# Probabilistic Timed Automata 

Simon Wimmer and Johannes Hölzl

September 13, 2023


#### Abstract

We present a formalization of probabilistic timed automata (PTA) for which we try to follow the formula "MDP $+\mathrm{TA}=$ 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 suntil ..... 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 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 m 'b b bool) = 'a set }=>\mathrm{ 'b set }=>\mathrm{ bool
    where rel-set-strong R A B \longleftrightarrow(\forallxy.R x y \longrightarrow(x\inA\longleftrightarrowy\inB))
```

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: $\wedge s t . R s t \longleftrightarrow(s \in S \wedge f s=t)$
assumes $A[$ measurable $]: A \in$ sets (stream-space (count-space UNIV))
and $B[$ measurable $]: B \in$ sets (stream-space (count-space UNIV))
and $A B$ : rel-set-strong (stream-all2 $R$ ) $A B$ and $K L$ : rel-fun $R(r e l-p m f R) K L$ and $x y: R x y$
shows $M C$-syntax.T $K$ x $A=M C$-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 $\langle R x y$ by (auto simp: $R$-def)
define $g$ where $g t=(S O M E$ 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 $x f$, THEN rel-pmf-imp-rel-set] by (auto simp: rel-set-def $R$-def)
have in-S: $A E \omega$ 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: $L y=\operatorname{map-pmf} f(K x)$ if $x y: R x y$ for $x y$
proof -
have rel-pmf $(\lambda x y . x=y)($ map-pmf $f(K x))(L y)$
using $K L[T H E N$ 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 ? $D=\lambda x$. distr $(K . T x) K . S(\operatorname{smap} f)$
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- $D: ? D x=? D x^{\prime}$ if $R x y R x^{\prime} y$ for $x x^{\prime} y$
proof (rule stream-space-eq-sstart)
define $A$ where $A=K . a c c$ " $\left\{x, x^{\prime}\right\}$
have $x$ - $A: x \in A x^{\prime} \in A$ by (auto simp: $A$-def)
let $? \Omega=f^{\text {' }} A$
show countable? $\Omega$
unfolding $A$-def by (intro countable-image K.countable-acc) auto
show prob-space (?D $x$ ) prob-space (? $D x^{\prime}$ ) by (auto intro!: K.T.prob-space-distr)
show sets $(? D x)=$ sets $L . S$ sets $\left(? D x^{\prime}\right)=$ sets $L . S$ by auto
have $A E$-streams: $A E x$ in ? $D x^{\prime \prime} . x \in$ streams $? \Omega$ if $x^{\prime \prime} \in A$ for $x^{\prime \prime}$
apply (simp add: space-stream-space streams-sets AE-distr-iff)
using K.AE-T-reachable[of $\left.x^{\prime \prime}\right]$ unfolding alw-HLD-iff-streams
proof eventually-elim
fix $s$ assume $s \in$ streams (K.acc" $\left\{x^{\prime \prime}\right\}$ )
moreover have K.acc " $\left\{x^{\prime \prime}\right\} \subseteq A$
using $\left\langle x^{\prime \prime} \in A\right\rangle$ by (auto simp: A-def Image-def intro: rtrancl-trans)
ultimately show smap $f s \in$ streams ( $f$ ' $A$ )
by (auto intro: smap-streams)
qed
with $x$ - $A$ show $A E x$ in ? $D x^{\prime} . x \in$ streams ? $\Omega A E x$ in ? $D x . x \in$ streams ? $\Omega$
by auto
from $\langle x \in A\rangle\left\langle x^{\prime} \in A\right\rangle$ that show ? D $x\left(\operatorname{sstart}\left(f^{\prime} A\right) x s\right)=? D x^{\prime}\left(\operatorname{sstart}\left(f^{\prime} A\right) x s\right)$ for $x s$
proof (induction xs arbitrary: $x x^{\prime} y$ )
case Nil
moreover have ?D $x\left(\right.$ streams $\left.\left(f^{\prime} A\right)\right)=1$ if $x \in A$ 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 $x x^{\prime} y$ )
have rel-pmf $\left(R O O R^{-1-1}\right)(K x)\left(K x^{\prime}\right)$
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 $p::\left({ }^{\prime} s \times\right.$ 's) pmf where $p: \bigwedge a b .(a, b) \in p \Longrightarrow\left(R O O R^{-1-1}\right) a b$ and
eq: map-pmf fst $p=K x$ map-pmf snd $p=K x^{\prime}$
by (auto simp: pmf.in-rel)
let ? $S=$ stream-space (count-space UNIV)
have $*:(\# \#) y-‘ \operatorname{smap} f-‘ \operatorname{sstart}(f ‘ A)(z \# z s)=($ if $f y=z$ then smap $f-‘ \operatorname{sstart}(f ‘ A) z s$ else
\{\}) for $y z z s$
by auto
have $* *$ : ? $D x\left(\operatorname{sstart}\left(f^{\prime} A\right)(z \# z s)\right)=\left(\int^{+} y^{\prime}\right.$. (if $f y^{\prime}=z$ then ?D $y^{\prime}\left(\operatorname{sstart}\left(f^{\prime} A\right) z s\right)$ else 0$\left.) \partial K x\right)$
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 $f s t-A: f s t a b \in A$ if $a b \in p$ for $a b$
proof -
have fst $a b \in K x$ using $\langle a b \in p\rangle$ set-map-pmf [of fst $p]$ by (auto simp: eq)
with $\langle x \in A\rangle$ show $f s t a b \in A$
by (auto simp: A-def intro: rtrancl.rtrancl-into-rtrancl)
qed
have snd-A: snd $a b \in A$ if $a b \in p$ for $a b$
proof -
have snd $a b \in K x^{\prime}$ using $\langle a b \in p\rangle$ set-map-pmf [of snd $p$ ] by (auto simp: eq)
with $\left\langle x^{\prime} \in A\right\rangle$ show snd $a b \in A$
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 $a b$ using $p[o f f s t a b$ snd $a b]$ 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 〈 \(R x y\rangle\)
proof (coinduction arbitrary: \(x\) y rule: L.T-coinduct)
    case (cont x y)
    then have \(K x\)-Ly: rel-pmf \(R(K x)(L y)\)
        by (rule KL[THEN rel-funD])
    then have \(*: y^{\prime} \in L y \Longrightarrow \exists x^{\prime} \in K x . R x^{\prime} y^{\prime}\) for \(y^{\prime}\)
        by (auto dest!: rel-pmf-imp-rel-set simp: rel-set-def)
    have \(* *: y^{\prime} \in L y \Longrightarrow R\left(g y^{\prime}\right) y^{\prime}\) for \(y^{\prime}\)
        using \(*\left[\right.\) of \(\left.y^{\prime}\right]\) unfolding \(g\)-def by (auto intro: someI)
    have D-SCons-eq-D-D: distr (K.T i) K.S ( \(\lambda x . z \# \# \operatorname{smap} f x)=\operatorname{distr}(? D i) K . S(\lambda x . z \# \# x)\) for \(i z\)
        by (subst distr-distr) (auto simp: comp-def)
    have \(D\)-eq- \(D-g i\) : ? \(D i=? D(g(f i))\) if \(i: i \in K x\) for \(i\)
    proof -
        obtain \(j\) where \(j \in L y R i j f i=j\)
            using \(K x-L y\) i by (force dest!: rel-pmf-imp-rel-set simp: rel-set-def \(R\)-def)
        then show ?thesis
            by (auto intro!: \(D-e q-D[O F\langle R i j\rangle] g\) )
    qed
    have \(* * *: ~ ? D ~ x=\) measure-pmf \((L y) \gg(\lambda y . \operatorname{distr}(? D(g y)) K . S((\# \#) y))\)
        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 - \(\lambda t\). 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 \(t \longleftrightarrow(s \in\) streams \(S \wedge \operatorname{smap} f s=t)\) for \(s t\)
proof safe
    show stream-all2 \(R s t \Longrightarrow s \in\) 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 \(t \Longrightarrow\) smap \(f s=t\)
    apply (coinduction arbitrary: st 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 \(\omega \in\) streams \(S \Longrightarrow \omega \in A \longleftrightarrow \operatorname{smap} f \omega \in B\) for \(\omega\)
    using \(A B\) by (auto simp: rel-set-strong-def)
with in-S have \(K . T x A=K . T x\left(\operatorname{smap} f-{ }^{\prime} B \cap \operatorname{space}(K . T x)\right)\)
    by (auto intro!: emeasure-eq-AE streams-sets)
also have \(\ldots=(\operatorname{distr}(K . T x) K . S(\operatorname{smap} f)) B\)
    by (intro emeasure-distr[symmetric]) auto
also have \(\ldots=(L . T y) B\) unfolding \(L-e q-D .\).
```

finally show ?thesis.
qed
no-notation ccval $(\{-\}\}[100])$
hide-const succ

## 2 Additional Facts on Regions

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

Defining the closest successor of a region. Only exists if at least one interval is upper-bounded.
abbreviation is-upper-right where
is-upper-right $R \equiv(\forall t \geq 0 . \forall u \in R . u \oplus t \in R)$

## definition

succ $\mathcal{R} R \equiv$
if is-upper-right $R$ then $R$ else
$\left(\right.$ THE $R^{\prime} . R^{\prime} \neq R \wedge R^{\prime} \in \operatorname{Succ} \mathcal{R} R \wedge\left(\forall u \in R . \forall t \geq 0 .(u \oplus t) \notin R \longrightarrow\left(\exists t^{\prime} \leq t .\left(u \oplus t^{\prime}\right) \in R^{\prime} \wedge 0 \leq\right.\right.$ $\left.t^{\prime}\right)$ )
lemma region-continuous:
assumes valid-region $X k I r$
defines $R$ : $R \equiv$ region X I r
assumes between: $0 \leq t 1 t 1 \leq t 2$
assumes elem: $u \in R u \oplus t 2 \in R$
shows $u \oplus t 1 \in R$
unfolding $R$
proof
from $\langle 0 \leq t 1\rangle\langle u \in R\rangle$ show $\forall x \in X .0 \leq(u \oplus t 1) x$ by (auto simp: $R$ cval-add-def)
have intv-elem $x(u \oplus t 1)(I x)$ if $x \in X$ for $x$
proof -
from elem that have intv-elem $x u(I x)$ intv-elem $x(u \oplus t 2)(I x)$ by (auto simp: $R$ )
with between show ?thesis by (cases I x, auto simp: cval-add-def)
qed
then show $\forall x \in X$. intv-elem $x(u \oplus t 1)(I x)$ by blast
let ? $X_{0}=\{x \in X . \exists d . I x=$ Intv $d\}$
show ? $X_{0}=? X_{0} .$.
from elem have $\forall x \in ? X_{0} . \forall y \in ? X_{0} .(x, y) \in r \longleftrightarrow$ frac $(u x) \leq$ frac ( $u y$ ) by (auto simp: $R$ )
moreover
$\{$ fix $x y c d$ assume $A: x \in X y \in X I x=$ Intv $c$ I $y=$ Intv $d$
from $A$ elem between have $*$ :
$c<u x u x<c+1 c<u x+t 1 u x+t 1<c+1$
by (fastforce simp: cval-add-def $R$ )+
moreover from $A(2,4)$ elem between have $* *$ :

