dest!: rel-pmf-imp-rel-set simp: rel-set-def R-def)
  then show ?thesis
    by (auto intro!: D-eq-D[OF ⟨R i j⟩] g)
qed

have ***: ?D x = measure-pmf (L y) ≫ (λy. distr (?D (g y)) K.S ((##) y))
  apply (subst K.T-eq-bind)
  apply (subst distr-bind[of - - K.S])
  apply (rule measurable-distr2[of - - K.S])
  apply (simp-all add: Pi-iff)
  apply (simp add: distr-distr comp-def L-eq[OF cont] map-pmf-rep-eq)
  apply (subst bind-distr[where K=K.S])
  apply measurable []
  apply (rule measurable-distr2[of - - K.S])
  apply measurable []
  apply (rule measurable-compose[OF measurable-g])
  apply measurable []
  apply simp
  apply (rule bind-measure-pmf-cong[where N=K.S])
  apply (auto simp: space-subprob-algebra space-stream-space intro!: K.T.subprob-space-distr)
  unfolding D-SCons-eq-D-D D-eq-D-gi ..
show ?case
  by (intro exI[of - λt. distr (K.T (g t)) (stream-space (count-space UNIV)) (smap f)])
    (auto simp add: K.T.prob-space-distr *** dest: **)
qed (auto intro: K.T.prob-space-distr)

have stream-all2 R s t ⇔ (s ∈ streams S ∧ smap f s = t) for s t
proof safe
  show stream-all2 R s t ⇒ s ∈ streams S
    apply (coinduction arbitrary: s t)
    subgoal for s t by (cases s; cases t) (auto simp: R-def)
    done
  show stream-all2 R s t ⇒ smap f s = t
    apply (coinduction arbitrary: s t rule: stream.coinduct)
    subgoal for s t by (cases s; cases t) (auto simp: R-def)
    done
qed (auto intro!: stream.rel-refl-strong simp: stream.rel-map R-def streams-iff-sset)
then have ω ∈ streams S ⇒ ω ∈ A ⇔ smap f ω ∈ B for ω
  using AB by (auto simp: rel-set-strong-def)
with in-S have K.T x A = K.T x (smap f -' B ∩ space (K.T x))
  by (auto intro!: emeasure-eq-AE streams-sets)
also have ... = (distr (K.T x) K.S (smap f)) B
  by (intro emeasure-distr[symmetric]) auto
also have ... = (L.T y) B unfolding L-eq-D ..

```

finally show *?thesis* .

qed

no-notation *cval* ($\{\cdot\}$ [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* $R'. R' \neq R \wedge R' \in \text{Succ } \mathcal{R} R \wedge (\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t'))$)

lemma *region-continuous*:

assumes *valid-region* $X k I r$

defines $R: R \equiv \text{region } X I r$

assumes *between*: $0 \leq t1 \ t1 \leq t2$

assumes *elem*: $u \in R \ u \oplus t2 \in R$

shows $u \oplus t1 \in R$

unfolding R

proof

from $\langle 0 \leq t1 \rangle \langle u \in R \rangle$ show $\forall x \in X. 0 \leq (u \oplus t1) x$ by (*auto simp: R cval-add-def*)

have *intv-elem* $x (u \oplus t1) (I x)$ if $x \in X$ for x

proof -

from *elem* that have *intv-elem* $x u (I x)$ *intv-elem* $x (u \oplus t2) (I x)$ by (*auto simp: R*)

with *between* show *?thesis* by (*cases I x, auto simp: cval-add-def*)

qed

then show $\forall x \in X. \text{intv-elem } x (u \oplus t1) (I x)$ by *blast*

let $?X_0 = \{x \in X. \exists d. I x = \text{Intv } d\}$

show $?X_0 = ?X_0 ..$

from *elem* have $\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow \text{frac } (u x) \leq \text{frac } (u y)$ by (*auto simp: R*)

moreover

{ fix $x y c d$ assume $A: x \in X \ y \in X \ I x = \text{Intv } c \ I y = \text{Intv } d$

from A *elem between* have *:

$c < u x \ x < c + 1 \ c < u x + t1 \ u x + t1 < c + 1$

by (*fastforce simp: cval-add-def R*)+

moreover from $A(2,4)$ *elem between* have **:

$d < u y \ y < d + 1 \ d < u y + t1 \ u y + t1 < d + 1$

by (*fastforce simp: cval-add-def R*)+

ultimately have $u x = c + \text{frac } (u x) \ u y = d + \text{frac } (u y)$ using *nat-intv-frac-decomp* by *auto*

then have

$\text{frac } (u x + t1) = \text{frac } (u x) + t1 \ \text{frac } (u y + t1) = \text{frac } (u y) + t1$

using $*(3,4) \ ***(3,4)$ *nat-intv-frac-decomp* by *force*+

then have

$\text{frac } (u x) \leq \text{frac } (u y) \longleftrightarrow \text{frac } ((u \oplus t1) x) \leq \text{frac } ((u \oplus t1) y)$

by (*auto simp: cval-add-def*)

}

ultimately show

$\forall x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longleftrightarrow \text{frac}((u \oplus t1) x) \leq \text{frac}((u \oplus t1) y)$
by (*auto simp: cval-add-def*)
qed

lemma *upper-right-eq*:

assumes *finite X valid-region X k I r*
shows $(\forall x \in X. \text{isGreater}(I x)) \longleftrightarrow \text{is-upper-right}(\text{region } X I r)$
using *assms*
proof (*safe, goal-cases*)
case (*1 t u*)
then show *?case*
by $-$ (*standard, force simp: cval-add-def*)
next
case (*2 x*)

from *region-not-empty[OF assms]* **obtain** *u* **where** $u: u \in \text{region } X I r ..$
moreover have $(1 :: \text{real}) \geq 0$ **by** *auto*
ultimately have $(u \oplus 1) \in \text{region } X I r$ **using** *2* **by** *auto*
with $\langle x \in X \rangle u$ **have** *intv-elem x u (I x) intv-elem x (u \oplus 1) (I x)* **by** *auto*
then show *?case* **by** (*cases I x, auto simp: cval-add-def*)
qed

lemma *bounded-region*:

assumes *finite X valid-region X k I r*
defines $R: R \equiv \text{region } X I r$
assumes $\neg \text{is-upper-right } R u \in R$
shows $u \oplus 1 \notin R$
proof $-$
from *upper-right-eq[OF assms(1,2)] assms(4)* **obtain** *x* **where** $x \in X \neg \text{isGreater}(I x)$
by (*auto simp: R*)
with *assms* **have** *intv-elem x u (I x)* **by** *auto*
with $x(2)$ **have** $\neg \text{intv-elem } x (u \oplus 1) (I x)$ **by** (*cases I x, auto simp: cval-add-def*)
with $x(1)$ *assms* **show** *?thesis* **by** *auto*
qed

context *AlphaClosure*
begin

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

lemma *succ-ex*:

assumes $R \in \mathcal{R}$
shows $\text{succ } \mathcal{R} R \in \mathcal{R}$ (**is** *?G1*) **and** $\text{succ } \mathcal{R} R \in \text{Succ } \mathcal{R} R$ (**is** *?G2*)
and $\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in \text{succ } \mathcal{R} R \wedge 0 \leq t')$ (**is** *?G3*)
proof $-$
from $\langle R \in \mathcal{R} \rangle$ **obtain** $I r$ **where** $R: R = \text{region } X I r \text{ valid-region } X k I r$
unfolding *\mathcal{R}-def* **by** *auto*
from *region-not-empty[OF finite] R* **obtain** *u* **where** $u: u \in R$
by *blast*
let $?Z = \{x \in X . \exists c. I x = \text{Const } c\}$
let $?succ =$
 $\lambda R'. R' \neq R \wedge R' \in \text{Succ } \mathcal{R} R$
 $\wedge (\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t'))$
consider (*upper-right*) $\forall x \in X. \text{isGreater}(I x) \mid (\text{intv}) \exists x \in X. \exists d. I x = \text{Intv } d \wedge ?Z = \{\}$
 $\mid (\text{const}) ?Z \neq \{\}$
apply (*cases* $\forall x \in X. \text{isGreater}(I x)$)
apply *fast*
apply (*cases* $?Z = \{\}$)

apply *safe*
apply (*rename-tac* x)
apply (*case-tac* I x)
by *auto*
then have $?G1 \wedge ?G2 \wedge ?G3$
proof cases
 case *const*
 with *upper-right-eq*[*OF finite* $R(2)$] **have** \neg *is-upper-right* R **by** (*auto simp: R(1)*)
 from *closest-prestable-1*($1,2$)[*OF const finite* $R(2)$] *closest-valid-1*[*OF const finite* $R(2)$] $R(1)$
 obtain R' **where** R' :
 $\forall u \in R. \forall t > 0. \exists t' \leq t. (u \oplus t') \in R' \wedge t' \geq 0 \wedge R' \in \mathcal{R} \wedge \forall u \in R'. \forall t \geq 0. (u \oplus t) \notin R$
 unfolding \mathcal{R} -*def* **by** *auto*
 with *region-not-empty*[*OF finite*] **obtain** u' **where** $u' \in R'$ **unfolding** \mathcal{R} -*def* **by** *blast*
 with $R'(3)$ **have** *neg: R' \neq R* **by** (*fastforce simp: cval-add-def*)
 obtain $t :: \text{real}$ **where** $t > 0$ **by** (*auto intro: that[of 1]*)
 with $R'(1,2) \langle u \in R \rangle$ **obtain** t **where** $t \geq 0 \wedge u \oplus t \in R'$ **by** *auto*
 with $\langle R \in \mathcal{R} \rangle \langle R' \in \mathcal{R} \rangle \langle u \in R \rangle$ **have** $R' \in \text{Succ } \mathcal{R} \ R$ **by** (*intro SuccI3*)
 moreover have ($\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t')$)
 using $R'(1)$ **unfolding** *cval-add-def*
 apply *clarsimp*
 subgoal for u t
 by (*cases t = 0*) *auto*
 done
 ultimately have $*$: *?succ R'* **using** *neg* **by** *auto*
 have *succ* $\mathcal{R} \ R = R'$ **unfolding** *succ-def*
proof (*simp add: $\langle \neg$ is-upper-right R*, *intro the-equality, rule *, goal-cases*)
 case *prems: (1 R'')*
 from *prems* **obtain** $t' u'$ **where** R'' :
 $R'' \in \mathcal{R} \wedge R'' \neq R \wedge t' \geq 0 \wedge R'' = [u' \oplus t']_{\mathcal{R}} \wedge u' \in R$
 using $R'(1)$ **by** *fastforce*
 from *this(1)* **obtain** $I r$ **where** $R''2$:
 $R'' = \text{region } X \ I \ r \ \text{valid-region } X \ k \ I \ r$
 by (*auto simp: R-def*)
 from R'' **have** $u' \oplus t' \notin R$ **using** *assms region-unique-spec* **by** *blast*
 with $* \langle t' \geq 0 \rangle \langle u' \in R \rangle$ **obtain** t'' **where** $t'' : t'' \leq t' \wedge u' \oplus t'' \in R' \wedge t'' \geq 0$ **by** *auto*
 from *this(2)* *neg* **have** $u' \oplus t'' \notin R$ **using** $R'(2)$ *assms region-unique-spec* **by** *auto*
 with t'' *prems* $\langle u' \in R \rangle$ **obtain** t''' **where** t''' :
 $t''' \leq t'' \wedge u' \oplus t''' \in R'' \wedge t''' \geq 0$
 by *auto*
 with *region-continuous*[*OF R''2(2) - - t'''(2)[unfolded R''2(1)]*, *of t'' - t''' t' - t'''*]
 $t'' R''$ *regions-closed'-spec*[*OF $\langle R \in \mathcal{R} \rangle R''(5,3)$*]
 have $u' \oplus t'' \in R''$ **by** (*auto simp: R''2 cval-add-def*)
 with $t''(2)$ **show** *?case* **using** $R''(1) \ R''(2)$ *region-unique-spec* **by** *blast*
 qed
with $R' *$ **show** *?thesis* **by** *auto*
next
 case *intv*
 then have $*$: $\forall x \in X. \neg \text{Regions.isConst } (I \ x)$ **by** *auto*
 let $?X_0 = \{x \in X. \text{isIntv } (I \ x)\}$
 let $?M = \{x \in ?X_0. \forall y \in ?X_0. (x, y) \in r \longrightarrow (y, x) \in r\}$
 from *intv* **obtain** $x \ c$ **where** $x : x \in X \wedge \neg \text{isGreater } (I \ x)$ **and** $c : I \ x = \text{Intv } c$ **by** *auto*
 with $\langle x \in X \rangle$ **have** $?X_0 \neq \{\}$ **by** *auto*
 have $?X_0 = \{x \in X. \exists d. I \ x = \text{Intv } d\}$ **by** *auto*
 with $R(2)$ **have** r : *total-on* $?X_0$ *r trans r* **by** *auto*
 from *total-finite-trans-max*[*OF $\langle ?X_0 \neq \{\} \rangle$ - this] *finite*
 obtain x' **where** $x' : x' \in ?X_0 \wedge \forall y \in ?X_0. x' \neq y \longrightarrow (y, x') \in r$ **by** *fastforce*
 from *this(2)* **have** $\forall y \in ?X_0. (x', y) \in r \longrightarrow (y, x') \in r$ **by** *auto*
 with $x'(1)$ **have** $*$: $?M \neq \{\}$ **by** *fastforce*
 with *upper-right-eq*[*OF finite* $R(2)$] **have** \neg *is-upper-right* R **by** (*auto simp: R(1)*)
 from *closest-prestable-2*($1,2$)[*OF * finite* $R(2) **$] *closest-valid-2*[*OF * finite* $R(2) **$] $R(1)$
 obtain R' **where** R' :*

$(\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t')) R' \in \mathcal{R}$
 $\forall u \in R'. \forall t \geq 0. (u \oplus t) \notin R$
unfolding \mathcal{R} -def **by** *auto*
with *region-not-empty*[*OF finite*] **obtain** u' **where** $u' \in R'$ **unfolding** \mathcal{R} -def **by** *blast*
with $R'(3)$ **have** *neg*: $R' \neq R$ **by** (*fastforce simp: cval-add-def*)
from *bounded-region*[*OF finite* $R(2)$, *folded* $R(1)$, *OF* $\langle \neg \text{is-upper-right } R \rangle$ u] **have**
 $u \oplus (1 :: t) \notin R$ $(1 :: t) \geq 0$
by *auto*
with $R'(1)$ u **obtain** t' **where** $t' \leq (1 :: t)$ $(u \oplus t') \in R'$ $0 \leq t'$ **by** *fastforce*
with $\langle R \in \mathcal{R} \rangle \langle R' \in \mathcal{R} \rangle \langle u \in R \rangle$ **have** $R' \in \text{Succ } \mathcal{R} R$ **by** (*intro SuccI3*)
with $R'(1)$ *neg* **have** $*$: $?succ R'$ **by** *auto*
have $\text{succ } \mathcal{R} R = R'$ **unfolding** *succ-def*
proof (*simp add:* $\langle \neg \text{is-upper-right } R \rangle$, *intro the-equality*, *rule* $*$, *goal-cases*)
case *prems*: $(1 R'')$
from *prems* **obtain** $t' u'$ **where** R'' :
 $R'' \in \mathcal{R}$ $R'' \neq R$ $t' \geq 0$ $R'' = [u' \oplus t']_{\mathcal{R}}$ $u' \in R$
using $R'(1)$ **by** *fastforce*
from *this*(1) **obtain** $I r$ **where** $R''2$:
 $R'' = \text{region } X I r$ *valid-region* $X k I r$
by (*auto simp: R-def*)
from R'' **have** $u' \oplus t' \notin R$ **using** *assms region-unique-spec* **by** *blast*
with $* \langle t' \geq 0 \rangle \langle u' \in R \rangle$ **obtain** t'' **where** t'' : $t'' \leq t'$ $u' \oplus t'' \in R'$ $t'' \geq 0$ **by** *auto*
from *this*(2) *neg* **have** $u' \oplus t'' \notin R$ **using** $R'(2)$ *assms region-unique-spec* **by** *auto*
with t'' *prems* $\langle u' \in R \rangle$ **obtain** t''' **where** t''' :
 $t''' \leq t''$ $u' \oplus t''' \in R''$ $t''' \geq 0$
by *auto*
with *region-continuous*[*OF* $R''2(2)$ - - $t'''(2)$ [*unfolded* $R''2(1)$], *of* $t'' - t'''$ $t' - t''$]
 $t'' R''$ *regions-closed'-spec*[*OF* $\langle R \in \mathcal{R} \rangle R''(5,3)$]
have $u' \oplus t'' \in R''$ **by** (*auto simp: cval-add-def* $R''2$)
with $t''(2)$ **show** $?case$ **using** $R''(1)$ $R''(2)$ *region-unique-spec* **by** *blast*
qed
with $R' *$ **show** $?thesis$ **by** *auto*
next
case *upper-right*
with *upper-right-eq*[*OF finite* $R(2)$] **have** $\text{succ } \mathcal{R} R = R$ **by** (*auto simp: R succ-def*)
with $\langle R \in \mathcal{R} \rangle u$ **show** $?thesis$ **by** (*fastforce simp: cval-add-def intro: SuccI3*)
qed
then **show** $?G1 ?G2 ?G3$ **by** *auto*
qed

lemma *region-set'-closed*:

fixes $d :: \text{nat}$

assumes $R \in \mathcal{R}$ $d \geq 0$ $\forall x \in \text{set } r. d \leq k x$ $\text{set } r \subseteq X$

shows *region-set'* $R r d \in \mathcal{R}$

proof -

from *region-not-empty*[*OF finite*] *assms*(1) **obtain** u **where** $u \in R$ **using** \mathcal{R} -def **by** *blast*

from *region-set'-id*[*OF* - - *finite*, *of* - k , *folded* \mathcal{R} -def] *assms* *this* **show** $?thesis$ **by** *fastforce*

qed

lemma *clock-set-cong*[*simp*]:

assumes $\forall c \in \text{set } r. u c = d$

shows $[r \rightarrow d]u = u$

proof *standard*

fix c

from *assms* **show** $([r \rightarrow d]u) c = u c$ **by** (*cases* $c \in \text{set } r$; *auto*)

qed

lemma *region-reset-not-Succ*:

notes *regions-closed'-spec*[*intro*]

```

assumes  $R \in \mathcal{R}$  set  $r \subseteq X$ 
shows region-set'  $R$   $r$   $0 = R \vee$  region-set'  $R$   $r$   $0 \notin \text{Succ } \mathcal{R} R$  (is  $?R = R \vee -$ )
proof -
from assms finite obtain  $u$  where  $u \in R$  by (meson Succ.cases succ-ex(2))
with  $\langle R \in \mathcal{R} \rangle$  have  $u \in V$   $[u]_{\mathcal{R}} = R$  by (auto simp: region-unique-spec dest: region-V)
with region-set'-id[OF  $\langle R \in \mathcal{R} \rangle$  [unfolding  $\mathcal{R}$ -def]  $\langle u \in R \rangle$  finite] assms(2) have
   $?R = [[r \rightarrow 0]u]_{\mathcal{R}}$ 
by (force simp:  $\mathcal{R}$ -def)
show ?thesis
proof (cases  $\forall x \in \text{set } r. u x = 0$ )
  case True
    then have  $[r \rightarrow 0]u = u$  by simp
    with  $\langle ?R = - \rangle \langle - = R \rangle$  have  $?R = R$  by (force simp:  $\mathcal{R}$ -def)
    then show ?thesis ..
  next
    case False
    then obtain  $x$  where  $x \in \text{set } r$   $u x \neq 0$  by auto
    { assume  $?R \in \text{Succ } \mathcal{R} R$ 
      with  $\langle u \in R \rangle \langle R \in \mathcal{R} \rangle$  obtain  $t$  where
         $t \geq 0$   $[u \oplus t]_{\mathcal{R}} = ?R$   $?R \in \mathcal{R}$ 
        by (meson Succ.cases set-of-regions-spec)
        with  $\langle u \in R \rangle$  assms(1) have  $u \oplus t \in ?R$  by blast
        moreover from  $\langle ?R = - \rangle \langle u \in R \rangle$  have  $[r \rightarrow 0]u \in ?R$  by (fastforce simp: region-set'-def)
        moreover from  $x \langle t \geq 0 \rangle \langle u \in V \rangle$  assms have  $(u \oplus t) x > 0$  by (force simp: cval-add-def V-def)
        moreover from  $x$  have  $([r \rightarrow 0]u) x = 0$  by auto
        ultimately have False using  $\langle ?R \in \mathcal{R} \rangle x(1)$  by (fastforce simp: region-set'-def)
      }
    then show ?thesis by auto
  qed
qed
end

```

2.1 Justifying Timed Until vs *suntil*

lemma *guard-continuous*:

```

assumes  $u \vdash g$   $u \oplus t \vdash g$   $0 \leq (t'::'t::\text{time})$   $t' \leq t$ 
shows  $u \oplus t' \vdash g$ 
using assms
by (induction g;
  auto 4 3
  simp: cval-add-def order-le-less-subst2 order-subst2 add-increasing2
  intro: less-le-trans
)

```

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

$$('c, 't, 's) \text{ pta} = ('c, 't, 's) \text{ transition set} * ('c, 't, 's) \text{ invassn}$$
definition

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

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

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

$$\text{trans-of} :: ('c, 't, 's) \text{ pta} \Rightarrow ('c, 't, 's) \text{ transition set}$$
where

$$\text{trans-of} \equiv \text{fst}$$
definition

$$\text{inv-of} :: ('c, 'time, 's) \text{ pta} \Rightarrow ('c, 'time, 's) \text{ invassn}$$
where

$$\text{inv-of} \equiv \text{snd}$$

no-notation *transition* $(- \vdash - \longrightarrow_{\text{inv-of}} - [61,61,61,61,61,61] \ 61)$

abbreviation *transition* ::
$$('c, 'time, 's) \text{ pta} \Rightarrow 's \Rightarrow ('c, 'time) \text{ cconstraint} \Rightarrow ('c \text{ set} * 's) \text{ pmf} \Rightarrow 'c \text{ set} \Rightarrow 's \Rightarrow \text{bool}$$

$$(- \vdash - \longrightarrow_{\text{inv-of}} - [61,61,61,61,61,61] \ 61) \text{ where}$$

$$(A \vdash l \longrightarrow_{g,p,X} l') \equiv (l, g, p, X, l') \in \text{Edges } A$$
definition

$$\text{locations} :: ('c, 't, 's) \text{ pta} \Rightarrow 's \text{ set}$$
where

$$\text{locations } A \equiv (\text{fst} \text{ ' Edges } A) \cup ((\text{snd} \circ \text{snd} \circ \text{snd} \circ \text{snd}) \text{ ' Edges } A)$$

3.1.1 Collecting Information About Clocks

definition *collect-clkt* :: $('c, 't :: \text{time}, 's) \text{ transition set} \Rightarrow ('c * 't) \text{ set}$

where

$$\text{collect-clkt } S = \bigcup \{\text{collect-clock-pairs } (\text{fst } (\text{snd } t)) \mid t. t \in S\}$$

definition *collect-clki* :: $('c, 't :: \text{time}, 's) \text{ invassn} \Rightarrow ('c * 't) \text{ set}$

where

$$\text{collect-clki } I = \bigcup \{\text{collect-clock-pairs } (I \ x) \mid x. \text{True}\}$$

definition *clkp-set* :: $('c, 't :: \text{time}, 's) \text{ pta} \Rightarrow ('c * 't) \text{ set}$

where

$$\text{clkp-set } A = \text{collect-clki } (\text{inv-of } A) \cup \text{collect-clkt } (\text{trans-of } A)$$

definition *collect-clkvt* :: $('c, 't :: \text{time}, 's) \text{ pta} \Rightarrow 'c \text{ set}$

where

$$\text{collect-clkvt } A = \bigcup ((\text{fst} \circ \text{snd} \circ \text{snd} \circ \text{snd}) \text{ ' Edges } A)$$

abbreviation *clocks* **where** $\text{clocks } A \equiv \text{fst} \text{ ' clkp-set } A \cup \text{collect-clkvt } A$

definition *valid-abstraction***where**

$$\text{valid-abstraction } A \ X \ k \equiv$$

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

assumes *valid-abstraction* $A \ X \ k$

shows $(\forall (x, m) \in \text{clkp-set } A. m \leq k \ x \wedge x \in X \wedge m \in \mathbb{N}) \text{ collect-clkvt } A \subseteq X \text{ finite } X$

using *assms unfolding valid-abstraction-def* by *auto*

lemma *valid-abstractionI*[*intro*]:

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

using *assms unfolding valid-abstraction-def* by *auto*

3.2 Operational Semantics as an MDP

abbreviation (*input*) *clock-set-set* :: $'c \ \text{set} \Rightarrow 't::\text{time} \Rightarrow ('c,'t) \ \text{cval} \Rightarrow ('c,'t) \ \text{cval}$
($[-:=]-$)- [65,65,65] 65)

where

$[X:=t]u \equiv \text{clock-set } (\text{SOME } r. \text{set } r = X) \ t \ u$

term *region-set'*

abbreviation *region-set-set* :: $'c \ \text{set} \Rightarrow 't::\text{time} \Rightarrow ('c,'t) \ \text{zone} \Rightarrow ('c,'t) \ \text{zone}$
($[-::=]-$)- [65,65,65] 65)

where

$[X::=t]R \equiv \text{region-set}' \ R \ (\text{SOME } r. \text{set } r = X) \ t$

no-notation *zone-set* $(- \rightarrow_0$ [71] 71)

abbreviation *zone-set-set* :: $('c, 't::\text{time}) \ \text{zone} \Rightarrow 'c \ \text{set} \Rightarrow ('c, 't) \ \text{zone}$
 $(- \rightarrow_0$ [71] 71)

where

$Z_{X \rightarrow_0} \equiv \text{zone-set } Z \ (\text{SOME } r. \text{set } r = X)$

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

locale *Probabilistic-Timed-Automaton* =

fixes $A :: ('c, 't :: \text{time}, 's) \ \text{pta}$

assumes *admissible-targets*:

$(l, g, \mu) \in \text{trans-of } A \Longrightarrow (X, l') \in \mu \Longrightarrow \{\!|g|\!\}_{X \rightarrow_0} \subseteq \{\!|\text{inv-of } A \ l'\!\}$

$(l, g, \mu) \in \text{trans-of } A \Longrightarrow (X, l') \in \mu \Longrightarrow X \subseteq \text{clocks } A$

— Not necessarily what we want to have

begin

3.3 Syntactic Definition

definition $L = \text{locations } A$

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

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

inductive-set

$K :: ('s * ('c, 't) \ \text{cval}) \Rightarrow ('s * ('c, 't) \ \text{cval}) \ \text{pmf set}$ **for** $st :: ('s * ('c, 't) \ \text{cval})$

where

— Passage of time *delay*:

$st \in S \Longrightarrow st = (l, u) \Longrightarrow t \geq 0 \Longrightarrow u \oplus t \vdash \text{inv-of } A \ l \Longrightarrow \text{return-pmf } (l, u \oplus t) \in K \ st \mid$

— Discrete transitions *action*:

$st \in S \Longrightarrow st = (l, u) \Longrightarrow (l, g, \mu) \in \text{trans-of } A \Longrightarrow u \vdash g$
 $\Longrightarrow \text{map-pmf } (\lambda (X, l). (l, ([X := 0]u))) \ \mu \in K \ st \mid$

— Self loops – Note that this does not assume $st \in S$ *loop*:

$\text{return-pmf } st \in K \ st$

declare $K.\text{intros}$ [*intro*]

sublocale *MDP: Markov-Decision-Process* K **by** (*standard, auto*)

end

4 Constructing the Corresponding Finite MDP on Regions

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

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

4.1 Syntactic Definition

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

lemma *S-alt-def: $\mathcal{S} = \{(l, u) . l \in L \wedge u \in V \wedge u \vdash \text{inv-of } A \ l\}$ **unfolding** *V-def S-def* **by** *auto**

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

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

where

— Passage of time *delay*:

$st \in \mathcal{S} \Longrightarrow st = (l, R) \Longrightarrow R' \in \text{Succ } \mathcal{R} \ R \Longrightarrow R' \subseteq \{\text{inv-of } A \ l\} \Longrightarrow \text{return-pmf } (l, R') \in \mathcal{K} \ st \mid$

— Discrete transitions *action*:

$st \in \mathcal{S} \Longrightarrow st = (l, R) \Longrightarrow (l, g, \mu) \in \text{trans-of } A \Longrightarrow R \subseteq \{g\}$
 $\Longrightarrow \text{map-pmf } (\lambda (X, l). (l, \text{region-set}' R (\text{SOME } r. \text{set } r = X) \ 0)) \ \mu \in \mathcal{K} \ st \mid$

— Self loops – Note that this does not assume $st \in \mathcal{S}$ *loop*:

$\text{return-pmf } st \in \mathcal{K} \ st$

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

4.2 Many Closure Properties

lemma *transition-def*:

$(A \vdash l \longrightarrow^{g, \mu, X} l') = ((l, g, \mu) \in \text{trans-of } A \wedge (X, l') \in \mu)$
unfolding *Edges-def edges-def trans-of-def* **by** *auto*

lemma *transitionI[intro]*:

$A \vdash l \longrightarrow^{g, \mu, X} l'$ **if** $(l, g, \mu) \in \text{trans-of } A \ (X, l') \in \mu$
using that **unfolding** *transition-def* **..**

lemma *transitionD[dest]*:

$(l, g, \mu) \in \text{trans-of } A \ (X, l') \in \mu$ **if** $A \vdash l \longrightarrow^{g, \mu, X} l'$
using that **unfolding** *transition-def* **by** *auto*

lemma *bex-Edges*:

$(\exists x \in \text{Edges } A. P \ x) = (\exists l \ g \ \mu \ X \ l'. A \vdash l \longrightarrow^{g, \mu, X} l' \wedge P \ (l, g, \mu, X, l'))$
by *fastforce*

lemma *L-trans[intro]*:

assumes $(l, g, \mu) \in \text{trans-of } A \ (X, l') \in \mu$
shows $l \in L \ l' \in L$

using *assms* **unfolding** *L-def locations-def* **by** (*auto simp: image-iff beX-Edges transition-def*)

lemma *transition-X*:

$X \subseteq \mathcal{X}$ **if** $A \vdash l \longrightarrow^{g,\mu,X} l'$

using *that* **unfolding** *X-def collect-clkvt-def clkp-set-def* **by** *auto*

lemma *admissible-targets-alt*:

$A \vdash l \longrightarrow^{g,\mu,X} l' \implies \{g\}_X \rightarrow 0 \subseteq \{\text{inv-of } A \ l'\}$

$A \vdash l \longrightarrow^{g,\mu,X} l' \implies X \subseteq \text{clocks } A$

by (*intro admissible-targets; blast*)⁺

lemma *V-reset-closed*[*intro*]:

assumes $u \in V$

shows $[r \rightarrow (d::\text{nat})]u \in V$

using *assms* **unfolding** *V-def*

apply *safe*

subgoal for x

by (*cases* $x \in \text{set } r$; *auto*)

done

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 –

from *assms* *regions-partition*[*OF meta-eq-to-obj-eq*[*OF R-def*]] **have**

$\exists !R. R \in \mathcal{R} \wedge u \in R$

unfolding *V-def* **by** *auto*

then show $[u]_{\mathcal{R}} \in \mathcal{R} \ u \in [u]_{\mathcal{R}}$

using *alpha-interp.region-unique-spec* **by** *auto*

qed

lemma *rep-R-ex*[*intro*]:

assumes $R \in \mathcal{R}$

shows $(\text{SOME } u. u \in R) \in R$

proof –

from *assms* *region-not-empty*[*OF finite*(1)] **have** $\exists u. u \in R$ **unfolding** *R-def* **by** *auto*

then show *?thesis* ..

qed

lemma *V-nn-closed*[*intro*]:

$u \in V \implies t \geq 0 \implies u \oplus t \in V$

unfolding *V-def cval-add-def* **by** *auto*

lemma *K-S-closed*[*intro*]:

assumes $\mu \in K \ s \ s' \in \mu \ s \in S$

shows $s' \in S$

using *assms*

by (*cases* *rule: K.cases, auto simp: S-alt-def* *dest: admissible-targets*[*unfolded zone-set-def*])

lemma *S-V*[*intro*]:

$(l, u) \in S \implies u \in V$

unfolding *S-alt-def* **by** *auto*

lemma *L-V*[*intro*]:

$(l, u) \in S \implies l \in L$

unfolding *S-def* **by** *auto*

lemma *S-V*[*intro*]:

$(l, R) \in \mathcal{S} \implies R \in \mathcal{R}$
unfolding \mathcal{S} -def by auto

lemma *admissible-targets'*:

assumes $(l, g, \mu) \in \text{trans-of } A \ (X, l') \in \mu \ R \subseteq \{g\}$
shows $\text{region-set}' \ R \ (\text{SOME } r. \text{ set } r = X) \ 0 \subseteq \{\text{inv-of } A \ l'\}$
using *admissible-targets(1)[OF assms(1,2)] assms(3)* **unfolding** *region-set'-def zone-set-def* by auto

4.3 The Region Graph is a Finite MDP

lemma \mathcal{S} -finite:

finite \mathcal{S}
using *finite finite- \mathcal{R}* **unfolding** \mathcal{S} -def by auto

lemma \mathcal{K} -finite:

finite (\mathcal{K} st)
proof –
let $?B1 = \{(R', l, R). \text{ st} \in \mathcal{S} \wedge \text{ st} = (l, R) \wedge R' \in \text{Succ } \mathcal{R} \ R \wedge R' \subseteq \{\text{inv-of } A \ l'\}\}$
let $?S1 = (\lambda(R', l, R). \text{ return-pmf } (l, R')) \text{ ' } ?B1$
let $?S1 = \{\text{return-pmf } (l, R') \mid R' \text{ l } R. \text{ st} \in \mathcal{S} \wedge \text{ st} = (l, R) \wedge R' \in \text{Succ } \mathcal{R} \ R \wedge R' \subseteq \{\text{inv-of } A \ l'\}\}$
let $?S2 = \{\text{map-pmf } (\lambda(X, l). (l, \text{region-set}' \ R \ (\text{SOME } r. \text{ set } r = X) \ 0)) \ \mu$
 $\mid R \ \mu. \exists l \ g. \text{ st} \in \mathcal{S} \wedge \text{ st} = (l, R) \wedge (l, g, \mu) \in \text{trans-of } A \wedge R \subseteq \{g\}\}$
have $?B1 \subseteq \{(R', l, R). R' \in \mathcal{R} \wedge (l, R) \in \mathcal{S}\}$ **unfolding** \mathcal{S} -def by auto
with \mathcal{S} -finite *finite- \mathcal{R}* **have** *finite* $?B1$ **by** – (rule *finite-subset, auto*)
moreover **have** $?S1 = (\lambda(R', l, R). \text{ return-pmf } (l, R')) \text{ ' } ?B1$ **by** (*auto simp: image-def*)
ultimately **have** *: *finite* $?S1$ **by** auto
have $\{\mu. \exists l \ g. (l, g, \mu) \in \text{PTA.trans-of } A\} = ((\lambda(l, g, \mu). \mu) \text{ ' } \text{PTA.trans-of } A)$ **by** force
with *finite(3)* *finite- \mathcal{R}* **have** *finite* $\{(R, \mu). \exists l \ g. R \in \mathcal{R} \wedge (l, g, \mu) \in \text{trans-of } A\}$ **by** auto
moreover **have**
 $\{(R, \mu). \exists l \ g. \text{ st} \in \mathcal{S} \wedge \text{ st} = (l, R) \wedge (l, g, \mu) \in \text{trans-of } A \wedge R \subseteq \{g\}\} \subseteq \dots$
unfolding \mathcal{S} -def by *fastforce*
ultimately **have** **:
finite $\{(R, \mu). \exists l \ g. \text{ st} \in \mathcal{S} \wedge \text{ st} = (l, R) \wedge (l, g, \mu) \in \text{trans-of } A \wedge R \subseteq \{g\}\}$
unfolding \mathcal{S} -def by (*blast intro: finite-subset*)
then **have** *finite* $?S2$ **unfolding** \mathcal{S} -def by auto
have \mathcal{K} st = $?S1 \cup ?S2 \cup \{\text{return-pmf st}\}$ **by** (*safe, cases rule: \mathcal{K} .cases, auto*)
with * ** **show** *?thesis* by auto

qed

lemma \mathcal{R} -not-empty:

$\mathcal{R} \neq \{\}$
proof –
let $?r = \{\}$
let $?I = \lambda c. \text{ Const } 0$
let $?R = \text{region } \mathcal{X} \ ?I \ ?r$
have *valid-region* $\mathcal{X} \ k \ ?I \ ?r$
proof
show $\{\} = \{x \in \mathcal{X}. \exists d. \text{ Const } 0 = \text{Intv } d\}$ **by** auto
show *refl-on* $\{\} \ \{\}$ **and** *trans* $\{\} \ \{\}$ **and** *total-on* $\{\} \ \{\}$ **unfolding** *trans-def* by auto
show $\forall x \in \mathcal{X}. \text{Regions.valid-intv } (k \ x) \ (\text{Const } 0)$ **by** auto
qed
then **have** $?R \in \mathcal{R}$ **unfolding** \mathcal{R} -def by auto
then **show** $\mathcal{R} \neq \{\}$ by *blast*

qed

lemma \mathcal{S} -not-empty:

$\mathcal{S} \neq \{\}$
proof –
from *not-trivial* **obtain** $l \ u$ where $\text{st}: l \in L \ u \in V \ u \vdash \text{inv-of } A \ l$ **by** *blast*
then **obtain** R where $R: R \in \mathcal{R} \ u \in R$ **using** \mathcal{R} -V by auto

from *valid* **have**
 $\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l). \ m \leq \text{real } (k \ x) \wedge x \in \mathcal{X} \wedge m \in \mathbf{N}$
by (*fastforce simp: clkp-set-def collect-clki-def*)
from *ccompatible*[*OF this, folded \mathcal{R} -def*] *R st(3)* **have**
 $R \subseteq \{\text{inv-of } A \ l\}$
unfolding *ccompatible-def ccval-def* **by** *auto*
with *st(1) R(1)* **show** *?thesis unfolding \mathcal{S} -def* **by** *auto*
qed

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

assumes $s \in \mathcal{S}$
shows $(\bigcup D \in \mathcal{K} \ s. \ \text{set-pmf } D) \subseteq \mathcal{S}$
proof (*safe, cases rule: \mathcal{K} .cases, blast, goal-cases*)
case ($1 \ x \ a \ b \ l \ R$)
then show *?case unfolding \mathcal{S} -def* **by** (*auto intro: alpha-interp.succ-ex(1)*)
next
case ($3 \ a \ b \ x$)
with $\langle s \in \mathcal{S} \rangle$ **show** *?case* **by** *auto*
next
case *prems: (2 l' R' p l R g μ)*
then obtain X **where** $*$: $(X, l') \in \text{set-pmf } \mu \ R' = \text{region-set}' \ R \ (\text{SOME } r. \ \text{set } r = X) \ 0$ **by** *auto*

show *?case unfolding \mathcal{S} -def*

proof *safe*

from $*(1)$ **have** $(l, g, \mu, X, l') \in \text{edges } (l, g, \mu)$ **unfolding** *edges-def* **by** *auto*
with *prems(6)* **have** $(l, g, \mu, X, l') \in \text{Edges } A$ **unfolding** *Edges-def trans-of-def* **by** *auto*
then show $l' \in L$ **unfolding** *L-def locations-def* **by** *force*

show $u \vdash \text{inv-of } A \ l'$ **if** $u \in R'$ **for** u

using *admissible-targets'*[*OF prems(6) *(1) prems(7)*] $*(2)$ **that** **by** *auto*

from *admissible-targets(2)*[*OF prems(6) *(1)*] **have** $X \subseteq \mathcal{X}$ **unfolding** *\mathcal{X} -def* **by** *auto*

with *finite(1)* **have** *finite* X **by** (*blast intro: finite-subset*)

then obtain r **where** $\text{set } r = X$ **using** *finite-list* **by** *auto*

then have $\text{set } (\text{SOME } r. \ \text{set } r = X) = X$ **by** (*rule someI*)

with $\langle X \subseteq \mathcal{X} \rangle$ **have** $\text{set } (\text{SOME } r. \ \text{set } r = X) \subseteq \mathcal{X}$ **by** *auto*

with *alpha-interp.region-set'-closed*[*of R 0 SOME r. set r = X*] *prems(4,5) *(2)*

show $R' \in \mathcal{R}$ **unfolding** *\mathcal{S} -def \mathcal{X} -def* **by** *auto*

qed

qed

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

by (*standard, auto simp: \mathcal{S} -finite \mathcal{S} -not-empty \mathcal{K} -finite \mathcal{K} - \mathcal{S} -closed*)

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

5 Relating the MDPs

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

lemma *ccompatible-inv*:

shows *ccompatible* \mathcal{R} (*inv-of* $A \ l$)

proof –

from *valid* **have**

$\forall (x, m) \in \text{collect-clock-pairs } (\text{inv-of } A \ l). \ m \leq \text{real } (k \ x) \wedge x \in \mathcal{X} \wedge m \in \mathbf{N}$

unfolding *valid-abstraction-def clkp-set-def collect-clki-def* **by** *auto*

with *ccompatible*[*of - k \mathcal{X} , folded \mathcal{R} -def*] **show** *?thesis* **by** *auto*

qed

lemma *ccompatible-guard*:

assumes $(l, g, \mu) \in \text{trans-of } A$
shows *ccompatible* $\mathcal{R} \ g$
proof –
from *assms valid* **have**
 $\forall (x, m) \in \text{collect-clock-pairs} \ g. \ m \leq \text{real } (k \ x) \wedge x \in \mathcal{X} \wedge m \in \mathbb{N}$
unfolding *valid-abstraction-def clkp-set-def collect-clkt-def trans-of-def* **by** *fastforce*
with *assms ccompatible[of - k \mathcal{X} , folded \mathcal{R} -def]* **show** *?thesis* **by** *auto*
qed

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

lemma *region-set'-eq:*

fixes $X :: 'c \ \text{set}$
assumes $R \in \mathcal{R} \ u \in R$
and $A \vdash l \longrightarrow g, \mu, X \ l'$
shows
 $[[X:=0]u]_{\mathcal{R}} = \text{region-set}' \ R \ (\text{SOME } r. \ \text{set } r = X) \ 0 \ [[X:=0]u]_{\mathcal{R}} \in \mathcal{R} \ [X:=0]u \in [[X:=0]u]_{\mathcal{R}}$
proof –
let $?r = (\text{SOME } r. \ \text{set } r = X)$
from *admissible-targets-alt[OF assms(3)] \mathcal{X} -def finite* **have** *finite X*
by (*auto intro: finite-subset*)
then obtain r **where** $\text{set } r = X$ **using** *finite-list* **by** *blast*
then have $\text{set } ?r = X$ **by** (*intro someI*)
with *valid assms(3)* **have** $\text{set } ?r \subseteq \mathcal{X}$
by (*simp add: transition- \mathcal{X}*)
from *region-set'-id[of - \mathcal{X} k, folded \mathcal{R} -def, OF assms(1,2) finite(1) - - this]*
show
 $[[X:=0]u]_{\mathcal{R}} = \text{region-set}' \ R \ (\text{SOME } r. \ \text{set } r = X) \ 0 \ [[X:=0]u]_{\mathcal{R}} \in \mathcal{R} \ [X:=0]u \in [[X:=0]u]_{\mathcal{R}}$
by *force+*
qed

lemma *regions-part-ex-reset:*

assumes $u \in V$
shows $[r \rightarrow (d::\text{nat})]u \in [[r \rightarrow d]u]_{\mathcal{R}} \ [[r \rightarrow d]u]_{\mathcal{R}} \in \mathcal{R}$
using *assms* **by** *auto*

lemma *reset-sets-all-equiv:*

assumes $u \in V \ u' \in [[r \rightarrow (d::\text{nat})]u]_{\mathcal{R}} \ x \in \text{set } r \ \text{set } r \subseteq \mathcal{X} \ d \leq k \ x$
shows $u' \ x = d$
proof –
from *assms(1)* **have** $[r \rightarrow d]u \in [[r \rightarrow d]u]_{\mathcal{R}} \ [[r \rightarrow d]u]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*
then obtain $I \ \varrho$ **where** $I: [[r \rightarrow d]u]_{\mathcal{R}} = \text{region } \mathcal{X} \ I \ \varrho$ *valid-region \mathcal{X} k I ϱ*
by (*auto simp: \mathcal{R} -def*)
with $u(1)$ *assms(3-)* **have** *intv-elem x ([r → d]u) (I x) valid-intv (k x) (I x)* **by** *fastforce+*
moreover from *assms* **have** $([r \rightarrow d]u) \ x = d$ **by** *simp*
ultimately have $I \ x = \text{Const } d$ **using** *assms(5)* **by** (*cases I x*) *auto*
moreover from I *assms(2-)* **have** *intv-elem x u' (I x)* **by** *fastforce*
ultimately show $u' \ x = d$ **by** *auto*
qed

lemma *reset-eq:*

assumes $u \in V \ ([[r \rightarrow 0]u]_{\mathcal{R}}) = ([[r' \rightarrow 0]u]_{\mathcal{R}}) \ \text{set } r \subseteq \mathcal{X} \ \text{set } r' \subseteq \mathcal{X}$
shows $[r \rightarrow 0]u = [r' \rightarrow 0]u$ **using** *assms*
proof –
have $*$: $u' \ x = 0$ **if** $u' \in [[r \rightarrow 0]u]_{\mathcal{R}} \ x \in \text{set } r$ **for** $u' \ x$
using *reset-sets-all-equiv[of u u' r 0 x] that assms* **by** *auto*
have $u' \ x = 0$ **if** $u' \in [[r' \rightarrow 0]u]_{\mathcal{R}} \ x \in \text{set } r'$ **for** $u' \ x$
using *reset-sets-all-equiv[of u u' r' 0 x] that assms* **by** *auto*
from *regions-part-ex-reset[OF assms(1), of - 0] assms(2)* **have** $**$:
 $([r' \rightarrow 0]u) \in [[r \rightarrow 0]u]_{\mathcal{R}} \ ([r \rightarrow 0]u) \in [[r \rightarrow 0]u]_{\mathcal{R}} \ [[r \rightarrow 0]u]_{\mathcal{R}} \in \mathcal{R}$
by *auto*