$$
d<u y u y<d+1 d<u y+t 1 u y+t 1<d+1
$$

by (fastforce simp: cval-add-def $R$ )+
ultimately have $u x=c+\operatorname{frac}(u x) u y=d+f r a c(u y)$ using nat-intv-frac-decomp by auto
then have
frac $(u x+t 1)=$ frac $(u x)+t 1$ frac $(u y+t 1)=$ frac $(u y)+t 1$
using $*(3,4) * *(3,4)$ nat-intv-frac-decomp by force+
then have
frac $(u x) \leq$ frac $(u y) \longleftrightarrow$ frac $((u \oplus t 1) x) \leq \operatorname{frac}((u \oplus t 1) y)$
by (auto simp: cval-add-def)
\}
ultimately show

```
    \forallx\in? . X . }\forally\in? ? X . . (x,y) \inr\longleftrightarrow frac ((u\oplust1) x) \leq frac ((u\oplust1)y
    by (auto simp: cval-add-def)
qed
lemma upper-right-eq:
    assumes finite X valid-region X k I r
    shows (\forallx 就isGreater (I x)) \longleftrightarrow is-upper-right (region X I r)
using assms
proof (safe, goal-cases)
    case (1tu)
    then show ?case
        by - (standard, force simp: cval-add-def)+
next
    case (2 x)
```

    from region-not-empty[OF assms] obtain \(u\) where \(u: u \in\) region X Ir ..
    moreover have ( \(1::\) real) \(\geq 0\) by auto
    ultimately have \((u \oplus 1) \in\) 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 $R$ : $R \equiv$ region $X I r$
assumes $\neg$ is-upper-right $R u \in R$
shows $u \oplus 1 \notin R$
proof -
from upper-right-eq[OF $\operatorname{assms}(1,2)] \operatorname{assms}(4)$ obtain $x$ where $x$ :
$x \in X \neg$ isGreater ( $I x$ )
by (auto simp: $R$ )
with assms have intv-elem $x u(I x)$ by auto
with $x(2)$ have $\neg$ 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 succ $\mathcal{R} R \in \mathcal{R}$ (is ?G1) and succ $\mathcal{R} R \in S u c c \mathcal{R} R$ (is ?G2)
and $\forall u \in R . \forall t \geq 0 .(u \oplus t) \notin R \longrightarrow\left(\exists t^{\prime} \leq t .\left(u \oplus t^{\prime}\right) \in \operatorname{succ} \mathcal{R} R \wedge 0 \leq t^{\prime}\right)($ is ? $G 3)$
proof -
from $\langle R \in \mathcal{R}\rangle$ obtain Ir where $R$ : $R=$ region X Ir valid-region X $k$ Ir
unfolding $\mathcal{R}$-def by auto
from region-not-empty[OF finite] $R$ obtain $u$ where $u: u \in R$
by blast
let $? Z=\{x \in X . \exists c . I x=$ Const $c\}$
let ?succ =
$\lambda R^{\prime} . R^{\prime} \neq R \wedge R^{\prime} \in \operatorname{Succ} \mathcal{R} R$
$\wedge\left(\forall u \in R . \forall t \geq 0 .(u \oplus t) \notin R \longrightarrow\left(\exists t^{\prime} \leq t .\left(u \oplus t^{\prime}\right) \in R^{\prime} \wedge 0 \leq t^{\prime}\right)\right)$
consider (upper-right) $\forall x \in X$. isGreater $(I x) \mid$ (intv) $\exists x \in X . \exists d . I x=$ Intv $d \wedge ? Z=\{ \}$
$\mid$ (const) $? Z \neq\{ \}$
apply (cases $\forall x \in X$. isGreater (I x))
apply fast
apply (cases ? $Z=\{ \})$

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

```
    (\forallu\inR.\forallt\geq0.(u\oplust)\not\inR\longrightarrow(\exists\mp@subsup{t}{}{\prime}\leqt.(u\oplus\mp@subsup{t}{}{\prime})\in\mp@subsup{R}{}{\prime}\wedge0\leq\mp@subsup{t}{}{\prime}))}\mp@subsup{R}{}{\prime}\in\mathcal{R
    \forallu\inR'.}\forallt\geq0.(u\oplust)\not\in
    unfolding }\mathcal{R}\mathrm{ -def by auto
    with region-not-empty[OF finite] obtain }\mp@subsup{u}{}{\prime}\mathrm{ where }\mp@subsup{u}{}{\prime}\in\mp@subsup{R}{}{\prime}\mathrm{ unfolding }\mathcal{R}\mathrm{ -def by blast
    with }\mp@subsup{R}{}{\prime}(3)\mathrm{ have neq: R'}=R\mathrm{ by (fastforce simp: cval-add-def)
    from bounded-region[OF finite R(2), folded R(1), OF \prec\neg is-upper-right R>u] have
    u\oplus(1 ::t)\not\inR(1 :: t)\geq0
    by auto
    with R'(1)u obtain t' where t'\leq(1:: t) (u\oplus t')\in R' 0 \leq t' by fastforce
    with}\langleR\in\mathcal{R}\rangle\langle\mp@subsup{R}{}{\prime}\in\mathcal{R}\rangle\langleu\inR\rangle\mathrm{ have }\mp@subsup{R}{}{\prime}\in\mathrm{ Succ }\mathcal{R}R\mathrm{ by (intro SuccI3)
    with R'(1) neq have *: ?succ R' by auto
    have succ }\mathcal{R}R=\mp@subsup{R}{}{\prime}\mathrm{ unfolding succ-def
    proof (simp add:<\neg is-upper-right R`, intro the-equality, rule *, goal-cases)
    case prems: (1 R')
    from prems obtain t' u' where }\mp@subsup{R}{}{\prime\prime}\mathrm{ :
```



```
    using R'(1) by fastforce
    from this(1) obtain Ir where R'\prime2:
        R'\prime}=\mathrm{ region X I r valid-region X k I r
    by (auto simp: \mathcal{R-def)}
    from }\mp@subsup{R}{}{\prime\prime}\mathrm{ have }\mp@subsup{u}{}{\prime}\oplus\mp@subsup{t}{}{\prime}\not\inR\mathrm{ using assms region-unique-spec by blast
    with *\langle\mp@subsup{t}{}{\prime}\geq0\rangle\langle\mp@subsup{u}{}{\prime}\inR\rangle\mathrm{ obtain }\mp@subsup{t}{}{\prime\prime}\mathrm{ where }\mp@subsup{t}{}{\prime\prime}:\mp@subsup{t}{}{\prime\prime}\leq\mp@subsup{t}{}{\prime}\mp@subsup{u}{}{\prime}\oplus\mp@subsup{t}{}{\prime\prime}\in\mp@subsup{R}{}{\prime}\mp@subsup{t}{}{\prime\prime}\geq0}\mathrm{ by auto
    from this(2) neq have }\mp@subsup{u}{}{\prime}\oplus\mp@subsup{t}{}{\prime\prime}\not\inR\mathrm{ using }\mp@subsup{R}{}{\prime}(2)\mathrm{ assms region-unique-spec by auto
    with t" prems <u'\inR> obtain t'/\prime where t'\prime\prime
        t ^ { \prime \prime \prime } \leq t ^ { \prime \prime } u ^ { \prime } \oplus t ^ { \prime \prime \prime } \in R ^ { \prime \prime } t ^ { \prime \prime \prime } \geq 0
    by auto
    with region-continuous[OF R'I2(2) - - t'\prime\prime(2)[unfolded R'I'2(1)], of t'\prime - t'\prime\prime }\mp@subsup{t}{}{\prime}-\mp@subsup{t}{}{\prime\prime\prime}
                t' }\mp@subsup{R}{}{\prime\prime}\mathrm{ regions-closed'-spec[OF }\langleR\in\mathcal{R}\\mp@subsup{R}{}{\prime\prime}(5,3)
    have }\mp@subsup{u}{}{\prime}\oplus\mp@subsup{t}{}{\prime\prime}\in\mp@subsup{R}{}{\prime\prime}\mathrm{ by (auto simp: cval-add-def R'2)
    with }\mp@subsup{t}{}{\prime\prime}(2)\mathrm{ show ?case using }\mp@subsup{R}{}{\prime\prime}(1)\mp@subsup{R}{}{\prime}(2) region-unique-spec by blas
    qed
    with }\mp@subsup{R}{}{\prime}*\mathrm{ show ?thesis by auto
    next
    case upper-right
    with upper-right-eq[OF finite R(2)] have succ \mathcal{R }R=R by (auto simp: R succ-def)
    with}\langleR\in\mathcal{R}\rangleu\mathrm{ 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 :: nat
    assumes R}\in\mathcal{R}d\geq0\forallx\in\mathrm{ set r.d 
    shows region-set' R rd {\mathcal{R}
proof -
    from region-not-empty[OF finite] assms(1) obtain u where }u\inR\mathrm{ using }\mathcal{R}\mathrm{ -def by blast
    from region-set'-id[OF - finite, of - k, folded \mathcal{R}
qed
lemma clock-set-cong[simp]:
    assumes }\forallc\in\mathrm{ set r. u c=d
    shows [r->d]u=u
proof standard
    fix c
    from assms show ([r->d]u) c=uc by (cases c f 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 \(^{\prime} R\) r \(0 \notin \operatorname{Succ} \mathcal{R} R\) (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 \(\langle R \in \mathcal{R}\rangle[u n f o l d e d \mathcal{R}\)-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 \(\forall x \in\) 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 \(x: x \in\) set \(r u x \neq 0\) by auto
        \{ assume ?R \(\in\) 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 \(\langle u \in R\rangle \operatorname{assms}(1)\) 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\vdashgu\oplust\vdashg0\leq(\mp@subsup{t}{}{\prime}::'t::time) t'\leq
    shows }u\oplus\mp@subsup{t}{}{\prime}\vdash
    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

$\left({ }^{\prime} c,{ }^{\prime} t,{ }^{\prime} s\right)$ transition $=' s *\left({ }^{\prime} c,{ }^{\prime} t\right)$ cconstraint $*\left({ }^{\prime} c\right.$ set $*$ 's) pmf

## type-synonym

$$
\left({ }^{\prime} c,,^{\prime} t,{ }^{\prime} s\right) p t a=\left({ }^{\prime} c,,^{\prime} t,{ }^{\prime} s\right) \text { transition set } *\left({ }^{\prime} c, ' \text { ' } s\right) \text { invassn }
$$

```
definition
    edges :: ('c, 't, 's) transition }=>('s*('c, 't) cconstraint * ('c set * 's) pmf * 'c set * 's) se
where
    edges }\equiv\lambda(l,g,p).{(l,g,p,X,\mp@subsup{l}{}{\prime})|X\mp@subsup{l}{}{\prime}.(X,\mp@subsup{l}{}{\prime})\in\mathrm{ set-pmf p}
```


## definition

        Edges \(A \equiv \bigcup\{\) edges \(t \mid t . t \in\) fst \(A\}\)
    definition
trans-of :: ('c, 't, 's) pta $\Rightarrow\left({ }^{\prime} c\right.$, ' $t$, 's) transition set
where
trans-of $\equiv f s t$
definition
inv-of $::($ ('c, 'time, 's) pta $\Rightarrow(' c$, 'time, 's) invassn
where
inv-of $\equiv$ snd
no-notation transition $\left(-\vdash-\longrightarrow{ }^{-,-,-}-[61,61,61,61,61,61] 61\right)$
abbreviation transition ::

```
('c, 'time, 's) pta \(\Rightarrow\) 's \(\Rightarrow\left({ }^{\prime} c,{ }^{\prime}\right.\) 'time \()\) cconstraint \(\Rightarrow\left({ }^{\prime} c\right.\) set \(*\) 's) pmf \(\Rightarrow{ }^{\prime}\) c set \(\Rightarrow\) 's bool
```

$\left(-\vdash-\longrightarrow \longrightarrow^{-,--}-[61,61,61,61,61,61] 61\right)$ where
$\left(A \vdash l \longrightarrow g, p, X \quad l^{\prime}\right) \equiv\left(l, g, p, X, l^{\prime}\right) \in$ Edges $A$

## definition

        locations :: ('c, 't, 's) pta \(\Rightarrow\) 's set
    where
locations $A \equiv($ fst 'Edges $A) \cup(($ snd o snd o snd o snd $)$ 'Edges $A)$

### 3.1.1 Collecting Information About Clocks

```
definition collect-clkt :: ('c, 't::time, 's) transition set \(\Rightarrow\left({ }^{\prime} c *^{\prime} t\right)\) set
where
    collect-clkt \(S=\bigcup\{\) collect-clock-pairs \((\) fst \((\) snd \(t)) \mid t . t \in S\}\)
definition collect-clki \(::\left({ }^{\prime} c\right.\), 't :: time, 's) invassn \(\Rightarrow\left({ }^{\prime} c *^{\prime} t\right)\) set
where
    collect-clki \(I=\bigcup\{\) collect-clock-pairs \((\) I x) \(\mid x\). True \(\}\)
definition clkp-set :: ('c, 't :: time, 's) pta \(\Rightarrow\left({ }^{\prime} c *^{\prime} t\right)\) set
where
        clkp-set \(A=\) collect-clki \((\) inv-of \(A) \cup\) collect-clkt (trans-of \(A)\)
definition collect-clkvt :: ('c, 't :: time, 's) pta \(\Rightarrow{ }^{\prime} c\) set
where
    collect-clkvt \(A=\bigcup((\) fst o snd o snd o snd \()\) 'Edges A)
```

abbreviation clocks where clocks $A \equiv f s t$ 'clkp-set $A \cup$ collect-clkvt $A$
definition valid-abstraction
where
valid-abstraction A $X k \equiv$
$(\forall(x, m) \in$ clkp-set $A . m \leq k x \wedge x \in X \wedge m \in \mathbb{N}) \wedge$ collect-clkvt $A \subseteq X \wedge$ finite $X$
lemma valid-abstraction $D[$ dest $]$ :
assumes valid-abstraction $A X k$
shows $(\forall(x, m) \in$ clkp-set $A . m \leq k x \wedge x \in X \wedge m \in \mathbb{N})$ collect-clkvt $A \subseteq X$ finite $X$
using assms unfolding valid-abstraction-def by auto

```
lemma valid-abstractionI[intro]:
    assumes \((\forall(x, m) \in\) clkp-set \(A . m \leq k x \wedge x \in X \wedge m \in \mathbb{N})\) collect-clkvt \(A \subseteq X\) finite \(X\)
    shows valid-abstraction \(A X k\)
using assms unfolding valid-abstraction-def by auto
```


### 3.2 Operational Semantics as an MDP

abbreviation (input) clock-set-set :: 'c set $\Rightarrow{ }^{\prime} t::$ time $\Rightarrow\left({ }^{\prime} c,{ }^{\prime} t\right)$ cval $\Rightarrow\left({ }^{\prime} c,{ }^{\prime} t\right)$ cval
([-:=-]- [65,65,65] 65)

## where

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

term region-set'
abbreviation region-set-set $::$ 'c set $\Rightarrow{ }^{\prime} t::$ time $\Rightarrow\left({ }^{\prime} c\right.$, 't) zone $\Rightarrow\left({ }^{\prime} c,{ }^{\prime} t\right)$ zone
([-::=-]- [65,65,65] 65)
where

$$
[X::=t] R \equiv \text { region-set }^{\prime} R(S O M E \text {. set } r=X) t
$$

no-notation zone-set ( - - 0 [71] 71)
abbreviation zone-set-set :: ( ${ }^{\prime}$, 't::time) zone $\Rightarrow{ }^{\prime} c$ set $\Rightarrow\left({ }^{\prime} c,{ }^{\prime} t\right)$ zone (-- $\rightarrow 0$ [71] 71)
where

$$
Z_{X} \rightarrow 0 \equiv \text { zone-set } Z(S O M E \text { r. set } r=X)
$$

abbreviation (input) ccval $(\{-\}[100])$ where $c c v a l ~ c c \equiv\{v . v \vdash c c\}$

```
locale Probabilistic-Timed-Automaton \(=\)
    fixes \(A::\left({ }^{\prime} c\right.\), 't :: time, 's) pta
    assumes admissible-targets:
        \((l, g, \mu) \in\) trans-of \(A \Longrightarrow\left(X, l^{\prime}\right) \in \mu \Longrightarrow\{g\}_{X} \rightarrow 0 \subseteq\left\{\right.\) inv-of \(\left.A l^{\prime}\right\}\)
        \((l, g, \mu) \in\) trans-of \(A \Longrightarrow\left(X, l^{\prime}\right) \in \mu \Longrightarrow X \subseteq\) clocks \(A\)
```

    - Not necessarily what we want to have
    begin

### 3.3 Syntactic Definition

```
definition L = locations A
definition \mathcal{X = clocks }A
definition S\equiv{(l,u).l\inL^(\forallx\in\mathcal{X}.ux\geq0)\wedgeu\vdash inv-of A l}
inductive-set
    K :: ('s * ('c, 't) cval) =>('s*('c, 't) cval) pmf set for st :: ('s * ('c, 't) cval)
where
    - Passage of time delay:
    st }\inS\Longrightarrowst=(l,u)\Longrightarrowt\geq0\Longrightarrowu\oplust\vdash\mathrm{ inv-of A l ב return-pmf (l,u }\oplust)\inK st |
    - Discrete transitions action:
    st }\inS\Longrightarrowst=(l,u)\Longrightarrow(l,g,\mu)\in\mathrm{ trans-of A בu}\vdash)
    map-pmf (\lambda (X,l).(l,([X:= 0]u))) \mu\inK st |
    - Self loops - Note that this does not assume st \inS loop:
    return-pmf st }\inK\mathrm{ st
declare K.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 $\mathcal{X}$ for $A::\left({ }^{\prime} c, t,{ }^{\prime} s\right) p t a+$

- The following are necessary to obtain a finite MDP
assumes finite: finite $\mathcal{X}$ finite $L$ finite (trans-of $A$ )
assumes not-trivial: $\exists l \in L . \exists u \in V . u \vdash$ inv-of $A l$
assumes valid: valid-abstraction $A \mathcal{X} k$
begin
lemmas finite- $\mathcal{R}=$ finite- $\mathcal{R}[$ OF finite $(1)$, of $k$, folded $\mathcal{R}$-def $]$


### 4.1 Syntactic Definition

definition $\mathcal{S} \equiv\{(l, R) . l \in L \wedge R \in \mathcal{R} \wedge R \subseteq\{u . u \vdash$ inv-of $A l\}\}$
lemma $S$-alt-def: $S=\{(l, u) . l \in L \wedge u \in V \wedge u \vdash$ 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

```
\(\mathcal{K}::\left({ }^{\prime} s *\left({ }^{\prime} c, t\right)\right.\) cval set \() \Rightarrow\left({ }^{\prime} s *\left({ }^{\prime} c, t\right)\right.\) cval set) pmf set for \(s t::\left({ }^{\prime} s *\left({ }^{\prime} c, t\right) c v a l\right.\) set \()\)
where
    - Passage of time delay:
    \(s t \in \mathcal{S} \Longrightarrow s t=(l, R) \Longrightarrow R^{\prime} \in \operatorname{Succ} \mathcal{R} R \Longrightarrow R^{\prime} \subseteq\{\) inv-of \(A l\} \Longrightarrow\) return-pmf \(\left(l, R^{\prime}\right) \in \mathcal{K}\) st \(\mid\)
    - Discrete transitions action:
    st \(\in \mathcal{S} \Longrightarrow s t=(l, R) \Longrightarrow(l, g, \mu) \in\) trans-of \(A \Longrightarrow R \subseteq\{g\}\)
    \(\Longrightarrow\) map-pmf \(\left(\lambda(X, l) .\left(l\right.\right.\), region-set \({ }^{\prime} R(S O M E\). set \(\left.\left.r=X) 0\right)\right) \mu \in \mathcal{K}\) st \(\mid\)
    - Self loops - Note that this does not assume st \(\in \mathcal{S}\) loop:
    return-pmf st \(\in \mathcal{K}\) st
lemmas \([\) intro \(]=\mathcal{K}\). intros
```