```

have ( $([r \rightarrow 0]u) x = ([r' \rightarrow 0]u) x$ ) for  $x$ 
proof (cases  $x \in \text{set } r$ )
  case True
    then have  $([r \rightarrow 0]u) x = 0$  by simp
    moreover from  $** \text{True}$  have  $([r' \rightarrow 0]u) x = 0$  by auto
    ultimately show ?thesis ..
next
  case False
    then have id:  $([r \rightarrow 0]u) x = u x$  by simp
    show ?thesis
    proof (cases  $x \in \text{set } r'$ )
      case True
        then have reset:  $([r' \rightarrow 0]u) x = 0$  by simp
        show ?thesis
        proof (cases  $x \in \mathcal{X}$ )
          case True
            from  $**(\beta)$  obtain  $I \varrho$  where
               $(([r \rightarrow 0]u)_{\mathcal{R}}) = \text{Regions.region } \mathcal{X} \ I \ \varrho \ \text{Regions.valid-region } \mathcal{X} \ k \ I \ \varrho$ 
            by (auto simp: R-def)
            with  $** \langle x \in \mathcal{X} \rangle$  have  $***$ :
               $\text{intv-elem } x \ ([r' \rightarrow 0]u) \ (I \ x) \ \text{intv-elem } x \ ([r \rightarrow 0]u) \ (I \ x)$ 
            by auto
            with reset have  $I \ x = \text{Const } 0$  by (cases  $I \ x$ , auto)
            with  $***(2)$  have  $([r \rightarrow 0]u) x = 0$  by auto
            with reset show ?thesis by auto
          next
            case False
              with assms( $\beta-$ ) have  $x \notin \text{set } r \ x \notin \text{set } r'$  by auto
              then show ?thesis by simp
          qed
        next
          case False
            then have reset:  $([r' \rightarrow 0]u) x = u x$  by simp
            with id show ?thesis by simp
          qed
        then show ?thesis ..
    qed

```

lemma *admissible-targets-clocks*:

```

assumes  $(l, g, \mu) \in \text{trans-of } A \ (X, l') \in \mu$ 
shows  $X \subseteq \mathcal{X} \ \text{set} \ (\text{SOME } r. \ \text{set } r = X) \subseteq \mathcal{X}$ 
proof  $-$ 
  from admissible-targets(2)[OF assms] finite have
    finite  $X \ X \subseteq \mathcal{X}$ 
  by (auto intro: finite-subset simp: X-def)
  then obtain  $r$  where  $\text{set } r = X$  using finite-list by blast
  with  $\langle X \subseteq \mathcal{X} \rangle$  show  $X \subseteq \mathcal{X} \ \text{set} \ (\text{SOME } r. \ \text{set } r = X) \subseteq \mathcal{X}$ 
  by (metis (mono-tags, lifting) someI-ex) $+$ 
qed

```

lemma

```

 $\text{rel-pmf} \ (\lambda a \ b. \ f \ a = b) \ \mu \ (\text{map-pmf } f \ \mu)$ 
by (subst pmf.rel-map(2)) (rule rel-pmf-reflI, auto)

```

lemma *K-pmf-rel*:

```

defines  $f \equiv \lambda (l, u). \ (l, [u]_{\mathcal{R}})$ 
shows  $\text{rel-pmf} \ (\lambda (l, u) \ st. \ (l, [u]_{\mathcal{R}}) = st) \ \mu \ (\text{map-pmf } f \ \mu)$  unfolding f-def
by (subst pmf.rel-map(2)) (rule rel-pmf-reflI, auto)

```

lemma *K-pmf-rel*:

assumes $A: \mu \in \mathcal{K} (l, R)$
defines $f \equiv \lambda (l, u). (l, \text{SOME } u. u \in R)$
shows $\text{rel-pmf } (\lambda (l, u) \text{ st. } (l, \text{SOME } u. u \in R) = \text{st}) \mu (\text{map-pmf } f \mu)$ **unfolding** $f\text{-def}$
by $(\text{subst pmf.rel-map}(2)) (\text{rule rel-pmf-reflI, auto})$

lemma $K\text{-elem-abs-inj}$:

assumes $A: \mu \in \mathcal{K} (l, u)$
defines $f \equiv \lambda (l, u). (l, [u]_{\mathcal{R}})$
shows $\text{inj-on } f \mu$
proof –
have $(l1, u1) = (l2, u2)$
if $\text{id: } (l1, [u1]_{\mathcal{R}}) = (l2, [u2]_{\mathcal{R}})$ **and** $\text{elem: } (l1, u1) \in \mu (l2, u2) \in \mu$ **for** $l1\ l2\ u1\ u2$
proof –
from id **have** $[\text{simp}]: l2 = l1$ **by** auto
from A
show $?thesis$
proof $(\text{cases, safe, goal-cases})$
case $(\lambda - - \tau \mu')$
from $\langle \mu = \rightarrow \text{elem} \text{ obtain } X1\ X2 \text{ where}$
 $u1 = [(SOME\ r.\ \text{set } r = X1) \rightarrow 0]u (X1, l1) \in \mu'$
 $u2 = [(SOME\ r.\ \text{set } r = X2) \rightarrow 0]u (X2, l1) \in \mu'$
by auto
with $\langle - \in \text{trans-of } \rightarrow \text{admissible-targets-clocks} \text{ have}$
 $\text{set } (SOME\ r.\ \text{set } r = X1) \subseteq \mathcal{X}$ $\text{set } (SOME\ r.\ \text{set } r = X2) \subseteq \mathcal{X}$
by auto
with $\text{id } \langle u1 = \rightarrow \langle u2 = \rightarrow \text{reset-eq}[of\ u] \langle - \in S \rangle$ **show** $?case$ **by** $(\text{auto simp: } S\text{-def } V\text{-def})$
qed $(-, \text{insert elem, simp})+$
qed
then show $?thesis$ **unfolding** $f\text{-def inj-on-def}$ **by** auto
qed

lemma $K\text{-elem-repr-inj}$:

notes $\text{alpha-interp.valid-regions-distinct-spec}[\text{intro}]$
assumes $A: \mu \in \mathcal{K} (l, R)$
defines $f \equiv \lambda (l, R). (l, \text{SOME } u. u \in R)$
shows $\text{inj-on } f \mu$
proof –
have $(l1, R1) = (l2, R2)$
if $\text{id: } (l1, \text{SOME } u. u \in R1) = (l2, \text{SOME } u. u \in R2)$ **and** $\text{elem: } (l1, R1) \in \mu (l2, R2) \in \mu$
for $l1\ l2\ R1\ R2$
proof –
let $?r1 = \text{SOME } u. u \in R1$ **and** $?r2 = \text{SOME } u. u \in R2$
from id **have** $[\text{simp}]: l2 = l1\ ?r2 = ?r1$ **by** auto
{ fix $g\ \mu'\ x$
assume $(l, R) \in \mathcal{S} (l, g, \mu') \in \text{PTA.trans-of } A\ R \subseteq \{v. v \vdash g\}$
and $\mu = \text{map-pmf } (\lambda(X, l). (l, \text{region-set}'\ R (SOME\ r.\ \text{set } r = X)\ 0)) \mu'$
from $\langle \mu = \rightarrow \text{elem} \text{ obtain } X1\ X2 \text{ where}$
 $R1 = \text{region-set}'\ R (SOME\ r.\ \text{set } r = X1)\ 0 (X1, l1) \in \mu'$
 $R2 = \text{region-set}'\ R (SOME\ r.\ \text{set } r = X2)\ 0 (X2, l1) \in \mu'$
by auto
with $\langle - \in \text{trans-of } \rightarrow \text{admissible-targets-clocks} \text{ have}$
 $\text{set } (SOME\ r.\ \text{set } r = X1) \subseteq \mathcal{X}$ $\text{set } (SOME\ r.\ \text{set } r = X2) \subseteq \mathcal{X}$
by auto
with $\text{alpha-interp.region-set}'\text{-closed}[of\ -\ 0] \langle R1 = \rightarrow \langle R2 = \rightarrow \langle - \in \mathcal{S} \rangle$ **have**
 $R1 \in \mathcal{R}\ R2 \in \mathcal{R}$
unfolding $\mathcal{S}\text{-def}$ **by** auto
with $\text{region-not-empty}[OF\ \text{finite}(1)]$ **have**
 $R1 \neq \{\}\ R2 \neq \{\} \exists u. u \in R1 \exists u. u \in R2$
by $(\text{auto simp: } \mathcal{R}\text{-def})$
from $\text{someI-ex}[OF\ \text{this}(3)]\ \text{someI-ex}[OF\ \text{this}(4)]$ **have** $?r1 \in R1\ ?r1 \in R2$ **by** $\text{simp}+$
with $\langle R1 \in \mathcal{R} \rangle \langle R2 \in \mathcal{R} \rangle$ **have** $R1 = R2$ **..**

```

}
from A elem this show ?thesis by (cases, auto)
qed
then show ?thesis unfolding f-def inj-on-def by auto
qed

```

lemma *K-elem-pmf-map-abs*:
assumes $A: \mu \in \mathcal{K} (l, u) (l', u') \in \mu$
defines $f \equiv \lambda (l, u). (l, [u]_{\mathcal{R}})$
shows $\text{pmf } (\text{map-pmf } f \ \mu) (f (l', u')) = \text{pmf } \mu (l', u')$
using A **unfolding** *f-def* **by** (*blast intro: pmf-map-inj K-elem-abs-inj*)

lemma *K-elem-pmf-map-repr*:
assumes $A: \mu \in \mathcal{K} (l, R) (l', R') \in \mu$
defines $f \equiv \lambda (l, R). (l, \text{SOME } u. u \in R)$
shows $\text{pmf } (\text{map-pmf } f \ \mu) (f (l', R')) = \text{pmf } \mu (l', R')$
using A **unfolding** *f-def* **by** (*blast intro: pmf-map-inj K-elem-repr-inj*)

definition *transp* :: $(s * (c, t) \text{ cval} \Rightarrow \text{bool}) \Rightarrow s * (c, t) \text{ cval set} \Rightarrow \text{bool}$ **where**
transp $\varphi \equiv \lambda (l, R). \forall u \in R. \varphi (l, u)$

5.2 Translating Configurations

5.2.1 States

definition
abss :: $s * (c, t) \text{ cval} \Rightarrow s * (c, t) \text{ cval set}$
where
abss $\equiv \lambda (l, u). \text{if } u \in V \text{ then } (l, [u]_{\mathcal{R}}) \text{ else } (l, -V)$

definition
reps :: $s * (c, t) \text{ cval set} \Rightarrow s * (c, t) \text{ cval}$
where
reps $\equiv \lambda (l, R). \text{if } R \in \mathcal{R} \text{ then } (l, \text{SOME } u. u \in R) \text{ else } (l, \lambda-. -1)$

lemma *S-reps-S[intro]*:
assumes $s \in \mathcal{S}$
shows $\text{reps } s \in \mathcal{S}$
using *assms R-V* **unfolding** *S-def S-def reps-def V-def* **by force**

lemma *S-abss-S[intro]*:
assumes $s \in \mathcal{S}$
shows $\text{abss } s \in \mathcal{S}$
using *assms ccompatible-inv* **unfolding** *S-def S-alt-def abss-def ccompatible-def* **by force**

lemma *S-abss-reps[simp]*:
 $s \in \mathcal{S} \Longrightarrow \text{abss } (\text{reps } s) = s$
using *R-V alpha-interp.region-unique-spec* **by** (*auto simp: S-def S-def reps-def abss-def; blast*)

lemma *map-pmf-abs-reps*:
assumes $s \in \mathcal{S} \ \mu \in \mathcal{K} \ s$
shows $\text{map-pmf } \text{abss } (\text{map-pmf } \text{reps } \mu) = \mu$
proof –
have $\text{map-pmf } \text{abss } (\text{map-pmf } \text{reps } \mu) = \text{map-pmf } (\text{abss } \circ \text{reps}) \ \mu$ **by** (*simp add: pmf.map-comp*)
also have $\dots = \mu$
proof (*rule map-pmf-idI, safe, goal-cases*)
case *prems*: $(1 \ l' \ R')$
with *assms* **have** $(l', R') \in \mathcal{S} \ \text{reps } (l', R') \in \mathcal{S}$ **by** *auto*
then show *?case* **by** *auto*

qed
 finally show *?thesis* by auto
 qed

lemma *abss-reps-id*:
 notes *R-G.cfg-onD-state[simp del]*
 assumes $s' \in \mathcal{S}$ $s \in \text{set-pmf (action cfg) cfg} \in R\text{-G.cfg-on } s'$
 shows $\text{abss (reps } s) = s$
 proof –
 from *assms* have $s \in \mathcal{S}$ by auto
 then show *?thesis* by auto
 qed

lemma *abss-S[intro]*:
 assumes $(l, u) \in S$
 shows $\text{abss } (l, u) = (l, [u]_{\mathcal{R}})$
 using *assms* unfolding *abss-def* by auto

lemma *reps-S[intro]*:
 assumes $(l, R) \in \mathcal{S}$
 shows $\text{reps } (l, R) = (l, \text{SOME } u. u \in R)$
 using *assms* unfolding *reps-def* by auto

lemma *fst-abss*:
 $\text{fst (abss } st) = \text{fst } st$ for st
 by (*cases st*) (*auto simp: abss-def*)

lemma *K-elem-abss-inj*:
 assumes $A: \mu \in K$ $(l, u) (l, u) \in S$
 shows *inj-on abss* μ
 proof –
 from *assms* have $\text{abss } s' = (\lambda (l, u). (l, [u]_{\mathcal{R}})) s'$ if $s' \in \mu$ for s'
 using *that* by (*auto split: prod.split*)
 from *inj-on-cong[OF this] K-elem-abs-inj[OF A(1)]* show *?thesis* by force
 qed

lemma *K-elem-reps-inj*:
 assumes $A: \mu \in \mathcal{K}$ $(l, R) (l, R) \in \mathcal{S}$
 shows *inj-on reps* μ
 proof –
 from *assms* have $\text{reps } s' = (\lambda (l, R). (l, \text{SOME } u. u \in R)) s'$ if $s' \in \mu$ for s'
 using *that* by (*auto split: prod.split*)
 from *inj-on-cong[OF this] K-elem-repr-inj[OF A(1)]* show *?thesis* by force
 qed

lemma *P-elem-pmf-map-abss*:
 assumes $A: \mu \in K$ $(l, u) (l, u) \in S$ $s' \in \mu$
 shows $\text{pmf (map-pmf abss } \mu) (\text{abss } s') = \text{pmf } \mu s'$
 using A by (*blast intro: pmf-map-inj K-elem-abss-inj*)

lemma *K-elem-pmf-map-reps*:
 assumes $A: \mu \in \mathcal{K}$ $(l, R) (l, R) \in \mathcal{S}$ $(l', R') \in \mu$
 shows $\text{pmf (map-pmf reps } \mu) (\text{reps } (l', R')) = \text{pmf } \mu (l', R')$
 using A by (*blast intro: pmf-map-inj K-elem-reps-inj*)

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

lemma *not-S-reps*:
 $(l, R) \notin \mathcal{S} \implies \text{reps } (l, R) \notin S$
 proof –
 assume $(l, R) \notin \mathcal{S}$
 let $?u = \text{SOME } u. u \in R$

have $\neg ?u \vdash \text{inv-of } A \text{ l}$ **if** $R \in \mathcal{R} \text{ l} \in L$
proof –

from *region-not-empty*[*OF finite(1)*] $\langle R \in \mathcal{R} \rangle$ **have** $\exists u. u \in R$ **by** (*auto simp: R-def*)
from *someI-ex*[*OF this*] **have** $?u \in R$.
moreover from $\langle (l, R) \notin \mathcal{S} \rangle$ **that have** $\neg R \subseteq \{\text{inv-of } A \text{ l}\}$ **by** (*auto simp: S-def*)
ultimately show *?thesis*
using *ccompatible-inv*[*of l*] $\langle R \in \mathcal{R} \rangle$ **unfolding** *ccompatible-def* **by** *fastforce*
qed
with *non-empty* $\langle (l, R) \notin \mathcal{S} \rangle$ **show** *?thesis* **unfolding** *S-def S-def reps-def* **by** *auto*
qed

lemma *neq-V-not-region*:
 $\neg V \notin \mathcal{R}$
using *R-V rep-R-ex* **by** *auto*

lemma *S-abss-S*:
 $\text{abss } s \in \mathcal{S} \implies s \in S$
unfolding *abss-def S-def S-def*
apply *safe*
subgoal for - - - u
by (*cases u \in V*) *auto*
subgoal for - - - u
using *neq-V-not-region* **by** (*cases u \in V, (auto simp: V-def; fail), auto*)
subgoal for $l' y \text{ l } u$
using *neq-V-not-region* **by** (*cases u \in V; auto dest: regions-part-ex*)
done

lemma *S-pred-stream-abss-S*:
 $\text{pred-stream } (\lambda s. s \in S) \text{ xs} \longleftrightarrow \text{pred-stream } (\lambda s. s \in \mathcal{S}) (\text{smap abss xs})$
using *S-abss-S S-abss-S* **by** (*auto simp: stream.pred-set*)

sublocale *MDP: Markov-Decision-Process-Invariant K S* **by** (*standard, auto*)

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

lemma *K-closed*:
 $s \in S \implies (\bigcup D \in K \text{ s. set-pmf } D) \subseteq S$
by *auto*

5.2.2 Intermezzo

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

lemma *bisim-loc-id*[*intro*]:
 $(l, u) \sim (l', u') \implies l = l'$
unfolding *abss-def* **by** (*cases u \in V; cases u' \in V; simp*)

lemma *bisim-val-id*[*intro*]:
 $[u]_{\mathcal{R}} = [u']_{\mathcal{R}}$ **if** $u \in V \text{ (} l, u) \sim (l', u')$
proof –
have $(l', - V) \neq (l, [u]_{\mathcal{R}})$
using *that* **by** *blast*
with *that* **have** $u' \in V$
by (*force simp: abss-def*)
with *that* **show** *?thesis*
by (*simp add: abss-def*)
qed

lemma *bisim-symmetric*:
 $(l, u) \sim (l', u') = (l', u') \sim (l, u)$
by (*rule eq-commute*)

lemma *bisim-val-id2[intro]*:
 $u' \in V \implies (l, u) \sim (l', u') \implies [u]_{\mathcal{R}} = [u']_{\mathcal{R}}$
apply (*subst (asm) eq-commute*)
apply (*subst eq-commute*)
apply (*rule bisim-val-id*)
by *auto*

lemma *K-bisim-unique*:
assumes $s \in S \ \mu \in K \ s \ x \in \mu \ x' \in \mu \ x \sim x'$
shows $x = x'$
using *assms(2,1,3-)*
proof (*cases rule: K.cases*)
case *prems: (action l u τ μ')*
with *assms* **obtain** $l1 \ l2 \ X1 \ X2$ **where** *A*:
 $(X1, l1) \in \text{set-pmf } \mu' \ (X2, l2) \in \text{set-pmf } \mu'$
 $x = (l1, [X1:=0]u) \ x' = (l2, [X2:=0]u)$
by *auto*
from $\langle x \sim x' \rangle \ A \ \langle s \in S \rangle \ \langle s = (l, u) \rangle$ **have** $[[X1:=0]u]_{\mathcal{R}} = [[X2:=0]u]_{\mathcal{R}}$
using *bisim-val-id[OF S-V] K-S-closed assms(2-4)* **by** (*auto intro!: bisim-val-id[OF S-V]*)
then **have** $[X1:=0]u = [X2:=0]u$
using *A admissible-targets-clocks(2)[OF prems(4)] prems(2,3)* **by** $-$ (*rule reset-eq, force*)
with $A \ \langle x \sim x' \rangle$ **show** *?thesis* **by** *auto*
next
case *delay*
with *assms(3-)* **show** *?thesis* **by** *auto*
next
case *loop*
with *assms(3-)* **show** *?thesis* **by** *auto*
qed

5.2.3 Predicates

definition *absp* **where**
 $\text{absp } \varphi \equiv \varphi \circ \text{reps}$

definition *repp* **where**
 $\text{repp } \varphi \equiv \varphi \circ \text{absp}$

5.2.4 Distributions

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

lemma *abss-SD*:
assumes $\text{abss } s \in \mathcal{S}$
obtains $l \ u$ **where** $s = (l, u) \ u \in [u]_{\mathcal{R}} \ [u]_{\mathcal{R}} \in \mathcal{R}$
proof $-$
obtain $l \ u$ **where** $s = (l, u)$ **by** *force*
moreover **from** $\mathcal{S}\text{-abss-}\mathcal{S}[\text{OF } \text{assms}]$ **have** $s \in \mathcal{S}$.
ultimately **have** $\text{abss } s = (l, [u]_{\mathcal{R}}) \ u \in V \ u \in [u]_{\mathcal{R}} \ [u]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*
with $\langle s = \cdot \rangle$ **show** *?thesis* **by** (*auto intro: that*)
qed

lemma *abss-SD'*:

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 –
from $abss\text{-}SD[OF\ assms(1)]$ **obtain** $l'\ u$ **where** u :
 $s = (l', u)\ u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}$
by *blast+*
with $\mathcal{R}\text{-}V$ **have** $u \in V$ **by** *auto*
with $\langle s = \cdot \rangle\ assms(2)$ **have** $l' = l\ R = [u]_{\mathcal{R}}$ **unfolding** $abss\text{-}def$ **by** *auto*
with u **show** *?thesis* **by** (*auto intro: that*)
qed

definition $infR\ R \equiv \lambda\ c.\ of\text{-}int\ [(SOME\ u.\ u \in R)\ c]$

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

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

lemma $delayedR\text{-}correct\text{-}aux\text{-}aux$:

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 –

have $f1: a + b < real\ (c + 1)$
using $assms(4)$ **by** *auto*
have $f2: \bigwedge r\ ra.\ (r::real) + (-\ r + ra) = ra$
by *linarith*
have $f3: \bigwedge r.\ (r::real) = -\ (-\ r)$
by *linarith*
have $f4: \bigwedge r\ ra.\ -\ (r::real) + (ra + r) = ra$
by *linarith*
then have $f5: \bigwedge r\ n.\ r + -\ frac\ r = real\ n \vee \neg\ r < real\ (n + 1) \vee \neg\ real\ n < r$
using $f2$ **by** (*metis\ nat\text{-}intv\text{-}frac\text{-}decomp*)
then have $frac\ a + real\ c = a$
using $f4\ f3$ **by** (*metis\ One\text{-}nat\text{-}def\ add.\textit{right}\text{-}neutral\ add\text{-}Suc\text{-}right\ assms(1)\ assms(2)*)
then show *?thesis*
using $f5\ f1\ assms(1)\ assms(3)$ **by** *fastforce*

qed

lemma $delayedR\text{-}correct\text{-}aux$:

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 $u \oplus t \in R$ **unfolding** $R\text{-}def$

proof
from *assms* **have** $R \in \mathcal{R}$ **unfolding** \mathcal{R} -*def* **by** *auto*
with $\langle u \in R \rangle$ \mathcal{R} -*V* **have** $u \in V$ **by** *auto*
with $\langle t \geq 0 \rangle$ **show** $\forall x \in \mathcal{X}. 0 \leq (u \oplus t) x$ **unfolding** *V-def* **by** (*auto simp: cval-add-def*)
have *intv-elem* $x (u \oplus t) (I x)$ **if** $x \in \mathcal{X}$ **for** x
proof (*cases I x*)
 case *Const*
 with *assms* $\langle x \in \mathcal{X} \rangle$ **show** *?thesis* **by** *auto*
next
 case (*Intv c*)
 with *assms* $\langle x \in \mathcal{X} \rangle$ **show** *?thesis* **by** (*simp add: cval-add-def*) (*rule; force*)
next
 case (*Greater c*)
 with *assms* $\langle x \in \mathcal{X} \rangle$ **show** *?thesis* **by** (*fastforce simp add: cval-add-def*)
qed
then **show** $\forall x \in \mathcal{X}. \text{intv-elem } x (u \oplus t) (I x) ..$

let $?X_0 = \{x \in \mathcal{X}. \exists d. I x = \text{Intv } d\}$
show $?X_0 = ?X_0$ **by** *auto*

have $\text{frac } (u x + t) = \text{frac } (u x) + t$ **if** $x \in ?X_0$ **for** x
proof –
 show *?thesis*
 apply (*rule delayedR-correct-aux-aux*[**where** $c = \text{intv-const } (I x)$])
 using *assms* $\langle x \in ?X_0 \rangle$ **by** (*force simp add: cval-add-def*)+
qed
then **have** $\text{frac } (u x) \leq \text{frac } (u y) \iff \text{frac } (u x + t) \leq \text{frac } (u y + t)$ **if** $x \in ?X_0$ $y \in ?X_0$ **for** $x y$
using *that* **by** *auto*
with *assms* **show**
 $\forall x \in ?X_0. \forall y \in ?X_0. ((x, y) \in r) = (\text{frac } ((u \oplus t) x) \leq \text{frac } ((u \oplus t) y))$
unfolding *cval-add-def* **by** *auto*
qed

lemma *delayedR-correct-aux'*:
fixes $I r$
defines $R \equiv \text{region } \mathcal{X} I r$
assumes $u \oplus t1 \in R$ *valid-region* $\mathcal{X} k I r \forall c \in \mathcal{X}. \neg \text{isConst } (I c)$
 $\forall c \in \mathcal{X}. \text{isIntv } (I c) \implies (u \oplus t2) c < \text{intv-const } (I c) + 1$
 $t1 \leq t2$
shows $u \oplus t2 \in R$
proof –
 have $(u \oplus t1) \oplus (t2 - t1) \in R$ **unfolding** *R-def*
 using *assms* **by** – (*rule delayedR-correct-aux, auto simp: cval-add-def*)
 then **show** $u \oplus t2 \in R$ **by** (*simp add: cval-add-def*)
qed

lemma *valid-regions-intv-distinct*:
valid-region $X k I r \implies \text{valid-region } X k I' r' \implies u \in \text{region } X I r \implies u \in \text{region } X I' r'$
 $\implies x \in X \implies I x = I' x$
proof *goal-cases*
 case *A: 1*
 note $x = \langle x \in X \rangle$
 with *A* **have** *valid-intv* $(k x) (I x)$ **by** *auto*
 moreover **from** *A(2)* x **have** *valid-intv* $(k x) (I' x)$ **by** *auto*
 moreover **from** *A(3)* x **have** *intv-elem* $x u (I x)$ **by** *auto*
 moreover **from** *A(4)* x **have** *intv-elem* $x u (I' x)$ **by** *auto*
 ultimately **show** $I x = I' x$ **using** *valid-intv-distinct* **by** *fastforce*
qed

lemma *delayedR-correct*:

fixes $I r$

defines $R' \equiv \text{region } \mathcal{X} I r$

assumes $u \in R R \in \mathcal{R}$ *valid-region* $\mathcal{X} k I r \forall c \in \mathcal{X}. \neg \text{isConst } (I c) R' \in \text{Succ } \mathcal{R} R$

shows

delayedR $R' u \in R'$

$\exists t \geq 0. \text{delayedR } R' u = u \oplus t$

$\wedge t \geq (1 - \text{Max } (\{\text{frac } (u c) \mid c. c \in \mathcal{X} \wedge \text{isIntv } (I c)\} \cup \{0\})) / 2$

proof –

let $?u = \text{SOME } u. u \in R$

let $?I = \text{SOME } I. \exists r. \text{valid-region } \mathcal{X} k I r \wedge R' = \text{region } \mathcal{X} I r$

let $?S = \{\text{frac } (u c) \mid c. c \in \mathcal{X} \wedge \text{isIntv } (I c)\}$

let $?m = 1 - \text{Max } (?S \cup \{0\})$

let $?t = \text{SOME } t. u \oplus t \in R' \wedge t \geq ?m / 2$

have $\text{Max } (?S \cup \{0\}) \geq 0 \ ?m \leq 1$ **using** *finite(1)* **by** *auto*

have $\text{Max } (?S \cup \{0\}) \in ?S \cup \{0\}$ **using** *finite(1)* **by** – (*rule Max-in; auto*)

with *frac-lt-1* **have** $\text{Max } (?S \cup \{0\}) \leq 1 \ ?m \geq 0$ **by** *auto*

from *assms(3, 6)* $\langle u \in R \rangle$ **obtain** t **where** t :

$u \oplus t \in R' t \geq 0$

by (*metis alpha-interp.regions-closed'-spec alpha-interp.set-of-regions-spec*)

have *I-cong*: $\forall c \in \mathcal{X}. I' c = I c$ **if** *valid-region* $\mathcal{X} k I' r' R' = \text{region } \mathcal{X} I' r'$ **for** $I' r'$

using *valid-regions-intv-distinct assms(4) t(1)* **that** **unfolding** *R'-def* **by** *auto*

have *I-cong*: $?I c = I c$ **if** $c \in \mathcal{X}$ **for** c

proof –

from *assms* **have**

$\exists r. \text{valid-region } \mathcal{X} k ?I r \wedge R' = \text{region } \mathcal{X} ?I r$

by – (*rule someI[where P = $\lambda I. \exists r. \text{valid-region } \mathcal{X} k I r \wedge R' = \text{region } \mathcal{X} I r$]; auto*)

with *I-cong* **that** **show** *thesis* **by** *auto*

qed

then **have** $?S = \{\text{frac } (u c) \mid c. c \in \mathcal{X} \wedge \text{isIntv } (?I c)\}$ **by** *auto*

have *upper-bound*: $(u \oplus ?m / 2) c < \text{intv-const } (I c) + 1$ **if** $c \in \mathcal{X}$ *isIntv* $(I c)$ **for** c

proof (*cases* $u c > \text{intv-const } (I c)$)

case *True*

from t **that** *assms* **have** $u c + t < \text{intv-const } (I c) + 1$ **unfolding** *cval-add-def* **by** *fastforce*

with $\langle t \geq 0 \rangle$ *True* **have** $*$: $\text{intv-const } (I c) < u c \ u c < \text{intv-const } (I c) + 1$ **by** *auto*

have $\text{frac } (u c) \leq \text{Max } (?S \cup \{0\})$ **using** *finite(1)* **that** **by** – (*rule Max-ge; auto*)

then **have** $?m \leq 1 - \text{frac } (u c)$ **by** *auto*

then **have** $?m / 2 < 1 - \text{frac } (u c)$ **using** $*$ *nat-intv-frac-decomp* **by** *fastforce*

then **have** $(u \oplus ?m / 2) c < u c + 1 - \text{frac } (u c)$ **unfolding** *cval-add-def* **by** *auto*

also **from** $*$ **have**

$\dots = \text{intv-const } (I c) + 1$

using *nat-intv-frac-decomp of-nat-1 of-nat-add* **by** *fastforce*

finally **show** *thesis* .

next

case *False*

then **have** $u c \leq \text{intv-const } (I c)$ **by** *auto*

moreover **from** $\langle 0 \leq ?m \rangle \langle ?m \leq 1 \rangle$ **have** $?m / 2 < 1$ **by** *auto*

ultimately **have** $u c + ?m / 2 < \text{intv-const } (I c) + 1$ **by** *linarith*

then **show** *thesis* **by** (*simp add: cval-add-def*)

qed

have $?t \geq 0 \wedge u \oplus ?t \in R' \wedge ?t \geq ?m / 2$

proof (*cases* $t \geq ?m / 2$)

case *True*

from $\langle t \geq ?m / 2 \rangle t \langle \text{Max } (?S \cup \{0\}) \leq 1 \rangle$ **have** $u \oplus ?t \in R' \wedge ?t \geq ?m / 2$

by – (*rule someI; auto*)

with $\langle ?m \geq 0 \rangle$ **show** *thesis* **by** *auto*

next

case *False*

have $u \oplus ?m / 2 \in R'$ **unfolding** *R'-def*

apply (*rule delayedR-correct-aux'*)

apply (rule $t[\text{unfolded } R'\text{-def}]$)
apply (rule assms)
using upper-bound False **by** auto
with $\langle ?m \geq 0 \rangle$ **show** $?thesis$ **by** – (rule someI2 ; fastforce)
qed
then show $\text{delayedR } R' u \in R' \exists t \geq 0. \text{delayedR } R' u = u \oplus t \wedge t \geq ?m / 2$
by (auto simp: delayedR-def $\langle ?S = \rightarrow \rangle$)
qed

definition

$\text{rept} :: 's * ('c, t) \text{cval} \Rightarrow ('s * ('c, t) \text{cval set}) \text{pmf} \Rightarrow ('s * ('c, t) \text{cval}) \text{pmf}$

where

$\text{rept } s \mu\text{-abs} \equiv \text{let } (l, u) = s \text{ in}$
 if $(\exists R'. (l, u) \in S \wedge \mu\text{-abs} = \text{return-pmf } (l, R') \wedge$
 $(([u]_{\mathcal{R}} = R' \wedge (\forall c \in \mathcal{X}. u \text{ c} > k \text{ c}))))$
 then $\text{return-pmf } (l, u \oplus 0.5)$
 else if
 $(\exists R'. (l, u) \in S \wedge \mu\text{-abs} = \text{return-pmf } (l, R') \wedge R' \in \text{Succ } \mathcal{R} ([u]_{\mathcal{R}}) \wedge [u]_{\mathcal{R}} \neq R'$
 $\wedge (\forall u \in R'. \forall c \in \mathcal{X}. \nexists d. d \leq k \text{ c} \wedge u \text{ c} = \text{real } d))$
 then $\text{return-pmf } (l, \text{delayedR } (\text{SOME } R'. \mu\text{-abs} = \text{return-pmf } (l, R')) u)$
 else $\text{SOME } \mu. \mu \in K \text{ s} \wedge \text{abst } \mu = \mu\text{-abs}$

lemma S-L:

$l \in L$ if $(l, R) \in \mathcal{S}$
using that **unfolding** $\mathcal{S}\text{-def}$ **by** auto

lemma S-inv:

$(l, R) \in \mathcal{S} \implies R \subseteq \{\text{inv-of } A \ l\}$
unfolding $\mathcal{S}\text{-def}$ **by** auto

lemma upper-right-closed:

assumes $\forall c \in \mathcal{X}. \text{real } (k \text{ c}) < u \text{ c} \ u \in R \ R \in \mathcal{R} \ t \geq 0$
shows $u \oplus t \in R$

proof –

from $\langle R \in \mathcal{R} \rangle$ **obtain** $I \ r$ **where** R :
 $R = \text{region } \mathcal{X} \ I \ r \ \text{valid-region } \mathcal{X} \ k \ I \ r$
unfolding $\mathcal{R}\text{-def}$ **by** auto
from $\text{assms } \mathcal{R}\text{-V}$ **have** $u \in V$ **by** auto
from $\text{assms } R$ **have** $\forall c \in \mathcal{X}. I \ c = \text{Greater } (k \text{ c})$ **by** safe (case-tac $I \ c$; fastforce)
with $R \ \langle u \in V \rangle$ assms **show**
 $u \oplus t \in R$
unfolding $V\text{-def}$ **by** safe (rule; force simp: cval-add-def)
qed

lemma S-I[intro]:

$(l, u) \in S$ if $l \in L \ u \in V \ u \vdash \text{inv-of } A \ l$
using that **by** (auto simp: $\mathcal{S}\text{-def}$ $V\text{-def}$)

lemma rept-ex:

assumes $\mu \in \mathcal{K} (\text{abss } s)$
shows $\text{rept } s \ \mu \in K \ \text{s} \wedge \text{abst } (\text{rept } s \ \mu) = \mu$ **using** assms

proof cases

case prems : (delay $l \ R \ R'$)
then **have** $R \in \mathcal{R}$ **by** auto
from $\text{prems}(2)$ **have** $s \in S$ **by** (auto intro: $\mathcal{S}\text{-abss-S}$)
from $\text{abss-SD}[OF \ \text{prems}(2)]$ **obtain** $l' \ u'$ **where** $s = (l', u') \ u' \in [u]_{\mathcal{R}}$
by metis
with $\text{prems}(3)$ **have** $*$: $s = (l, u') \wedge u' \in R$
apply simp
apply (subst (asm) $\text{abss-S}[OF \ \mathcal{S}\text{-abss-S}]$)