### 4.2 Many Closure Properties

lemma transition-def:
$\left(A \vdash l \longrightarrow g, \mu, X \quad l^{\prime}\right)=\left((l, g, \mu) \in\right.$ trans-of $\left.A \wedge\left(X, l^{\prime}\right) \in \mu\right)$
unfolding Edges-def edges-def trans-of-def by auto
lemma transitionI [intro]:
$A \vdash l \longrightarrow g, \mu, X \quad l^{\prime}$ if $(l, g, \mu) \in$ trans-of $A\left(X, l^{\prime}\right) \in \mu$
using that unfolding transition-def ..
lemma transition $D[$ dest $]$ :
$(l, g, \mu) \in$ trans-of $A\left(X, l^{\prime}\right) \in \mu$ if $A \vdash l \longrightarrow g, \mu, X l^{\prime}$
using that unfolding transition-def by auto
lemma bex-Edges:
$(\exists x \in$ Edges A. $P x)=\left(\exists l g \mu X l^{\prime} . A \vdash l \longrightarrow \longrightarrow^{g}, \mu, X l^{\prime} \wedge P\left(l, g, \mu, X, l^{\prime}\right)\right)$
by fastforce
lemma $L$-trans $[$ intro $]$ :
assumes $(l, g, \mu) \in$ trans-of $A\left(X, l^{\prime}\right) \in \mu$
shows $l \in L l^{\prime} \in L$
using assms unfolding L-def locations-def by (auto simp: image-iff bex-Edges transition-def)

```
lemma transition- \(\mathcal{X}\) :
    \(X \subseteq \mathcal{X}\) if \(A \vdash l \longrightarrow \longrightarrow^{g, \mu, X} l^{\prime}\)
    using that unfolding \(\mathcal{X}\)-def collect-clkvt-def clkp-set-def by auto
lemma admissible-targets-alt:
    \(A \vdash l \longrightarrow g, \mu, X l^{\prime} \Longrightarrow\{g\}_{X} \rightarrow 0 \subseteq\left\{\right.\) inv-of \(\left.A l^{\prime}\right\}\)
    \(A \vdash l \longrightarrow g, \mu, X \quad l^{\prime} \Longrightarrow X \subseteq\) clocks \(A\)
    by (intro admissible-targets; blast) +
lemma \(V\)-reset-closed[intro]:
    assumes \(u \in V\)
    shows \([r \rightarrow(d:: n a t)] u \in V\)
using assms unfolding \(V\)-def
    apply safe
    subgoal for \(x\)
        by (cases \(x \in\) set \(r\); auto)
    done
lemmas \(V\)-reset-closed \({ }^{\prime}[\) 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 \(\mathcal{R}\)-def]] have
        \(\exists!R . R \in \mathcal{R} \wedge u \in R\)
        unfolding \(V\)-def by auto
    then show \([u]_{\mathcal{R}} \in \mathcal{R} \quad u \in[u]_{\mathcal{R}}\)
        using alpha-interp.region-unique-spec by auto
qed
lemma rep- \(\mathcal{R}\)-ex[intro]:
    assumes \(R \in \mathcal{R}\)
    shows (SOME \(u . u \in R) \in R\)
proof -
    from assms region-not-empty[OF finite(1)] have \(\exists u . u \in R\) unfolding \(\mathcal{R}\)-def by auto
    then show ?thesis ..
qed
lemma \(V\)-nn-closed[intro]:
    \(u \in V \Longrightarrow t \geq 0 \Longrightarrow u \oplus t \in V\)
unfolding \(V\)-def cual-add-def by auto
lemma \(K\) - \(S\)-closed[intro]:
    assumes \(\mu \in K s s^{\prime} \in \mu s \in S\)
    shows \(s^{\prime} \in S\)
    using assms
    by (cases rule: \(K\).cases, auto simp: S-alt-def dest: admissible-targets[unfolded zone-set-def])
lemma \(S\) - \(V[\) intro \(]\) :
    \((l, u) \in S \Longrightarrow u \in V\)
unfolding \(S\)-alt-def by auto
lemma \(L-V[\) intro \(]\) :
    \((l, u) \in S \Longrightarrow l \in L\)
unfolding \(S\)-def by auto
lemma \(\mathcal{S}-V[\) intro \(]\) :
```

$$
(l, R) \in \mathcal{S} \Longrightarrow R \in \mathcal{R}
$$

unfolding $\mathcal{S}$-def by auto
lemma admissible-targets':
assumes $(l, g, \mu) \in$ trans-of $A\left(X, l^{\prime}\right) \in \mu R \subseteq\{g\}$
shows region-set $R$ (SOME $r$. set $r=X) 0 \subseteq\left\{\right.$ inv-of $\left.A l^{\prime}\right\}$
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 $=\left\{\left(R^{\prime}, l, R\right)\right.$. st $\in \mathcal{S} \wedge$ st $=(l, R) \wedge R^{\prime} \in \operatorname{Succ} \mathcal{R} R \wedge R^{\prime} \subseteq\{$ inv-of Al\} $\}$
let ? $S 1=\left(\lambda\left(R^{\prime}, l, R\right)\right.$. return-pmf $\left.\left(l, R^{\prime}\right)\right) \cdot$ ?B1
let ? $S 1=\left\{\right.$ return-pmf $\left(l, R^{\prime}\right) \mid R^{\prime} l$ R. st $\in \mathcal{S} \wedge$ st $=(l, R) \wedge R^{\prime} \in S u c c \mathcal{R} R \wedge R^{\prime} \subseteq\{$ inv-of $\left.A l\}\right\}$
let ? SZ $=\left\{\right.$ map-pmf $\left(\lambda(X, l)\right.$. $\left(l\right.$, region-set ${ }^{\prime} R(S O M E$ r. set $\left.\left.r=X) 0\right)\right) \mu$
$\mid R \mu . \exists l g . s t \in \mathcal{S} \wedge s t=(l, R) \wedge(l, g, \mu) \in$ trans-of $A \wedge R \subseteq\{g\}\}$
have ? $B 1 \subseteq\left\{\left(R^{\prime}, l, R\right) . R^{\prime} \in \mathcal{R} \wedge(l, R) \in \mathcal{S}\right\}$ unfolding $\mathcal{S}$-def by auto
with $\mathcal{S}$-finite finite- $\mathcal{R}$ have finite ? B1 by - (rule finite-subset, auto)
moreover have ? $S 1=\left(\lambda\left(R^{\prime}, l, R\right)\right.$. return-pmf $\left.\left(l, R^{\prime}\right)\right)$ '? B1 by (auto simp: image-def)
ultimately have $*$ : finite ?S1 by auto
have $\{\mu . \exists l g .(l, g, \mu) \in P T A . t r a n s-o f ~ A\}=((\lambda(l, g, \mu) . \mu)$ 'PTA.trans-of $A)$ by force
with finite(3) finite- $\mathcal{R}$ have finite $\{(R, \mu) . \exists l g . R \in \mathcal{R} \wedge(l, g, \mu) \in$ trans-of $A\}$ by auto
moreover have
$\{(R, \mu) . \exists l g . s t \in \mathcal{S} \wedge s t=(l, R) \wedge(l, g, \mu) \in$ trans-of $A \wedge R \subseteq\{g\}\} \subseteq \ldots$
unfolding $\mathcal{S}$-def by fastforce
ultimately have $* *$ :
finite $\{(R, \mu) . \exists l g . s t \in \mathcal{S} \wedge s t=(l, R) \wedge(l, g, \mu) \in$ trans-of $A \wedge R \subseteq\{g\}\}$
unfolding $\mathcal{S}$-def by (blast intro: finite-subset)
then have finite ? S 2 unfolding $\mathcal{S}$-def by auto
have $\mathcal{K}$ st $=$ ? S1 $\cup$ ?S $2 \cup\{$ return-pmf st $\}$ by (safe, cases rule: $\mathcal{K} . c a s e s$, auto)
with $* * *$ show ?thesis by auto
qed
lemma $\mathcal{R}$-not-empty:
$\mathcal{R} \neq\{ \}$
proof -
let $? r=\{ \}$
let $? I=\lambda c$. Const 0
let $? R=$ region $\mathcal{X}$ ?I ?r
have valid-region $\mathcal{X} k$ ?I ?r
proof
show $\}=\{x \in \mathcal{X} . \exists d$. Const $0=$ Intv $d\}$ by auto
show refl-on $\}\}$ and trans $\}$ and total-on $\}\}$ unfolding trans-def by auto
show $\forall x \in \mathcal{X}$. Regions.valid-intv $(k x)$ (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 st: $l \in L u \in V u \vdash$ 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)\incollect-clock-pairs (inv-of A l). m\leq real (kx)\wedgex\in\mathcal{X}\wedgem\in\mathbb{N}
    by (fastforce simp: clkp-set-def collect-clki-def)
    from ccompatible[OF this, folded \mathcal{R-def] R st(3) have}
        R\subseteq{inv-of A l}
    unfolding ccompatible-def ccval-def by auto
    with st(1) R(1) show ?thesis unfolding S-def by auto
qed
lemma }\mathcal{K}-\mathcal{S}\mathrm{ -closed:
    assumes }s\in\mathcal{S
    shows }(\bigcupD\in\mathcal{K}\mathrm{ s. set-pmf D)}\subseteq\mathcal{S
proof (safe, cases rule: K.cases, blast, goal-cases)
    case (1 x abl R)
    then show ?case unfolding S-def by (auto intro: alpha-interp.succ-ex(1))
next
    case (3 a b x)
    with }\langles\in\mathcal{S}\rangle\mathrm{ show ?case by auto
next
    case prems:(2 l' R'plRg )
    then obtain X where *:(X, l') \in set-pmf \mu R'= region-set' R (SOME r. set r=X)0 by auto
    show ?case unfolding S-def
    proof safe
        from *(1) have (l,g, \mu,X, l') \in edges ( l,g, \mu) unfolding edges-def by auto
        with prems(6) have (l,g, , , X, l') \in Edges A unfolding Edges-def trans-of-def by auto
        then show }\mp@subsup{l}{}{\prime}\inL\mathrm{ unfolding L-def locations-def by force
        show }u\vdash\mathrm{ inv-of A l' if u}\in\mp@subsup{R}{}{\prime}\mathrm{ for }
        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}\mathrm{ unfolding }\mathcal{X}\mathrm{ -def by auto}
        with finite(1) have finite X by (blast intro: finite-subset)
        then obtain r where set r = X using finite-list by auto
        then have set (SOME r. set r=X)=X by (rule someI)
        with }\langleX\subseteq\mathcal{X}\rangle\mathrm{ have set (SOME r. set r=X) }\subseteq\mathcal{X}\mathrm{ by auto
        with alpha-interp.region-set'-closed[of R 0 SOME r. set r = X] prems(4,5)*(2)
        show }\mp@subsup{R}{}{\prime}\in\mathcal{R}\mathrm{ unfolding }\mathcal{S}\mathrm{ -def }\mathcal{X}\mathrm{ -def by auto
    qed
qed
sublocale R-G: Finite-Markov-Decision-Process \mathcal{K S}
by (standard, auto simp: \mathcal{S}}\mathrm{ -finite }\mathcal{S}\mathrm{ -not-empty }\mathcal{K}\mathrm{ -finite }\mathcal{K}\mathrm{ -S-closed)
lemmas }\mathcal{K}\mathrm{ -S-closed'[intro] =R-G.set-pmf-closed
```


## 5 Relating the MDPs

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

```
lemma ccompatible-inv:
    shows ccompatible \mathcal{R}}\mathrm{ (inv-of A l)
proof -
    from valid have
        \forall(x,m)\incollect-clock-pairs (inv-of A l). m\leqreal (kx)\wedgex\in\mathcal{X}\wedgem\in\mathbb{N}
    unfolding valid-abstraction-def clkp-set-def collect-clki-def by auto
    with ccompatible[of-k\mathcal{X}\mathrm{ , folded }\mathcal{R}\mathrm{ -def] show ?thesis by auto}
qed
lemma ccompatible-guard:
```

assumes $(l, g, \mu) \in$ trans-of $A$
shows ccompatible $\mathcal{R} g$
proof -
from assms valid have
$\forall(x, m) \in$ collect-clock-pairs $g . m \leq \operatorname{real}(k x) \wedge x \in \mathcal{X} \wedge m \in \mathbb{N}$
unfolding valid-abstraction-def clkp-set-def collect-clkt-def trans-of-def by fastforce
with assms ccompatible $[o f-k \mathcal{X}$, folded $\mathcal{R}$-def] show ?thesis by auto
qed
lemmas ccompatible-def $=$ ccompatible-def[unfolded ccval-def]
lemma region-set'-eq:
fixes $X::$ 'c set
assumes $R \in \mathcal{R} u \in R$
and $A \vdash l \longrightarrow g, \mu, X \quad l^{\prime}$
shows

$$
[[X:=0] u]_{\mathcal{R}}=\text { region-set }{ }^{\prime} R(\text { SOME } r \text {. set } r=X) 0[[X:=0] u]_{\mathcal{R}} \in \mathcal{R}[X:=0] u \in\left[[X:=0]_{u}\right]_{\mathcal{R}}
$$

proof -
let $? r=(S O M E$ r. 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 set $r=X$ using finite-list by blast
then have set ? $r=X$ by (intro someI)
with valid $\operatorname{assms}(3)$ have set ?r $\subseteq \mathcal{X}$
by (simp add: transition- $\mathcal{X}$ )
from region-set'-id[of - $\mathcal{X} k$, folded $\mathcal{R}$-def, OF $\operatorname{assms}(1,2)$ finite $(1)-$ this]
show

$$
[[X:=0] u]_{\mathcal{R}}=\text { region-set }^{\prime} R(S O M E \text { r. set } r=X) 0[[X:=0] u]_{\mathcal{R}} \in \mathcal{R}[X:=0] u \in\left[[X:=0]_{u}\right]_{\mathcal{R}}
$$

by force+
qed
lemma regions-part-ex-reset:
assumes $u \in V$
shows $[r \rightarrow(d:: n a t)] u \in[[r \rightarrow d] u]_{\mathcal{R}}[[r \rightarrow d] u]_{\mathcal{R}} \in \mathcal{R}$
using assms by auto
lemma reset-sets-all-equiv:
assumes $u \in V u^{\prime} \in[[r \rightarrow(d:: \text { nat })] u]_{\mathcal{R}} x \in$ set $r$ set $r \subseteq \mathcal{X} d \leq k x$
shows $u^{\prime} x=d$
proof -
from assms(1) have $u:[r \rightarrow d] u \in[[r \rightarrow d] u]_{\mathcal{R}}[[r \rightarrow d] u]_{\mathcal{R}} \in \mathcal{R}$ by auto
then obtain $I \varrho$ where $I:[[r \rightarrow d] u]_{\mathcal{R}}=$ region $\mathcal{X} I \varrho$ valid-region $\mathcal{X} k I \varrho$
by (auto simp: $\mathcal{R}$-def)
with $u(1) \operatorname{assms}(3-)$ have intv-elem $x([r \rightarrow d] u)(I x)$ valid-intv $(k x)(I x)$ by fastforce+
moreover from assms have $([r \rightarrow d] u) x=d$ by simp
ultimately have $I x=$ Const $d$ using assms(5) by (cases $I x$ ) auto
moreover from $I \operatorname{assms}(2-)$ have intv-elem $x u^{\prime}(I x)$ by fastforce
ultimately show $u^{\prime} x=d$ by auto
qed
lemma reset-eq:
assumes $u \in V\left([[r \rightarrow 0] u]_{\mathcal{R}}\right)=\left(\left[\left[r^{\prime} \rightarrow 0\right] u\right]_{\mathcal{R}}\right)$ set $r \subseteq \mathcal{X}$ set $r^{\prime} \subseteq \mathcal{X}$
shows $[r \rightarrow 0] u=\left[r^{\prime} \rightarrow 0\right] u$ using assms
proof -
have $*: u^{\prime} x=0$ if $u^{\prime} \in[[r \rightarrow 0] u]_{\mathcal{R}} x \in \operatorname{set} r$ for $u^{\prime} x$
using reset-sets-all-equiv[of $\left.u u^{\prime} r 0 x\right]$ that assms by auto
have $u^{\prime} x=0$ if $u^{\prime} \in\left[\left[r^{\prime} \rightarrow 0\right] u\right]_{\mathcal{R}} x \in$ set $r^{\prime}$ for $u^{\prime} x$
using reset-sets-all-equiv $\left[\right.$ of $\left.u u^{\prime} r^{\prime} 0 x\right]$ that assms by auto
from regions-part-ex-reset $[$ OF $\operatorname{assms}(1)$, of - 0] $\operatorname{assms}(2)$ have $* *$ :

$$
\left(\left[r^{\prime} \rightarrow 0\right] u\right) \in[[r \rightarrow 0] u]_{\mathcal{R}}([r \rightarrow 0] u) \in\left[[r \rightarrow 0]_{u}\right]_{\mathcal{R}}[[r \rightarrow 0] u]_{\mathcal{R}} \in \mathcal{R}
$$

by auto

```
    have (([r->0]u)x)=(([r'->0]u)x) for x
    proof (cases x 新 r)
        case True
        then have ([r->0]u)x=0 by simp
    moreover from *** True have ([r'->0]u) x=0 by auto
    ultimately show ?thesis ..
    next
    case False
    then have id: ([r->0]u)x=ux by simp
    show ?thesis
    proof (cases x f set r')
        case True
        then have reset: ([r'->0]u) x=0 by simp
        show ?thesis
        proof (cases x }\in\mathcal{X}
            case True
            from **(3) obtain I \varrho where
                ([([r->0]u)\mp@subsup{]}{\mathcal{R}}{})=\mathrm{ Regions.region X I @ Regions.valid-region X X k I @}
            by (auto simp: \mathcal{R-def)}
            with ** \langlex\in\mathcal{X}\rangle\mathrm{ have ***:}
                intv-elem x ([r'诠掠)(Ix) intv-elem x ([r->0]u) (I x)
            by auto
            with reset have I x = Const 0 by (cases I x, auto)
            with ***(2) have ([r->0]u) x=0 by auto
            with reset show ?thesis by auto
        next
            case False
            with assms(3-) have x\not\in set r x & set r' by auto
            then show ?thesis by simp
        qed
    next
        case False
        then have reset: ([r'->0]u) x=ux by simp
        with id show ?thesis by simp
    qed
    qed
    then show ?thesis ..
qed
lemma admissible-targets-clocks:
    assumes (l,g, \mu)\in trans-of A (X, l') \in \mu
    shows X\subseteq\mathcal{X}}\mathrm{ set (SOME r. 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: \mathcal{X-def)}
    then obtain r where set r = X using finite-list by blast
    with}\langleX\subseteq\mathcal{X}\rangle\mathrm{ show }X\subseteq\mathcal{X}\mathrm{ set (SOME r. set r = X) }\subseteq\mathcal{X
        by (metis (mono-tags, lifting) someI-ex)+
qed
lemma
    rel-pmf ( }\lambda\mathrm{ a b. fa=b) }\mu(\mathrm{ map-pmff }\mu
by (subst pmf.rel-map(2)) (rule rel-pmf-reflI, auto)
lemma K-pmf-rel:
    defines }f\equiv\lambda(l,u).(l,[u\mp@subsup{]}{\mathcal{R}}{}
    shows rel-pmf ( }\lambda(l,u)\mathrm{ st. (l, [u] 疎) =st) }\mu\mathrm{ (map-pmff }\mu\mathrm{ ) 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 \(f \equiv \lambda(l, u) .(l\), SOME \(u . u \in R)\)
    shows rel-pmf \((\lambda(l, u)\) st. \((l, S O M E u . u \in R)=s t) \mu\) (map-pmff \(\mu\) ) unfolding \(f\)-def
by (subst pmf.rel-map(2)) (rule rel-pmf-reflI, auto)
lemma \(K\)-elem-abs-inj:
    assumes \(A: \mu \in K(l, u)\)
    defines \(f \equiv \lambda(l, u) .\left(l,[u]_{\mathcal{R}}\right)\)
    shows inj-on \(f \mu\)
proof -
    have \((l 1, u 1)=(l 2, u 2)\)
        if \(i d:\left(l 1,[u 1]_{\mathcal{R}}\right)=\left(l 2,[u 2]_{\mathcal{R}}\right)\) and elem: \((l 1, u 1) \in \mu(l 2, u 2) \in \mu\) for 11 l2 u1 u2
    proof -
        from id have [simp]: \(12=11\) by auto
        from \(A\)
        show ?thesis
        proof (cases, safe, goal-cases)
            case (4-- \(\tau \mu^{\prime}\) )
            from \(\langle\mu=-\rangle\) elem obtain \(X 1\) X2 where
                \(u 1=[(S O M E\) r. set \(r=X 1) \rightarrow 0] u(X 1, l 1) \in \mu^{\prime}\)
                \(u 2=[(S O M E\) r. set \(r=X 2) \rightarrow 0] u(X 2, l 1) \in \mu^{\prime}\)
            by auto
            with \(\langle-\in\) trans-of -〉 admissible-targets-clocks have
                set \((S O M E r\). set \(r=X 1) \subseteq \mathcal{X}\) set \((S O M E r\). set \(r=X 2) \subseteq \mathcal{X}\)
            by auto
            with \(i d\langle u 1=-\rangle\langle u 2=-\rangle\) reset-eq \([\) of \(u]\langle-\in S\rangle\) show ?case by (auto simp: \(S\)-def \(V\)-def)
        qed ( - , insert elem, simp) +
    qed
    then show ?thesis unfolding \(f\)-def inj-on-def by auto
qed
lemma K-elem-repr-inj:
    notes alpha-interp.valid-regions-distinct-spec[intro]
    assumes \(A: \mu \in \mathcal{K}(l, R)\)
    defines \(f \equiv \lambda(l, R) .(l, S O M E u . u \in R)\)
    shows inj-on f \(\mu\)
proof -
    have \((l 1, R 1)=(l 2, R 2)\)
        if \(i d:(l 1, S O M E u . u \in R 1)=(l 2, S O M E u . u \in R 2)\) and elem: \((l 1, R 1) \in \mu(l 2, R 2) \in \mu\)
        for l1 l2 R1 R2
    proof -
        let \({ }^{2} r 1=\) SOME \(u . u \in R 1\) and \(? r 2=S O M E u . u \in R 2\)
        from id have \([\) simp \(]: 12=l 1\) ? \(22=\) ? r 1 by auto
        \{ fix \(g \mu^{\prime} x\)
            assume \((l, R) \in \mathcal{S}\left(l, g, \mu^{\prime}\right) \in P T A . t r a n s-o f A R \subseteq\{v . v \vdash g\}\)
                and \(\mu=\) map-pmf \((\lambda(X, l)\). (l, region-set \(R(S O M E\) r. set \(r=X) 0)) \mu^{\prime}\)
            from \(\langle\mu=-\rangle\) elem obtain X1 X2 where
                \(R 1=\) region-set \({ }^{\prime} R(\) SOME \(r\). set \(r=X 1) 0(X 1, l 1) \in \mu^{\prime}\)
                \(R 2=\) region-set \({ }^{\prime} R(S O M E r\). set \(r=X 2) 0(X 2, l 1) \in \mu^{\prime}\)
            by auto
            with \(\langle-\in\) trans-of -> admissible-targets-clocks have
                set \((\) SOME \(r\). set \(r=X 1) \subseteq \mathcal{X}\) set \((S O M E\) r. set \(r=X 2) \subseteq \mathcal{X}\)
            by auto
            with alpha-interp.region-set'-closed \([\) of -0\(]\langle R 1=-\rangle\langle R 2=-\rangle\langle-\in \mathcal{S}\rangle\) have
                \(R 1 \in \mathcal{R} R 2 \in \mathcal{R}\)
            unfolding \(\mathcal{S}\)-def by auto
            with region-not-empty[OF finite(1)] have
                \(R 1 \neq\{ \} R 2 \neq\{ \} \exists u . u \in R 1 \exists u . u \in R 2\)
            by (auto simp: \(\mathcal{R}\)-def)
            from someI-ex[OF this(3)] someI-ex[OF this(4)] have ?r1 \(\quad\) R1 ? r \(1 \in R 2\) by simp +
            with \(\langle R 1 \in \mathcal{R}\rangle\langle R 2 \in \mathcal{R}\rangle\) have \(R 1=R 2 .\).
```