```

using prems(2) by auto
with prems(4) alpha-interp.set-of-regions-spec[OF ‹R ∈ ℛ›] obtain t where R':
  t ≥ 0 R' = [u' ⊕ t]ℛ
by auto
with ‹s ∈ S› * have u' ⊕ t ∈ R' u' ⊕ t ∈ V l ∈ L by auto
with prems(5) have (l, u' ⊕ t) ∈ S unfolding S-def V-def by auto
with ‹R' = [u' ⊕ t]ℛℛ = R by (simp add: * ‹R ∈ ℛ› alpha-interp.region-unique-spec)
from ‹R' ∈ ℛ› obtain I r where R':
  R' = region ℳ I r valid-region ℳ k I r
unfolding ℛ-def by auto
have u' ∈ V using * prems ℛ-V by force
let ?μ' = return-pmf (l, u' ⊕ 0.5)
have elapsed: abst (return-pmf (l, u' ⊕ t)) = μ return-pmf (l, u' ⊕ t) ∈ K s
  if u' ⊕ t ∈ R' t ≥ 0 for t
proof -
  let ?u = u' ⊕ t let ?μ' = return-pmf (l, u' ⊕ t)
  from ‹?u ∈ R'› ‹R' ∈ ℛ› ℛ-V have ?u ∈ V by auto
  with ‹?u ∈ R'› ‹R' ∈ ℛ› have [?u]ℛ = R' using alpha-interp.region-unique-spec by auto
  with ‹?u ∈ V› ‹?u ∈ R'› ‹l ∈ L› prems(4,5) have abss (l, ?u) = (l, R')
    by (subst abss-S) auto
  with prems(1) have abst ?μ' = μ by (auto simp: abst-def)
  moreover from * ‹?u ∈ R'› ‹s ∈ S› prems ‹t ≥ 0› have ?μ' ∈ K s by auto
  ultimately show abst ?μ' = μ ?μ' ∈ K s by auto
qed
show ?thesis
proof (cases R = R')
  case T: True
  show ?thesis
  proof (cases ∀ c ∈ ℳ. u' c > k c)
    case True
    with T * R prems(1,4) ‹s ∈ S› have
      rept s μ = return-pmf (l, u' ⊕ 0.5) (is - = ?μ)
    unfolding rept-def by auto
    from upper-right-closed[OF True] * ‹R' ∈ ℛ› T have u' ⊕ 0.5 ∈ R' by auto
    with elapsed ‹rept - - = -› show ?thesis by auto
  next
    case False
    with T * R prems(1) have
      rept s μ = (SOME μ'. μ' ∈ K s ∧ abst μ' = μ)
    unfolding rept-def by auto
    with default show ?thesis by simp (rule someI; auto)
  qed
next
  case F: False
  show ?thesis
  proof (cases ∀ u ∈ R'. ∀ c ∈ ℳ. ∯ d. d ≤ k c ∧ u c = real d)
    case False
    with F * R prems(1) have
      rept s μ = (SOME μ'. μ' ∈ K s ∧ abst μ' = μ)
    unfolding rept-def by auto
    with default show ?thesis by simp (rule someI; auto)
  next
    case True
    with True F * R prems(1,4) ‹s ∈ S› have
      rept s μ = return-pmf (l, delayedR (SOME R'. μ = return-pmf (l, R')) u')

```

```

    (is - = return-pmf (l, delayedR ?R u'))
  unfolding rept-def by auto
  let ?u = delayedR ?R u'
  from prems(1) have  $\mu = \text{return-pmf } (l, ?R)$  by auto
  with prems(1) have  $?R = R'$  by auto
  moreover from  $R' \text{ True } \langle - \in R' \rangle$  have  $\forall c \in \mathcal{X}. \neg \text{Regions.isConst } (I c)$  by fastforce
  moreover note  $\text{delayedR-correct}[of u' R I r] * \langle R \in \mathcal{R} \rangle R' \text{ True } \langle R' \in \text{Succ } \mathcal{R} R \rangle$ 
  ultimately obtain  $t$  where  $** : \text{delayedR } R' u' \in R' t \geq 0 \text{ delayedR } R' u' = u' \oplus t$  by auto
  moreover from  $\langle ?R = - \rangle \text{rept } - - = - \rangle$  have  $\text{rept } s \mu = \text{return-pmf } (l, \text{delayedR } R' u')$  by auto
  ultimately show  $?thesis$  using elapsed by auto
qed
qed
next
case prems: (action l R  $\tau \mu'$ )
from abss-SD'[OF prems(2,3)] obtain u where u:
  s = (l, u) u  $\in [u]_{\mathcal{R}}$   $[u]_{\mathcal{R}} \in \mathcal{R}$  R =  $[u]_{\mathcal{R}}$ 
by auto
with  $\langle - \in \mathcal{S} \rangle$  have  $(l, u) \in S$  by (auto intro: S-abss-S)
let  $?\mu = \text{map-pmf } (\lambda(X, l). (l, [X:=0]u)) \mu'$ 
from u prems have  $?\mu \in K s$  by (fastforce intro: S-abss-S)
moreover have  $\text{abst } ?\mu = \mu$  unfolding prems(1) abst-def
proof (subst map-pmf-comp, rule pmf.map-cong, safe, goal-cases)
  case A: (1 X l')
  from u have  $u \in V$  using  $\mathcal{R}-V$  by auto
  then have  $[X:=0]u \in V$  by auto
  from prems(1) A
  have  $(l', \text{region-set}' R (\text{SOME } r. \text{set } r = X) 0) \in \mu$  by auto
  from A prems R-G.K-closed  $\langle \mu \in - \rangle$  have
     $l' \in L \text{ region-set}' R (\text{SOME } r. \text{set } r = X) 0 \subseteq \{\text{inv-of } A l'\}$ 
  by (force dest: S-L S-inv)+
  with u have  $[X:=0]u \vdash \text{inv-of } A l'$  unfolding region-set'-def by auto
  with  $\langle l' \in L \rangle \langle [X:=0]u \in V \rangle$  have  $(l', [X:=0]u) \in S$  unfolding S-def V-def by auto
  then have  $\text{abss } (l', [X:=0]u) = (l', [[X:=0]u]_{\mathcal{R}})$  by auto
  also have
     $\dots = (l', \text{region-set}' R (\text{SOME } r. \text{set } r = X) 0)$ 
  using region-set'-eq(1)[unfolded transition-def] prems A u by force
  finally show  $?case$  .
qed
ultimately have default:  $?thesis$  if  $\text{rept } s \mu = (\text{SOME } \mu'. \mu' \in K s \wedge \text{abst } \mu' = \mu)$  using that
by simp (rule someI; auto)
show  $?thesis$ 
proof (cases  $\exists R. \mu = \text{return-pmf } (l, R)$ )
  case False
  with  $\langle s = (l, u) \rangle$  have  $\text{rept } s \mu = (\text{SOME } \mu'. \mu' \in K s \wedge \text{abst } \mu' = \mu)$  unfolding rept-def by auto
  with default show  $?thesis$  by auto
next
case True
then obtain R' where  $R' : \mu = \text{return-pmf } (l, R')$  by auto
show  $?thesis$ 
proof (cases  $R = R'$ )
  case False
  from R' prems(1) have
     $\forall (X, l') \in \mu'. (l', \text{region-set}' R (\text{SOME } r. \text{set } r = X) 0) = (l, R')$ 
  by (auto simp: map-pmf-eq-return-pmf-iff[of -  $\mu'$  (l, R')])
  then obtain X where
     $\text{region-set}' R (\text{SOME } r. \text{set } r = X) 0 = R' (X, l) \in \mu'$ 
  using set-pmf-not-empty by force
  with prems(4) have  $X \subseteq \mathcal{X}$  by (simp add: admissible-targets-clocks(1))
  moreover then have
     $\text{set } (\text{SOME } r. \text{set } r = X) = X$ 
  by - (rule someI-ex, metis finite-list finite(1) finite-subset)

```

ultimately have $\text{set } (\text{SOME } r. \text{ set } r = X) \subseteq \mathcal{X}$ **by auto**
with $\text{alpha-interp.region-reset-not-Succ False } \langle - = R' \rangle u(3,4)$ **have** $R' \notin \text{Succ } \mathcal{R} R$ **by auto**
with $\langle s = (l, u) \rangle R' u(4)$ **False have**
 $\text{rept } s \mu = (\text{SOME } \mu'. \mu' \in K s \wedge \text{abst } \mu' = \mu)$
unfolding rept-def **by auto**
with default show $?thesis$ **by auto**
next
case $T: \text{True}$
show $?thesis$
proof $(\text{cases } \forall c \in \mathcal{X}. \text{real } (k c) < u c)$
case False
with $T \langle s = (l, u) \rangle R' u(4)$ **have**
 $\text{rept } s \mu = (\text{SOME } \mu'. \mu' \in K s \wedge \text{abst } \mu' = \mu)$
unfolding rept-def **by auto**
with default show $?thesis$ **by auto**
next
case True
with $T \langle s = (l, u) \rangle R' u(4) \langle (l, u) \in S \rangle$ **have**
 $\text{rept } s \mu = \text{return-pmf } (l, u \oplus 0.5)$
unfolding rept-def **by auto**
from $\text{upper-right-closed}[OF \text{True}] T u \mathcal{R}\text{-}V$ **have** $u \oplus 0.5 \in R' u \oplus 0.5 \in V$ **by force+**
moreover then have $[u \oplus 0.5]_{\mathcal{R}} = R'$
using $T \text{alpha-interp.region-unique-spec } u(3,4)$ **by blast**
moreover note $* = \langle \text{rept } - - = \rangle R' \langle \text{abss } s \in S \rangle \langle \text{abss } s = \rangle \text{prems}(5)$
ultimately have $\text{abst } (\text{rept } s \mu) = \mu$
apply $(\text{simp add: abst-def})$
apply (subst abss-S)
by $(\text{auto simp: S-L S-def V-def T dest: S-inv})$
moreover from $* \langle s = \rangle \langle (l, u) \in S \rangle \langle - \in R' \rangle$ **have**
 $\text{rept } s \mu \in K s$
apply simp
apply (rule K.delay)
by $(\text{auto simp: T dest: S-inv})$
ultimately show $?thesis$ **by auto**
qed
qed
qed
next
case loop
obtain $l u$ **where** $s = (l, u)$ **by force**
show $?thesis$
proof $(\text{cases } s \in S)$
case $T: \text{True}$
with $\langle s = \rangle$ **have** $*: l \in L u \in [u]_{\mathcal{R}} [u]_{\mathcal{R}} \in \mathcal{R} \text{abss } s = (l, [u]_{\mathcal{R}})$ **by auto**
then have $\text{abss } s = (l, [u]_{\mathcal{R}})$ **by auto**
with $\langle s \in S \rangle S\text{-abss-S}$ **have** $(l, [u]_{\mathcal{R}}) \in S$ **by auto**
with $S\text{-inv}$ **have** $[u]_{\mathcal{R}} \subseteq \{u. u \vdash \text{inv-of } A l\}$ **by auto**
show $?thesis$
proof $(\text{cases } \forall c \in \mathcal{X}. \text{real } (k c) < u c)$
case True
with $* \langle \mu = \rangle \langle s = \rangle \langle s \in S \rangle$ **have**
 $\text{rept } s \mu = \text{return-pmf } (l, u \oplus 0.5)$
unfolding rept-def **by auto**
from $\text{upper-right-closed}[OF \text{True}] *$ **have** $u \oplus 0.5 \in [u]_{\mathcal{R}}$ **by auto**
moreover with $* \mathcal{R}\text{-}V$ **have** $u \oplus 0.5 \in V$ **by auto**
moreover with $\text{calculation } * \text{alpha-interp.region-unique-spec}$ **have** $[u \oplus 0.5]_{\mathcal{R}} = [u]_{\mathcal{R}}$ **by blast**
moreover note $* \langle \text{rept } - - = \rangle \langle s = \rangle T \langle \mu = \rangle \langle (l, -) \in S \rangle S\text{-inv}$
ultimately show $?thesis$ **unfolding** rept-def
apply simp
apply safe
apply fastforce

```

  apply (simp add: abst-def)
  apply (subst abst-def abss-S)
  by fastforce+
next
case False
with * ⟨s = -⟩ ⟨μ = -⟩ have
  rept s μ = (SOME μ'. μ' ∈ K s ∧ abst μ' = μ)
  unfolding rept-def by auto
  with ⟨μ = -⟩ show ?thesis by simp (rule someI[where x = return-pmf s], auto simp: abst-def)
qed
next
case False
with ⟨s = -⟩ ⟨μ = -⟩ have
  rept s μ = (SOME μ'. μ' ∈ K s ∧ abst μ' = μ)
  unfolding rept-def by auto
  with ⟨μ = -⟩ show ?thesis by simp (rule someI[where x = return-pmf s], auto simp: abst-def)
qed
qed

```

```

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

```

```

lemma abst-rept2:
  assumes μ ∈ K s s ∈ S
  shows abst (rept (reps s) μ) = μ
using assms by auto

```

```

lemma rept-K2:
  assumes μ ∈ K s s ∈ S
  shows rept (reps s) μ ∈ K (reps s)
using assms by auto

```

```

lemma theI':
  assumes P a
  and ∧x. P x ⇒ x = a
  shows P (THE x. P x) ∧ (∀ y. P y ⇒ y = (THE x. P x))
using theI assms by metis

```

```

lemma cont-cfg-defined:
  fixes cfg s
  assumes cfg ∈ valid-cfg s ∈ abst (action cfg)
  defines x ≡ THE x. abss x = s ∧ x ∈ action cfg
  shows (abss x = s ∧ x ∈ action cfg) ∧ (∀ y. abss y = s ∧ y ∈ action cfg ⇒ y = x)
proof -
  from assms(2) obtain s' where s' ∈ action cfg s = abss s' unfolding abst-def by auto
  with assms show ?thesis unfolding x-def
  by -(rule theI'[of - s'], auto intro: K-bisim-unique MDP.valid-cfg-state-in-S dest: MDP.valid-cfgD)
qed

```

```

definition
  absc' :: ('s * ('c, t) eval) cfg ⇒ ('s * ('c, t) eval set) cfg
where
  absc' cfg = cfg-corec
    (abss (state cfg))
    (abst o action)
    (λ cfg s. cont cfg (THE x. abss x = s ∧ x ∈ action cfg)) cfg

```

5.2.5 Configuration

```

definition

```

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

definition

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

definition

$\text{repc cfg} = \text{repcs (reps (state cfg)) cfg}$

lemma \mathcal{S} -state-absc-repc[simp]:

$\text{state cfg} \in \mathcal{S} \Longrightarrow \text{state (absc (repc cfg))} = \text{state cfg}$

by (simp add: absc-def repc-def repcs-def)

lemma action-repc:

$\text{action (repc cfg)} = \text{rept (reps (state cfg)) (action cfg)}$

unfolding repc-def repcs-def **by** simp

lemma action-absc:

$\text{action (absc cfg)} = \text{abst (action cfg)}$

unfolding absc-def **by** simp

lemma action-absc':

$\text{action (absc cfg)} = \text{map-pmf abss (action cfg)}$

unfolding absc-def **unfolding** abst-def **by** simp

lemma

notes $R\text{-}G.\text{cfg-onD-state}$ [simp del]

assumes $\text{state cfg} \in \mathcal{S}$ $s' \in \text{set-pmf (action (repc cfg))}$ $\text{cfg} \in R\text{-}G.\text{cfg-on (state cfg)}$

shows $\text{cont (repc cfg) s'} = \text{repcs s' (cont cfg (abss s'))}$

using *assms* **by** (auto simp: repc-def repcs-def abss-reps-id)

lemma cont-repcs1:

notes $R\text{-}G.\text{cfg-onD-state}$ [simp del]

assumes $\text{abss s} \in \mathcal{S}$ $s' \in \text{set-pmf (action (repcs s cfg))}$ $\text{cfg} \in R\text{-}G.\text{cfg-on (abss s)}$

shows $\text{cont (repcs s cfg) s'} = \text{repcs s' (cont cfg (abss s'))}$

using *assms* **by** (auto simp: repc-def repcs-def abss-reps-id)

lemma cont-absc-1:

notes $MDP.\text{cfg-onD-state}$ [simp del]

assumes $\text{cfg} \in \text{valid-cfg}$ $s' \in \text{set-pmf (action cfg)}$

shows $\text{cont (absc cfg) (abss s')} = \text{absc (cont cfg s')}$

proof –

define x **where** $x \equiv \text{THE } x. x \sim s' \wedge x \in \text{set-pmf (action cfg)}$

from *assms*(2) **have** $\text{abss s'} \in \text{set-pmf (abst (action cfg))}$ **unfolding** abst-def **by** auto

from cont-cfg-defined [OF *assms*(1) *this*] **have**

$(x \sim s' \wedge x \in \text{set-pmf (action cfg)}) \wedge (\forall y. y \sim s' \wedge y \in \text{set-pmf (action cfg)} \longrightarrow y = x)$

unfolding $x\text{-def}$.

with *assms* **have** $s' = x$ **by** fastforce

then show *?thesis*

unfolding absc-def abst-def repc-def $x\text{-def}$ **using** *assms*(2) **by** auto

qed

lemma *state-repc*:

state (repc cfg) = reps (state cfg)

unfolding *repc-def repcs-def* **by** *simp*

lemma *abss-reps-id'*:

notes *R-G.cfg-onD-state[*simp del*]*

assumes *cfg ∈ R-G.valid-cfg s ∈ set-pmf (action cfg)*

shows *abss (reps s) = s*

using *assms* **by** (*auto intro: abss-reps-id R-G.valid-cfg-state-in-S R-G.valid-cfgD*)

lemma *valid-cfg-coinduct[coinduct set: valid-cfg]*:

assumes *P cfg*

assumes $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{state } \text{cfg} \in S$

assumes $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{action } \text{cfg} \in K (\text{state } \text{cfg})$

assumes $\bigwedge \text{cfg } t. P \text{ cfg} \implies t \in \text{action } \text{cfg} \implies P (\text{cont } \text{cfg } t)$

shows *cfg ∈ valid-cfg*

proof –

from *assms* **have** *cfg ∈ MDP.cfg-on (state cfg)* **by** (*coinduction arbitrary: cfg*) *auto*

moreover from *assms* **have** *state cfg ∈ S* **by** *auto*

ultimately show *?thesis* **by** (*intro MDP.valid-cfgI*)

qed

lemma *state-repcD[*simp*]*:

assumes *cfg ∈ R-G.cfg-on s*

shows *state (repc cfg) = reps s*

using *assms* **unfolding** *repc-def repcs-def* **by** *auto*

lemma *ccompatible-subs[*intro*]*:

assumes *ccompatible R g R ∈ R u ∈ R u ⊢ g*

shows $R \subseteq \{u. u \vdash g\}$

using *assms* **unfolding** *ccompatible-def* **by** *auto*

lemma *action-abscD[*dest*]*:

cfg ∈ MDP.cfg-on s \implies *action (absc cfg) ∈ K (abss s)*

unfolding *absc-def abst-def*

proof *simp*

assume *cfg: cfg ∈ MDP.cfg-on s*

then have *action cfg ∈ K s* **by** *auto*

then show *map-pmf abss (action cfg) ∈ K (abss s)*

proof *cases*

case *prems: (delay l u t)*

then have $[u \oplus t]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*

moreover with *prems ccompatible-inv[of l]* **have**

$[u \oplus t]_{\mathcal{R}} \subseteq \{v. v \vdash \text{PTA.inv-of } A \ l\}$

unfolding *ccompatible-def* **by** *force*

moreover from *prems* **have** *abss (l, u ⊕ t) = (l, [u ⊕ t]_R)* **by** (*subst abss-S*) *auto*

ultimately show *?thesis* **using** *prems* **by** *auto*

next

case *prems: (action l u g μ)*

then have $[u]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*

moreover with *prems ccompatible-guard* **have** $[u]_{\mathcal{R}} \subseteq \{u. u \vdash g\}$

by (*intro ccompatible-subs*) *auto*

moreover have

map-pmf abss (action cfg)

$= \text{map-pmf } (\lambda(X, l). (l, \text{region-set}' ([u]_{\mathcal{R}}) (\text{SOME } r. \text{set } r = X) \ 0)) \ \mu$

proof –

have *abss (l', [X:=0]u) = (l', region-set' ([u]_R) (SOME r. set r = X) 0)*

if $(X, l') \in \mu$ **for** $X \ l'$

```

proof –
  from that prems have  $A \vdash l \longrightarrow^{g,\mu,X} l'$ 
    by auto
  from that prems  $MDP.action-closed[OF - cfg]$  have  $(l', [X:=0]u) \in S$  by force
  then have  $abss (l', [X:=0]u) = (l', [[X:=0]u]_{\mathcal{R}})$  by auto
  also have
     $\dots = (l', region-set' ([u]_{\mathcal{R}}) (SOME r. set r = X) 0)$ 
    using  $region-set'-eq(1)[OF - - \langle A \vdash l \longrightarrow^{g,\mu,X} l' \rangle]$  prems by auto
  finally show ?thesis .
  qed
  then show ?thesis
    unfolding  $prems(1)$ 
    by (auto intro: pmf.map-cong simp: map-pmf-comp)
  qed
  ultimately show ?thesis using prems by auto
next
  case prems: loop
  then show ?thesis by auto
qed
qed

lemma repcs-valid[intro]:
  assumes  $cfg \in R-G.valid-cfg$   $abss\ s = state\ cfg$ 
  shows  $repcs\ s\ cfg \in valid-cfg$ 
using assms
proof (coinduction arbitrary: cfg s)
  case 1
  then show ?case
  by (auto simp: repcs-def S-abss-S dest: R-G.valid-cfg-state-in-S)
next
  case (2  $cfg'\ s$ )
  then show ?case
  by (simp add: repcs-def) (rule rept-K, auto dest: R-G.valid-cfgD)
next
  case prems: (3 s' cfg)
  let  $?cfg = cont\ cfg\ (abss\ s')$ 
  from prems have  $abss\ s' \in abst\ (rept\ s\ (action\ cfg))$  unfolding repcs-def abst-def by auto
  with prems have
     $abss\ s' \in action\ cfg$ 
  by (subst (asm) abst-rept-id) (auto dest: R-G.valid-cfgD)
  with prems show ?case
  by (inst-existentials ?cfg s', subst cont-repcs1)
    (auto dest: R-G.valid-cfg-state-in-S intro: R-G.valid-cfgD R-G.valid-cfg-cont)
qed

lemma repc-valid[intro]:
  assumes  $cfg \in R-G.valid-cfg$ 
  shows  $repc\ cfg \in valid-cfg$ 
using assms unfolding repc-def by (force dest: R-G.valid-cfg-state-in-S)

lemma action-abst-repcs:
  assumes  $cfg \in R-G.valid-cfg$   $abss\ s = state\ cfg$ 
  shows  $abst\ (action\ (repcs\ s\ cfg)) = action\ cfg$ 
proof –
  from assms show ?thesis
  unfolding repc-def repcs-def
  apply simp
  apply (subst abst-rept-id)
  by (auto dest: R-G.cfg-onD-action R-G.valid-cfgD)
qed

```

lemma *action-abst-repc*:
assumes $cfg \in R-G.valid-cfg$
shows $abst (action (repc\ cfg)) = action\ cfg$
proof –
from *assms* **have** $abss (reps (state\ cfg)) = state\ cfg$ **by** (*auto dest: R-G.valid-cfg-state-in-S*)
with *action-abst-repcs[OF assms]* **show** *?thesis* **unfolding** *repc-def* **by** *auto*
qed

lemma *state-absc*:
 $state (absc\ cfg) = abss (state\ cfg)$
unfolding *absc-def* **by** *auto*

lemma *state-repcs[simp]*:
 $state (repcs\ s\ cfg) = s$
unfolding *repcs-def* **by** *auto*

lemma *repcs-bisim*:
notes *R-G.cfg-onD-state[simp del]*
assumes $cfg \in R-G.valid-cfg\ x \in S\ x \sim x'\ abss\ x = state\ cfg$
shows $absc (repcs\ x\ cfg) = absc (repcs\ x'\ cfg)$
using *assms*
proof –
from *assms* **have** $abss\ x' = state\ cfg$ **by** *auto*
from *assms* **have** $abss\ x' \in S$ **by** *auto*
then **have** $x' \in S$ **by** (*auto intro: S-abss-S*)
with *assms* **show** *?thesis*
proof (*coinduction arbitrary: cfg x x'*)
case *state*
then **show** *?case* **by** (*simp add: state-absc*)
next
case *action*
then **show** *?case* **unfolding** *absc-def repcs-def* **by** (*auto dest: R-G.valid-cfgD*)
next
case *prems: (cont s cfg x x')*
define cfg' **where** $cfg' = cont\ cfg\ s$
define t **where** $t \equiv THE\ y.\ abss\ y = s \wedge y \in action (repcs\ x\ cfg)$
define t' **where** $t' \equiv THE\ y.\ abss\ y = s \wedge y \in action (repcs\ x'\ cfg)$
from *prems* **have** *valid: repcs x cfg ∈ valid-cfg* **by** (*intro repcs-valid*)
from *prems* **have** $s \in abst (action (repcs\ x\ cfg))$
unfolding *cfg'-def* **by** (*simp add: action-absc*)
with *prems* **have** $s \in action\ cfg$ **by** (*auto dest: R-G.valid-cfgD simp: repcs-def*)
with *prems* **have** $s \in S$ **by** (*auto intro: R-G.valid-cfg-action*)
from *cont-cfg-defined[OF valid *]* **have** t :
 $abss\ t = s\ t \in action (repcs\ x\ cfg)$
unfolding *t-def* **by** *auto*
have $cont (absc (repcs\ x\ cfg))\ s = cont (absc (repcs\ x\ cfg)) (abss\ t)$ **using** t **by** *auto*
have $cont (absc (repcs\ x\ cfg))\ s = absc (cont (repcs\ x\ cfg))\ t$
using t *valid* **by** (*auto simp: cont-absc-1*)
also **have** $\dots = absc (repcs\ t (cont\ cfg\ s))$
using *prems t* **by** (*subst cont-repcs1*) (*auto dest: R-G.valid-cfgD*)
finally **have** *cont-x: cont (absc (repcs x cfg)) s = absc (repcs t (cont cfg s))* .
from *prems* **have** *valid: repcs x' cfg ∈ valid-cfg* **by** *auto*
from $\langle s \in action\ cfg \rangle$ *prems* **have** $s \in abst (action (repcs\ x'\ cfg))$
by (*auto dest: R-G.valid-cfgD simp: repcs-def*)
from *cont-cfg-defined[OF valid this]* **have** t' :
 $abss\ t' = s\ t' \in action (repcs\ x'\ cfg)$
unfolding *t'-def* **by** *auto*
have $cont (absc (repcs\ x'\ cfg))\ s = cont (absc (repcs\ x'\ cfg)) (abss\ t')$ **using** t' **by** *auto*
have $cont (absc (repcs\ x'\ cfg))\ s = absc (cont (repcs\ x'\ cfg))\ t'$