```
    }
    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\inK(l,u)(\mp@subsup{l}{}{\prime},\mp@subsup{u}{}{\prime})\in
    defines }f\equiv\lambda(l,u).(l,[u\mp@subsup{]}{\mathcal{R}}{}
    shows pmf (map-pmff }\mu)(f(\mp@subsup{l}{}{\prime},\mp@subsup{u}{}{\prime}))=pmf \mu(\mp@subsup{l}{}{\prime},\mp@subsup{u}{}{\prime}
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)(\mp@subsup{l}{}{\prime},\mp@subsup{R}{}{\prime})\in
    defines }f\equiv\lambda(l,R).(l, SOME u.u\inR
    shows pmf (map-pmff }\mu)(f(\mp@subsup{l}{}{\prime},\mp@subsup{R}{}{\prime}))=pmf \mu(\mp@subsup{l}{}{\prime},\mp@subsup{R}{}{\prime}
using A unfolding f-def by (blast intro: pmf-map-inj K-elem-repr-inj)
```

definition transp :: ( $' s *\left({ }^{\prime} c, t\right)$ cval $\Rightarrow$ bool $) \Rightarrow{ }^{\prime} s *\left({ }^{\prime} c, t\right)$ cval set $\Rightarrow$ 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) cval $\Rightarrow{ }^{\prime} s *\left({ }^{\prime} c, t\right)$ cval set
where

$$
\text { abss } \equiv \lambda(l, u) . \text { if } u \in V \text { then }\left(l,[u]_{\mathcal{R}}\right) \text { else }(l,-V)
$$

## definition

```
    reps :: 's * ('c, t) cval set \(\Rightarrow{ }^{\prime} s *\left({ }^{\prime} c, t\right)\) cval
```

where
reps $\equiv \lambda(l, R)$. if $R \in \mathcal{R}$ then $(l, S O M E u . u \in R)$ else $(l, \lambda-.-1)$
lemma $\mathcal{S}$-reps-S $[$ intro $]$ :
assumes $s \in \mathcal{S}$
shows reps $s \in S$
using assms $\mathcal{R}$ - $V$ unfolding $S$-def $\mathcal{S}$-def reps-def $V$-def by force
lemma $S$-abss- $\mathcal{S}[$ intro]:
assumes $s \in S$
shows abss $s \in \mathcal{S}$
using assms ccompatible-inv unfolding $\mathcal{S}$-def $S$-alt-def abss-def ccompatible-def by force
lemma $\mathcal{S}$-abss-reps $[$ simp $]$ :
$s \in \mathcal{S} \Longrightarrow$ abss $($ reps $s)=s$
using $\mathcal{R}$-V alpha-interp.region-unique-spec by (auto simp: $S$-def $\mathcal{S}$-def reps-def abss-def; blast)
lemma map-pmf-abs-reps:
assumes $s \in \mathcal{S} \mu \in \mathcal{K} s$
shows map-pmf abss (map-pmf reps $\mu$ ) $=\mu$
proof -
have map-pmf abss (map-pmf reps $\mu$ ) $=$ map-pmf (abss o reps) $\mu$ by (simp add: pmf.map-comp)
also have $\ldots=\mu$
proof (rule map-pmf-idI, safe, goal-cases)
case prems: $\left(1 l^{\prime} R^{\prime}\right)$
with assms have $\left(l^{\prime}, R^{\prime}\right) \in \mathcal{S}$ reps $\left(l^{\prime}, R^{\prime}\right) \in S$ 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 set-pmf (action cfg) cfg \inR-G.cfg-on s'
    shows abss (reps s)=s
proof -
    from assms have s\in\mathcal{S}\mathrm{ by auto}
    then show ?thesis by auto
qed
lemma abss-S[intro]:
    assumes }(l,u)\in
    shows abss (l,u)=(l, [u\mp@subsup{]}{\mathcal{R}}{})
using assms unfolding abss-def by auto
lemma reps-\mathcal{S}[intro]:
    assumes (l, R)\in\mathcal{S}
    shows reps (l,R)=(l, SOME u.u\inR)
using assms unfolding reps-def by auto
lemma fst-abss:
    fst (abss st) = fst st for st
    by (cases st) (auto simp: abss-def)
lemma K-elem-abss-inj:
    assumes A: }\mu\inK(l,u)(l,u)\in
    shows inj-on abss }
proof -
    from assms have abss s}\mp@subsup{s}{}{\prime}=(\lambda(l,u).(l,[u\mp@subsup{]}{\mathcal{R}}{}))\mp@subsup{s}{}{\prime}\mathrm{ if }\mp@subsup{s}{}{\prime}\in\mu\mathrm{ 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}\mathcal{K}\mathrm{ -elem-reps-inj:
    assumes A: }\mu\in\mathcal{K}(l,R)(l,R)\in\mathcal{S
    shows inj-on reps }
proof -
    from assms have reps s}\mp@subsup{s}{}{\prime}=(\lambda(l,R).(l,SOME u.u\inR)) 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\inK(l,u)(l,u)\inS s'\in
    shows pmf (map-pmf abss \mu) (abss s')=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}(\mp@subsup{l}{}{\prime},\mp@subsup{R}{}{\prime})\in
    shows pmf (map-pmf reps \mu)(reps ( l', R')) = pmf \mu( l', R')
using A by (blast intro: pmf-map-inj \mathcal{K}\mathrm{ -elem-reps-inj)}
We need that \(\mathcal{X}\) is non-trivial here
lemma not-S-reps:
        (l,R)\not\in\mathcal{S Creps }(l,R)\not\inS
proof -
    assume (l, R)\not\in\mathcal{S}
    let ?u = SOME u.u\inR
```

```
have ᄀ?u\vdash inv-of Al if R\in\mathcal{R}l\inL
proof -
    from region-not-empty[OF finite(1)]<R \in\mathcal{R}> have }\existsu.u\inR by (auto simp: \mathcal{R-def)
    from someI-ex[OF this] have ?u\inR.
```



```
    ultimately show ?thesis
        using ccompatible-inv[of l]<R <\mathcal{R}\ unfolding ccompatible-def by fastforce
qed
with non-empty «(l,R)\not\in\mathcal{S}`\mathrm{ show ?thesis unfolding }\mathcal{S}\mathrm{ -def S-def reps-def by auto}
qed
```

```
lemma neq- \(V\)-not-region:
    \(-V \notin \mathcal{R}\)
using \(\mathcal{R}\) - \(V\) rep- \(\mathcal{R}\)-ex by auto
lemma \(\mathcal{S}\)-abss-S:
    abss \(s \in \mathcal{S} \Longrightarrow s \in S\)
    unfolding abss-def \(\mathcal{S}\)-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 \(l^{\prime} y l u\)
        using neq- \(V\)-not-region by (cases \(u \in V\); auto dest: regions-part-ex)
    done
```

lemma $S$-pred-stream-abss-S :
pred-stream $(\lambda s . s \in S) x s \longleftrightarrow$ pred-stream $(\lambda s . s \in \mathcal{S})$ (smap abss xs)
using $S$-abss- $\mathcal{S} \mathcal{S}$-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:
$s \in S \Longrightarrow(\bigcup D \in K$ s. set-pmf $D) \subseteq S$
by auto

### 5.2.2 Intermezzo

```
abbreviation timed-bisim (infixr \(\sim 60\) ) where
    \(s^{\sim} s^{\prime} \equiv\) abss \(s=\) abss \(s^{\prime}\)
lemma bisim-loc-id[intro]:
    \((l, u) \sim\left(l^{\prime}, u^{\prime}\right) \Longrightarrow l=l^{\prime}\)
unfolding abss-def by (cases \(u \in V\); cases \(u^{\prime} \in V\); simp)
lemma bisim-val-id[intro]:
    \([u]_{\mathcal{R}}=[u]_{\mathcal{R}}\) if \(u \in V(l, u)^{\sim}\left(l^{\prime}, u^{\prime}\right)\)
proof -
    have \(\left(l^{\prime},-V\right) \neq\left(l,[u]_{\mathcal{R}}\right)\)
        using that by blast
    with that have \(u^{\prime} \in V\)
        by (force simp: abss-def)
    with that show ?thesis
        by (simp add: abss-def)
qed
```

```
lemma bisim-symmetric:
    (l,u) ~ (l', u') = (l',}\mp@subsup{u}{}{\prime})~(l,u
by (rule eq-commute)
lemma bisim-val-id2[intro]:
    u}\mp@subsup{}{\prime}{\prime}=V\Longrightarrow(l,u)~(\mp@subsup{l}{}{\prime},u')\Longrightarrow[u\mp@subsup{]}{\mathcal{R}}{}=[u\mp@subsup{]}{\mathcal{R}}{
apply (subst (asm) eq-commute)
apply (subst eq-commute)
apply (rule bisim-val-id)
by auto
lemma K-bisim-unique:
    assumes s\inS \mu\inKs x\in\mu \mp@subsup{x}{}{\prime}\in\mux~}\mp@subsup{}{~}{~}\mp@subsup{x}{}{\prime
    shows x = \mp@subsup{x}{}{\prime}
using assms(2,1,3-)
proof (cases rule: K.cases)
    case prems:(action l u \tau \mu')
    with assms obtain l1 l2 X1 X2 where A:
        (X1,l1) \in set-pmf }\mp@subsup{\mu}{}{\prime}(X2,l2)\in set-pmf \mp@subsup{\mu}{}{\prime
        x=(l1,[X1:=0]u) \mp@subsup{x}{}{\prime}=(l2,[X2:=0]u)
    by auto
```



```
        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<x ~ x'〉 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 $a b s p$ where absp $\varphi \equiv \varphi$ o reps
definition repp where
repp $\varphi \equiv \varphi$ o absp

### 5.2.4 Distributions

## definition

    abst :: ('s * ('c, t) cval) pmf \(\Rightarrow(' s *(' c, t) c v a l ~ s e t) ~ p m f ~\)
    where
$a b s t=m a p-p m f a b s s$
lemma abss-SD:
assumes abss $s \in \mathcal{S}$
obtains $l u$ where $s=(l, u) u \in[u]_{\mathcal{R}}[u]_{\mathcal{R}} \in \mathcal{R}$
proof -
obtain $l u$ where $s=(l, u)$ by force
moreover from $\mathcal{S}$-abss- $S[O F$ assms $]$ have $s \in S$.
ultimately have abss $s=\left(l,[u]_{\mathcal{R}}\right) u \in V u \in[u]_{\mathcal{R}}[u]_{\mathcal{R}} \in \mathcal{R}$ by auto
with $\langle s=-\rangle$ 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-S \(D[O F \operatorname{assms}(1)]\) obtain \(l^{\prime} u\) where \(u\) :
        \(s=\left(l^{\prime}, u\right) u \in[u]_{\mathcal{R}}[u]_{\mathcal{R}} \in \mathcal{R}\)
    by blast+
    with \(\mathcal{R}-V\) have \(u \in V\) by auto
    with \(\langle s=-\rangle \operatorname{assms}(2)\) have \(l^{\prime}=l R=[u]_{\mathcal{R}}\) unfolding abss-def by auto
    with \(u\) show ?thesis by (auto intro: that)
qed
```

definition $\inf R R \equiv \lambda c$. of-int $\lfloor(S O M E u . u \in R) c\rfloor$
term let $a=3$ in $b$
definition delayed $R$ R $u \equiv$
$u \oplus($
let $I=(S O M E I . \exists r$. valid-region $\mathcal{X} k I r \wedge R=$ region $\mathcal{X} I r)$;
$m=1-\operatorname{Max}(\{\operatorname{frac}(u c) \mid c . c \in \mathcal{X} \wedge \operatorname{isIntv}(I c)\} \cup\{0\})$
in SOME $t . u \oplus t \in R \wedge t \geq m / 2$
)
lemma delayed $R$-correct-aux-aux:
fixes $c::$ nat
fixes $a b$ :: real
assumes $c<a a<$ Suc $c b \geq 0 a+b<$ Suc $c$
shows frac $(a+b)=$ frac $a+b$
proof -
have f1: $a+b<\operatorname{real}(c+1)$ using assms(4) by auto
have f2: $\bigwedge r r a .(r::$ real $)+(-r+r a)=r a$ by linarith
have f3: $\wedge r .(r::$ real $)=-(-r)$ by linarith
have $\mathrm{f}_{4}: \bigwedge r r a .-(r::$ real $)+(r a+r)=r a$ by linarith
then have f5: $\wedge r n . r+-$ frac $r=$ real $n \vee \neg r<\operatorname{real}(n+1) \vee \neg$ real $n<r$ using f2 by (metis nat-intv-frac-decomp)
then have frac $a+$ real $c=a$
using $f 4$ f3 by (metis One-nat-def add.right-neutral add-Suc-right assms(1) assms(2))
then show ?thesis
using f5 f1 assms(1) assms(3) by fastforce
qed
lemma delayedR-correct-aux:
fixes $I r$
defines $R \equiv$ region $\mathcal{X}$ Ir
assumes $u \in R$ valid-region $\mathcal{X}$ k $\operatorname{Ir} \forall c \in \mathcal{X}$. $\neg$ isConst $(I c)$
$\forall c \in \mathcal{X}$. isIntv $(I c) \longrightarrow(u \oplus t) c<i n t v-c o n s t(I c)+1$
$t \geq 0$
shows $u \oplus t \in R$ unfolding $R$-def

```
proof
    from assms have R}\in\mathcal{R}\mathrm{ unfolding }\mathcal{R}\mathrm{ -def by auto
    with }\langleu\inR\rangle\mathcal{R}-V\mathrm{ have }u\inV\mathrm{ by auto
    with }\langlet\geq0\rangle\mathrm{ show }\forallx\in\mathcal{X}.0\leq(u\oplust)x\mathrm{ unfolding V-def by (auto simp:cval-add-def)
    have intv-elem x (u\oplust) (Ix) if x\in\mathcal{X}\mathrm{ for }x
    proof (cases I x)
        case Const
        with assms }\langlex\in\mathcal{X}\rangle\mathrm{ show ?thesis by auto
    next
        case (Intv c)
        with assms <x \in\mathcal{X}>\mathrm{ show ?thesis by (simp add: cval-add-def) (rule; force)}
    next
        case (Greater c)
        with assms }\langlex\in\mathcal{X}\rangle\mathrm{ show ?thesis by (fastforce simp add: cval-add-def)
    qed
    then show }\forallx\in\mathcal{X}\mathrm{ . intv-elem x (u }\oplust)(Ix).
    let ? }\mp@subsup{X}{0}{}={x\in\mathcal{X}.\existsd.Ix= Intv d
    show ? }\mp@subsup{X}{0}{}=?\mathrm{ ? }\mp@subsup{X}{0}{}\mathrm{ by auto
    have frac (ux+t)= frac(ux)+t if x\in? (X for x
    proof -
        show ?thesis
            apply (rule delayedR-correct-aux-aux[where c=intv-const (I x)])
        using assms «x \in? }\mp@subsup{X}{0}{}\rangle\mathrm{ by (force simp add: cval-add-def)+
    qed
    then have frac (ux)\leqfrac(uy)\longleftrightarrow \longleftrightarrowfrac (ux+t)\leqfrac(uy+t) if x\in? ( 
    using that by auto
    with assms show
        \forallx\in? X X . }\forally\in?\mp@subsup{?}{0}{}.((x,y)\inr)=(frac ((u\oplust)x)\leqfrac ((u\oplust)y)
    unfolding cval-add-def by auto
qed
lemma delayedR-correct-aux':
    fixes I r
    defines }R\equiv\mathrm{ region X I r
    assumes u\oplust1\inR valid-region \mathcal{X k I r}\forallc\in\mathcal{X}.\negisConst (I c)
                \forallc\in\mathcal{X}.isIntv (I c)\longrightarrow(u\oplust2) c<intv-const (I c) + 1
        t1\leqt2
    shows }u\oplust2\in
proof -
    have (u\oplust1)\oplus(t2 - t1)\inR unfolding R-def
        using assms by - (rule delayedR-correct-aux, auto simp: cval-add-def)
    then show }u\oplust2\inR\mathrm{ by (simp add: cval-add-def)
qed
lemma valid-regions-intv-distinct:
valid-region \(X k\) Ir \(\Longrightarrow\) valid-region \(X k I^{\prime} r^{\prime} \Longrightarrow u \in\) region \(X I r \Longrightarrow u \in\) region \(X I^{\prime} r^{\prime}\)
\(\Longrightarrow x \in X \Longrightarrow I x=I^{\prime} x\)
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 \((k x)\left(I^{\prime} x\right)\) by auto
moreover from \(A(3) x\) have intv-elem \(x u(I x)\) by auto
moreover from \(A(4) x\) have intv-elem \(x u\left(I^{\prime} x\right)\) by auto
ultimately show \(I x=I^{\prime} x\) using valid-intv-distinct by fastforce
qed
```

```
lemma delayedR-correct:
    fixes Ir
    defines }\mp@subsup{R}{}{\prime}\equiv\mathrm{ \egion X I Ir
    assumes u\inR R \in\mathcal{R}\mathrm{ valid-region X k Ir }\forallc\in\mathcal{X}.\negisConst (I c) R'\inSucc \mathcal{R}R
    shows
        delayed R R' u \in R'
        \exists t\geq0.delayedR R'}u=u\oplus
            \wedget\geq(1-Max ({frac (uc)|c.c\in\mathcal{X}\wedge isIntv (Ic)}\cup{0})) / 2
proof -
    let ?u=SOME u.u\inR
    let ?I =SOME I. \exists r.valid-region \mathcal{X k I r }\wedge \mp@subsup{R}{}{\prime}=\mathrm{ region X X I r}
    let ?S ={frac (uc)|c.c\in\mathcal{X}\wedge isIntv (Ic)}
    let ?m}=1-\operatorname{Max}(?S\cup{0}
    let ?t = SOME t. u }\oplust\in\mp@subsup{R}{}{\prime}\wedget\geq?m/
    have Max (?S\cup{0})\geq0?m\leq1 using finite(1) by auto
    have Max (?S \cup{0})\in?S\cup{0} using finite(1) by - (rule Max-in; auto)
    with frac-lt-1 have Max (?S\cup{0})\leq1 ?m \geq0 by auto
    from assms(3,6)<u\inR> obtain t where t:
        u}\oplust\in\mp@subsup{R}{}{\prime}t\geq
    by (metis alpha-interp.regions-closed'-spec alpha-interp.set-of-regions-spec)
```



```
    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}\mathrm{ for c}<0
    proof -
        from assms have
            \exists r.valid-region \mathcal{X}k ?I r }\wedge\mp@subsup{R}{}{\prime}=\mathrm{ region X ? ?I r
        by - (rule someI[where P=\lambdaI.\existsr.valid-region \mathcal{X kIr }^\mp@subsup{|}{}{\prime}=\mathrm{ region X I I r]; auto)})
        with I-cong that show ?thesis by auto
    qed
    then have ?S ={frac (uc)| c.c\in\mathcal{X}\wedge isIntv (?I c)} by auto
    have upper-bound: (u\oplus?m/2) c<intv-const (Ic)+1 if c\in\mathcal{X}}\mathrm{ isIntv (I c) for c
    proof (cases u c> intv-const (I c))
        case True
        from t that assms have uc+t< intv-const (I c) + 1 unfolding cval-add-def by fastforce
        with 〈t\geq0\rangle True have *: intv-const (I c)<u c u c<intv-const (I c) + 1 by auto
        have frac (u c)\leqMax (?S\cup{0}) using finite(1) that by - (rule Max-ge; auto)
        then have ?m}\leq1-frac (uc) by aut
        then have ?m / 2 < 1 - frac (u c) using * nat-intv-frac-decomp by fastforce
        then have (u\oplus?m / 2) c<uc+1 - frac (uc) unfolding cval-add-def by auto
        also from * have
        ... = 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 uc\leqintv-const (I c) by auto
    moreover from }\langle0\leq?m\rangle\langle?m\leq1\rangle\mathrm{ have ?m / 2< 1 by auto
    ultimately have uc+?m / 2 < intv-const (Ic)+1 by linarith
    then show ?thesis by (simp add: cval-add-def)
    qed
    have ?t }\geq0\wedgeu\oplus?t\in\mp@subsup{R}{}{\prime}\wedge??t\geq?m/
    proof (cases t\geq?m / 2)
        case True
```



```
        by - (rule someI; auto)
    with <? m \geq0\rangle show ?thesis by auto
    next
    case False
    have }u\oplus?m/2\in\mp@subsup{R}{}{\prime}\mathrm{ unfolding R'-def
        apply (rule delayedR-correct-aux')
```

```
            apply (rule t[unfolded R'-def])
            apply (rule assms)+
    using upper-bound False by auto
    with〈?m}\geq0\rangle\mathrm{ show ?thesis by - (rule someI2; fastforce)
    qed
    then show delayedR R' }u\in\mp@subsup{R}{}{\prime}\existst\geq0.delayedR R' u=u\oplust\wedget\geq?m / 2
        by (auto simp: delayed R-def <?S = ->)
qed
definition
    rept :: 's*('c,t) cval =>('s*('c,t) cval set) pmf => ('s*('c,t) cval) pmf
where
    rept s \mu-abs\equivlet (l,u)=s in
        if }(\exists\mp@subsup{R}{}{\prime}.(l,u)\inS\wedge\mu-abs=return-pmf (l, R')\wedge
            (([u\mp@subsup{]}{\mathcal{R}}{}=\mp@subsup{R}{}{\prime}\wedge(\forallc\in\mathcal{X}.uc>kc))))
        then return-pmf (l,u\oplus0.5)
        else if
            (\exists R'.(l,u) \inS ^ -abs = return-pmf (l, R')^ R'\inSucc \mathcal{R}}([u\mp@subsup{]}{\mathcal{R}}{\prime})\wedge[u\mp@subsup{]}{\mathcal{R}}{}\not=\mp@subsup{R}{}{\prime
                \wedge(\forallu\in R'.}\forallc\in\mathcal{X}.#d.d\leqkc\wedgeuc=reald)
    then return-pmf (l, delayedR (SOME R'. }\mu\mathrm{ -abs = return-pmf (l, R'))u)
    else SOME }\mu.\mu\inKs\wedge\mathrm{ abst }\mu=\mu\mathrm{ -abs
```