```

    using t' valid by (auto simp: cont-absc-1)
  also have ... = absc (repcs t' (cont cfg s))
    using prems t' by (subst cont-repcs1) (auto dest: R-G.valid-cfgD)
  finally have cont (absc (repcs x' cfg)) s = absc (repcs t' (cont cfg s)) .
  with cont-x ⟨s ∈ action cfg⟩ prems(1) t t' ⟨s ∈ S⟩
  show ?case
    by (inst-existentials cont cfg s t t')
      (auto intro: S-abss-S R-G.valid-cfg-action R-G.valid-cfg-cont)
qed
qed

```

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 using assms
proof (subst eq-commute, coinduction arbitrary: cfg s)
  case state
  then show ?case by (simp add: absc-def repc-def repcs-def)
next
  case prems: (action cfg)
  then show ?case by (auto simp: action-abst-repcs action-absc)
next
  case prems: (cont s^)
  define cfg' where cfg' ≡ repcs s cfg
  define t where t ≡ THE x. abss x = s^ ∧ x ∈ set-pmf (action cfg')
  from prems have cfg ∈ R-G.cfg-on (state cfg) state cfg ∈ S by (auto dest: R-G-I)
  then have *: cfg ∈ R-G.cfg-on (abss (reps (state cfg))) abss (reps (state cfg)) ∈ S by auto
  from prems have s^ ∈ S by (auto intro: R-G.valid-cfg-action)
  from prems have valid: cfg' ∈ valid-cfg unfolding cfg'-def by (intro repcs-valid)
  from prems have s^ ∈ abst (action cfg') unfolding cfg'-def by (subst action-abst-repcs)
  from cont-cfg-defined[OF valid this] have t:
    abss t = s^ t ∈ action cfg'
  unfolding t-def cfg'-def by auto
  with prems have t ∼ reps (abss t)
  apply –
  apply (subst S-abss-reps)
  by (auto intro: R-G.valid-cfg-action)
  have cont (absc cfg^) s^ = cont (absc cfg^) (abss t) using t by auto
  have cont (absc cfg^) s^ = absc (cont cfg' t) using t valid by (auto simp: cont-absc-1)
  also have ... = absc (repcs t (cont cfg s^)) using prems t * ⟨t ∼  $\rightarrow$  valid⟩
  by (fastforce dest: R-G-I intro: repcs-bisim simp: cont-repcs1 cfg'-def)
  finally show ?case
  apply –
  apply (rule exI[where x = cont cfg s^], rule exI[where x = t])
  unfolding cfg'-def using prems t by (auto intro: R-G.valid-cfg-cont)
qed

```

lemma *absc-repc-id*:

```

  notes R-G.cfg-onD-state[simp del]
  assumes cfg ∈ R-G.valid-cfg
  shows absc (repc cfg) = cfg using assms
unfolding repc-def using assms by (subst absc-repcs-id) (auto dest: R-G-I)

```

lemma *K-cfg-map-absc*:

```

cfg ∈ valid-cfg ⇒ K-cfg (absc cfg) = map-pmf absc (K-cfg cfg)

```

by (auto simp: K-cfg-def map-pmf-comp action-absc abst-def cont-absc-1 intro: map-pmf-cong)

lemma *smap-comp*:

(*smap f o smap g*) = *smap (f o g)*

by (auto simp: stream.map-comp)

lemma *state-abscD[simp]*:

assumes *cfg* ∈ *MDP.cfg-on s*

shows *state (absc cfg)* = *abss s*

using *assms* **unfolding** *absc-def* **by** *auto*

lemma *R-G-valid-cfg-coinduct[coinduct set: valid-cfg]*:

assumes *P cfg*

assumes $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{state } \text{cfg} \in \mathcal{S}$

assumes $\bigwedge \text{cfg}. P \text{ cfg} \implies \text{action } \text{cfg} \in \mathcal{K} (\text{state } \text{cfg})$

assumes $\bigwedge \text{cfg } t. P \text{ cfg} \implies t \in \text{action } \text{cfg} \implies P (\text{cont } \text{cfg } t)$

shows *cfg* ∈ *R-G.valid-cfg*

proof –

from *assms* **have** *cfg* ∈ *R-G.cfg-on (state cfg)* **by** (*coinduction arbitrary: cfg*) *auto*

moreover from *assms* **have** *state cfg* ∈ *S* **by** *auto*

ultimately show *?thesis* **by** (*intro R-G.valid-cfgI*)

qed

lemma *absc-valid[intro]*:

assumes *cfg* ∈ *valid-cfg*

shows *absc cfg* ∈ *R-G.valid-cfg*

using *assms*

proof (*coinduction arbitrary: cfg*)

case *1*

then show *?case* **by** (*auto simp: absc-def dest: MDP.valid-cfg-state-in-S*)

next

case (*2 cfg'*)

then show *?case* **by** (*subst state-abscD*) (*auto intro: MDP.valid-cfgD action-abscD*)

next

case *prems: (∃ s' cfg)*

define *t* **where** *t* ≡ *THE x. abss x = s' ∧ x ∈ set-pmf (action cfg)*

let *?cfg = cont cfg t*

from *prems* **obtain** *s* **where** *s' = abss s s ∈ action cfg* **by** (*auto simp: action-absc'*)

with *cont-cfg-defined[OF prems(1), of s']* **have**

abss t = s' t ∈ set-pmf (action cfg)

$\forall y. \text{abss } y = s' \wedge y \in \text{set-pmf } (\text{action } \text{cfg}) \longrightarrow y = t$

unfolding *t-def abst-def* **by** *auto*

with *prems* **show** *?case*

by (*inst-existentials ?cfg*)

(*auto intro: MDP.valid-cfg-cont simp: abst-def action-absc absc-def t-def*)

qed

lemma *K-cfg-set-absc*:

assumes *cfg* ∈ *valid-cfg* *cfg'* ∈ *K-cfg cfg*

shows *absc cfg'* ∈ *K-cfg (absc cfg)*

using *assms* **by** (*auto simp: K-cfg-map-absc*)

lemma *abst-action-repcs*:

assumes *cfg* ∈ *R-G.valid-cfg* *abss s = state cfg*

shows *abst (action (repcs s cfg)) = action cfg*

unfolding *repc-def repcs-def* **using** *assms* **by** (*simp, subst abst-rept-id*) (*auto intro: R-G-I*)

lemma *abst-action-repc*:

assumes *cfg* ∈ *R-G.valid-cfg*

shows $abst (action (repc\ cf\ g)) = action\ cf\ g$
using *assms* **unfolding** *repc-def* **by** (*auto* *intro*: *abst-action-repcs simp*: *R-G-I*)

lemma *K-elem-abss-inj'*:

assumes $\mu \in K\ s$
and $s \in S$

shows *inj-on* *abss* (*set-pmf* μ)

using *assms* *K-elem-abss-inj* **by** (*simp* *add*: *K-bisim-unique inj-onI*)

lemma *K-cfg-rept-aux*:

assumes $cfg \in R-G.valid-cfg\ abss\ s = state\ cf\ g\ x \in rept\ s\ (action\ cf\ g)$

defines $t \equiv \lambda\ cf\ g'.\ THE\ s'.\ s' \in rept\ s\ (action\ cf\ g) \wedge s' \sim x$

shows $t\ cf\ g' = x$

proof –

from *assms* **have** $rept\ s\ (action\ cf\ g) \in K\ s\ s \in S$ **by** (*auto* *simp*: *R-G-I S-abss-S*)

from *K-bisim-unique*[*OF* *this*(2,1) - *assms*(3)] *assms*(3) **show** *?thesis* **unfolding** *t-def* **by** *blast*

qed

lemma *K-cfg-rept-action*:

assumes $cfg \in R-G.valid-cfg\ abss\ s = state\ cf\ g\ cf\ g' \in set-pmf\ (K-cfg\ cf\ g)$

shows $abss\ (THE\ s'.\ s' \in rept\ s\ (action\ cf\ g) \wedge abss\ s' = state\ cf\ g') = state\ cf\ g'$

proof –

let $?mu = rept\ s\ (action\ cf\ g)$

from *abst-rept-id* *assms* **have** $action\ cf\ g = abst\ ?mu$ **by** (*auto* *simp*: *R-G-I*)

moreover **from** *assms* **have** $state\ cf\ g' \in action\ cf\ g$ **by** (*auto* *simp*: *set-K-cfg*)

ultimately **have** $state\ cf\ g' \in abst\ ?mu$ **by** *simp*

then **obtain** s' **where** $s' \in ?mu\ abss\ s' = state\ cf\ g'$ **by** (*auto* *simp*: *abst-def pmf.set-map*)

with *K-cfg-rept-aux*[*OF* *assms*(1,2) *this*(1)] **show** *?thesis* **by** *auto*

qed

lemma *K-cfg-map-repcs*:

assumes $cfg \in R-G.valid-cfg\ abss\ s = state\ cf\ g$

defines $repc' \equiv (\lambda\ cf\ g'.\ repcs\ (THE\ s'.\ s' \in rept\ s\ (action\ cf\ g) \wedge abss\ s' = state\ cf\ g')\ cf\ g')$

shows $K-cfg\ (repcs\ s\ cf\ g) = map-pmf\ repc'\ (K-cfg\ cf\ g)$

proof –

let $?mu = rept\ s\ (action\ cf\ g)$

define t **where** $t \equiv \lambda\ cf\ g'.\ THE\ s.\ s \in ?mu \wedge abss\ s = state\ cf\ g'$

have $t:\ t\ (cont\ cf\ g\ (abss\ s')) = s'$ **if** $s' \in ?mu$ **for** s'

using *K-cfg-rept-aux*[*OF* *assms*(1,2) *that*] **unfolding** *t-def* **by** *auto*

show *?thesis*

unfolding *K-cfg-def* **using** t

by (*subst* *abst-action-repcs*[*symmetric*])

(*auto* *simp*: *repc-def repcs-def t-def map-pmf-comp abst-def assms intro*: *map-pmf-cong*)

qed

lemma *K-cfg-map-repc*:

assumes $cfg \in R-G.valid-cfg$

defines

$repc'\ cf\ g' \equiv repcs\ (THE\ s.\ s \in rept\ (rep\ (state\ cf\ g))\ (action\ cf\ g) \wedge abss\ s = state\ cf\ g')\ cf\ g'$

shows

$K-cfg\ (repc\ cf\ g) = map-pmf\ repc'\ (K-cfg\ cf\ g)$

using *assms* **unfolding** *repc'-def repc-def* **by** (*auto* *simp*: *R-G-I K-cfg-map-repcs*)

lemma *R-G-K-cfg-valid-cfgD*:

assumes $cfg \in R-G.valid-cfg\ cf\ g' \in K-cfg\ cf\ g$

shows $cf\ g' = cont\ cf\ g\ (state\ cf\ g')\ state\ cf\ g' \in action\ cf\ g$

proof –

from *assms*(2) **obtain** s **where** $s \in action\ cf\ g\ cf\ g' = cont\ cf\ g\ s$ **by** (*auto* *simp*: *set-K-cfg*)

with *assms* **show**

$cf\ g' = cont\ cf\ g\ (state\ cf\ g')\ state\ cf\ g' \in action\ cf\ g$

by (*auto* *intro*: *R-G.valid-cfg-state-in-S R-G.valid-cfgD*)

qed

lemma *K-cfg-valid-cfgD*:

assumes $cfg \in \text{valid-cfg}$ $cfg' \in K\text{-cfg}$ cfg

shows $cfg' = \text{cont } cfg \text{ (state } cfg')$ $state\ cfg' \in \text{action } cfg$

proof -

from *assms*(2) obtain s where $s \in \text{action } cfg$ $cfg' = \text{cont } cfg\ s$ by (*auto simp: set-K-cfg*)

with *assms* show

$cfg' = \text{cont } cfg \text{ (state } cfg')$ $state\ cfg' \in \text{action } cfg$

by *auto*

qed

lemma *absc-bisim-abss*:

assumes $\text{absc } x = \text{absc } x'$

shows $state\ x \sim state\ x'$

proof -

from *assms* have $state\ (\text{absc } x) = state\ (\text{absc } x')$ by *simp*

then show *?thesis* by (*simp add: state-absc*)

qed

lemma *K-cfg-bisim-unique*:

assumes $cfg \in \text{valid-cfg}$ and $x \in K\text{-cfg}$ $cfg\ x' \in K\text{-cfg}$ cfg and $state\ x \sim state\ x'$

shows $x = x'$

proof -

define t where $t \equiv \text{THE } x'. x' \sim state\ x \wedge x' \in \text{set-pmf } (\text{action } cfg)$

from *K-cfg-valid-cfgD* *assms* have *:

$x = \text{cont } cfg \text{ (state } x)$ $state\ x \in \text{action } cfg$

$x' = \text{cont } cfg \text{ (state } x')$ $state\ x' \in \text{action } cfg$

by *auto*

with *assms* have

$cfg \in \text{valid-cfg}$ $\text{abss } (state\ x) \in \text{set-pmf } (\text{abst } (\text{action } cfg))$

unfolding *abst-def* by *auto*

with *cont-cfg-defined*[of cfg $\text{abss } (state\ x)$] have

$\forall y. y \sim state\ x \wedge y \in \text{set-pmf } (\text{action } cfg) \longrightarrow y = t$

unfolding *t-def* by *auto*

with * *assms*(4) have $state\ x' = t$ $state\ x = t$ by *fastforce+*

with * show *?thesis* by *simp*

qed

lemma *absc-distr-self*:

$MDP.MC.T \text{ (absc } cfg) = \text{distr } (MDP.MC.T\ cfg) \text{ MDP.MC.S } (\text{smap } \text{absc})$ if $cfg \in \text{valid-cfg}$

using $\langle \text{cfg} \in \rightarrow \rangle$

proof (*coinduction arbitrary: cfg rule: MDP.MC.T-coinduct*)

case *prob*

show *?case* by (*rule MDP.MC.T.prob-space-distr, simp*)

next

case *sets*

show *?case* by *auto*

next

case *prems*: (*cont* cfg)

define t where $t \equiv \lambda y. \text{THE } x. y = \text{absc } x \wedge x \in K\text{-cfg } cfg$

define M' where $M' \equiv \lambda cfg. \text{distr } (MDP.MC.T \text{ (} t\ \text{cfg)}) \text{ MDP.MC.S } (\text{smap } \text{absc})$

show *?case*

proof (*rule exI*[where $x = M'$], *safe, goal-cases*)

case *A*: (1 y)

from *A* *prems* obtain x' where $y = \text{absc } x'$ $x' \in K\text{-cfg } cfg$ by (*auto simp: K-cfg-map-absc*)

with *K-cfg-bisim-unique*[OF *prems* - - *absc-bisim-abss*] have

$y = \text{absc } (t\ y)$ $x' = t\ y$

unfolding *t-def* by (*auto intro: theI2*)

```

moreover have  $x' \in \text{valid-cfg}$  using  $\langle x' \in \cdot \rangle \text{prems}$  by auto
ultimately show  $?case$  unfolding  $M'\text{-def}$  by auto
next
  case 5
show  $?case$  unfolding  $M'\text{-def}$ 
  apply (subst distr-distr)
  prefer 3
  apply (subst MDP.MC.T-eq-bind)
  apply (subst distr-bind)
  prefer 4
  apply (subst distr-distr)
  prefer 3
  apply (subst K-cfg-map-absc)
  apply (rule prems)
  apply (subst map-pmf-rep-eq)
  apply (subst bind-distr)
  prefer 4
  apply (rule bind-measure-pmf-cong)
  prefer 3
subgoal premises  $A$  for  $x$ 
proof –
  have  $t(\text{absc } x) = x$  unfolding  $t\text{-def}$ 
  proof (rule the-equality, goal-cases)
  case 1 with  $A$  show  $?case$  by simp
next
  case (2  $x'$ )
  with  $K\text{-cfg-bisim-unique}[OF \text{prems} - A \text{absc-bisim-abss}]$  show  $?case$  by simp
qed
  then show  $?thesis$  by (auto simp: comp-def)
qed
by (fastforce
  simp: space-subprob-algebra MC-syntax.in-S
  intro: bind-measure-pmf-cong MDP.MC.T.subprob-space-distr MDP.MC.T.prob-space-distr
  )+
qed (auto simp: M'\text{-def intro: MDP.MC.T.prob-space-distr)
qed

lemma  $R\text{-G-trace-space-distr-eq}$ :
assumes  $\text{cfg} \in R\text{-G.valid-cfg}$   $\text{abss } s = \text{state } \text{cfg}$ 
shows  $\text{MDP.MC.T } \text{cfg} = \text{distr } (\text{MDP.MC.T } (\text{repcs } s \text{cfg})) \text{MDP.MC.S } (\text{smap } \text{absc})$ 
using assms
proof (coinduction arbitrary: cfg s rule: MDP.MC.T-coinduct)
  case prob
  show  $?case$  by (rule MDP.MC.T.prob-space-distr, simp)
next
  case sets
  show  $?case$  by auto
next
  case prems: (cont cfg s)
  let  $?\mu = \text{rept } s \text{ (action } \text{cfg})$ 
  define  $\text{repc}'$  where  $\text{repc}' \equiv \lambda \text{cfg}'. \text{repcs } (\text{THE } s. s \in ?\mu \wedge \text{abss } s = \text{state } \text{cfg}')$   $\text{cfg}'$ 
  define  $M'$  where  $M' \equiv \lambda \text{cfg}. \text{distr } (\text{MDP.MC.T } (\text{repc}' \text{cfg})) \text{MDP.MC.S } (\text{smap } \text{absc})$ 
  show  $?case$ 
  proof (intro exI[where  $x = M'$ ], safe, goal-cases)
  case  $A: (1 \text{cfg}')$ 
  with  $K\text{-cfg-rept-action}[OF \text{prems}]$  have
     $\text{abss } (\text{THE } s. s \in ?\mu \wedge \text{abss } s = \text{state } \text{cfg}') = \text{state } \text{cfg}'$ 
  by auto
  moreover from  $A$  prems have  $\text{cfg}' \in R\text{-G.valid-cfg}$  by auto
  ultimately show  $?case$  unfolding  $M'\text{-def repc}'\text{-def}$  by best
next

```



```

case 4
show ?case unfolding M'-def by (rule MDP.MC.T.prob-space-distr, simp)
next
case 5
have *: smap absc ∘ (##) (repc' cfg') = (##) cfg' ∘ smap absc
if cfg' ∈ set-pmf (K-cfg cfg) for cfg'
proof -
  from K-cfg-rept-action[OF prems that] have
    abss (THE s. s ∈ ?μ ∧ abss s = state cfg') = state cfg'
  .
  with prems that have *:
    absc (repc' cfg') = cfg'
  unfolding repc'-def by (subst absc-repcs-id, auto)
  then show (smap absc ∘ (##) (repc' cfg')) = ((##) cfg' ∘ smap absc) by auto
qed
from prems show ?case unfolding M'-def
apply (subst distr-distr)
  apply simp+
  apply (subst MDP.MC.T.eq-bind)
  apply (subst distr-bind)
  prefer 2
  apply simp
  apply (rule MDP.MC.distr-Stream-subprob)
  apply simp
  apply (subst distr-distr)
  apply simp+
  apply (subst K-cfg-map-repcs[OF prems])
  apply (subst map-pmf-rep-eq)
  apply (subst bind-distr)
  by (fastforce simp: *[unfolded repc'-def] repc'-def space-subprob-algebra MC-syntax.in-S
      intro: bind-measure-pmf-cong MDP.MC.T.subprob-space-distr)+
qed (simp add: M'-def)+
qed

```

```

lemma repc-inj-on-K-cfg:
  assumes cfg ∈ R-G.cfg-on s s ∈ S
  shows inj-on repc (set-pmf (K-cfg cfg))
  using assms
  by (intro inj-on-inverseI[where g = absc], subst absc-repc-id)
  (auto intro: R-G.valid-cfgD R-G.valid-cfgI R-G.valid-cfg-state-in-S)

```

```

lemma smap-absc-iff:
  assumes ∧ x y. x ∈ X ⇒ smap abss x = smap abss y ⇒ y ∈ X
  shows (smap state xs ∈ X) = (smap (λz. abss (state z)) xs ∈ smap abss ` X)
proof (safe, goal-cases)
  case 1
  then show ?case unfolding image-def
    by clarify (inst-existentials smap state xs, auto simp: stream.map-comp)
next
  case prems: (2 xs')
  have
    smap (λz. abss (state z)) xs = smap abss (smap state xs)
  by (auto simp: comp-def stream.map-comp)
  with prems have smap abss (smap state xs) = smap abss xs' by simp
  with prems(2) assms show ?case by auto
qed

```

```

lemma valid-abss-reps[simp]:
  assumes cfg ∈ R-G.valid-cfg
  shows abss (reps (state cfg)) = state cfg
using assms by (subst S-abss-reps) (auto intro: R-G.valid-cfg-state-in-S)

```

lemma *in-space-UNIV*: $x \in \text{space}$ (*count-space UNIV*)
by *simp*

lemma *S-reps-S-aux*:
 $\text{reps } (l, R) \in S \implies (l, R) \in \mathcal{S}$
using *ccompatible-inv unfolding reps-def ccompatible-def S-def S-def*
by (*cases R ∈ R; auto simp: non-empty*)

lemma *S-reps-S[intro]*:
 $\text{reps } s \in S \implies s \in \mathcal{S}$
using *S-reps-S-aux* **by** (*metis surj-pair*)

lemma *absc-valid-cfg-eq*:
 $\text{absc } \text{ ` } \text{valid-cfg} = R\text{-G.}\text{valid-cfg}$
apply *safe*
subgoal
by *auto*
subgoal for *cfg*
using *absc-repcs-id*[**where** $s = \text{reps } (\text{state } \text{cfg})$]
by – (*frule repcs-valid*[**where** $s = \text{reps } (\text{state } \text{cfg})$]); *force intro: imageI*
done

lemma *action-repcs*:
 $\text{action } (\text{repcs } (l, u) \text{cfg}) = \text{rept } (l, u) (\text{action } \text{cfg})$
by (*simp add: repcs-def*)

5.3 Equalities Between Measures of Trace Spaces

lemma *path-measure-eq-absc1-new*:
fixes *cfg s*
defines $\text{cfg}' \equiv \text{absc } \text{cfg}$
assumes *valid: cfg ∈ valid-cfg*
assumes $X[\text{measurable}]: X \in R\text{-G.}\text{St}$ **and** $Y[\text{measurable}]: Y \in \text{MDP.}\text{St}$
assumes $P: \text{AE } x \text{ in } (R\text{-G.}\text{T } \text{cfg}')$. $P \ x$ **and** $Q: \text{AE } x \text{ in } (\text{MDP.}\text{T } \text{cfg})$. $Q \ x$
assumes $P'[\text{measurable}]: \text{Measurable.pred } R\text{-G.}\text{St } P$
and $Q'[\text{measurable}]: \text{Measurable.pred } \text{MDP.}\text{St } Q$
assumes $X\text{-}Y\text{-closed}: \bigwedge x \ y. P \ x \implies \text{smap } \text{abss } y = x \implies x \in X \implies y \in Y \wedge Q \ y$
assumes $Y\text{-}X\text{-closed}: \bigwedge x \ y. Q \ y \implies \text{smap } \text{abss } y = x \implies y \in Y \implies x \in X \wedge P \ x$
shows
 $\text{emeasure } (R\text{-G.}\text{T } \text{cfg}') \ X = \text{emeasure } (\text{MDP.}\text{T } \text{cfg}) \ Y$
proof –
have *: $\text{stream-all2 } (\lambda s. (=) (\text{absc } s)) \ x \ y = \text{stream-all2 } (=) (\text{smap } \text{absc } x) \ y$ **for** $x \ y$
by *simp*
have *: $\text{stream-all2 } (\lambda s \ t. t = \text{absc } s) \ x \ y = \text{stream-all2 } (=) \ y (\text{smap } \text{absc } x)$ **for** $x \ y$
using *stream.rel-conversep*[*of* $\lambda s \ t. t = \text{absc } s$]
by (*simp add: conversep-iff*[*abs-def*])
from P **have** $\text{emeasure } (R\text{-G.}\text{T } \text{cfg}') \ X = \text{emeasure } (R\text{-G.}\text{T } \text{cfg}') \ \{x \in X. P \ x\}$
by (*auto intro: emeasure-eq-AE*)
moreover from Q **have** $\text{emeasure } (\text{MDP.}\text{T } \text{cfg}) \ Y = \text{emeasure } (\text{MDP.}\text{T } \text{cfg}) \ \{y \in Y. Q \ y\}$
by (*auto intro: emeasure-eq-AE*)
moreover show *?thesis*
apply (*simp only: calculation*)
unfolding *R-G.T-def MDP.T-def*
apply (*simp add: emeasure-distr*)
apply (*rule sym*)
apply (*rule T-eq-rel-half*[**where** $f = \text{absc}$ **and** $S = \text{valid-cfg}$])
apply (*rule HOL.refl*)
apply *measurable*

```

  apply (simp add: space-stream-space)
subgoal
  unfolding rel-set-strong-def stream.rel-eq
  apply (intro allI impI)
  apply (drule stream.rel-mono-strong[where Ra =  $\lambda s t. t = \text{absc } s$ ])
  apply (simp; fail)
  subgoal for x y
  using Y-X-closed[of smap state x smap state (smap absc x) for x y]
  using X-Y-closed[of smap state (smap absc x) smap state x for x y]
  by (auto simp: * stream.rel-eq stream.map-comp state-absc)+
done
subgoal
  apply (auto intro!: rel-funI)
  apply (subst K-cfg-map-absc)
  defer
  apply (subst pmf.rel-map(2))
  apply (rule rel-pmf-reflI)
  by auto
subgoal
  using valid unfolding cfg'-def by simp
done
qed

```

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-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:  $\bigwedge x y. P x \implies \text{smap abss } y = x \implies x \in X \implies y \in Y \wedge Q y$ 
assumes Y-X-closed:  $\bigwedge x y. Q y \implies \text{smap abss } y = x \implies y \in Y \implies x \in X \wedge P x$ 
shows
  emeasure (R-G.T cfg) X = emeasure (MDP.T cfg') Y
proof -
  have *: stream-all2 ( $\lambda s t. t = \text{absc } s$ ) x y = stream-all2 (=) y (smap absc x) for x y
  using stream.rel-conversep[of  $\lambda s t. t = \text{absc } s$ ]
  by (simp add: conversep-iff[abs-def])
  from P X have
    emeasure (R-G.T cfg) X = emeasure (R-G.T cfg) {x  $\in$  X. P x}
  by (auto intro: emeasure-eq-AE)
  moreover from Q Y have
    emeasure (MDP.T cfg') Y = emeasure (MDP.T cfg') {y  $\in$  Y. Q y}
  by (auto intro: emeasure-eq-AE)
  moreover show ?thesis
  apply (simp only: calculation)
  unfolding R-G.T-def MDP.T-def
  apply (simp add: emeasure-distr)
  apply (rule sym)
  apply (rule T-eq-rel-half[where f = absc and S = valid-cfg])
  apply (rule HOL.refl)
  apply measurable
  apply (simp add: space-stream-space)
subgoal
  unfolding rel-set-strong-def stream.rel-eq
  apply (intro allI impI)
  apply (drule stream.rel-mono-strong[where Ra =  $\lambda s t. t = \text{absc } s$ ])
  apply (simp; fail)

```

```

subgoal for  $x y$ 
  using  $Y$ - $X$ -closed[of  $\text{smap state } x \text{ smap state } (\text{smap absc } x)$  for  $x y$ ]
  using  $X$ - $Y$ -closed[of  $\text{smap state } (\text{smap absc } x) \text{ smap state } x$  for  $x y$ ]
  by (auto simp: * stream.rel-eq stream.map-comp state-absc)+
done
subgoal
  apply (auto intro!: rel-funI)
  apply (subst  $K$ -cfg-map-absc)
  defer
  apply (subst pmf.rel-map(2))
  apply (rule rel-pmf-reflI)
  by auto
subgoal
  using valid unfolding  $\text{cfg}'$ -def by (auto simp:  $s$  absc-repcs-id)
done
qed

lemma region-compatible-suntil1:
  assumes (holds  $(\lambda x. \varphi (\text{reps } x))$  until holds  $(\lambda x. \psi (\text{reps } x))$ ) ( $\text{smap abss } x$ )
    and  $\text{pred-stream } (\lambda s. \varphi (\text{reps } (\text{abss } s)) \longrightarrow \varphi s) x$ 
    and  $\text{pred-stream } (\lambda s. \psi (\text{reps } (\text{abss } s)) \longrightarrow \psi s) x$ 
  shows (holds  $\varphi$  until holds  $\psi$ )  $x$  using  $\text{assms}$ 
proof (induction  $\text{smap abss } x$  arbitrary:  $x$  rule: until.induct)
  case base
  then show ?case by (auto intro: until.base simp: stream.pred-set)
next
  case step
  have
     $\text{pred-stream } (\lambda s. \varphi (\text{reps } (\text{abss } s)) \longrightarrow \varphi s) (\text{stl } x)$ 
     $\text{pred-stream } (\lambda s. \psi (\text{reps } (\text{abss } s)) \longrightarrow \psi s) (\text{stl } x)$ 
    using step.prem1 apply (cases  $x$ ; auto)
    using step.prem2 apply (cases  $x$ ; auto)
  done
  with step.hyps(3)[of  $\text{stl } x$ ] have (holds  $\varphi$  until holds  $\psi$ ) ( $\text{stl } x$ ) by auto
  with step.prem3 step.hyps(1-2) show ?case by (auto intro: until.step simp: stream.pred-set)
qed

lemma region-compatible-suntil2:
  assumes (holds  $\varphi$  until holds  $\psi$ )  $x$ 
    and  $\text{pred-stream } (\lambda s. \varphi s \longrightarrow \varphi (\text{reps } (\text{abss } s))) x$ 
    and  $\text{pred-stream } (\lambda s. \psi s \longrightarrow \psi (\text{reps } (\text{abss } s))) x$ 
  shows (holds  $(\lambda x. \varphi (\text{reps } x))$  until holds  $(\lambda x. \psi (\text{reps } x))$ ) ( $\text{smap abss } x$ ) using  $\text{assms}$ 
proof (induction  $x$  rule: until.induct)
  case (base  $x$ )
  then show ?case by (auto intro: until.base simp: stream.pred-set)
next
  case (step  $x$ )
  have
     $\text{pred-stream } (\lambda s. \varphi s \longrightarrow \varphi (\text{reps } (\text{abss } s))) (\text{stl } x)$ 
     $\text{pred-stream } (\lambda s. \psi s \longrightarrow \psi (\text{reps } (\text{abss } s))) (\text{stl } x)$ 
    using step.prem1 apply (cases  $x$ ; auto)
    using step.prem2 apply (cases  $x$ ; auto)
  done
  with step show ?case by (auto intro: until.step simp: stream.pred-set)
qed

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

```

using *assms region-compatible-suntil1 region-compatible-suntil2* **unfolding** *stream.pred-set* **by** *blast*

lemma *reps-abss-S*:

assumes *reps* (*abss s*) $\in S$
shows *s* $\in S$

by (*simp add: S-reps-S S-abss-S assms*)

lemma *measurable-sset*[*measurable (raw)*]:

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

proof –

have *: $(\lambda x. \forall s \in \text{sset} (f x). P s) = (\lambda x. \forall i. P (f x !! i))$
by (*simp add: sset-range*)

show *?thesis*

unfolding * **by** *measurable*

qed

lemma *path-measure-eq-repcs''-new*:

notes *in-space-UNIV*[*measurable*]

fixes *cfg* φ ψ *s*

defines *cfg'* \equiv *repcs s cfg*

defines $\varphi' \equiv$ *absp* φ **and** $\psi' \equiv$ *absp* ψ

assumes *s*: *abss s* = *state cfg*

assumes *valid*: *cfg* $\in R\text{-G.valid-cfg}$

assumes *valid'*: *cfg'* \in *valid-cfg*

assumes *equiv- φ* : $\bigwedge x. \text{pred-stream } (\lambda s. s \in S) x$

$\implies \text{pred-stream } (\lambda s. \varphi (\text{reps } (\text{abss } s))) \longleftrightarrow \varphi s$ (*state cfg' ## x*)

and *equiv- ψ* : $\bigwedge x. \text{pred-stream } (\lambda s. s \in S) x$

$\implies \text{pred-stream } (\lambda s. \psi (\text{reps } (\text{abss } s))) \longleftrightarrow \psi s$ (*state cfg' ## x*)

shows

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

unfolding *cfg'-def*

apply (*rule path-measure-eq-repcs1-new*[**where** $P = \text{pred-stream } (\lambda s. s \in S)$ **and** $Q = \text{pred-stream } (\lambda s. s \in S)$])

apply *fact*

apply *fact*

apply *measurable*

subgoal

unfolding *R-G.T-def*

apply (*subst AE-distr-iff*)

apply (*auto*; *fail*)

apply (*auto simp: stream.pred-set*; *fail*)

apply (*rule AE-mp*[*OF MDP.MC.AE-T-enabled AE-I2*])

using *R-G.pred-stream-cfg-on*[*OF valid*] **by** (*auto simp: stream.pred-set*)

subgoal

unfolding *MDP.T-def*

apply (*subst AE-distr-iff*)

apply (*auto*; *fail*)

apply (*auto simp: stream.pred-set*; *fail*)

apply (*rule AE-mp*[*OF MDP.MC.AE-T-enabled AE-I2*])

using *MDP.pred-stream-cfg-on*[*OF valid'*, *unfolded cfg'-def*] **by** (*auto simp: stream.pred-set*)

apply *measurable*

subgoal **premises** *prems* **for** *ys xs*

apply *safe*

apply *measurable*

unfolding φ' -*def* ψ' -*def* *absp-def*

apply (*subst region-compatible-suntil*[*symmetric*])

subgoal

proof –

from *prems* **have** *pred-stream* $(\lambda s. s \in S) xs$ **using** *S-abss-S* **by** (*auto simp: stream.pred-set*)

```

  with equiv-φ show ?thesis by (simp add: cfg'-def)
qed
subgoal
proof -
  from prems have pred-stream (λs. s ∈ S) xs using S-abss-S by (auto simp: stream.pred-set)
  with equiv-ψ show ?thesis by (simp add: cfg'-def)
qed
using valid prems
  apply (auto simp: s comp-def φ'-def ψ'-def absp-def dest: R-G.valid-cfg-state-in-S)
  apply (auto simp: stream.pred-set intro: S-abss-S dest: R-G.valid-cfg-state-in-S)
done
subgoal premises prems for ys xs
  apply safe
  using prems apply (auto simp: stream.pred-set S-abss-S; measurable; fail)
  using prems unfolding φ'-def ψ'-def absp-def comp-def apply (simp add: stream.map-comp)
  apply (subst (asm) region-compatible-suntil[symmetric])
subgoal
proof -
  from prems have pred-stream (λs. s ∈ S) xs using S-abss-S by auto
  with equiv-φ show ?thesis using valid by (simp add: cfg'-def repc-def)
qed
subgoal
proof -
  from prems have pred-stream (λs. s ∈ S) xs using S-abss-S by auto
  with equiv-ψ show ?thesis using valid by (simp add: cfg'-def)
qed
using valid prems by (auto simp: s S-abss-S stream.pred-set dest: R-G.valid-cfg-state-in-S)
done
end

end
theory PTA-Reachability
  imports PTA
begin

```

6 Classifying Regions for Divergence

6.1 Pairwise

coinductive *pairwise* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a stream ⇒ bool **for** *P* **where**
 $P\ a\ b \implies \text{pairwise}\ P\ (b\ \#\#\ xs) \implies \text{pairwise}\ P\ (a\ \#\#\ b\ \#\#\ xs)$

lemma *pairwise-Suc*:

$\text{pairwise}\ P\ xs \implies P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i))$
by (*induction* *i* *arbitrary: xs*) (*force elim: pairwise.cases*)+

lemma *Suc-pairwise*:

$\forall\ i.\ P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i)) \implies \text{pairwise}\ P\ xs$
apply (*coinduction arbitrary: xs*)
apply (*subst stream.collapse[symmetric]*)
apply (*rewrite in stl - stream.collapse[symmetric]*)
apply (*intro exI conjI, rule HOL.refl*)
apply (*erule allE[where x = 0]; simp; fail*)
by *simp (metis snth.simps(2))*

lemma *pairwise-iff*:

$\text{pairwise}\ P\ xs \longleftrightarrow (\forall\ i.\ P\ (xs\ !!\ i)\ (xs\ !!\ (Suc\ i)))$
using *pairwise-Suc Suc-pairwise* **by** *blast*

lemma *pairwise-stlD*:

pairwise P xs \implies pairwise P (stl xs)

by (*auto elim: pairwise.cases*)

lemma *pairwise-pairD*:

pairwise P xs \implies P (shd xs) (shd (stl xs))

by (*auto elim: pairwise.cases*)

lemma *pairwise-mp*:

assumes *pairwise P xs* **and** *lift: $\bigwedge x y. x \in \text{sset } xs \implies y \in \text{sset } xs \implies P x y \implies Q x y$*

shows *pairwise Q xs* **using** *assms*

apply (*coinduction arbitrary: xs*)

subgoal for *xs*

apply (*subst stream.collapse[symmetric]*)

apply (*rewrite in stl - stream.collapse[symmetric]*)

apply (*intro exI conjI*)

apply (*rule HOL.refl*)

by (*auto intro: stl-sset dest: pairwise-pairD pairwise-stlD*)

done

lemma *pairwise-sdropD*:

pairwise P (sdrop i xs) **if** *pairwise P xs*

using *that*

proof (*coinduction arbitrary: i xs*)

case (*pairwise i xs*)

then show *?case*

apply (*inst-existentials shd (sdrop i xs) shd (stl (sdrop i xs)) stl (stl (sdrop i xs))*)

subgoal

by (*auto dest: pairwise-Suc*) (*metis sdrop-simps(1) sdrop-stl stream.collapse*)

subgoal

by (*inst-existentials i - 1 stl xs*) (*auto dest: pairwise-Suc pairwise-stlD*)

by (*metis sdrop-simps(2) stream.collapse*)

qed

6.2 Regions

lemma *gt-GreaterD*:

assumes *u \in region X I r valid-region X k I r c \in X u c $>$ k c*

shows *I c = Greater (k c)*

proof –

from *assms* **have** *intv-elem c u (I c) valid-intv (k c) (I c)* **by** *auto*

with *assms(4)* **show** *?thesis* **by** (*cases I c*) *auto*

qed

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 –

from *assms* **have** *intv-elem c u (I c) valid-intv (k c) (I c)* **by** *auto*

with *assms(4,5)* **show** *?thesis* **by** (*cases I c*) *auto*

qed

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 –

from *assms* **have** *intv-elem x u (I x) valid-intv (k x) (I x)* **by** *auto*

with *assms(1)* **show** *u x \leq k x* **by** (*cases I x*) *auto*

qed

lemma *Greater-closed*:

```

fixes  $t :: \text{real}$ 
assumes  $u \in \text{region } X \text{ I } r \text{ valid-region } X \text{ k I } r \text{ c} \in X \text{ I } c = \text{Greater } (k \text{ c}) \text{ t} > k \text{ c}$ 
shows  $u(c := t) \in \text{region } X \text{ I } r$ 
using assms
apply (intro region.intros)
  apply (auto; fail)
  apply standard
subgoal for  $x$ 
  by (cases x = c; cases I x; force intro!: intv-elem.intros)
by auto

```

```

lemma Greater-unbounded-aux:
  assumes finite X valid-region X k I r c  $\in X \text{ I } c = \text{Greater } (k \text{ c})$ 
  shows  $\exists u \in \text{region } X \text{ I } r. u \text{ c} > t$ 
using assms Greater-closed[OF - assms(2-4)]
proof -
  let  $?R = \text{region } X \text{ I } r$ 
  let  $?t = \text{if } t > k \text{ c} \text{ then } t + 1 \text{ else } k \text{ c} + 1$ 
  have  $t: ?t > k \text{ c}$  by auto
  from region-not-empty[OF assms(1,2)] obtain  $u$  where  $u: u \in ?R$  by auto
  from Greater-closed[OF this assms(2-4) t] have  $u(c:=?t) \in ?R$  by auto
  with  $t$  show ?thesis by (inst-existentials u(c:=?t)) auto
qed

```

6.3 Unbounded and Zero Regions

```

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

```

```

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

```

```

lemma Greater-unbounded:
  assumes finite X valid-region X k I r c  $\in X \text{ I } c = \text{Greater } (k \text{ c})$ 
  shows unbounded c (region X I r)
using Greater-unbounded-aux[OF assms] unfolding unbounded-def by blast

```

```

lemma unbounded-Greater:
  assumes valid-region X k I r c  $\in X$  unbounded c (region X I r)
  shows  $I \text{ c} = \text{Greater } (k \text{ c})$ 
using assms unfolding unbounded-def by (auto intro: gt-GreaterD)

```

```

lemma Const-zero:
  assumes  $c \in X \text{ I } c = \text{Const } 0$ 
  shows zero c (region X I r)
using assms unfolding zero-def by force

```

```

lemma zero-Const:
  assumes finite X valid-region X k I r c  $\in X$  zero c (region X I r)
  shows  $I \text{ c} = \text{Const } 0$ 
proof -
  from assms obtain  $u$  where  $u \in \text{region } X \text{ I } r$  by atomize-elim (auto intro: region-not-empty)
  with assms show ?thesis unfolding zero-def by (auto intro: const-ConstD)
qed

```

```

lemma zero-all:
  assumes finite X valid-region X k I r c  $\in X$   $u \in \text{region } X \text{ I } r$   $u \text{ c} = 0$ 
  shows zero c (region X I r)
proof -
  from assms have intv-elem c u (I c) valid-intv (k c) (I c) by auto
  then have  $I \text{ c} = \text{Const } 0$  using assms(5) by cases auto
  with assms have  $u' \text{ c} = 0$  if  $u' \in \text{region } X \text{ I } r$  for  $u'$  using that by force
  then show ?thesis unfolding zero-def by blast

```


qed

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

begin

definition $\varphi' \equiv \text{absp } \varphi$

definition $\psi' \equiv \text{absp } \psi$

definition $s' \equiv \text{abss } s$

lemma *s-s'-cfg-on*[*intro*]:

assumes *cfg* \in *MDP.cfg-on s*
shows *absc cfg* \in *R-G.cfg-on s'*

proof –

from *assms s* **have** *cfg* \in *valid-cfg* **unfolding** *MDP.valid-cfg-def* **by** *auto*
then **have** *absc cfg* \in *R-G.cfg-on* (*state (absc cfg)*) **by** (*auto intro: R-G.valid-cfgD*)
with *assms* **show** *?thesis* **unfolding** *s'-def* **by** (*auto simp: state-absc*)

qed

lemma *s'-S*[*simp*, *intro*]:

$s' \in S$
unfolding *s'-def* **using** *s* **by** *auto*

lemma *s'-s-cfg-on*[*intro*]:

assumes *cfg* \in *R-G.cfg-on s'*
shows *repcs s cfg* \in *MDP.cfg-on s*

proof –

from *assms s* **have** *cfg* \in *R-G.valid-cfg* **unfolding** *R-G.valid-cfg-def* **by** *auto*
with *assms* **have** *repcs s cfg* \in *valid-cfg* **by** (*auto simp: s'-def intro: R-G.valid-cfgD*)
then **show** *?thesis* **by** (*auto dest: MDP.valid-cfgD*)

qed

lemma (**in** *Probabilistic-Timed-Automaton-Regions*) *compatible-stream*:

assumes φ : $\bigwedge x y. x \in S \Rightarrow x \sim y \Rightarrow \varphi x \longleftrightarrow \varphi y$
assumes *pred-stream* ($\lambda s. s \in S$) *xs*
and [*intro*]: $x \in S$
shows *pred-stream* ($\lambda s. \varphi$ (*reps* (*abss s*)) = φs) ($x \## xs$)

unfolding *stream.pred-set* **proof** *clarify*

fix *l u*

assume *A*: (l, u) \in *sset* ($x \## xs$)

from *assms* **have** *pred-stream* ($\lambda s. s \in S$) ($x \## xs$) **by** *auto*

with *A* **have** (l, u) \in *S* **by** (*fastforce simp: stream.pred-set*)

then **have** *abss* (l, u) \in *S* **by** *auto*

then **have** *reps* (*abss* (l, u)) \sim (l, u) **by** *simp*

with φ $\langle l, u \in S \rangle$ **show** φ (*reps* (*abss* (l, u))) = φ (l, u) **by** *blast*

qed

lemma *φ-stream'*:

pred-stream ($\lambda s. \varphi$ (*reps* (*abss s*)) = φs) ($x \## xs$) **if** *pred-stream* ($\lambda s. s \in S$) *xs* $x \in S$
using *compatible-stream*[*of* φ , *OF* φ *that*].

lemma ψ -stream':

$\text{pred-stream } (\lambda s. \psi (\text{reps } (\text{abss } s)) = \psi s) (x \#\# xs)$ **if** $\text{pred-stream } (\lambda s. s \in S) xs x \in S$
using $\text{compatible-stream}[of \psi, OF \psi \text{ that}]$.

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

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

7.2 Easier Result on All Configurations

lemma suntil-reps :

assumes

$\forall s \in \text{set } (\text{smap } \text{abss } y). s \in S$
 $(\text{holds } \varphi' \text{ suntil holds } \psi') (s' \#\# \text{smap } \text{abss } y)$

shows $(\text{holds } \varphi \text{ suntil holds } \psi) (s \#\# y)$

using assms

by $(\text{subst } \text{region-compatible-suntil}[\text{symmetric}]; (\text{intro } \varphi\text{-stream } \psi\text{-stream})?)$
 $(\text{auto simp: } \varphi'\text{-def } \psi'\text{-def } \text{absp-def } \text{stream.pred-set } S\text{-abss-}S \text{ } s'\text{-def } \text{comp-def})$

lemma suntil-abss :

assumes

$\forall s \in \text{set } y. s \in S$
 $(\text{holds } \varphi \text{ suntil holds } \psi) (s \#\# y)$

shows

$(\text{holds } \varphi' \text{ suntil holds } \psi') (s' \#\# \text{smap } \text{abss } y)$

using assms

by $(\text{subst } (\text{asm } \text{region-compatible-suntil}[\text{symmetric}]; (\text{intro } \varphi\text{-stream } \psi\text{-stream})?)$
 $(\text{auto simp: } \varphi'\text{-def } \psi'\text{-def } \text{absp-def } \text{stream.pred-set } s'\text{-def } \text{comp-def})$

theorem P -sup-suntil-eq:

notes $[\text{measurable}] = \text{in-space-UNIV}$ **and** $[\text{iff}] = \text{pred-stream-iff}$

shows

$(\text{MDP.P-sup } s (\lambda x. (\text{holds } \varphi \text{ suntil holds } \psi) (s \#\# x)))$
 $= (\text{R-G.P-sup } s' (\lambda x. (\text{holds } \varphi' \text{ suntil holds } \psi') (s' \#\# x)))$

unfolding $\text{MDP.P-sup-def } \text{R-G.P-sup-def}$

proof $(\text{rule } \text{SUP-eq, goal-cases})$

case $\text{prems: } (1 \text{ cfg})$

let $?cfg' = \text{absc } \text{cfg}$

from prems **have** $\text{cfg} \in \text{valid-cfg}$ **by** $(\text{auto intro: } \text{MDP.valid-cfgI})$

then **have** $?cfg' \in \text{R-G.valid-cfg}$ **by** $(\text{auto intro: } \text{R-G.valid-cfgI})$

from $\langle \text{cfg} \in \text{valid-cfg} \rangle$ **have** $\text{alw-}S$: $\text{almost-everywhere } (\text{MDP.T } \text{cfg}) (\text{pred-stream } (\lambda s. s \in S))$

by $(\text{rule } \text{MDP.alw-}S)$

from $\langle ?cfg' \in \text{R-G.valid-cfg} \rangle$ **have** $\text{alw-}S$: $\text{almost-everywhere } (\text{R-G.T } ?\text{cfg}') (\text{pred-stream } (\lambda s. s \in S))$

by $(\text{rule } \text{R-G.alw-}S)$

have $\text{emeasure } (\text{MDP.T } \text{cfg}) \{x \in \text{space } \text{MDP.St. } (\text{holds } \varphi \text{ suntil holds } \psi) (s \#\# x)\}$

$= \text{emeasure } (\text{R-G.T } ?\text{cfg}') \{x \in \text{space } \text{R-G.St. } (\text{holds } \varphi' \text{ suntil holds } \psi') (s' \#\# x)\}$

apply $(\text{rule } \text{path-measure-eq-absc1-new}[\text{symmetric, where } P = \text{pred-stream } (\lambda s. s \in S)])$

and $Q = \text{pred-stream } (\lambda s. s \in S)$

)

using $\text{prems } \text{alw-}S \text{ } \text{alw-}S$ **apply** $(\text{auto intro: } \text{MDP.valid-cfgI } \text{simp: })[\gamma]$

by $(\text{auto simp: } S\text{-abss-}S \text{ intro: } S\text{-abss-}S \text{ intro!: } \text{suntil-abss } \text{suntil-reps, measurable})$

with prems **show** $?case$ **by** $(\text{inst-existentials } ?\text{cfg}') \text{ auto}$

next

case $\text{prems: } (2 \text{ cfg})$

let $?cfg' = \text{repcs } s \text{ } \text{cfg}$

have $s = \text{state } ?\text{cfg}'$ **by** simp

from prems **have** $s' = \text{state } \text{cfg}$ **by** auto

have $\text{pred-stream } (\lambda s. \varphi (\text{reps } (\text{abss } s)) = \varphi s) (\text{state } (\text{repcs } s \text{ } \text{cfg}) \#\# x)$

if $\text{pred-stream } (\lambda s. s \in S) x$ **for** x

using prems **that** **by** $(\text{intro } \varphi\text{-stream}) \text{ auto}$

moreover
have $\text{pred-stream } (\lambda s. \psi (\text{reps } (\text{abss } s)) = \psi s) (\text{state } (\text{reps } s \text{ cfg}) \#\# x)$
if $\text{pred-stream } (\lambda s. s \in S) x$ **for** x
using *prems that* **by** (*intro* $\psi\text{-stream}$) *auto*
ultimately
have $\text{emeasure } (R\text{-}G.T \text{ cfg}) \{x \in \text{space } R\text{-}G.St. (\text{holds } \varphi' \text{ until holds } \psi') (s' \#\# x)\}$
 $= \text{emeasure } (MDP.T (\text{reps } s \text{ cfg})) \{x \in \text{space } MDP.St. (\text{holds } \varphi \text{ until holds } \psi) (s \#\# x)\}$
apply (*rewrite in* $s \#\# - \langle s = - \rangle$)
apply (*subst* $\langle s' = - \rangle$)
unfolding $\varphi'\text{-def } \psi'\text{-def } s'\text{-def}$
apply (*rule path-measure-eq-reps''-new*)
using *prems by* (*auto* $\& \& \text{simp: } s'\text{-def intro: } R\text{-}G.\text{valid-cfgI } MDP.\text{valid-cfgI}$)
with *prems show* $?case$ **by** (*inst-existentials* $?cfg'$) *auto*
qed
end

7.3 Divergent Adversaries

context *Probabilistic-Timed-Automaton*
begin

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

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

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

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

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

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

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

end

context *Probabilistic-Timed-Automaton-Regions*
begin

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

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

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

unfolding $\mathcal{R}\text{-div-def}$

by (*intro*
 $\text{pred-intros-finite}[OF \text{ beta-interp.finite}]$
 $\text{pred-intros-logic } \text{pred-intros-countable}$
 $\text{measurable-count-space-const measurable-compose}[OF \text{ measurable-snth}]$
 $) \text{measurable}$

lemma *elapsed-ge0[simp]: elapsed x y ≥ 0*

unfolding *elapsed-def using finite(1) by auto*

lemma *dur-pos:*

```

dur xs i ≥ 0
apply (induction i arbitrary: xs)
apply (auto; fail)
subgoal for i xs
  apply (subst stream.collapse[symmetric])
  apply (rewrite at stl xs stream.collapse[symmetric])
  apply (subst dur.simps)
by simp
done

```

```

lemma dur-mono:
   $i \leq j \implies \text{dur } xs \ i \leq \text{dur } xs \ j$ 
proof (induction i arbitrary: xs j)
  case 0 show ?case by (auto intro: dur-pos)
next
  case (Suc i xs j)
  obtain x y ys where xs: xs = x ## y ## ys using stream.collapse by metis
  from Suc obtain j' where j': j = Suc j' by (cases j) auto
  with xs have dur xs j = elapsed x y + dur (y ## ys) j' by auto
  also from Suc j' have ... ≥ elapsed x y + dur (y ## ys) i by auto
  also have elapsed x y + dur (y ## ys) i = dur xs (Suc i) by (simp add: xs)
  finally show ?case .
qed

```

```

lemma dur-monoD:
  assumes dur xs i < dur xs j
  shows i < j using assms
by - (rule ccontr; auto 4 4 dest: leI dur-mono[where xs = xs])

```

```

lemma elapsed-0D:
  assumes c ∈ X elapsed u u' ≤ 0
  shows u' c - u c ≤ 0
proof -
  from assms have u' c - u c ∈ {u' c - u c | c. c ∈ X} ∪ {0} by auto
  with finite(1) have u' c - u c ≤ Max ({u' c - u c | c. c ∈ X} ∪ {0}) by auto
  with assms(2) show ?thesis unfolding elapsed-def by auto
qed

```

```

lemma elapsed-ge:
  assumes eq-elapsed u u' c ∈ X
  shows elapsed u u' ≥ u' c - u c
  using assms unfolding eq-elapsed-def by (auto intro: elapsed-ge0 order.trans[OF elapsed-0D])

```

```

lemma elapsed-eq:
  assumes eq-elapsed u u' c ∈ X u' c - u c ≥ 0
  shows elapsed u u' = u' c - u c
  using elapsed-ge[OF assms(1,2)] assms unfolding eq-elapsed-def by auto

```

```

lemma dur-shift:
   $\text{dur } \omega \ (i + j) = \text{dur } \omega \ i + \text{dur} \ (\text{sdrop } i \ \omega) \ j$ 
apply (induction i arbitrary: ω)
apply simp
subgoal for i ω
  apply simp
  apply (subst stream.collapse[symmetric])
  apply (rewrite at stl ω stream.collapse[symmetric])
  apply (subst dur.simps)
  apply (rewrite in dur ω stream.collapse[symmetric])
  apply (rewrite in dur (- ## □) (Suc -) stream.collapse[symmetric])
  apply (subst dur.simps)
apply simp

```

done
done

lemma *dur-zero*:

assumes
 $\forall i. xs !! i \in \omega !! i \forall j \leq i. zero\ x\ (\omega !! j)\ x \in \mathcal{X}$
 $\forall i. eq\text{-elapsd}\ (xs !! i)\ (xs !! Suc\ i)$
shows $dur\ xs\ i = 0$ using *assms*
proof (induction *i* arbitrary: $xs\ \omega$)
 case 0
 then show *?case* by *simp*
next
 case (Suc *i xs ω*)
 let *?x* = $xs !! 0$
 let *?y* = $xs !! 1$
 let *?ys* = $stl\ (stl\ xs)$
 have $xs: xs = ?x\ \#\#\ ?y\ \#\#\ ?ys$ by *auto*
 from *Suc.prem*s have
 $\forall i. (?y\ \#\#\ ?ys) !! i \in stl\ \omega !! i \forall j \leq i. zero\ x\ (stl\ \omega !! j)$
 $\forall i. eq\text{-elapsd}\ (stl\ xs !! i)\ (stl\ xs !! Suc\ i)$
 by (metis *snth.simps*(2) | *auto*)
 from *Suc.IH*[*OF* *this*(1,2) $\langle x \in \cdot \rangle$] *this*(3) have [*simp*]: $dur\ (stl\ xs)\ i = 0$ by *auto*
 from *Suc.prem*s(1,2) have $?y\ x = 0\ ?x\ x = 0$ unfolding *zero-def* by *force+*
 then have $*$: $?y\ x - ?x\ x = 0$ by *simp*
 have $dur\ xs\ (Suc\ i) = elapsed\ ?x\ ?y$
 apply (subst *xs*)
 apply (subst *dur.simps*)
 by *simp*
 also have $\dots = 0$
 apply (subst *elapsed-eq*[*OF* - $\langle x \in \cdot \rangle$])
 unfolding *One-nat-def* using *Suc.prem*s(4) apply *blast*
 using $*$ by *auto*
 finally show *?case* .
qed

lemma *dur-zero-tail*:

assumes $\forall i. xs !! i \in \omega !! i \forall k \geq i. k \leq j \longrightarrow zero\ x\ (\omega !! k)\ x \in \mathcal{X}\ j \geq i$
 $\forall i. eq\text{-elapsd}\ (xs !! i)\ (xs !! Suc\ i)$
shows $dur\ xs\ j = dur\ xs\ i$
proof -
 from $\langle j \geq i \rangle$ *dur-shift*[*of* $xs\ i\ j - i$] have
 $dur\ xs\ j = dur\ xs\ i + dur\ (sdrop\ i\ xs)\ (j - i)$
 by *simp*
 also have $\dots = dur\ xs\ i$
 using *assms*
 by (rewrite in $dur\ (sdrop\ -\ -) - dur\ zero$ [where $\omega = sdrop\ i\ \omega$])
 (*auto dest: prop-nth-sdrop-pair*[*of* *eq-elapsd*] *prop-nth-sdrop prop-nth-sdrop-pair*[*of* (\in)])
 finally show *?thesis* .
qed

lemma *elapsed-ge-pos*:

fixes $u :: ('c, t)\ cval$
assumes $eq\text{-elapsd}\ u\ u'\ c \in \mathcal{X}\ u \in V\ u' \in V$
shows $elapsed\ u\ u' \leq u'\ c$
proof (cases $elapsed\ u\ u' = 0$)
 case True
 with *assms* show *?thesis* by (auto simp: *V-def*)
next
 case False
 from $\langle u \in V \rangle \langle c \in \mathcal{X} \rangle$ have $u\ c \geq 0$ by (auto simp: *V-def*)
 from False *assms* have $elapsed\ u\ u' = u'\ c - u\ c$

unfolding *eq-elapsed-def* **by** (*auto simp add: less-le*)
also from $\langle u \ c \geq 0 \rangle$ **have** $\dots \leq u' \ c$ **by** *simp*
finally show *?thesis* .
qed

lemma *dur-Suc*:

dur xs (Suc i) - dur xs i = elapsed (xs !! i) (xs !! Suc i)
apply (*induction i arbitrary: xs*)
apply *simp*
apply (*subst stream.collapse[symmetric]*)
apply (*rewrite in stl - stream.collapse[symmetric]*)
apply (*subst dur.simps*)
apply *simp*
apply *simp*
subgoal for *i xs*
apply (*subst stream.collapse[symmetric]*)
apply (*rewrite in stl - stream.collapse[symmetric]*)
apply (*subst dur.simps*)
apply *simp*
apply (*rewrite in dur xs (Suc -) stream.collapse[symmetric]*)
apply (*rewrite at stl xs in - ## stl xs stream.collapse[symmetric]*)
apply (*subst dur.simps*)
apply *simp*
done
done

inductive trans where

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

abbreviation *stream-trans* \equiv *pairwise trans*

lemma *K-cfg-trans*:

assumes *cfg* \in *MDP.cfg-on* (*l, R*) *cfg'* \in *K-cfg cfg state* *cfg' = (l', R')*
shows *trans R R'*
using *assms*
apply (*simp add: set-K-cfg*)
apply (*drule MDP.cfg-onD-action*)
apply (*cases rule: K.cases*)
apply (*auto intro: trans.intros*)
using *admissible-targets-clocks(2)* **by** (*blast intro: trans.intros(2)*)

lemma *enabled-stream-trans*:

assumes *cfg* \in *valid-cfg MDP.MC.enabled* *cfg xs*
shows *stream-trans (smap (snd o state) xs)*
using *assms*
proof (*coinduction arbitrary: cfg xs*)
case *prems: (pairwise cfg xs)*
let *?xs = stl (stl xs)* **let** *?x = shd xs* **let** *?y = shd (stl xs)*
from *MDP.pred-stream-cfg-on[OF prems]* **have** $*$:
*pred-stream ($\lambda \text{cfg. state } \text{cfg} \in S \wedge \text{cfg} \in \text{MDP.cfg-on (state } \text{cfg})$) *xs* .*
obtain *l R l' R'* **where** *eq: state ?x = (l, R) state ?y = (l', R')* **by** *force*
moreover from $*$ **have** *?x* \in *MDP.cfg-on (state ?x)* *?x* \in *valid-cfg*
by (*auto intro: MDP.valid-cfgI simp: stream.pred-set*)
moreover from *prems(2)* **have** *?y* \in *K-cfg ?x* **by** (*auto elim: MDP.MC.enabled.cases*)
ultimately have *trans R R'*
by (*intro K-cfg-trans[where $\text{cfg} = ?x$ and $\text{cfg}' = ?y$ and $l = l$ and $l' = l'$]*) *metis+*
with $\langle ?x \in \text{valid-cfg} \rangle$ *prems(2)* **show** *?case*
apply (*inst-existentials R R' smap (snd o state) ?xs*)
apply (*simp add: eq; fail*) $+$

```

  apply (rule disjI1, inst-existentials ?x stl xs)
  by (auto simp: eq elim: MDP.MC.enabled.cases)
qed

```

```

lemma stream-trans-trans:
  assumes stream-trans xs
  shows trans (xs !! i) (stl xs !! i)
using pairwise-Suc assms by auto

```

```

lemma trans-eq-elapsed:
  assumes trans u u' u ∈ V
  shows eq-elapsed u u'
using assms
proof cases
  case (succ t)
  with finite(1) show ?thesis by (auto simp: cval-add-def elapsed-def max-def eq-elapsed-def)
next
  case prems: (reset l)
  then have u' c - u c ≤ 0 if c ∈ X for c
  using that ⟨u ∈ V⟩ by (cases c ∈ set l) (auto simp: V-def)
  then have elapsed u u' = 0 unfolding elapsed-def using finite(1)
  apply simp
  apply (subst Max-insert2)
  by auto
  then show ?thesis by (auto simp: eq-elapsed-def)
next
  case id
  then show ?thesis
  using finite(1) by (auto simp: Max-gr-iff elapsed-def eq-elapsed-def)
qed

```

```

lemma pairwise-trans-eq-elapsed:
  assumes stream-trans xs pred-stream (λ u. u ∈ V) xs
  shows pairwise eq-elapsed xs
using trans-eq-elapsed assms by (auto intro: pairwise-mp simp: stream.pred-set)

```

```

lemma not-reset-dur:
  assumes ∀ k > i. k ≤ j → ¬ zero c ([xs !! k]R) j ≥ i c ∈ X stream-trans xs
  ∨ i. eq-elapsed (xs !! i) (xs !! Suc i) ∨ i. xs !! i ∈ V
  shows dur xs j - dur xs i = (xs !! j) c - (xs !! i) c
  using assms
proof (induction j)
  case 0 then show ?case by simp
next
  case (Suc j)
  from stream-trans-trans[OF Suc.prems(4)] have trans: trans (xs !! j) (xs !! Suc j) by auto
  from Suc.prems have *:
    ¬ zero c ([xs !! Suc j]R) eq-elapsed (xs !! j) (xs !! Suc j) if Suc j > i
  using that by auto
  from Suc.prems(6) have xs !! j ∈ V xs !! Suc j ∈ V by blast+
  then have regions: [xs !! j]R ∈ R [xs !! Suc j]R ∈ R by auto
  from trans have (xs !! Suc j) c - (xs !! j) c ≥ 0 if Suc j > i
  proof (cases)
    case succ
    with regions show ?thesis by (auto simp: cval-add-def)
  next
    case prems: (reset l)
    show ?thesis
  proof (cases c ∈ set l)
    case False
    with prems show ?thesis by auto

```

```

next
  case True
  with prems have (xs !! Suc j) c = 0 by auto
  moreover from assms have xs !! Suc j ∈ [xs !! Suc j]R by blast
  ultimately have
    zero c ([xs !! Suc j]R)
    using zero-all[OF finite(1) - ⟨c ∈ X⟩] regions(2) by (auto simp: R-def)
  with * that show ?thesis by auto
qed
next
  case id then show ?thesis by simp
qed
with * ⟨c ∈ X⟩ elapsed-eq have
  *: elapsed (xs !! j) (xs !! Suc j) = (xs !! Suc j) c - (xs !! j) c
  if Suc j > i
  using that by blast
show ?case
proof (cases i = Suc j)
  case False
  with Suc have
    dur xs (Suc j) - dur xs i = dur xs (Suc j) - dur xs j + (xs !! j) c - (xs !! i) c
    by auto
  also have ... = elapsed (xs !! j) (xs !! Suc j) + (xs !! j) c - (xs !! i) c
    by (simp add: dur-Suc)
  also have
    ... = (xs !! Suc j) c - (xs !! j) c + (xs !! j) c - (xs !! i) c
    using * False Suc.prems by auto
  also have ... = (xs !! Suc j) c - (xs !! i) c by simp
  finally show ?thesis by auto
next
  case True
  then show ?thesis by simp
qed
qed

lemma not-reset-dur':
  assumes ∀j≥i. ¬ zero c ([xs !! j]R) j ≥ i c ∈ X stream-trans xs
    ∀ i. eq-elapsed (xs !! i) (xs !! Suc i) ∀ j. xs !! j ∈ V
  shows dur xs j - dur xs i = (xs !! j) c - (xs !! i) c
using assms not-reset-dur by auto

lemma not-reset-unbounded:
  assumes ∀j≥i. ¬ zero c ([xs !! j]R) j ≥ i c ∈ X stream-trans xs
    ∀ i. eq-elapsed (xs !! i) (xs !! Suc i) ∀ j. xs !! j ∈ V
    unbounded c ([xs !! i]R)
  shows unbounded c ([xs !! j]R)
proof -
  let ?u = xs !! i let ?u' = xs !! j let ?R = [xs !! i]R
  from assms have ?u ∈ ?R by auto
  from assms(6) have ?R ∈ R by auto
  then obtain I r where ?R = region X I r valid-region X k I r unfolding R-def by auto
  with assms(3,7) unbounded-Greater ⟨?u ∈ ?R⟩ have ?u c > k c by force
  also from not-reset-dur'[OF assms(1-6)] dur-mono[OF ⟨j ≥ i⟩, of xs] have ?u' c ≥ ?u c by auto
  finally have ?u' c > k c by auto
  let ?R' = [xs !! j]R
  from assms have ?u' ∈ ?R' by auto
  from assms(6) have ?R' ∈ R by auto
  then obtain I r where ?R' = region X I r valid-region X k I r unfolding R-def by auto
  moreover with ⟨?u' c > -⟩ ⟨?u' ∈ -⟩ gt-GreaterD ⟨c ∈ X⟩ have I c = Greater (k c) by auto
  ultimately show ?thesis using Greater-unbounded[OF finite(1) - ⟨c ∈ X⟩] by auto
qed

```


lemma *gt-unboundedD*:
assumes $u \in R$
and $R \in \mathcal{R}$
and $c \in \mathcal{X}$
and $\text{real } (k\ c) < u\ c$
shows $\text{unbounded } c\ R$
proof –
from *assms* **obtain** $I\ r$ **where** $R = \text{region } \mathcal{X}\ I\ r\ \text{valid-region } \mathcal{X}\ k\ I\ r$
unfolding $\mathcal{R}\text{-def}$ **by** *auto*
with $\text{Greater-unbounded}[of\ \mathcal{X}\ k\ I\ r\ c]$ $\text{gt-GreaterD}[of\ u\ \mathcal{X}\ I\ r\ k\ c]$ *assms* $\text{finite}(1)$ **show** *?thesis*
by *auto*
qed

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

lemma *zeroI*:
assumes $c \in \mathcal{X}\ u \in V\ u\ c = 0$
shows $\text{zero } c\ ([u]_{\mathcal{R}})$
proof –
from *assms* **have** $u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*
then obtain $I\ r$ **where** $[u]_{\mathcal{R}} = \text{region } \mathcal{X}\ I\ r\ \text{valid-region } \mathcal{X}\ k\ I\ r$ **unfolding** $\mathcal{R}\text{-def}$ **by** *auto*
with $\text{zero-all}[OF\ \text{finite}(1)\ \text{this}(2)\ \langle c \in \mathcal{X} \rangle\ \langle u \in [u]_{\mathcal{R}} \rangle\ \langle u\ c = 0 \rangle]$ **show** *?thesis* **by** *auto*
qed

lemma *zeroD*:
 $u\ x = 0$ **if** $\text{zero } x\ ([u]_{\mathcal{R}})\ u \in V$
using *that* **by** (*metis* $\text{regions-part-ex}(1)$ zero-def)

lemma *not-zeroD*:
assumes $\neg\ \text{zero } x\ ([u]_{\mathcal{R}})\ u \in V\ x \in \mathcal{X}$
shows $u\ x > 0$
proof –
from *zeroI* *assms* **have** $u\ x \neq 0$ **by** *auto*
moreover from *assms* **have** $u\ x \geq 0$ **unfolding** $V\text{-def}$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *not-const-intv*:
assumes $u \in V\ \forall\ c \in \mathcal{X}. \nexists d. d \leq k\ c \wedge u\ c = \text{real } d$
shows $\forall\ c \in \mathcal{X}. \forall\ u \in [u]_{\mathcal{R}}. \nexists d. d \leq k\ c \wedge u\ c = \text{real } d$
proof –
from *assms* **have** $u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}$ **by** *auto*
then obtain $I\ r$ **where** $I: [u]_{\mathcal{R}} = \text{region } \mathcal{X}\ I\ r\ \text{valid-region } \mathcal{X}\ k\ I\ r$ **unfolding** $\mathcal{R}\text{-def}$ **by** *auto*
have $\nexists d. d \leq k\ c \wedge u'\ c = \text{real } d$ **if** $c \in \mathcal{X}\ u' \in [u]_{\mathcal{R}}$ **for** $c\ u'$
proof *safe*
fix d **assume** $A: d \leq k\ c\ u'\ c = \text{real } d$
from I **that** **have** $\text{intv-elem } c\ u'\ (I\ c)\ \text{valid-intv } (k\ c)\ (I\ c)$ **by** *auto*
then show *False*
using $A\ I\ \langle u \in [u]_{\mathcal{R}} \rangle\ \langle c \in \mathcal{X} \rangle$ *assms*(2) **by** (*cases*; *fastforce*)
qed
then show *?thesis* **by** *auto*
qed

lemma *K-cfg-trans'*:

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

shows $\text{trans}' u u'$

using *assms*

apply (*simp add: set-K-cfg*)

apply (*drule MDP.cfg-onD-action*)

apply (*cases rule: K.cases*)

apply *assumption*

proof *goal-cases*

case *prems: (1 l u t)*

from *assms* $\langle - = (l, u) \rangle$ **have** $\text{repcs } (l, u) \text{ cfg} \in \text{valid-cfg}$ **by** (*auto intro: MDP.valid-cfgI*)

then **have** $\text{absc } (\text{repcs } (l, u) \text{ cfg}) \in \text{R-G.valid-cfg}$ **by** *auto*

from *prems* **have** $*$: $\text{rept } (l, u) \text{ (action } \text{cfg}) = \text{return-pmf } (l, u \oplus t)$ **unfolding** *repcs-def* **by** *auto*

from $\langle \text{abss } - = - \rangle \langle - = (l, u) \rangle \langle \text{cfg} \in \text{R-G.valid-cfg} \rangle$ **have**
 $\text{action } \text{cfg} \in \mathcal{K} \text{ (abss } (l, u))$

by (*auto dest: R-G-I*)

from *abst-rept-id[OF this]* $*$ **have** $\text{action } \text{cfg} = \text{abst } (\text{return-pmf } (l, u \oplus t))$ **by** *auto*

with *prems* **have** $**$: $\text{action } \text{cfg} = \text{return-pmf } (l, [u \oplus t]_{\mathcal{R}})$ **unfolding** *abst-def* **by** *auto*

show *?thesis*

proof (*cases* $\forall c \in \mathcal{X}. u \text{ c} > k \text{ c}$)

case *True*

from *prems* **have** $u \oplus t \in [u]_{\mathcal{R}}$ **by** (*auto intro: upper-right-closed[OF True]*)

with *prems* **have** $[u \oplus t]_{\mathcal{R}} = [u]_{\mathcal{R}}$ **by** (*auto dest: alpha-interp.region-unique-spec*)

with $**$ **have** $\text{action } \text{cfg} = \text{return-pmf } (l, [u]_{\mathcal{R}})$ **by** *simp*

with *True* **have** $\text{rept } (l, u) \text{ (action } \text{cfg}) = \text{return-pmf } (l, u \oplus 0.5)$

unfolding *rept-def* **using** *prems* **by** *auto*

with $*$ **have** $u \oplus t = u \oplus 0.5$ **by** *auto*

moreover **from** *prems* **have** $u' = u \oplus t$ **by** *auto*

moreover **from** *prems* *True* **have** $\forall c \in \mathcal{X}. u' \text{ c} > k \text{ c}$ **by** (*auto simp: cval-add-def*)

ultimately **show** *?thesis* **using** *True* $\langle - = (l, u) \rangle$ **unfolding** *trans'-def* **by** *auto*

next

case *F: False*

show *?thesis*

proof (*cases* $\exists c \in \mathcal{X}. u \text{ c} = 0 \wedge 0 < u' \text{ c} \wedge (\forall c \in \mathcal{X}. \nexists d. d \leq k \text{ c} \wedge u' \text{ c} = \text{real } d)$)

case *True*

from *prems* **have** $u' \in [u']_{\mathcal{R}}$ **by** *auto*

from *prems* **have** $[u \oplus t]_{\mathcal{R}} \in \text{Succ } \mathcal{R} ([u]_{\mathcal{R}})$ **by** *auto*

from *True* **obtain** c **where** $c \in \mathcal{X} \text{ } u \text{ c} = 0 \text{ } u' \text{ c} > 0$ **by** *auto*

with *zeroI* *prems* **have** $\text{zero } c ([u]_{\mathcal{R}})$ **by** *auto*

moreover **from** $\langle u' \in - \rangle \langle u' \text{ c} > 0 \rangle$ **have** $\neg \text{zero } c ([u']_{\mathcal{R}})$ **unfolding** *zero-def* **by** *fastforce*

ultimately **have** $[u \oplus t]_{\mathcal{R}} \neq [u]_{\mathcal{R}}$ **using** *prems* **by** *auto*

moreover **from** *True* *not-const-intv* *prems* **have**
 $\forall u \in [u \oplus t]_{\mathcal{R}}. \forall c \in \mathcal{X}. \nexists d. d \leq k \text{ c} \wedge u \text{ c} = \text{real } d$

by *auto*

ultimately **have** $\exists R'. (l, u) \in S \wedge$
 $\text{action } \text{cfg} = \text{return-pmf } (l, R') \wedge$
 $R' \in \text{Succ } \mathcal{R} ([u]_{\mathcal{R}}) \wedge [u]_{\mathcal{R}} \neq R' \wedge (\forall u \in R'. \forall c \in \mathcal{X}. \nexists d. d \leq k \text{ c} \wedge u \text{ c} = \text{real } d)$

apply $-$

apply (*rule exI[where x = [u \oplus t]_{\mathcal{R}}*])

apply *safe*

using *prems* $**$ **by** *auto*

then **have**
 $\text{rept } (l, u) \text{ (action } \text{cfg})$
 $= \text{return-pmf } (l, \text{delayedR } (\text{SOME } R'. \text{action } \text{cfg} = \text{return-pmf } (l, R')) u)$

unfolding *rept-def* **by** *auto*

with $*$ $**$ *prems* **have** $u' = \text{delayedR } ([u \oplus t]_{\mathcal{R}}) u$ **by** *auto*

with *F* *True* *prems* **show** *?thesis* **unfolding** *trans'-def* **by** *auto*

next

case *False*

with *F* $\langle - = (l, u) \rangle$ **show** *?thesis* **unfolding** *trans'-def* **by** *auto*

qed
qed
next
case $\text{prems}: (2 - - \tau \mu)$
then obtain X **where** $X: u' = ([X := 0]u) (X, l') \in \text{set-pmf } \mu$ **by** *auto*
from $\langle \cdot \in S \rangle$ **have** $u \in V$ **by** *auto*
let $?r = \text{SOME } r$. **set** $r = X$
show $?case$
proof ($\text{cases } X = \{\}$)
case *True*
with X **have** $u = u'$ **by** *auto*
with *non-empty* **show** $?thesis$ **unfolding** *trans'-def* **by** *auto*
next
case *False*
then obtain x **where** $x \in X$ **by** *auto*
moreover have $X \subseteq \mathcal{X}$ **using** *admissible-targets-clocks(1)[OF prems(10) X(2)]* **by** *auto*
ultimately have $x \in \mathcal{X}$ **by** *auto*
from $\langle X \subseteq \mathcal{X} \rangle$ *finite(1)* **obtain** r **where** $\text{set } r = X$ **using** *finite-list finite-subset* **by** *blast*
then have r : $\text{set } ?r = X$ **by** (*rule someI*)
with $\langle x \in X \rangle$ X **have** $u' x = 0$ **by** *auto*
from $X r \langle u \in V \rangle \langle X \subseteq \mathcal{X} \rangle$ **have** $u' x \leq u x$ **for** x
by ($\text{cases } x \in X$; *auto simp: V-def*)
have *False* **if** $u' x > 0 \wedge u x = 0$ **for** x
using $\langle u' - \leq \cdot \rangle$ [*of x*] **that** **by** *auto*
with $\langle u' x = 0 \rangle$ **show** $?thesis$ **using** $\langle x \in \mathcal{X} \rangle$ **unfolding** *trans'-def* **by** *auto*
qed
next
case 3
with *non-empty* **show** $?case$ **unfolding** *trans'-def* **by** *auto*
qed

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

lemma *K-cfg-rept-in*:

assumes $\text{cfg} \in R\text{-G.valid-cfg}$
and $\text{abss } st = \text{state cfg}$
and $\text{cfg}' \in K\text{-cfg cfg}$
shows ($\text{THE } s'. s' \in \text{set-pmf } (\text{rept } st \text{ (action cfg)}) \wedge \text{abss } s' = \text{state cfg}'$)
 $\in \text{set-pmf } (\text{rept } st \text{ (action cfg)})$
proof –
from *assms(1,2)* **have** $\text{action cfg} \in \mathcal{K} (\text{abss } st)$ **by** (*auto simp: R-G-I*)
from $\langle \text{cfg}' \in \cdot \rangle$ **have**
 $\text{cfg}' = \text{cont cfg } (\text{state cfg}') \text{ state cfg}' \in \text{action cfg}$
by (*auto simp: set-K-cfg*)
with *abst-rept-id[OF action - \in \cdot]* *pmf.set-map* **have**
 $\text{state cfg}' \in \text{abss ' set-pmf } (\text{rept } st \text{ (action cfg)})$ **unfolding** *abst-def* **by** *metis*
then obtain st' **where**
 $st' \in \text{rept } st \text{ (action cfg)} \text{ abss } st' = \text{state cfg}'$
unfolding *abst-def* **by** *auto*
with *K-cfg-rept-aux[OF assms(1,2) this(1)]* **show** $?thesis$ **by** *auto*
qed

lemma *enabled-repcsI*:

assumes $\text{cfg} \in R\text{-G.valid-cfg}$ $\text{abss } st = \text{state cfg}$ $\text{MDP.MC.enabled } (\text{repcs } st \text{ cfg}) xs$
shows $\text{enabled-repcs } (\text{repcs } st \text{ cfg}) xs$ **using** *assms*
proof (*coinduction arbitrary: cfg xs st*)

case *prems*: (*enabled-repcs cfg xs st*)
let $?x = \text{shd } xs$ **and** $?y = \text{shd } (\text{stl } xs)$
let $?st = \text{THE } s'. s' \in \text{set-pmf } (\text{rept } st \text{ (action } \text{cfg})) \wedge \text{abss } s' = \text{state } (\text{absc } ?x)$
from *prems*(3) **have** $?x \in K\text{-cfg } (\text{repcs } st \text{ cfg})$ **by cases**
with *K-cfg-map-repcs*[*OF prems*(1,2)] **obtain** *cfg'* **where**
 $\text{cfg}' \in K\text{-cfg } \text{cfg } ?x = \text{repcs } (\text{THE } s'. s' \in \text{rept } st \text{ (action } \text{cfg})) \wedge \text{abss } s' = \text{state } \text{cfg}'$ *cfg'*
by auto
let $?st = \text{THE } s'. s' \in \text{rept } st \text{ (action } \text{cfg}) \wedge \text{abss } s' = \text{state } \text{cfg}'$
from *K-cfg-rept-action*[*OF prems*(1,2) $\langle \text{cfg}' \in - \rangle$] **have** $\text{abss } ?st = \text{state } \text{cfg}'$.
moreover from *K-cfg-rept-in*[*OF prems*(1,2) $\langle \text{cfg}' \in - \rangle$] **have** $?st \in \text{rept } st \text{ (action } \text{cfg})$.
moreover have $\text{cfg}' \in R\text{-G.valid-cfg}$ **using** $\langle \text{cfg}' \in K\text{-cfg } \text{cfg} \rangle$ *prems*(1) **by blast**
moreover from *absc-repcs-id*[*OF this* $\langle \text{abss } ?st = \text{state } \text{cfg}' \rangle$] $\langle ?x = - \rangle$ **have** $\text{absc } ?x = \text{cfg}'$
by auto
moreover from *prems*(3) **have** *MDP.MC.enabled* (*shd xs*) (*stl xs*) **by cases**
ultimately show *?case*
using $\langle ?x = - \rangle$ **by** (*inst-existentials xs ?st absc ?x st cfg*) *fastforce*+
qed

lemma *repcs-eq-rept*:

$\text{rept } st \text{ (action } \text{cfg}) = \text{rept } st'' \text{ (action } \text{cfg}'')$ **if** $\text{repcs } st \text{ cfg} = \text{repcs } st'' \text{ cfg}''$
by (*metis* (*mono-tags*, *lifting*) *action-cfg-corec old.prod.case repcs-def that*)

lemma *enabled-stream-trans'*:

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

using *assms*

proof (*coinduction arbitrary: cfg xs st*)

case *prems*: (*pairwise cfg xs*)

let $?xs = \text{stl } xs$

from *prems* **have** *A*: *enabled-repcs* (*repcs st cfg*) *xs* **by** (*auto intro: enabled-repcsI*)

then obtain $st' \text{ cfg}'$ **where**

$\text{enabled-repcs } (\text{shd } ?xs) (\text{stl } ?xs) \text{ shd } ?xs = \text{repcs } st' \text{ cfg}'$ $st' \in \text{rept } st \text{ (action } \text{cfg})$

$\text{abss } st' = \text{state } \text{cfg}'$ $\text{cfg}' \in R\text{-G.valid-cfg}$

apply *atomize-elim*

apply (*cases rule: enabled-repcs.cases*)

apply *assumption*

subgoal for $st' \text{ cfg}' st'' \text{ cfg}''$

by (*inst-existentials st' cfg'*) (*auto dest: repcs-eq-rept*)

done

then obtain $st'' \text{ cfg}''$ **where**

$\text{enabled-repcs } (\text{shd } ?xs) (\text{stl } ?xs)$

$\text{shd } ?xs = \text{repcs } st'' \text{ cfg}''$ $st'' \in \text{rept } st' \text{ (action } \text{cfg}') \text{ abss } st'' = \text{state } \text{cfg}''$

by *atomize-elim* (*subst* (*asm*)*enabled-repcs.simps*, *fastforce dest: repcs-eq-rept*)

let $?x = \text{shd } xs$ **let** $?y = \text{shd } (\text{stl } xs)$

let $?cfg = \text{repcs } st \text{ cfg}$

from *prems* **have** $?cfg \in \text{valid-cfg}$ **by auto**

from *MDP.pred-stream-cfg-on*[*OF* $\langle ?cfg \in \text{valid-cfg} \rangle$ *prems*(3)] **have** *:

pred-stream ($\lambda \text{cfg. state } \text{cfg} \in S \wedge \text{cfg} \in \text{MDP.cfg-on } (\text{state } \text{cfg})$) *xs* .

obtain $l \ u \ l' \ u'$ **where** $\text{eq: } st' = (l, u) \ st'' = (l', u')$

by *force*

moreover from * **have**

$?x \in \text{MDP.cfg-on } (\text{state } ?x) \ ?x \in \text{valid-cfg}$

by (*auto intro: MDP.valid-cfgI simp: stream.pred-set*)

moreover from *prems*(3) **have** $?y \in K\text{-cfg } ?x$ **by** (*auto elim: MDP.MC.enabled.cases*)

ultimately have *trans' u u'*

using $\langle ?x = - \rangle \langle ?y = - \rangle \langle \text{cfg}' \in - \rangle \langle \text{abss } st' = - \rangle$

by (*intro K-cfg-trans'*) (*auto dest: MDP.valid-cfg-state-in-S*)

with $\langle ?x \in \text{valid-cfg} \rangle \langle \text{cfg}' \in R\text{-G.valid-cfg} \rangle$ *prems*(3) $\langle \text{abss } - = \text{state } \text{cfg}' \rangle$ **show** *?case*

apply (*inst-existentials u u' smap* (*snd o state*) (*stl ?xs*))

apply (*simp add: eq* $\langle ?x = - \rangle \langle ?y = - \rangle$; *fail*)+

by ((*intro disjI1 exI*)*?*; *auto simp:* $\langle ?x = - \rangle \langle ?y = - \rangle$ *eq elim: MDP.MC.enabled.cases*)

qed

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

assumes *in-S*: *pred-stream* $(\lambda u. u \in V) xs$
and *div*: *divergent* *xs*
and *trans*: *stream-trans* *xs*
shows \mathcal{R} -*div* (*smap* $(\lambda u. [u]_{\mathcal{R}}) xs$) (is \mathcal{R} -*div* ? ω)
unfolding \mathcal{R} -*div-def* **proof** (*safe*, *simp-all*)
fix *x i*
assume *x*: $x \in \mathcal{X}$ and *bounded*: $\forall i. \exists j \geq i. \neg \text{unbounded } x ([xs !! j]_{\mathcal{R}})$
from *in-S* have *xs- ω* : $\forall i. xs !! i \in ?\omega !! i$ by (*auto simp: stream.pred-set*)
from *trans in-S* have *elapsed*:
 $\forall i. \text{eq-elapsed } (xs !! i) (xs !! \text{Suc } i)$
 by (*fastforce intro: pairwise-trans-eq-elapsed pairwise-Suc[where P = eq-elapsed]*)
{ assume *A*: $\forall j \geq i. \neg \text{zero } x ([xs !! j]_{\mathcal{R}})$
 let ?*t* = *dur* *xs* *i* + *k* *x*
 from *div* obtain *j* where *j*: *dur* *xs* *j* > *dur* *xs* *i* + *k* *x* unfolding *divergent-def* by *auto*
 then have *k* *x* < *dur* *xs* *j* - *dur* *xs* *i* by *auto*
 also with *not-reset-dur'*[*OF A less-imp-le[OF dur-monoD]*, of *xs*] $\langle x \in \mathcal{X} \rangle$ *assms elapsed* have
 ... = $(xs !! j) x - (xs !! i) x$
 by (*auto simp: stream.pred-set*)
 also have ... $\leq (xs !! j) x$
 using *assms*(1) $\langle x \in \mathcal{X} \rangle$ unfolding *V-def* by (*auto simp: stream.pred-set*)
 finally have *unbounded* $x ([xs !! j]_{\mathcal{R}})$
 using *assms* $\langle x \in \mathcal{X} \rangle$ by (*intro gt-unboundedD*) (*auto simp: stream.pred-set*)
 moreover from *dur-monoD*[of *xs* *i* *j*] *j A* have $\forall j' \geq j. \neg \text{zero } x ([xs !! j']_{\mathcal{R}})$ by *auto*
 ultimately have $\forall i \geq j. \text{unbounded } x ([xs !! i]_{\mathcal{R}})$
 using *elapsed assms* *x* by (*auto intro: not-reset-unbounded simp: stream.pred-set*)
 with *bounded* have *False* by *auto*
}
}
then show $\exists j \geq i. \text{zero } x ([xs !! j]_{\mathcal{R}})$ by *auto*
{ assume *A*: $\forall j \geq i. \text{zero } x ([xs !! j]_{\mathcal{R}})$
 from *div* obtain *j* where *j*: *dur* *xs* *j* > *dur* *xs* *i* unfolding *divergent-def* by *auto*
 then have *j* $\geq i$ by (*auto dest: dur-monoD*)
 from *A* have $\forall j \geq i. \text{zero } x (?\omega !! j)$ by *auto*
 with *dur-zero-tail*[*OF xs- ω - x* $\langle i \leq j \rangle$ *elapsed*] *j* have *False* by *simp*
}
}
then show $\exists j \geq i. \neg \text{zero } x ([xs !! j]_{\mathcal{R}})$ by *auto*

qed

lemma (in \neg)

fixes *f* :: *nat* \Rightarrow *real*
assumes $\forall i. f i \geq 0 \forall i. \exists j \geq i. f j > d d > 0$
shows $\exists n. (\sum i \leq n. f i) > t$
oops

lemma *dur-ev-exceedsI*:

assumes $\forall i. \exists j \geq i. \text{dur } xs \ j - \text{dur } xs \ i \geq d$ and $d > 0$
obtains *i* where *dur* *xs* *i* > *t*

proof -

have *base*: $\exists i. \text{dur } xs \ i > t$ if $t < d$ for *t*

proof -

from *assms* obtain *j* where *dur* *xs* *j* - *dur* *xs* 0 $\geq d$ by *fastforce*

with *dur-pos*[of *xs* 0] have *dur* *xs* *j* $\geq d$ by *simp*

with $\langle d > 0 \rangle \langle t < d \rangle$ show ?*thesis* by - (*rule exI[where x = j]*; *auto*)

qed

have *base2*: $\exists i. \text{dur } xs \ i > t$ if $t \leq d$ for *t*

proof (*cases* $t = d$)

case *False*

with $\langle t \leq d \rangle$ *base* show ?*thesis* by *simp*

```

next
  case True
  from base ⟨d > 0⟩ obtain i where dur xs i > 0 by auto
  moreover from assms obtain j where dur xs j - dur xs i ≥ d by auto
  ultimately have dur xs j > d by auto
  with ⟨t = d⟩ show ?thesis by auto
qed
show ?thesis
proof (cases t ≥ 0)
  case False
  with dur-pos have dur xs 0 > t by auto
  then show ?thesis by (fastforce intro: that)
next
  case True
  let ?m = nat ⌈t / d⌉
  from True have ∃ i. dur xs i > ?m * d
  proof (induction ?m arbitrary: t)
    case 0
    with base[OF ⟨0 < d⟩] show ?case by simp
  next
    case (Suc n t)
    let ?t = t - d
    show ?case
    proof (cases t ≥ d)
      case True
      have ?t / d = t / d - 1
    proof -
      have t / d + - 1 * ((t + - 1 * d) / d) + - 1 * (d / d) = 0
        by (simp add: diff-divide-distrib)
      then have t / d + - 1 * ((t + - 1 * d) / d) = 1
        using assms(2) by fastforce
      then show ?thesis
        by algebra
    qed
    then have ⌈?t / d⌉ = ⌈t / d⌉ - 1 by simp
    with ⟨Suc n = -⟩ have n = nat ⌈?t / d⌉ by simp
    with Suc ⟨t ≥ d⟩ obtain i where nat ⌈?t / d⌉ * d < dur xs i by fastforce
    from assms obtain j where dur xs j - dur xs i ≥ d j ≥ i by auto
    with ⟨dur xs i > -⟩ have nat ⌈?t / d⌉ * d + d < dur xs j by simp
    with True have dur xs j > nat ⌈t / d⌉ * d
    by (metis Suc.hyps(2) ⟨n = nat ⌈(t - d) / d⌉⟩ add.commute distrib-left mult.commute
      mult.right-neutral of-nat-Suc)
    then show ?thesis by blast
  next
    case False
    with ⟨t ≥ 0⟩ ⟨d > 0⟩ have nat ⌈t / d⌉ ≤ 1 by simp
    then have nat ⌈t / d⌉ * d ≤ d
    by (metis One-nat-def ⟨Suc n = -⟩ Suc-leI add.right-neutral le-antisym mult.commute
      mult.right-neutral of-nat-0 of-nat-Suc order-reft zero-less-Suc)
    with base2 show ?thesis by auto
  qed
qed
then obtain i where dur xs i > ?m * d by atomize-elim
moreover from ⟨t ≥ 0⟩ ⟨d > 0⟩ have ?m * d ≥ t
  using pos-divide-le-eq real-nat-ceiling-ge by blast
ultimately show ?thesis using that[of i] by simp
qed
qed

```

lemma not-reset-mono:

assumes *stream-trans xs shd xs c1 ≥ shd xs c2 stream-all* $(\lambda u. u \in V)$ *xs c2 ∈ X*
shows *(holds* $(\lambda u. u c1 \geq u c2)$ *until holds* $(\lambda u. u c1 = 0))$ *xs using* *assms*
proof *(coinduction arbitrary: xs)*
case *prems: (UNTIL xs)*
let *?xs = stl xs*
let *?x = shd xs*
let *?y = shd ?xs*
show *?case*
proof *(cases ?x c1 = 0)*
case *False*
show *?thesis*
proof *(cases ?y c1 = 0)*
case *False*
from *prems have trans ?x ?y by (intro pairwise-pairD[of trans])*
then have *?y c1 ≥ ?y c2*
proof *cases*
case *A: (reset t)*
show *?thesis*
proof *(cases c1 ∈ set t)*
case *True*
with *A False show ?thesis by auto*
next
case *False*
from *prems have ?x c2 ≥ 0 by (auto simp: V-def)*
with *A have ?y c2 ≤ ?x c2 by (cases c2 ∈ set t) auto*
with *A False <?x c1 ≥ ?x c2> show ?thesis by auto*
qed
qed *(use prems in <auto simp: cval-add-def>)*
moreover from prems have *stream-trans ?xs stream-all* $(\lambda u. u \in V)$ *?xs*
by *(auto intro: pairwise-stlD stl-sset)*
ultimately show *?thesis*
using *prems by auto*
qed *(use prems in <auto intro: UNTIL.base>)*
qed *auto*
qed

lemma R-divergent-divergent-aux:

fixes *xs :: ('c, t) cval stream*
assumes *stream-trans xs stream-all* $(\lambda u. u \in V)$ *xs*
 $(xs !! i) c1 = 0 \exists k > i. k \leq j \wedge (xs !! k) c2 = 0$
 $\forall k > i. k \leq j \longrightarrow (xs !! k) c1 \neq 0$
 $c1 \in X \ c2 \in X$
shows $(xs !! j) c1 \geq (xs !! j) c2$
proof *–*
from *assms obtain k where k: k > i k ≤ j (xs !! k) c2 = 0 by auto*
with *assms(5) <k ≤ j> have (xs !! k) c1 ≠ 0 by auto*
moreover from *assms(2) <c1 ∈ X> have (xs !! k) c1 ≥ 0 by (auto simp: V-def)*
ultimately have $(xs !! k) c1 > 0$ **by** *auto*
with $\langle (xs !! k) c2 = 0 \rangle$ **have** $\text{shd} (\text{sdrop } k \text{ } xs) c1 \geq \text{shd} (\text{sdrop } k \text{ } xs) c2$ **by** *auto*
from *not-reset-mono[OF - this] assms have*
 $(\text{holds } (\lambda u. u c2 \leq u c1) \text{ until holds } (\lambda u. u c1 = 0)) (\text{sdrop } k \text{ } xs)$
by *(auto intro: sset-sdrop pairwise-sdropD)*
from *assms(5) k(2) <k > i> have* $\forall m \leq j - k. (\text{sdrop } k \text{ } xs !! m) c1 \neq 0$ **by** *simp*
with *holds-untilD[OF <(- until -)>, of j - k] have*
 $(\text{sdrop } k \text{ } xs !! (j - k)) c2 \leq (\text{sdrop } k \text{ } xs !! (j - k)) c1$.
then show $(xs !! j) c2 \leq (xs !! j) c1$ **using** *k(1,2) by simp*
qed

lemma unbounded-all:

assumes $R \in \mathcal{R} \ u \in R$ *unbounded x R x ∈ X*

shows $u x > k x$
proof –
from *assms* **obtain** $I r$ **where** $R: R = \text{region } \mathcal{X} \text{ } I r \text{ valid-region } \mathcal{X} \text{ } k \text{ } I r$ **unfolding** $\mathcal{R}\text{-def}$ **by** *auto*
with *unbounded-Greater* $\langle x \in \mathcal{X} \rangle$ *assms*(3) **have** $I x = \text{Greater } (k x)$ **by** *simp*
with $\langle u \in R \rangle R \langle x \in \mathcal{X} \rangle$ **show** *?thesis* **by** *force*
qed

lemma *trans-not-delay-mono*:
 $u' c \leq u c$ **if** *trans* $u u' u \in V x \in \mathcal{X} u' x = 0 c \in \mathcal{X}$
using $\langle \text{trans } u u' \rangle$
proof (*cases*)
case (*reset l*)
with *that* **show** *?thesis* **by** (*cases c \in set l*) (*auto simp: V-def*)
qed (*use that in* $\langle \text{auto simp: cval-add-def V-def add-nonneg-eq-0-iff} \rangle$)

lemma *dur-reset*:
assumes *pairwise eq-elapsed xs pred-stream* $(\lambda u. u \in V) xs \text{ zero } x ([xs !! \text{Suc } i]_{\mathcal{R}}) x \in \mathcal{X}$
shows $\text{dur } xs (\text{Suc } i) - \text{dur } xs i = 0$
proof –
from *assms*(2) **have** $\text{in-}V: xs !! \text{Suc } i \in V$
unfolding *stream.pred-set* **by** *auto* (*metis snth.simps(2) snth-sset*)
with *elapsed-ge-pos*[*of xs !! i xs !! Suc i x*] *pairwise-Suc*[*OF assms(1)*] *assms*(2–) **have**
 $\text{elapsed } (xs !! i) (xs !! \text{Suc } i) \leq (xs !! \text{Suc } i) x$
unfolding *stream.pred-set* **by** *auto*
with $\text{in-}V$ *assms*(3) **have** $\text{elapsed } (xs !! i) (xs !! \text{Suc } i) \leq 0$ **by** (*auto simp: zeroD*)
with *elapsed-ge0*[*of xs !! i xs !! Suc i*] **have** $\text{elapsed } (xs !! i) (xs !! \text{Suc } i) = 0$
by *linarith*
then **show** *?thesis* **by** (*subst dur-Suc*)
qed

lemma *resets-mono-0'*:
assumes *pairwise eq-elapsed xs stream-all* $(\lambda u. u \in V) xs \text{ stream-trans } xs$
 $\forall j \leq i. \text{zero } x ([xs !! j]_{\mathcal{R}}) x \in \mathcal{X} c \in \mathcal{X}$
shows $(xs !! i) c = (xs !! 0) c \vee (xs !! i) c = 0$
using *assms* **proof** (*induction i*)
case 0
then **show** *?case* **by** *auto*
next
case ($\text{Suc } i$)
from *Suc.prem*s **have** $*$: $(xs !! \text{Suc } i) x = 0 (xs !! i) x = 0$
by (*blast intro: zeroD snth-sset, force intro: zeroD snth-sset*)
from *pairwise-Suc*[*OF Suc.prem*s(3)] **have** *trans* $(xs !! i) (xs !! \text{Suc } i)$.
then **show** *?case*
proof *cases*
case *prems: (succ t)*
with $*$ **have** $t = 0$ **unfolding** *cval-add-def* **by** *auto*
with *prems* **have** $(xs !! \text{Suc } i) c = (xs !! i) c$ **unfolding** *cval-add-def* **by** *auto*
with Suc **show** *?thesis* **by** *auto*
next
case *prems: (reset l)*
then **have** $(xs !! \text{Suc } i) c = 0 \vee (xs !! \text{Suc } i) c = (xs !! i) c$ **by** (*cases c \in set l*) *auto*
with Suc **show** *?thesis* **by** *auto*
next
case *id*
with Suc **show** *?thesis* **by** *auto*
qed
qed

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

shows $(xs !! j) c = (xs !! i) c \vee (xs !! j) c = 0$ using *assms*

proof –

from *assms* have 1: *stream-all* $(\lambda u. u \in V)$ $(sdrop\ i\ xs)$
 using *sset-sdrop unfolding stream.pred-set* by *force*

from *assms* have 2: *pairwise eq-elapsed* $(sdrop\ i\ xs)$ by $(intro\ pairwise-sdropD)$

from *assms* have 3: *stream-trans* $(sdrop\ i\ xs)$ by $(intro\ pairwise-sdropD)$

from *assms* have 4:
 $\forall k \leq j - i. zero\ x\ ([sdrop\ i\ xs !! k]_{\mathcal{R}})$
 by $(simp\ add:\ le-diff-conv2\ assms(6))$

from *resets-mono-0* $[OF\ 2\ 1\ 3\ 4\ assms(5,6)]\ \langle i \leq j \rangle$ show *?thesis* by *simp*

qed

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 $(xs !! j) c \leq (xs !! i) c$ using *assms*
 using *assms* by $(auto\ simp:\ V-def\ dest:\ resets-mono'[where\ c = c]\ simp:\ stream.pred-set)$

lemma *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 $\forall i. \exists j \geq i. \exists k > j. \exists P \in M. P\ j \wedge P\ k \wedge (\forall m < k. j < m \longrightarrow \neg P\ m)$
 $\wedge (\forall Q \in M. \exists m \leq k. j < m \wedge Q\ m)$

proof

fix i

let $?j1 = Max\ \{LEAST\ m. m > i \wedge P\ m \mid P. P \in M\}$

from $\langle M \neq \{\} \rangle$ obtain P where $P \in M$ by *auto*

let $?m = LEAST\ m. m > i \wedge P\ m$

from *assms(1)* $\langle P \in M \rangle$ obtain j where $j \geq Suc\ i\ P\ j$ by *auto*

then have $j > i\ P\ j$ by *auto*

with $\langle P \in M \rangle$ have $?m > i \wedge P\ ?m$ by $-(rule\ LeastI;\ auto)$

moreover with $\langle finite\ M \rangle\ \langle P \in M \rangle$ have $?j1 \geq ?m$ by $-(rule\ Max-ge;\ auto)$

ultimately have $?j1 \geq i$ by *simp*

moreover have $\exists m > i. m \leq ?j1 \wedge P\ m$ if $P \in M$ for P

proof –

let $?m = LEAST\ m. m > i \wedge P\ m$

from *assms(1)* $\langle P \in M \rangle$ obtain j where $j \geq Suc\ i\ P\ j$ by *auto*

then have $j > i\ P\ j$ by *auto*

with $\langle P \in M \rangle$ have $?m > i \wedge P\ ?m$ by $-(rule\ LeastI;\ auto)$

moreover with $\langle finite\ M \rangle\ \langle P \in M \rangle$ have $?j1 \geq ?m$ by $-(rule\ Max-ge;\ auto)$

ultimately show *?thesis* by *auto*

qed

ultimately obtain $j1$ where $j1: j1 \geq i \forall P \in M. \exists m > i. j1 \geq m \wedge P\ m$ by *auto*

define k where $k\ Q = (LEAST\ k. k > j1 \wedge Q\ k)$ for Q

let $?k = Max\ \{k\ Q \mid Q. Q \in M\}$

let $?P = SOME\ P. P \in M \wedge k\ P = ?k$

let $?j = Max\ \{j. i \leq j \wedge j \leq j1 \wedge ?P\ j\}$

have $?k \in \{k\ Q \mid Q. Q \in M\}$ using *assms* by $-(rule\ Max-in;\ auto)$

then obtain P where $P: k\ P = ?k\ P \in M$ by *auto*

have $?k \geq k\ Q$ if $Q \in M$ for Q using *assms* that by $-(rule\ Max-ge;\ auto)$

have $*$: $?P \in M \wedge k\ ?P = ?k$ using P by $-(rule\ someI[where\ x = P];\ auto)$

with $j1$ have $\exists m > i. j1 \geq m \wedge ?P\ m$ by *auto*

with $\langle finite\ \rightarrow \rangle$ have $?j \in \{j. i \leq j \wedge j \leq j1 \wedge ?P\ j\}$ by $-(rule\ Max-in;\ auto)$

have $k: k\ Q > j1 \wedge Q\ (k\ Q)$ if $Q \in M$ for Q

proof –

from *assms(1)* $\langle Q \in M \rangle$ obtain m where $m \geq Suc\ j1\ Q\ m$ by *auto*

then have $m > j1\ Q\ m$ by *auto*

then show $k\ Q > j1 \wedge Q\ (k\ Q)$ unfolding *k-def* by $-(rule\ LeastI;\ blast)$

qed

with $*$ $\langle ?j \in \rightarrow \rangle$ have $?P\ ?k\ ?j < ?k$ by *fastforce+*

have $\neg ?P\ m$ if $?j < m\ m < ?k$ for m

proof (*rule ccontr, simp*)
assume $?P\ m$
have $m > j1$
proof (*rule ccontr*)
assume $\neg j1 < m$
with $\langle ?j < m \rangle \langle ?j \in \rightarrow \rangle$ **have** $i \leq m\ m \leq j1$ **by** *auto*
with $\langle ?P\ m \rangle \langle \text{finite} \rightarrow \rangle$ **have** $?j \geq m$ **by** $-$ (*rule Max-ge; auto*)
with $\langle ?j < m \rangle$ **show** *False* **by** *simp*
qed
with $\langle ?P\ m \rangle \langle \text{finite} \rightarrow \rangle$ **have** $k\ ?P \leq m$ **unfolding** *k-def* **by** (*auto intro: Least-le*)
with $* \langle m < ?k \rangle$ **show** *False* **by** *auto*
qed
moreover **have** $\exists m \leq ?k. ?j < m \wedge Q\ m$ **if** $Q \in M$ **for** Q
proof $-$
from $k[OF \langle Q \in M \rangle]$ **have** $k\ Q > j1 \wedge Q\ (k\ Q)$.
moreover **with** $\langle \text{finite} \rightarrow \rangle \langle Q \in M \rangle$ **have** $k\ Q \leq ?k$ **by** $-$ (*rule Max-ge; auto*)
moreover **with** $\langle ?j \in \rightarrow \rangle \langle k\ Q > - \wedge \rightarrow \rangle$ **have** $?j < k\ Q$ **by** *auto*
ultimately **show** *?thesis* **by** *auto*
qed
ultimately **show**
 $\exists j \geq i. \exists k > j. \exists P \in M. P\ j \wedge P\ k \wedge (\forall m < k. j < m \longrightarrow \neg P\ m)$
 $\wedge (\forall Q \in M. \exists m \leq k. j < m \wedge Q\ m)$
using $\langle ?j < ?k \rangle \langle ?j \in \rightarrow \rangle \langle ?P\ ?k \rangle *$ **by** (*inst-existentials ?j ?k ?P; blast*)
qed

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

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

shows *divergent* xs

unfolding *divergent-def* **proof**

fix t

from *pairwise-trans-eq-elapsed* $[OF\ \text{trans } in-S]$ **have** *eq-elapsed: pairwise eq-elapsed* xs .

define $X1$ **where** $X1 = \{x. x \in \mathcal{X} \wedge (\exists i. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}}))\}$

let $?i = \text{Max } \{(SOME\ i. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}})) \mid x. x \in \mathcal{X}\}$

from *finite(1) non-empty* **have**

$?i \in \{(SOME\ i. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}})) \mid x. x \in \mathcal{X}\}$

by (*intro Max-in*) *auto*

have *unbounded* $x\ ([xs\ !!\ j]_{\mathcal{R}})$ **if** $x \in X1\ j \geq ?i$ **for** $x\ j$

proof $-$

have $X1 \subseteq \mathcal{X}$ **unfolding** *X1-def* **by** *auto*

with *finite(1) non-empty* $\langle x \in X1 \rangle$ **have** $*$:

$?i \geq (SOME\ i. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}}))$ (**is** $?i \geq ?k$)

by (*intro Max-ge*) *auto*

from $\langle x \in X1 \rangle$ **have** $\exists k. \forall j \geq k. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}})$ **by** (*auto simp: X1-def*)

then **have** $\forall j \geq ?k. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}})$ **by** (*rule someI-ex*)

moreover **from** $\langle j \geq ?i \rangle \langle ?i \geq \rightarrow \rangle$ **have** $j \geq ?k$ **by** *auto*

ultimately **show** *?thesis* **by** *blast*

qed

then **obtain** i **where** *unbounded:* $\forall x \in X1. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}})$

using *finite* **by** *auto*

show $\exists n. t < \text{dur } xs\ n$

proof (*cases* $\forall x \in \mathcal{X}. (\exists i. \forall j \geq i. \text{unbounded } x\ ([xs\ !!\ j]_{\mathcal{R}}))$)

case *True*

then **have** $X1 = \mathcal{X}$ **unfolding** *X1-def* **by** *auto*

have $\exists k \geq j. 0.5 \leq \text{dur } xs\ k - \text{dur } xs\ j$ **for** j

proof $-$

let $?u = xs\ !!\ \text{max } i\ j$

```

from in-S have  $?u \in [?u]_{\mathcal{R}} [?u]_{\mathcal{R}} \in \mathcal{R}$ 
  by (auto simp: stream.pred-set)
moreover from unbounded  $\langle X1 = \mathcal{X} \rangle$  have
   $\forall x \in \mathcal{X}. \text{unbounded } x ([?u]_{\mathcal{R}})$ 
  by force
ultimately have  $\forall x \in \mathcal{X}. ?u \ x > k \ x$ 
  by (auto intro: unbounded-all)
with unbounded-not-const have  $\neg \text{ev } (\text{alw } (\text{HLD } \{?u\})) \ x$ 
  unfolding HLD-iff by simp
then obtain r where
   $r \geq \max i \ j \ x$  !!  $r \neq x$  !! Suc r
  apply atomize-elim
  apply (simp add: not-ev-iff not-alw-iff)
  apply (drule alw-sdrop[where n = max i j])
  apply (drule alwD)
  apply (subst (asm) (3) stream.collapse[symmetric])
  apply simp
  apply (drule ev-neq-start-implies-ev-neq[simplified comp-def])
  using stream.collapse[of sdrop (max i j) x] by (auto 4 3 elim: ev-sdropD)
let  $?k = \text{Suc } r$ 
from in-S have  $x$  !!  $?k \in V$  using snth-sset unfolding stream.pred-set by blast
with in-S have  $*$ :
   $x$  !!  $r \in [x \ !! \ r]_{\mathcal{R}} [x \ !! \ r]_{\mathcal{R}} \in \mathcal{R}$ 
   $x$  !!  $?k \in [x \ !! \ ?k]_{\mathcal{R}} [x \ !! \ ?k]_{\mathcal{R}} \in \mathcal{R}$ 
  by (auto simp: stream.pred-set)
from  $\langle r \geq \rightarrow \rangle$  have  $r \geq i \ ?k \geq i$  by auto
with unbounded  $\langle X1 = \mathcal{X} \rangle$  have
   $\forall x \in \mathcal{X}. \text{unbounded } x ([x \ !! \ r]_{\mathcal{R}}) \ \forall x \in \mathcal{X}. \text{unbounded } x ([x \ !! \ ?k]_{\mathcal{R}})$ 
  by (auto simp del: snth.simps(2))
with in-S have  $\forall x \in \mathcal{X}. (x \ !! \ r) \ x > k \ x \ \forall x \in \mathcal{X}. (x \ !! \ ?k) \ x > k \ x$ 
  using  $*$  by (auto intro: unbounded-all)
moreover from trans' have trans'  $(x \ !! \ r) \ (x \ !! \ ?k)$ 
  using pairwise-Suc by auto
ultimately have  $(x \ !! \ ?k) = (x \ !! \ r) \oplus 0.5$ 
  unfolding trans'-def using  $\langle x \ !! \ r \neq \rightarrow \rangle$  by auto
moreover from pairwise-Suc[OF eq-elapsed] have eq-elapsed  $(x \ !! \ r) \ (x \ !! \ ?k)$ 
  by auto
ultimately have
   $\text{dur } x \ ?k - \text{dur } x \ r = 0.5$ 
  using non-empty by (auto simp: cval-add-def dur-Suc elapsed-eq)
with dur-mono[of j r x]  $\langle r \geq \max i \ j \rangle$  have  $\text{dur } x \ ?k - \text{dur } x \ j \geq 0.5$ 
  by auto
with  $\langle r \geq \max i \ j \rangle$  show ?thesis by  $-$  (rule exI[where x = ?k]; auto)
qed
then show ?thesis by  $-$  (rule dur-ev-exceedsI[where d = 0.5]; auto)
next
case False
define X2 where  $X2 = \mathcal{X} - X1$ 
from False have  $X2 \neq \{\}$  unfolding X1-def X2-def by fastforce
have inf-resets:
   $\forall i. (\exists j \geq i. \text{zero } x ([x \ !! \ j]_{\mathcal{R}})) \wedge (\exists j \geq i. \neg \text{zero } x ([x \ !! \ j]_{\mathcal{R}}))$  if  $x \in X2$  for  $x$ 
  using that div unfolding X1-def X2-def R-div-def by fastforce
have  $\exists j \geq i. \exists k > j. \exists x \in X2. \text{zero } x ([x \ !! \ j]_{\mathcal{R}}) \wedge \text{zero } x ([x \ !! \ k]_{\mathcal{R}})$ 
   $\wedge (\forall m. j < m \wedge m < k \longrightarrow \neg \text{zero } x ([x \ !! \ m]_{\mathcal{R}}))$ 
   $\wedge (\forall x \in X2. \exists m. j < m \wedge m \leq k \wedge \text{zero } x ([x \ !! \ m]_{\mathcal{R}}))$ 
   $\wedge (\forall x \in X1. \forall m \geq j. \text{unbounded } x ([x \ !! \ m]_{\mathcal{R}}))$  for  $i$ 
proof  $-$ 
from unbounded obtain  $i'$  where  $i' : \forall x \in X1. \forall m \geq i'. \text{unbounded } x ([x \ !! \ m]_{\mathcal{R}})$  by auto
then obtain  $i'$  where  $i'$ :
   $i' \geq i \ \forall x \in X1. \forall m \geq i'. \text{unbounded } x ([x \ !! \ m]_{\mathcal{R}})$ 
  by (cases  $i' \geq i$ ; auto)

```

from $\text{finite}(1)$ **have** $\text{finite } X2$ **unfolding** $X2\text{-def}$ **by** auto
with $\langle X2 \neq \{\} \rangle$ \mathcal{R} -*divergent-divergent-aux2* [**where** $M = \{\lambda i. \text{zero } x ([xs !! i]_{\mathcal{R}}) \mid x. x \in X2\}$]
inf-resets
have $\exists j \geq i'. \exists k > j. \exists P \in \{\lambda i. \text{zero } x ([xs !! i]_{\mathcal{R}}) \mid x. x \in X2\}. P j \wedge P k$
 $\wedge (\forall m < k. j < m \longrightarrow \neg P m) \wedge (\forall Q \in \{\lambda i. \text{zero } x ([xs !! i]_{\mathcal{R}}) \mid x. x \in X2\}. \exists m \leq k. j < m \wedge Q m)$
by force
then obtain $j k x$ **where**
 $j \geq i' k > j x \in X2 \text{ zero } x ([xs !! j]_{\mathcal{R}}) \text{ zero } x ([xs !! k]_{\mathcal{R}})$
 $\forall m. j < m \wedge m < k \longrightarrow \neg \text{zero } x ([xs !! m]_{\mathcal{R}})$
 $\forall Q \in \{\lambda i. \text{zero } x ([xs !! i]_{\mathcal{R}}) \mid x. x \in X2\}. \exists m \leq k. j < m \wedge Q m$
by auto
moreover from $\text{this}(7)$ **have** $\forall x \in X2. \exists m \leq k. j < m \wedge \text{zero } x ([xs !! m]_{\mathcal{R}})$ **by** auto
ultimately show $?thesis$ **using** i'
by ($\text{inst-existentials } j k x$) auto
qed
moreover have $\exists j' \geq j. \text{dur } xs j' - \text{dur } xs i \geq 0.5$
if $x: x \in X2 i < j \text{ zero } x ([xs !! i]_{\mathcal{R}}) \text{ zero } x ([xs !! j]_{\mathcal{R}})$
and $\text{not-reset}: \forall m. i < m \wedge m < j \longrightarrow \neg \text{zero } x ([xs !! m]_{\mathcal{R}})$
and $X2: \forall x \in X2. \exists m. i < m \wedge m \leq j \wedge \text{zero } x ([xs !! m]_{\mathcal{R}})$
and $X1: \forall x \in X1. \forall m \geq i. \text{unbounded } x ([xs !! m]_{\mathcal{R}})$
for $x i j$
proof $-$
have $\exists j' > j. \neg \text{zero } x ([xs !! j']_{\mathcal{R}})$
proof $-$
from $\text{inf-resets}[OF x(1)]$ **obtain** j' **where** $j' \geq \text{Suc } j \neg \text{zero } x ([xs !! j']_{\mathcal{R}})$ **by** auto
then show $?thesis$ **by** $-$ ($\text{rule } \text{exI}[\text{where } x = j']; \text{auto}$)
qed
from $\text{inf-resets}[OF x(1)]$ **obtain** j' **where** $j' \geq \text{Suc } j \neg \text{zero } x ([xs !! j']_{\mathcal{R}})$ **by** auto
with $\text{nat-eventually-critical-path}[OF x(4) \text{ this}(2)]$
obtain j' **where** j' :
 $j' > j \neg \text{zero } x ([xs !! j']_{\mathcal{R}}) \forall m \geq j. m < j' \longrightarrow \text{zero } x ([xs !! m]_{\mathcal{R}})$
by auto
from $\langle x \in X2 \rangle$ **have** $x \in \mathcal{X}$ **unfolding** $X2\text{-def}$ **by** simp
with $\langle i < j \rangle$ $\text{not-reset not-reset-dur } \langle \text{stream-trans } \rightarrow \text{in-S pairwise-Suc}[OF \text{eq-elapsed}]$ **have**
 $\text{dur } xs (j - 1) - \text{dur } xs i = (xs !! (j - 1)) x - (xs !! i) x$ (**is** $?d1 = ?d2$)
by ($\text{auto simp: stream.pred-set}$)
moreover from $\langle \text{zero } x ([xs !! i]_{\mathcal{R}}) \rangle$ in-S **have** $(xs !! i) x = 0$
by ($\text{auto intro: zeroD simp: stream.pred-set}$)
ultimately have
 $\text{dur } xs (j - 1) - \text{dur } xs i = (xs !! (j - 1)) x$ (**is** $?d1 = ?d2$)
by simp
show $?thesis$
proof ($\text{cases } ?d1 \geq 0.5$)
case True

with $\text{dur-mono}[of j - 1 j xs]$ **have**
 $5 / 10 \leq \text{dur } xs j - \text{dur } xs i$
by simp
then show $?thesis$ **by** blast
next
case False
have $j\text{-c-bound}: (xs !! j) c \leq ?d2$ **if** $c \in X2$ **for** c
proof ($\text{cases } (xs !! j) c = 0$)
case True
from $\text{in-S } \langle j > \rightarrow \text{True } \langle x \in \mathcal{X} \rangle$ **show** $?thesis$ **by** ($\text{auto simp: V-def stream.pred-set}$)
next
case False
from $X2 \langle c \in X2 \rangle$ in-S **have** $\exists k > i. k \leq j \wedge (xs !! k) c = 0$
by ($\text{force simp: zeroD stream.pred-set}$)
with False **have**
 $\exists k > i. k \leq j - \text{Suc } 0 \wedge (xs !! k) c = 0$

by (metis Suc-le-eq Suc-pred linorder-neqE-nat not-less not-less-zero)
moreover from that **have** $c \in \mathcal{X}$ **by** (auto simp: X2-def)
moreover from not-reset in-S $\langle x \in \mathcal{X} \rangle$ **have**
 $\forall k > i. k \leq j - 1 \longrightarrow (xs !! k) x \neq 0$
by (auto simp: zeroI stream.pred-set)
ultimately have
 $(xs !! (j - 1)) c \leq ?d2$
using trans in-S $\langle - x = 0 \rangle \langle x \in \mathcal{X} \rangle$
by (auto intro: \mathcal{R} -divergent-divergent-aux that simp: stream.pred-set)
moreover from
 trans-not-delay-mono[OF pairwise-Suc[OF trans], of $j - 1$]
 $\langle x \in \mathcal{X} \rangle \langle c \in \mathcal{X} \rangle \langle j > - \rangle$ in-S $x(4)$
have $(xs !! j) c \leq (xs !! (j - 1)) c$ **by** (auto simp: zeroD stream.pred-set)
ultimately show ?thesis **by** auto
qed
moreover from False $\langle ?d1 = ?d2 \rangle$ **have** $?d2 < 1$ **by** auto
moreover from in-S **have** $(xs !! j) c \geq 0$ **if** $c \in \mathcal{X}$ **for** c
using that **by** (auto simp: V-def stream.pred-set)
ultimately have frac-bound: $\text{frac}((xs !! j) c) \leq ?d2$ **if** $c \in X2$ **for** c
using that frac-le-1I **by** (force simp: X2-def)

let $?u = (xs !! j)$
from in-S **have** $[xs !! j]_{\mathcal{R}} \in \mathcal{R}$ **by** (auto simp: stream.pred-set)
then obtain $I r$ **where** region:
 $[xs !! j]_{\mathcal{R}} = \text{region } \mathcal{X} \ I r \ \text{valid-region } \mathcal{X} \ k \ I r$
unfolding \mathcal{R} -def **by** auto
let $?S = \{\text{frac} (?u c) \mid c. c \in \mathcal{X} \wedge \text{isIntv} (I c)\}$
have \mathcal{X} -X2: $c \in X2$ **if** $c \in \mathcal{X}$ **isIntv** $(I c)$ **for** c
proof –
from X1 $\langle j > i \rangle$ **have** $\forall x \in X1. \text{unbounded } x \ ([xs !! j]_{\mathcal{R}})$ **by** auto
with unbounded-Greater[OF region(2) $\langle c \in \mathcal{X} \rangle$] region(1) that(2) **have** $c \notin X1$ **by** auto
with $\langle c \in \mathcal{X} \rangle$ **show** $c \in X2$ **unfolding** X2-def **by** auto
qed
have frac-bound: $\text{frac}((xs !! j) c) \leq ?d2$ **if** $c \in \mathcal{X}$ **isIntv** $(I c)$ **for** c
using frac-bound[OF \mathcal{X} -X2] that .
have $\text{dur } xs \ (j' - 1) = \text{dur } xs \ j$ **using** $j' \langle x \in \mathcal{X} \rangle$ in-S eq-elapsed
by (subst dur-zero-tail[where $\omega = \text{smap} (\lambda u. [u]_{\mathcal{R}}) xs$])
 (auto dest: pairwise-Suc simp: stream.pred-set)
moreover from dur-reset[OF eq-elapsed in-S, of $x \ j - 1$] $\langle x \in \mathcal{X} \rangle \ x(4) \ \langle j > - \rangle$ **have**
 $\text{dur } xs \ j = \text{dur } xs \ (j - 1)$
by (auto simp: stream.pred-set)
ultimately have $\text{dur } xs \ (j' - 1) = \text{dur } xs \ (j - 1)$ **by** auto
moreover have $\text{dur } xs \ j' - \text{dur } xs \ (j' - 1) \geq (1 - ?d2) / 2$
proof –
from $\langle j' > - \rangle$ **have** $j' > 0$ **by** auto
with pairwise-Suc[OF trans', of $j' - 1$] **have**
 $\text{trans}' (xs !! (j' - 1)) (xs !! j')$
by auto
moreover from j' **have**
 $(xs !! (j' - 1)) x = 0 \ (xs !! j') x > 0$
using in-S $\langle x \in \mathcal{X} \rangle$ **by** (force intro: zeroD dest: not-zeroD simp: stream.pred-set)+
moreover note delayedR-aux = calculation
obtain t **where**
 $(xs !! j') = (xs !! (j' - 1)) \oplus t \ t \geq (1 - ?d2) / 2 \ t \geq 0$
proof –
from in-S **have** $[xs !! j']_{\mathcal{R}} \in \mathcal{R}$ **by** (auto simp: stream.pred-set)
then obtain $I' r'$ **where** region':
 $[xs !! j']_{\mathcal{R}} = \text{region } \mathcal{X} \ I' r' \ \text{valid-region } \mathcal{X} \ k \ I' r'$
unfolding \mathcal{R} -def **by** auto
let $?S' = \{\text{frac}((xs !! (j' - 1)) c) \mid c. c \in \mathcal{X} \wedge \text{Regions.isIntv} (I' c)\}$

```

from finite(1) have ?d2 ≥ Max (?S' ∪ {0})
  apply -
  apply (rule Max.boundedI)
  apply fastforce
  apply fastforce
  apply safe
subgoal premises prems for - c d
proof -
  from j' have (xs !! (j' - 1)) c = ?u c ∨ (xs !! (j' - 1)) c = 0
  by (intro resets-mono'[OF eq-elapsd in-S trans - ⟨x ∈ X⟩ ⟨c ∈ X⟩]; auto)
  then show ?thesis
  proof (standard, goal-cases)
    case A: 1
    show ?thesis
    proof (cases c ∈ X1)
      case True
      with X1 ⟨j' > j⟩ ⟨j > i⟩ have unbounded c ([xs !! j]ℛ) by auto
      with region' ⟨c ∈ X⟩ have I' c = Greater (k c)
      by (auto intro: unbounded-Greater)
      with prems show ?thesis by auto
    next
    case False
    with ⟨c ∈ X⟩ have c ∈ X2 unfolding X2-def by auto
    with j-c-bound have mono: (xs !! j) c ≤ (xs !! (j - 1)) x .
    from in-S ⟨c ∈ X⟩ have (xs !! (j' - 1)) c ≥ 0
      unfolding V-def stream.pred-set by auto
    then have
      frac ((xs !! (j' - 1)) c) ≤ (xs !! (j' - 1)) c
      using frac-le-self by auto
    with A mono show ?thesis by auto
  qed
next
case prems: 2

  have frac (0 :: real) = (0 :: real) by auto
  then have frac (0 :: real) ≤ (0 :: real) by linarith
  moreover from in-S ⟨x ∈ X⟩ have (xs !! (j - 1)) x ≥ 0
    unfolding V-def stream.pred-set by auto
  ultimately show ?thesis using prems by auto
qed
qed
using in-S ⟨x ∈ X⟩ by (auto simp: V-def stream.pred-set)
then have le: (1 - ?d2) / 2 ≤ (1 - Max (?S' ∪ {0})) / 2 by simp

let ?u = xs !! j'
let ?u' = xs !! (j' - 1)
from in-S have *: ?u' ∈ V [?u]ℛ ∈ ℛ ?u ∈ V [?u]ℛ ∈ ℛ
  by (auto simp: stream.pred-set)
from pairwise-Suc[OF trans, of j' - 1] ⟨j' > j⟩ have
  trans (xs !! (j' - 1)) (xs !! j')
  by auto
then have Succ:
  [xs !! j]ℛ ∈ Succ ℛ ([xs !! (j' - 1)]ℛ) ∧ (∃ t ≥ 0. ?u = ?u' ⊕ t)
proof cases
  case prems: (succ t)
  from * have ?u' ∈ [?u]ℛ by auto
  with prems * show ?thesis by auto
next
case (reset l)
with ⟨?u' ∈ V⟩ have ?u x ≤ ?u' x by (cases x ∈ set l) (auto simp: V-def)
from j' have zero x ([?u]ℛ) by auto

```

with $\langle ?u' \in V \rangle$ **have** $?u' x = 0$ **unfolding** *zero-def* **by** *auto*
with $\langle ?u x \leq \rightarrow \langle ?u x > 0 \rangle$ **show** *?thesis* **by** *auto*
next
case *id*
with $* \text{Succ-refl}[\text{of } \mathcal{R} \ \mathcal{X} \ k, \text{folded } \mathcal{R}\text{-def}, \text{OF} - \text{finite}(1)]$ **show** *?thesis*
unfolding *cval-add-def* **by** *auto*
qed
then obtain *t* **where** $t: ?u = xs !! (j' - 1) \oplus t \ t \geq 0$ **by** *auto*
note $\text{Succ} = \text{Succ}[\text{THEN } \text{conjunct1}]$

show *?thesis*
proof (*cases* $\exists c \in X2. \exists d :: \text{nat}. ?u \ c = d$)
case *True*
from *True* **obtain** *c* **and** $d :: \text{nat}$ **where** *c*:
 $c \in \mathcal{X} \ c \in X2 \ ?u \ c = d$
by (*auto simp: X2-def*)
have $?u \ x > 0$ **by** *fact*
from *pairwise-Suc[OF eq-elapsd, of j' - 1]* $\langle j' > j \rangle$ **have**
 $\text{eq-elapsd } (xs !! (j' - 1)) \ ?u$
by *auto*
moreover from
 $\text{elapsd-eq}[\text{OF this } \langle x \in \mathcal{X} \rangle \langle (xs !! (j' - 1)) \ x = 0 \rangle \langle (xs !! j') \ x > 0 \rangle]$
have $\text{elapsd } (xs !! (j' - 1)) \ (xs !! j') > 0$
by *auto*
ultimately have
 $?u \ c - (xs !! (j' - 1)) \ c > 0$
using $\langle c \in \mathcal{X} \rangle$ **unfolding** *eq-elapsd-def* **by** *auto*
moreover from *in-S* **have** $xs !! (j' - 1) \in V$ **by** (*auto simp: stream.pred-set*)
ultimately have $?u \ c > 0$ **using** $\langle c \in \mathcal{X} \rangle$ **unfolding** *V-def* **by** *auto*
from *region' in-S* $\langle c \in \mathcal{X} \rangle$ **have** $\text{intv-elem } c \ ?u \ (I' \ c)$
by (*force simp: stream.pred-set*)
with $\langle ?u \ c = d \rangle \langle ?u \ c > 0 \rangle$ **have** $?u \ c \geq 1$ **by** *auto*
moreover have $(xs !! (j' - 1)) \ c \leq 0.5$
proof –
have $(xs !! (j' - 1)) \ c \leq (xs !! j) \ c$

using $j'(1,3)$
by (*auto intro: resets-mono[OF eq-elapsd in-S trans - $\langle x \in \mathcal{X} \rangle \langle c \in \mathcal{X} \rangle$]*)
also have $\dots \leq ?d2$ **using** *j-c-bound[OF $\langle c \in X2 \rangle$]* .
also from $\langle ?d1 = ?d2 \rangle \langle \neg 5 / 10 \leq \rightarrow \rangle$ **have** $\dots \leq 0.5$ **by** *simp*
finally show *?thesis* .
qed
moreover have $?d2 \geq 0$ **using** *in-S* $\langle x \in \mathcal{X} \rangle$ **by** (*auto simp: V-def stream.pred-set*)
ultimately have $?u \ c - (xs !! (j' - 1)) \ c \geq (1 - ?d2) / 2$ **by** *auto*
with *t* **have** $t \geq (1 - ?d2) / 2$ **unfolding** *cval-add-def* **by** *auto*
with *t* **show** *?thesis* **by** (*auto intro: that*)
next
case *F: False*
have *not-const*: $\neg \text{isConst } (I' \ c)$ **if** $c \in \mathcal{X}$ **for** *c*
proof (*rule ccontr, simp*)
assume *A*: $\text{isConst } (I' \ c)$
show *False*
proof (*cases* $c \in X1$)
case *True*
with *X1* $\langle j' > j \rangle \langle j > \rightarrow \rangle$ **have** *unbounded* $c \ ([xs !! j]_{\mathcal{R}})$ **by** *auto*
with *unbounded-Greater* $\langle c \in \mathcal{X} \rangle$ *region'* **have** $\text{isGreater } (I' \ c)$ **by** *force*
with *A* **show** *False* **by** *auto*
next
case *False*
with $\langle c \in \mathcal{X} \rangle$ **have** $c \in X2$ **unfolding** *X2-def* **by** *auto*
from *region' in-S* $\langle c \in \mathcal{X} \rangle$ **have** $\text{intv-elem } c \ ?u \ (I' \ c)$

unfolding *stream.pred-set* **by** *force*
with $\langle c \in X2 \rangle A \text{ False } F$ **show** *False* **by** *auto*
qed
qed
have $\nexists x. x \leq k \ c \wedge (xs \ !! \ j') \ c = \text{real } x$ **if** $c \in \mathcal{X}$ **for** c
proof (*cases* $c \in X2$; *safe*)
fix d
assume $c \in X2 \ (xs \ !! \ j') \ c = \text{real } d$
with F **show** *False* **by** *auto*
next
fix d
assume $c \notin X2$
with *that* **have** $c \in X1$ **unfolding** *X2-def* **by** *auto*
with $X1 \ \langle j' > j \rangle \ \langle j > i \rangle$ **have** *unbounded* $c \ ([?u]_{\mathcal{R}})$ **by** *auto*
from *unbounded-all*[*OF* - - *this*] $\langle c \in \mathcal{X} \rangle$ *in-S* **have** $?u \ c > k \ c$
by (*force simp: stream.pred-set*)
moreover **assume** $?u \ c = \text{real } d \ d \leq k \ c$
ultimately **show** *False* **by** *auto*
qed
with *delayedR-aux* **have**
 $(xs \ !! \ j') = \text{delayedR} \ ([xs \ !! \ j']_{\mathcal{R}}) \ (xs \ !! \ (j' - 1))$
using $\langle x \in \mathcal{X} \rangle$ **unfolding** *trans'-def* **by** *auto*
from *not-const region'(1)* *in-S Succ(1)* **have**
 $\exists t \geq 0. \text{delayedR} \ ([xs \ !! \ j']_{\mathcal{R}}) \ (xs \ !! \ (j' - 1)) = xs \ !! \ (j' - 1) \oplus t \wedge$
 $(1 - \text{Max} \ (?S' \cup \{0\})) / 2 \leq t$
apply *simp*
apply (*rule delayedR-correct(2)*[*OF* - - *region'(2), simplified*])
by (*auto simp: stream.pred-set*)
with $le \ \langle - = \text{delayedR} \ - \ \rangle$ **show** *?thesis* **by** (*auto intro: that*)
qed
qed
moreover **from** *pairwise-Suc*[*OF eq-elapsed, of j' - 1*] $\langle j' > 0 \rangle$ **have**
eq-elapsed $(xs \ !! \ (j' - 1)) \ (xs \ !! \ j')$
by *auto*
ultimately **show** $\text{dur } xs \ j' - \text{dur } xs \ (j' - 1) \geq (1 - ?d2) / 2$
using $\langle j' > 0 \rangle$ *dur-Suc*[*of - j' - 1*] $\langle x \in \mathcal{X} \rangle$ **by** (*auto simp: cval-add-def elapsed-eq*)
qed
moreover **from** *dur-mono*[*of i j - 1 xs*] $\langle i < j \rangle$ **have** $\text{dur } xs \ i \leq \text{dur } xs \ (j - 1)$ **by** *simp*
ultimately **have** $\text{dur } xs \ j' - \text{dur } xs \ i \geq 0.5$ **unfolding** $\langle ?d1 = ?d2 \rangle$ [*symmetric*] **by** *auto*
then **show** *?thesis* **using** $\langle j < j' \rangle$ **by** - (*rule exI*[*where x = j'*]; *auto*)
qed
qed
moreover
have $\exists j' \geq i. \text{dur } xs \ j' - \text{dur } xs \ i \geq 0.5$ **for** i
proof -
from *calculation(1)*[*of i*] **obtain** $j \ k \ x$ **where**
 $j \geq i \ k > j \ x \in X2 \ \text{zero } x \ ([xs \ !! \ j]_{\mathcal{R}})$
 $\text{zero } x \ ([xs \ !! \ k]_{\mathcal{R}})$
 $\forall m. j < m \wedge m < k \longrightarrow \neg \text{zero } x \ ([xs \ !! \ m]_{\mathcal{R}})$
 $\forall x \in X2. \exists m > j. m \leq k \wedge \text{zero } x \ ([xs \ !! \ m]_{\mathcal{R}})$
 $\forall x \in X1. \forall m \geq j. \text{unbounded } x \ ([xs \ !! \ m]_{\mathcal{R}})$
by *auto*
from *calculation(2)*[*OF this(3,2,4-8)*] **obtain** j' **where**
 $j' \geq k \ 5 / 10 \leq \text{dur } xs \ j' - \text{dur } xs \ j$
by *auto*
with *dur-mono*[*of i j xs*] $\langle j \geq i \rangle \ \langle k > j \rangle$ **show** *?thesis* **by** (*intro exI*[*where x = j'*]; *auto*)
qed
then **show** *?thesis* **by** - (*rule dur-ev-exceedsI*[*where d = 0.5*]; *auto*)
qed
qed

lemma *cfg-on-div-abs*:
notes *in-space-UNIV*[*measurable*]
assumes $cfg \in \text{cfg-on-div}$ $st \in S$
shows $\text{absc } cfg \in R\text{-G-cfg-on-div } (\text{abss } st)$
proof –
from *assms* **have** $*$: $cfg \in \text{MDP.cfg-on}$ st $\text{state } cfg = st \text{ div-cfg } cfg$
unfolding *cfg-on-div-def* **by** *auto*
with *assms* **have** $cfg \in \text{valid-cfg}$ **by** (*auto intro: MDP.valid-cfgI*)
have *almost-everywhere* ($\text{MDP.MC.T } cfg$) ($\text{MDP.MC.enabled } cfg$)
by (*rule MDP.MC.AE-T-enabled*)
moreover from $*$ **have** *AE* x *in* $\text{MDP.MC.T } cfg$. *divergent* ($\text{smap } (snd \circ \text{state}) x$)
by (*simp add: div-cfg-def*)
ultimately have *AE* x *in* $\text{MDP.MC.T } cfg$. $\mathcal{R}\text{-div}$ ($\text{smap } (snd \circ \text{state}) (\text{smap } \text{absc } x)$)
proof *eventually-elim*
case (*elim* ω)
let $?xs = \text{smap } (snd \circ \text{state}) \omega$
from $\text{MDP.pred-stream-cfg-on}$ [*OF* $\langle - \in \text{valid-cfg} \rangle \langle \text{MDP.MC.enabled } - \rightarrow \rangle$] **have** $*$:
pred-stream ($\lambda x. x \in S$) ($\text{smap } \text{state } \omega$)
by (*auto simp: stream.pred-set*)
have $[\text{snd } (\text{state } x)]_{\mathcal{R}} = \text{snd } (\text{abss } (\text{state } x))$ **if** $x \in \text{sset } \omega$ **for** x
proof –
from $*$ **that** **have** $\text{state } x \in S$ **by** (*auto simp: stream.pred-set*)
then **have** $\text{snd } (\text{abss } (\text{state } x)) = [\text{snd } (\text{state } x)]_{\mathcal{R}}$ **by** (*metis abss-S snd-conv surj-pair*)
then **show** *?thesis* ..
qed
then **have** $\text{smap } (\lambda z. [\text{snd } (\text{state } z)]_{\mathcal{R}}) \omega = (\text{smap } (\lambda z. \text{snd } (\text{abss } (\text{state } z)))) \omega$ **by** *auto*
from $*$ **have** *pred-stream* ($\lambda u. u \in V$) $?xs$
apply (*simp add: map-def stream.pred-set*)
apply (*subst (asm) surjective-pairing*)
using $S\text{-}V$ **by** *blast*
moreover **have** *stream-trans* $?xs$
by (*rule enabled-stream-trans* $\langle - \in \text{valid-cfg} \rangle \langle \text{MDP.MC.enabled } - \rightarrow \rangle$)
ultimately show *?case* **using** $\langle \text{divergent } \rightarrow \rangle \langle \text{smap } - \omega = \rightarrow \rangle$
by – (*drule divergent- \mathcal{R} -divergent, auto simp add: stream.map-comp state-absc*)
qed
with $\langle cfg \in \text{valid-cfg} \rangle$ **have** $R\text{-G-div-cfg } (\text{absc } cfg)$ **unfolding** *R-G-div-cfg-def*
by (*subst absc-distr-self*) (*auto intro: MDP.valid-cfgI simp: AE-distr-iff*)
with $R\text{-G.valid-cfgD}$ $\langle cfg \in \text{valid-cfg} \rangle$ $*$ **show** *?thesis* **unfolding** *R-G-cfg-on-div-def* **by** *auto force*
qed

definition

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

lemma *K-cfg-same-loc-iff*:

$(\forall cfg' \in K\text{-cfg } cfg. \text{fst } (\text{state } cfg') = \text{fst } (\text{state } cfg))$
 $\longleftrightarrow (\forall cfg' \in K\text{-cfg } (\text{absc } cfg). \text{fst } (\text{state } cfg') = \text{fst } (\text{state } (\text{absc } cfg)))$
if $cfg \in \text{valid-cfg}$
using *that* **by** (*auto simp: state-absc fst-abss K-cfg-map-absc*)

lemma (*in* –) *stream-all2-flip*:

stream-all2 ($\lambda a b. R b a$) $xs\ ys = \text{stream-all2 } R\ ys\ xs$
by (*standard; coinduction arbitrary: xs ys; auto dest: sym*)

lemma *AE-alw-ev-same-loc-iff*:

assumes $cfg \in \text{valid-cfg}$
shows *alternating* $cfg \longleftrightarrow \text{alternating } (\text{absc } cfg)$
unfolding *alternating-def*
apply (*simp add: MDP.MC.T.AE-iff-emeasure-eq-1*)
subgoal

```

proof –
  show ?thesis (is (?x = 1) = (?y = 1))
  proof –
    have *: stream-all2 ( $\lambda s t. t = \text{absc } s$ )  $x y = \text{stream-all2 } (=) y (\text{smap } \text{absc } x)$  for  $x y$ 
      by (subst stream-all2-flip) simp
    have ?x = ?y
    apply (rule T-eq-rel-half[where  $f = \text{absc}$  and  $S = \text{valid-cfg}$ , OF HOL.refl, rotated 2])
    subgoal
      apply (simp add: space-stream-space rel-set-strong-def)
      apply (intro allI impI)
      apply (frule stream.rel-mono-strong[where  $Ra = \lambda s t. t = \text{absc } s$ ])
      by (auto simp: * stream.rel-eq stream-all2-refl alw-holds-pred-stream-iff[symmetric]
        K-cfg-same-loc-iff HLD-def elim!: alw-ev-cong)
    subgoal
      by (rule rel-funI) (auto intro!: rel-pmf-reflI simp: pmf.rel-map(2) K-cfg-map-absc)
    using ⟨cfg ∈ valid-cfg⟩ by simp+
    then show ?thesis
      by simp
  qed
qed
done

```

```

lemma AE-alw-ev-same-loc-iff':
  assumes  $\text{cfg} \in R\text{-G.cfg-on } (\text{abss } st) st \in S$ 
  shows alternating cfg  $\longleftrightarrow$  alternating (repcs st cfg)
proof –
  from assms have  $\text{cfg} \in R\text{-G.valid-cfg}$ 
    by (auto intro: R-G.valid-cfgI)
  with assms show ?thesis
    by (subst AE-alw-ev-same-loc-iff') (auto simp: absc-repcs-id)
qed

```

```

lemma (in –) cval-add-non-id:
  False if  $b \oplus d = b d > 0$  for  $d :: \text{real}$ 
proof –
  from that(1) have  $(b \oplus d) x = b x$ 
    by (rule fun-cong)
  with ⟨ $d > 0$ ⟩ show False
    unfolding cval-add-def by simp
qed

```

```

lemma repcs-unbounded-AE-non-loop-end-strong:
  assumes  $\text{cfg} \in R\text{-G.cfg-on } (\text{abss } st) st \in S$ 
  and alternating cfg
  shows AE  $\omega$  in MDP.MC.T (repcs st cfg).
    ( $\forall u :: ('c \Rightarrow \text{real}). (\forall c \in \mathcal{X}. u c > \text{real } (k c)) \longrightarrow$ 
       $\neg (\text{ev } (\text{alw } (\lambda xs. \text{shd } xs = u))) (\text{smap } (\text{snd } o \text{state}) \omega))$ ) (is AE  $\omega$  in ?M. ?P  $\omega$ )
proof –
  from assms have  $\text{cfg} \in R\text{-G.valid-cfg}$ 
    by (auto intro: R-G.valid-cfgI)
  with assms(1) have repcs st  $\text{cfg} \in \text{valid-cfg}$ 
    by auto
  from R-G.valid-cfgD[OF ⟨ $\text{cfg} \in R\text{-G.valid-cfg}$ ⟩] have  $\text{cfg} \in R\text{-G.cfg-on } (\text{state } \text{cfg})$  .
  let ?U =  $\lambda u. \bigcup l \in L. \{\mu \in K(l, u). \mu \neq \text{return-pmf}(l, u) \wedge (\forall x \in \mu. \text{fst } x = l)\}$ 
  let ?r =  $\lambda u. \text{Sup } (\{0\} \cup (\lambda \mu. \text{measure-pmf } \mu \{x. \text{snd } x = u\})) \text{ ' ?U } u$ 
  have lt-1: ?r  $u < 1$  for  $u$ 
  proof –
    have *:  $\text{emeasure } (\text{measure-pmf } \mu) \{x. \text{snd } x = u\} < 1$ 
      if  $\mu \neq \text{return-pmf}(l, u) \forall x \in \text{set-pmf } \mu. \text{fst } x = l$  for  $\mu$  and  $l :: 's$ 
    proof (rule ccontr)
      assume  $\neg \text{emeasure } (\text{measure-pmf } \mu) \{x. \text{snd } x = u\} < 1$ 

```

```

then have 1 = emeasure (measure-pmf  $\mu$ ) { $x$ . snd  $x$  =  $u$ }
  using measure-pmf.emeasure-ge-1-iff by force
also from that(2) have ...  $\leq$  emeasure (measure-pmf  $\mu$ ) {( $l$ ,  $u$ )}
  by (subst emeasure-Int-set-pmf[symmetric]) (auto intro!: emeasure-mono)
finally show False
  by (simp add: measure-pmf.emeasure-ge-1-iff measure-pmf-eq-1-iff that(1))
qed
let ?S =
  {map-pmf ( $\lambda$  ( $X$ ,  $l$ ). ( $l$ , ([ $X$  := 0] $u$ )))  $\mu$  |  $\mu$   $l$   $g$ . ( $l$ ,  $g$ ,  $\mu$ )  $\in$  trans-of  $A$ }
have ( $\lambda$   $\mu$ . measure-pmf  $\mu$  { $x$ . snd  $x$  =  $u$ }) ' ?U  $u$ 
   $\subseteq$  {0, 1}  $\cup$  ( $\lambda$   $\mu$ . measure-pmf  $\mu$  { $x$ . snd  $x$  =  $u$ }) ' ?S
  by (force elim!: K.cases)
moreover have finite ?S
proof -
  have ?S  $\subseteq$  ( $\lambda$  ( $l$ ,  $g$ ,  $\mu$ ). map-pmf ( $\lambda$  ( $X$ ,  $l$ ). ( $l$ , ([ $X$  := 0] $u$ )))  $\mu$ ) ' trans-of  $A$ 
    by force
  also from finite(3) have finite ... ..
  finally show ?thesis .
qed
ultimately have finite (( $\lambda$   $\mu$ . measure-pmf  $\mu$  { $x$ . snd  $x$  =  $u$ }) ' ?U  $u$ )
  by (auto intro: finite-subset)
then show ?thesis
  by (fastforce intro: * finite-imp-Sup-less)
qed
{ fix  $l$  :: 's and  $u$  :: 'c  $\Rightarrow$  real and  $cfg$  :: ('s  $\times$  ('c  $\Rightarrow$  real) set)  $cfg$ 
  assume unbounded:  $\forall$   $c \in \mathcal{X}$ .  $u$   $c >$   $k$   $c$  and  $cfg \in R-G.cfg-on$  (abss ( $l$ ,  $u$ )) abss ( $l$ ,  $u$ )  $\in \mathcal{S}$ 
  and same-loc:  $\forall$   $cfg' \in K-cfg$   $cfg$ . fst (state  $cfg'$ ) =  $l$ 
  then have  $cfg \in R-G.valid-cfg$  repcs ( $l$ ,  $u$ )  $cfg \in valid-cfg$ 
    by (auto intro: R-G.valid-cfgI)
  then have  $cfg-on$ : repcs ( $l$ ,  $u$ )  $cfg \in MDP.cfg-on$  ( $l$ ,  $u$ )
    by (auto dest: MDP.valid-cfgD)
  from  $\langle cfg \in R-G.cfg-on \rightarrow$  have action  $cfg \in \mathcal{K}$  (abss ( $l$ ,  $u$ ))
    by (rule R-G.cfg-onD-action)

  have K-cfg-rept: state ' K-cfg (repcs ( $l$ ,  $u$ )  $cfg$ ) = rept ( $l$ ,  $u$ ) (action  $cfg$ )
    unfolding K-cfg-def by (force simp: action-repcs)
  have  $l \in L$ 
    using MDP.valid-cfg-state-in-S  $\langle repcs$  ( $l$ ,  $u$ )  $cfg \in MDP.valid-cfg$  by fastforce
  moreover have rept ( $l$ ,  $u$ ) (action  $cfg$ )  $\neq$  return-pmf ( $l$ ,  $u$ )
proof (rule ccontr, simp)
  assume rept ( $l$ ,  $u$ ) (action  $cfg$ ) = return-pmf ( $l$ ,  $u$ )
  then have action  $cfg$  = return-pmf (abss ( $l$ ,  $u$ ))
    using abst-rept-id[OF  $\langle$ action  $cfg \in \rightarrow$ ]
    by (simp add: abst-def)
  moreover have ( $l$ ,  $u$ )  $\in S$ 
    using  $\langle - \in \mathcal{S} \rangle$  by (auto dest: S-abss-S)
  moreover have abss ( $l$ ,  $u$ ) = ( $l$ , [ $u$ ] $\mathcal{R}$ )
    by (metis abss-S calculation(2))
  ultimately show False
    using  $\langle$ rept ( $l$ ,  $u$ ) =  $\rightarrow$   $\rightarrow$  unbounded unfolding rept-def by (auto dest: eval-add-non-id)
qed
moreover have rept ( $l$ ,  $u$ ) (action  $cfg$ )  $\in K$  ( $l$ ,  $u$ )
proof -
  have action (repcs ( $l$ ,  $u$ )  $cfg$ )  $\in K$  ( $l$ ,  $u$ )
    using  $cfg-on$  by blast
  then show ?thesis
    by (simp add: repcs-def)
qed
moreover have  $\forall x \in set-pmf$  (rept ( $l$ ,  $u$ ) (action  $cfg$ )). fst  $x$  =  $l$ 
  using same-loc K-cfg-same-loc-iff[of repcs ( $l$ ,  $u$ )  $cfg$ ]
   $\langle repcs$  ( $l$ ,  $u$ ) =  $valid-cfg$   $\langle$   $cfg \in R-G.valid-cfg$   $\langle$   $cfg \in R-G.cfg-on \rightarrow$ 

```

```

  by (simp add: absc-repcs-id fst-abss K-cfg-rept[symmetric])
ultimately have rept (l, u) (action cfg) ∈ ?U u
  by blast
then have measure-pmf (rept (l, u) (action cfg)) {x. snd x = u} ≤ ?r u
  by (fastforce intro: Sup-upper)
moreover have rept (l, u) (action cfg) = action (repcs (l, u) cfg)
  by (simp add: repcs-def)
ultimately have measure-pmf (action (repcs (l, u) cfg)) {x. snd x = u} ≤ ?r u
  by auto
}
note * = this
let ?S = {cfg. ∃ cfg' s. cfg' ∈ R-G.valid-cfg ∧ cfg = repcs s cfg' ∧ abss s = state cfg'}
have start: repcs st cfg ∈ ?S
  using ⟨cfg ∈ R-G.valid-cfg⟩ assms unfolding R-G-cfg-on-div-def
  by clarsimp (inst-existentials cfg fst st snd st, auto)
have step: y ∈ ?S if y ∈ K-cfg x x ∈ ?S for x y
  using that apply safe
  subgoal for cfg' l u
    apply (inst-existentials absc y state y)
    subgoal
      by blast
    subgoal
      by (metis
        K-cfg-valid-cfgD R-G.valid-cfgD R-G.valid-cfg-state-in-S absc-repcs-id cont-absc-1
        cont-repcs1 repcs-valid
      )
    subgoal
      by (simp add: state-absc)
    done
  done
have **: x ∈ ?S if (repcs st cfg, x) ∈ MDP.MC.acc for x
proof –
  from MDP.MC.acc-relfunD[OF that] obtain n where ((λ a b. b ∈ K-cfg a)  $\rightsquigarrow$  n) (repcs st cfg) x .
  then show ?thesis
  proof (induction n arbitrary: x)
    case 0
    with start show ?case
      by simp
    next
    case (Suc n)
    from this(2)[simplified] show ?case
      apply (rule relcomppE)
      apply (erule step)
      apply (erule Suc.IH)
      done
  qed
qed
have ***: almost-everywhere (MDP.MC.T (repcs st cfg)) (alw (HLD ?S))
  by (rule AE-mp[OF MDP.MC.AE-T-reachable]) (fastforce dest: ** simp: HLD-iff elim: alw-mono)

from ⟨alternating cfg⟩ assms have alternating (repcs st cfg)
  by (simp add: AE-alw-ev-same-loc-iff'[of - st])
then have alw-ev-same2: almost-everywhere (MDP.MC.T (repcs st cfg))
  (alw (λω. HLD (state - ' snd - ' {u}) ω  $\longrightarrow$ 
    ev (HLD {cfg. ∃ cfg' ∈ set-pmf (K-cfg cfg). fst (state cfg') = fst (state cfg)})) ω))
  for u unfolding alternating-def by (auto elim: alw-mono)

let ?X = {cfg :: ('s × ('c ⇒ real)) cfg. ∀ c ∈ X. snd (state cfg) c > k c}
let ?Y = {cfg. ∃ cfg' ∈ K-cfg cfg. fst (state cfg') = fst (state cfg)}

have (AE ω in ?M. ?P ω)  $\longleftrightarrow$ 

```

```

(AE  $\omega$  in ?M.  $\forall u :: ('c \Rightarrow \text{real})$ .
  ( $\forall c \in \mathcal{X}. u c > k c) \wedge u \in \text{snd } ' \text{state } ' (MDP.MC.\text{acc } " \{ \text{repcs st cfg} \}) \longrightarrow$ 
   $\neg (ev (alw (\lambda xs. \text{shd } xs = u))) (smap (\text{snd } o \text{state}) \omega)) (\text{is } ?L \longleftrightarrow ?R)$ )
proof
  assume ?L
  then show ?R
  by eventually-elim auto
next
  assume ?R
  with MDP.MC.AE-T-reachable[of repcs st cfg] show ?L
  proof (eventually-elim, intro allI impI notI, goal-cases)
    case (1  $\omega$  u)
    then show ?case
    by - (intro alw-HLD-smap alw-disjoint-contr[where
      S = (snd o state) ' MDP.MC.\text{acc } " \{ \text{repcs st cfg} \}
      and R = \{ u \} and  $\omega = \text{smap } (\text{snd } o \text{state}) \omega$ 
      ]; auto simp: HLD-iff)
  qed
qed

also have ...  $\longleftrightarrow$ 
  ( $\forall u :: ('c \Rightarrow \text{real})$ .
    ( $\forall c \in \mathcal{X}. u c > k c) \wedge u \in \text{snd } ' \text{state } ' (MDP.MC.\text{acc } " \{ \text{repcs st cfg} \}) \longrightarrow$ 
    (AE  $\omega$  in ?M.  $\neg (ev (alw (\lambda xs. \text{shd } xs = u))) (smap (\text{snd } o \text{state}) \omega))$ )
    using MDP.MC.countable-reachable[of repcs st cfg]
    by - (rule AE-all-imp-countable,
      auto intro: countable-subset[where B = snd ' state ' MDP.MC.\text{acc } " \{ \text{repcs st cfg} \}])
  also show ?thesis
  unfolding calculation
  apply clarsimp
  subgoal for l u x
  apply (rule
    MDP.non-loop-tail-strong[simplified, of snd snd (state x) ?Y ?S ?r (snd (state x))]
  )
  subgoal
  apply safe
  subgoal premises prems for cfg l1 u1 - cfg' l2 u2
  proof -
    have [simp]: l2 = l1 u2 = u1
    subgoal
    by (metis MDP.cfg-onD-state Pair-inject prems(4) state-repcs)
    subgoal
    by (metis MDP.cfg-onD-state prems(4) snd-conv state-repcs)
    done
  with prems have [simp]: u2 = u
    by (metis  $\langle l, u \rangle = \text{state } x \rangle \langle \text{snd } (l1, u1) = \text{snd } (\text{state } x) \rangle \langle u2 = u1 \rangle \text{snd-conv}$ )
  have [simp]:  $\text{snd } - ' \{ \text{snd } (\text{state } x) \} = \{ y. \text{snd } y = \text{snd } (\text{state } x) \}$ 
    by (simp add: vimage-def)
  from prems show ?thesis
  apply simp
  apply (erule *[simplified])
  subgoal
  using prems(1) prems(2)[symmetric] prems(3-) by (auto simp: R-G.valid-cfg-def)
  subgoal
  using prems(1) prems(2)[symmetric] prems(3-) by (auto simp: R-G.valid-cfg-def)
  subgoal
  using K-cfg-same-loc-iff[of repcs (l1, snd (state x)) cfg']
  by (simp add: absc-repcs-id) (metis fst-abss fst-conv repcs-valid)
  done
  qed
done

```

```

subgoal
  by (auto intro: lt-1[simplified])
  apply (rule MDP.valid-cfgD[OF ‹repcs st cfg ∈ valid-cfg›]; fail)
subgoal
  using *** unfolding alw-holds-pred-stream-iff[symmetric] HLD-def .
subgoal
  by (rule alw-ev-same2)
done
done
qed

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 -
  let ?st = abss st
  let ?cfg = repcs st cfg
  from assms have *:
    cfg ∈ R-G.cfg-on ?st state cfg = ?st R-G-div-cfg cfg
  unfolding R-G-cfg-on-div-def by auto
  with assms have cfg ∈ R-G.valid-cfg by (auto intro: R-G.valid-cfgI)
  with ‹st ∈ S› ‹- = ?st› have ?cfg ∈ valid-cfg by auto
  from *(1) ‹st ∈ S› ‹alternating cfg› have
    AE ω in MDP.MC.T ?cfg. ∀ u. (∀ c ∈ X. real (k c) < u c) ⟶
      ¬ ev (alw (λxs. shd xs = u)) (smap (snd ∘ state) ω)
  by (rule repcs-unbounded-AE-non-loop-end-strong)
  — Move to lower level
  moreover from *(2,3) have AE ω in MDP.MC.T ?cfg. R-div (smap (snd ∘ state) (smap absc ω))
    unfolding R-G-div-cfg-def
    by (subst (asm) R-G-trace-space-distr-eq[OF ‹cfg ∈ R-G.valid-cfg›]; simp add: AE-distr-iff)
  ultimately have div-cfg ?cfg
    unfolding div-cfg-def using MDP.MC.AE-T-enabled[of ?cfg]
  proof eventually-elim
    case prems: (elim ω)
    let ?xs = smap (snd ∘ state) ω
    from MDP.pred-stream-cfg-on[OF ‹- ∈ valid-cfg› ‹MDP.MC.enabled - -›] have *:
      pred-stream (λ x. x ∈ S) (smap state ω)
    by (auto simp: stream.pred-set)
    have [snd (state x)]R = snd (abss (state x)) if x ∈ sset ω for x
    proof -
      from * that have state x ∈ S by (auto simp: stream.pred-set)
      then have snd (abss (state x)) = [snd (state x)]R by (metis abss-S snd-conv surj-pair)
      then show ?thesis ..
    qed
    then have smap (λz. [snd (state z)]R) ω = (smap (λz. snd (abss (state z)))) ω by auto
    from * have pred-stream (λ u. u ∈ V) ?xs
      by (simp add: map-def stream.pred-set, subst (asm) surjective-pairing, blast)
    moreover have stream-trans ?xs
      by (rule enabled-stream-trans ‹- ∈ valid-cfg› ‹MDP.MC.enabled - -›)+
    moreover have pairwise trans' ?xs
      using ‹- ∈ R-G.valid-cfg› ‹state cfg = -›[symmetric] ‹MDP.MC.enabled - -›
      by (rule enabled-stream-trans')
    moreover from prems(1) have
      ∀ u. (∀ c ∈ X. real (k c) < u c) ⟶ ¬ ev (alw (λxs. snd (shd xs) = u)) (smap state ω)
    by simp
    ultimately show ?case using ‹R-div -›
      by (simp add: stream.map-comp state-absc ‹smap - ω = -› R-divergent-divergent)
  qed
  with MDP.valid-cfgD ‹cfg ∈ R-G.valid-cfg› * show ?thesis unfolding cfg-on-div-def by auto force
qed

```

lemma *repcs-unbounded-AE-non-loop-end*:

assumes $cfg \in R-G.cfg\text{-on } (abss \ st) \ st \in S$

shows $AE \ \omega \text{ in } MDP.MC.T \ (repcs \ st \ cfg)$.

$(\forall \ s :: ('s \times ('c \Rightarrow \text{real})). (\forall \ c \in \mathcal{X}. \text{snd } s \ c > k \ c) \longrightarrow$
 $\neg (ev \ (alw \ (\lambda \ xs. \ \text{shd } xs = s))) \ (\text{smap } \text{state } \omega)) \ (\text{is } AE \ \omega \text{ in } ?M. \ ?P \ \omega)$

proof –

from *assms* **have** $cfg \in R-G.valid\text{-cfg}$

by (*auto intro: R-G.valid-cfgI*)

with *assms*(1) **have** $repcs \ st \ cfg \in valid\text{-cfg}$

by *auto*

from $R-G.valid\text{-cfgD}[OF \ \langle cfg \in R-G.valid\text{-cfg} \rangle]$ **have** $cfg \in R-G.cfg\text{-on } (state \ cfg)$.

let $?K = \lambda \ x. \ \{\mu \in K \ x. \ \mu \neq \text{return-pmf } x\}$

let $?r = \lambda \ x. \ \text{Sup } ((\lambda \ \mu. \ \text{measure-pmf } \mu \ \{x\}) \ ' ?K \ x)$

have *lt-1*: $?r \ x < 1$ **if** $\mu \in ?K \ x$ **for** $\mu \ x$

proof –

have $*$: $\text{emeasure } (\text{measure-pmf } \mu) \ \{x\} < 1$ **if** $\mu \neq \text{return-pmf } x$ **for** μ

proof (*rule ccontr*)

assume $\neg \text{emeasure } (\text{measure-pmf } \mu) \ \{x\} < 1$

then **have** $\text{emeasure } (\text{measure-pmf } \mu) \ \{x\} = 1$

using *measure-pmf.emeasure-ge-1-iff* **by** *force*

with *that* **show** *False*

by (*simp add: measure-pmf-eq-1-iff*)

qed

let $?S =$
 $\{\text{map-pmf } (\lambda \ (X, l). \ (l, \ ([X := 0]u))) \ \mu \mid \mu \ l \ u \ g.$
 $\ x = (l, u) \wedge (l, g, \mu) \in \text{trans-of } A\}$

have $(\lambda \ \mu. \ \text{measure-pmf } \mu \ \{x\}) \ ' ?K \ x$
 $\subseteq \{0, 1\} \cup (\lambda \ \mu. \ \text{measure-pmf } \mu \ \{x\}) \ ' ?S$

by (*force elim!: K.cases*)

moreover **have** *finite* $?S$

proof –

have $?S \subseteq (\lambda \ (l, g, \mu). \ \text{map-pmf } (\lambda \ (X, l). \ (l, \ ([X := 0](\text{snd } x)))) \ \mu) \ ' \text{trans-of } A$

by *force*

also **from** *finite*(3) **have** *finite* \dots ..

finally **show** *?thesis* .

qed

ultimately **have** *finite* $((\lambda \ \mu. \ \text{measure-pmf } \mu \ \{x\}) \ ' ?K \ x)$

by (*auto intro: finite-subset*)

then **show** *?thesis*

using *that* **by** (*auto intro: * finite-imp-Sup-less*)

qed

{ **fix** $s :: 's \times ('c \Rightarrow \text{real})$ **and** $cfg :: ('s \times ('c \Rightarrow \text{real}) \ \text{set}) \ \text{cfg}$

assume *unbounded*: $\forall \ c \in \mathcal{X}. \ \text{snd } s \ c > k \ c$ **and** $cfg \in R-G.cfg\text{-on } (abss \ s) \ abss \ s \in S$

then **have** $repcs \ s \ cfg \in valid\text{-cfg}$

by (*auto intro: R-G.valid-cfgI*)

then **have** *cfg-on*: $repcs \ s \ cfg \in MDP.cfg\text{-on } s$

by (*auto dest: MDP.valid-cfgD*)

from $\langle cfg \in \cdot \rangle$ **have** *action* $cfg \in \mathcal{K} \ (abss \ s)$

by (*rule R-G.cfg-onD-action*)

have *rept* $s \ (action \ cfg) \neq \text{return-pmf } s$

proof (*rule ccontr, simp*)

assume $\text{rept } s \ (action \ cfg) = \text{return-pmf } s$

then **have** *action* $cfg = \text{return-pmf } (abss \ s)$

using *abst-rept-id[OF \langle action \ cfg \in \cdot \rangle]*

by (*simp add: abst-def*)

moreover **have** $(fst \ s, \ \text{snd } s) \in S$

using $\langle \cdot \in S \rangle$ **by** (*auto dest: S-abss-S*)

moreover **have** $abss \ s = (fst \ s, \ [\text{snd } s]_{\mathcal{R}})$

by (*metis abss-S calculation(2) prod.collapse*)

ultimately **show** *False*

```

    using ⟨rept s - = -⟩ unbounded unfolding rept-def by (cases s) (auto dest: cval-add-non-id)
  qed
  moreover have rept s (action cfg) ∈ K s
  proof -
    have action (repcs s cfg) ∈ K s
      using cfg-on by blast
    then show ?thesis
      by (simp add: repcs-def)
  qed
  ultimately have rept s (action cfg) ∈ ?K s
    by blast
  then have measure-pmf (rept s (action cfg)) {s} ≤ ?r s
    by (auto intro: Sup-upper)
  moreover have rept s (action cfg) = action (repcs s cfg)
    by (simp add: repcs-def)
  ultimately have measure-pmf (action (repcs s cfg)) {s} ≤ ?r s
    by auto
  note this ⟨rept s (action cfg) ∈ ?K s⟩
}
note * = this
let ?S = {cfg. ∃ cfg' s. cfg' ∈ R-G.valid-cfg ∧ cfg = repcs s cfg' ∧ abss s = state cfg'}
have start: repcs st cfg ∈ ?S
  using ⟨cfg ∈ R-G.valid-cfg⟩ assms unfolding R-G.cfg-on-div-def
  by clarsimp (inst-existentials cfg fst st snd st, auto)
have step: y ∈ ?S if y ∈ K-cfg x x ∈ ?S for x y
  using that apply safe
  subgoal for cfg' l u
    apply (inst-existentials absc y state y)
    subgoal
      by blast
    subgoal
      by (metis
        K-cfg-valid-cfgD R-G.valid-cfgD R-G.valid-cfg-state-in-S absc-repcs-id cont-absc-1
        cont-repcs1 repcs-valid
      )
    subgoal
      by (simp add: state-absc)
    done
  done
have ** : x ∈ ?S if (repcs st cfg, x) ∈ MDP.MC.acc for x
proof -
  from MDP.MC.acc-relfunD[OF that] obtain n where  $((\lambda a b. b \in K\text{-cfg } a) \rightsquigarrow n)$  (repcs st cfg) x .
  then show ?thesis
  proof (induction n arbitrary: x)
    case 0
      with start show ?case
        by simp
    next
      case (Suc n)
        from this(2)[simplified] show ?case
          by (elim relcompPE step Suc.IH)
  qed
qed
have *** : almost-everywhere (MDP.MC.T (repcs st cfg)) (alw (HLD ?S))
  by (rule AE-mp[OF MDP.MC.AE-T-reachable]) (fastforce dest: ** simp: HLD-iff elim: alw-mono)

have (AE ω in ?M. ?P ω)  $\longleftrightarrow$ 
  (AE ω in ?M. ∀ s :: ('s × ('c ⇒ real)).
    ( $\forall c \in \mathcal{X}. \text{snd } s \ c > k \ c$ )  $\wedge$  s  $\in$  state ‘ (MDP.MC.acc “ {repcs st cfg}”)  $\longrightarrow$ 
     $\neg$  (ev (alw (λ xs. shd xs = s)) (smap state ω)) (is ?L  $\longleftrightarrow$  ?R))
proof

```



```

assume ?L
then show ?R
  by eventually-elim auto
next
  assume ?R
  with MDP.MC.AE-T-reachable[of repcs st cfg] show ?L
  proof (eventually-elim, intro allI impI notI, goal-cases)
    case (1  $\omega$  s)
    from this(1,2,5,6) show ?case
    by (intro alw-HLD-smap alw-disjoint-contr[where
      S = state ‘ MDP.MC.acc “ {repcs st cfg} and R = {s} and  $\omega$  = smap state  $\omega$ 
    ]; simp add: HLD-iff; blast)
  qed
qed

also have ...  $\longleftrightarrow$ 
  ( $\forall s :: ('s \times ('c \Rightarrow \text{real}))$ .
    ( $\forall c \in \mathcal{X}$ . snd s c > k c)  $\wedge$  s  $\in$  state ‘ (MDP.MC.acc “ {repcs st cfg})  $\longrightarrow$ 
    (AE  $\omega$  in ?M.  $\neg$  (ev (alw ( $\lambda xs$ . shd xs = s))) (smap state  $\omega$ )))
  using MDP.MC.countable-reachable[of repcs st cfg]
  by - (rule AE-all-imp-countable,
    auto intro: countable-subset[where B = state ‘ MDP.MC.acc “ {repcs st cfg}])
also show ?thesis
  unfolding calculation
  apply clarsimp
  subgoal for l u x
    apply (rule MDP.non-loop-tail'[simplified, of state x ?S ?r (state x)])
  subgoal
    apply safe
    subgoal premises prems for cfg cfg' l' u'
    proof -
      from prems have state x = (l', u')
      by (metis MDP.cfg-onD-state state-repcs)
      with  $\langle \cdot = \text{state } x \rangle$  have [simp]: l = l' u = u'
      by auto
      show ?thesis
      unfolding  $\langle \text{state } x = \cdot \rangle$  using prems(1,3-) by (auto simp: R-G.valid-cfg-def intro: *)
    qed
  done
  subgoal
    apply (drule **)
    apply clarsimp
    apply (rule lt-1)
    apply (rule *)
    apply (auto dest: R-G.valid-cfg-state-in-S R-G.valid-cfgD)
  done
  apply (rule MDP.valid-cfgD[OF  $\langle \text{repcs st cfg} \in \text{valid-cfg} \rangle$ ]; fail)
  using *** unfolding alw-holds-pred-stream-iff[symmetric] HLD-def .
  done
qed
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'
using that unfolding R-G-cfg-on-div-def R-G.valid-cfg-def **by** auto

lemma *cfg-on-valid*:

cfg ∈ *valid-cfg* **if** *cfg* ∈ *cfg-on-div s*

using that unfolding *cfg-on-div-def* *MDP.valid-cfg-def* **by auto**

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'*

apply safe

subgoal

unfolding *s'-def* **by** (*rule cfg-on-div-absc*) **auto**

subgoal

by (*simp add: AE-aw-ev-same-loc-iff' cfg-on-valid*)

subgoal for *cfg*

unfolding *s'-def*

by (*frule cfg-on-div-repcs-strong*)

(*auto 4 4*

simp: s'-def R-G-cfg-on-div-def AE-aw-ev-same-loc-iff'[symmetric]

intro: R-G-cfg-on-valid absc-repcs-id[symmetric]

)

done

Main theorem:

theorem *Min-Max-reachability*:

notes *in-space-UNIV[measurable]* **and** [*iff*] = *pred-stream-iff*

shows

(\sqcup *cfg* ∈ *progressive s*. *path-measure* (λ *x*. (*holds* φ *suntil holds* ψ) (*s* $\#\#$ *x*)) *cfg*)

= (\sqcup *cfg* ∈ *R-G-progressive s'*. *R-G-path-measure* (λ *x*. (*holds* φ' *suntil holds* ψ') (*s'* $\#\#$ *x*)) *cfg*)

∧ (\prod *cfg* ∈ *progressive s*. *path-measure* (λ *x*. (*holds* φ *suntil holds* ψ) (*s* $\#\#$ *x*)) *cfg*)

= (\prod *cfg* ∈ *R-G-progressive s'*. *R-G-path-measure* (λ *x*. (*holds* φ' *suntil holds* ψ') (*s'* $\#\#$ *x*)) *cfg*)

proof (*rule SUP-eq-and-INF-eq*; *rule bexI[rotated]*; *erule IntE*)

fix *cfg* **assume** *cfg-div: cfg* ∈ *R-G-cfg-on-div s'* **and** *cfg* ∈ *Collect alternating*

then have *alternating cfg*

by auto

let *?cfg' = repcs s cfg*

from \langle *alternating cfg* \rangle *cfg-div* **have** *alternating ?cfg'*

by (*simp add: R-G-cfg-on-div-def s'-def AE-aw-ev-same-loc-iff'[of - s]*)

with *cfg-div* \langle *alternating cfg* \rangle **show** *?cfg' ∈ cfg-on-div s* ∩ *Collect alternating*

by (*auto intro: cfg-on-div-repcs-strong simp: s'-def*)

show *emeasure (R-G.T cfg) {x ∈ space R-G.St. (holds* φ' *suntil holds* ψ') (*s'* $\#\#$ *x*)}

= *emeasure (MDP.T ?cfg') {x ∈ space MDP.St. (holds* φ *suntil holds* ψ) (*s* $\#\#$ *x*)}

(*is ?a = ?b*)

proof –

from *cfg-div* **have** *cfg* ∈ *R-G.valid-cfg*

by (*rule R-G-cfg-on-valid*)

from *cfg-div* **have** *cfg* ∈ *R-G.cfg-on s'*

unfolding *R-G-cfg-on-div-def* **by auto**

then have *state cfg = s'*

by auto

have *?a = ?b*

apply (*rule*

path-measure-eq-repcs''-new[

of s cfg φ ψ , *folded* φ' -*def* ψ' -*def*, *unfolded* \langle - = *s'* \rangle *state-repcs*

]

)

```

subgoal
  unfolding s'-def ..
subgoal
  by fact
subgoal
  using ⟨?cfg' ∈ cfg-on-div s ∩ -⟩ by (blast intro: cfg-on-valid)
subgoal premises prems for xs
  using prems s by (intro φ-stream)
subgoal premises prems
  using prems s by (intro ψ-stream)
done
then show ?thesis
  by simp
qed
next
fix cfg assume cfg-div: cfg ∈ cfg-on-div s and cfg ∈ Collect alternating
with absc-valid-cfg-eq show absc cfg ∈ R-G-cfg-on-div s' ∩ Collect alternating
  by auto
show emeasure (MDP.T cfg) {x ∈ space MDP.St. (holds φ until holds ψ) (s ## x)}
  = emeasure (R-G.T (absc cfg)) {x ∈ space R-G.St. (holds φ' until holds ψ') (s' ## x)}
  (is ?a = ?b)
proof -
  have absc cfg ∈ R-G.valid-cfg
    using R-G-cfg-on-valid ⟨absc cfg ∈ R-G-cfg-on-div s' ∩ -⟩ by blast
  from cfg-div have cfg ∈ valid-cfg
    by (simp add: cfg-on-valid)
  with ⟨absc cfg ∈ R-G.valid-cfg⟩ have ?b = ?a
    by (intro MDP.alw-S R-G.alw-S path-measure-eq-absc1-new
      [where P = pred-stream (λs. s ∈ S) and Q = pred-stream (λs. s ∈ S)]
    )
    (auto simp: S-abss-S intro: S-abss-S intro!: until-abss until-reps, measurable)
  then show ?a = ?b
    by simp
qed
qed
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.