## lemma $\mathcal{S}$ - $L$ :

## $l \in L$ if $(l, R) \in \mathcal{S}$

using that unfolding $\mathcal{S}$-def by auto
lemma $\mathcal{S}$-inv:
$(l, R) \in \mathcal{S} \Longrightarrow R \subseteq\{$ inv-of $A l\}$
unfolding $\mathcal{S}$-def by auto
lemma upper-right-closed:
assumes $\forall c \in \mathcal{X}$. real $(k c)<u c u \in R R \in \mathcal{R} t \geq 0$
shows $u \oplus t \in R$
proof -
from $\langle R \in \mathcal{R}\rangle$ obtain $I r$ where $R$ :
$R=$ region $\mathcal{X}$ I r valid-region $\mathcal{X} k$ Ir
unfolding $\mathcal{R}$-def by auto
from assms $\mathcal{R}-V$ have $u \in V$ by auto
from assms $R$ have $\forall c \in \mathcal{X}$. I $c=\operatorname{Greater}(k c)$ by safe (case-tac I c; fastforce)
with $R\langle u \in V\rangle$ assms show $u \oplus t \in R$
unfolding $V$-def by safe (rule; force simp: cval-add-def)
qed
lemma $S-I[$ intro $]$ :
$(l, u) \in S$ if $l \in L u \in V u \vdash$ inv-of $A l$
using that by (auto simp: $S$-def $V$-def)
lemma rept-ex:
assumes $\mu \in \mathcal{K}$ (abss s)
shows rept $s \mu \in K s \wedge$ abst (rept s $\mu$ ) $=\mu$ using assms
proof cases
case prems: (delay $\left.l R R^{\prime}\right)$
then have $R \in \mathcal{R}$ by auto
from $\operatorname{prems}(2)$ have $s \in S$ by (auto intro: $\mathcal{S}$-abss-S)
from abss-S $D[O F \operatorname{prems}(2)]$ obtain $l^{\prime} u^{\prime}$ where $s=\left(l^{\prime}, u^{\prime}\right) u^{\prime} \in[u]_{\mathcal{R}}$ by metis
with $\operatorname{prems}(3)$ have $*: s=\left(l, u^{\prime}\right) \wedge u^{\prime} \in R$
apply $\operatorname{simp}$
apply (subst (asm) abss-S[OF S-abss-S])
using $\operatorname{prems}(2)$ by auto
with $\operatorname{prems}(4)$ alpha-interp.set-of-regions-spec $[O F\langle R \in \mathcal{R}\rangle]$ obtain $t$ where $R^{\prime}$ :

$$
t \geq 0 R^{\prime}=\left[u^{\prime} \oplus t\right]_{\mathcal{R}}
$$

by auto
with $\langle s \in S\rangle *$ have $u^{\prime} \oplus t \in R^{\prime} u^{\prime} \oplus t \in V l \in L$ by auto
with $\operatorname{prems}(5)$ have $\left(l, u^{\prime} \oplus t\right) \in S$ unfolding $S$-def $V$-def by auto
with $\left\langle R^{\prime}=\left[u^{\prime} \oplus t\right]_{\mathcal{R}}\right\rangle$ have $* *$ : abss $\left(l, u^{\prime} \oplus t\right)=\left(l, R^{\prime}\right)$ by (auto simp: abss-S)
let $? \mu=$ return-pmf $\left(l, u^{\prime} \oplus t\right)$
have $? \mu \in K s$ using $*\langle s \in S\rangle\langle t \geq 0\rangle\left\langle u^{\prime} \oplus t \in R^{\prime}\right\rangle$ prems by blast
moreover have abst $? \mu=\mu$ by (simp add: ** abst-def prems(1))
moreover note default $=$ calculation
have $R^{\prime} \in \mathcal{R}$ using $\operatorname{prems(4)~by~auto~}$
have $R$ : $[u]_{\mathcal{R}}=R$ by (simp add: $*\langle R \in \mathcal{R}\rangle$ alpha-interp.region-unique-spec)
from $\left\langle R^{\prime} \in \mathcal{R}\right\rangle$ obtain $\operatorname{Ir}$ where $R^{\prime}$ :
$R^{\prime}=$ region $\mathcal{X}$ I r valid-region $\mathcal{X} k I r$
unfolding $\mathcal{R}$-def by auto
have $u^{\prime} \in V$ using $*$ prems $\mathcal{R}-V$ by force
let $? \mu^{\prime}=$ return-pmf $\left(l, u^{\prime} \oplus 0.5\right)$
have elapsed: abst (return-pmf $\left.\left(l, u^{\prime} \oplus t\right)\right)=\mu$ return-pmf $\left(l, u^{\prime} \oplus t\right) \in K s$
if $u^{\prime} \oplus t \in R^{\prime} t \geq 0$ for $t$
proof -
let ? $u=u^{\prime} \oplus t$ let $? \mu^{\prime}=$ return-pmf $\left(l, u^{\prime} \oplus t\right)$
from $\left\langle ? u \in R^{\prime}\right\rangle\left\langle R^{\prime} \in \mathcal{R}\right\rangle \mathcal{R}$ - $V$ have $? u \in V$ by auto
with $\left\langle ? u \in R^{\prime}\right\rangle\left\langle R^{\prime} \in \mathcal{R}\right\rangle$ have $[? u]_{\mathcal{R}}=R^{\prime}$ using alpha-interp.region-unique-spec by auto
with $\langle ? u \in V\rangle\left\langle ? u \in R^{\prime}\right\rangle\langle l \in L\rangle \operatorname{prems}(4,5)$ have abss $(l, ? u)=\left(l, R^{\prime}\right)$
by (subst abss-S) auto
with $\operatorname{prems}(1)$ have abst ? $\mu^{\prime}=\mu$ by (auto simp: abst-def)
moreover from $*\left\langle ? u \in R^{\prime}\right\rangle\langle s \in S\rangle$ prems $\langle t \geq 0\rangle$ have $? \mu^{\prime} \in K s$ by auto
ultimately show abst ? $\mu^{\prime}=\mu ? \mu^{\prime} \in K$ by auto
qed
show ?thesis
proof (cases $R=R^{\prime}$ )
case $T$ : True
show ?thesis
proof (cases $\forall c \in \mathcal{X} . u^{\prime} c>k c$ )
case True
with $T * R \operatorname{prems}(1,4)\langle s \in S\rangle$ have
rept s $\mu=$ return-pmf $\left(l, u^{\prime} \oplus 0.5\right)($ is $-=? \mu)$
unfolding rept-def by auto
from upper-right-closed[OF True] $*\left\langle R^{\prime} \in \mathcal{R}\right\rangle T$ have $u^{\prime} \oplus 0.5 \in R^{\prime}$ by auto
with elapsed 〈rept $--=-\rangle$ show ?thesis by auto
next
case False
with $T * R \operatorname{prems}(1)$ have
rept s $\mu=\left(S O M E \mu^{\prime} . \mu^{\prime} \in K s \wedge\right.$ abst $\left.\mu^{\prime}=\mu\right)$
unfolding rept-def by auto
with default show ?thesis by simp (rule someI; auto)
qed
next
case F: False
show ?thesis
proof (cases $\forall u \in R^{\prime} . \forall c \in \mathcal{X} . \nexists d . d \leq k c \wedge u c=$ real $\left.d\right)$
case False
with $F * R \operatorname{prems}(1)$ have
rept s $\mu=\left(\right.$ SOME $\mu^{\prime} . \mu^{\prime} \in K s \wedge$ abst $\left.\mu^{\prime}=\mu\right)$
unfolding rept-def by auto
with default show ?thesis by simp (rule someI; auto)
next
case True
from True $F * R \operatorname{prems}(1,4)\langle s \in S\rangle$ have
rept s $\mu=$ return-pmf (l, delayed $R\left(S O M E R^{\prime} . \mu=\right.$ return-pmf $\left.\left.\left(l, R^{\prime}\right)\right) u^{\prime}\right)$
(is - $=$ return-pmf $\left(l\right.$, delayed $\left.\left.R ? R u^{\prime}\right)\right)$
unfolding rept-def by auto
let $? u=$ delayed $R ? R u^{\prime}$
from $\operatorname{prems}(1)$ have $\mu=$ return-pmf (l,?R) by auto
with $\operatorname{prems}(1)$ have ? $R=R^{\prime}$ by auto
moreover from $R^{\prime}$ True $\left\langle-\in R^{\prime}\right\rangle$ have $\forall c \in \mathcal{X}$. $\neg$ Regions.isConst (I c) by fastforce
moreover note delayed $R$-correct $\left[\right.$ of $\left.u^{\prime} R I r\right] *\langle R \in \mathcal{R}\rangle R^{\prime}$ True $\left\langle R^{\prime} \in \operatorname{Succ} \mathcal{R} R\right\rangle$
ultimately obtain $t$ where $* *$ : delayed $R R^{\prime} u^{\prime} \in R^{\prime} t \geq 0$ delayed $R R^{\prime} u^{\prime}=u^{\prime} \oplus t$ by auto moreover from $\langle ? R=-\rangle\langle$ rept $--=-\rangle$ have rept s $\mu=$ return-pmf ( $l$, delayed $R R^{\prime} u^{\prime}$ ) by auto ultimately show ?thesis using elapsed by auto
qed
qed
next
case prems: (action $l$ R $\tau \mu^{\prime}$ )
from $a b s s-\mathcal{S} D^{\prime}[$ OF $\operatorname{prems}(2,3)]$ 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: $\mathcal{S}$-abss- $S$ )
let ? $\mu=\operatorname{map-pmf}(\lambda(X, l) .(l,[X:=0] u)) \mu^{\prime}$
from $u$ prems have ? $\mu \in K s$ by (fastforce intro: $\mathcal{S}$-abss-S)
moreover have abst ? $\mu=\mu$ unfolding prems(1) abst-def
proof (subst map-pmf-comp, rule pmf.map-cong, safe, goal-cases)
case $A:\left(1 X l^{\prime}\right)$
from $u$ have $u \in V$ using $\mathcal{R}-V$ by auto
then have $[X:=0] u \in V$ by auto
from $\operatorname{prems}(1) A$
have $\left(l^{\prime}\right.$, region-set $R(S O M E r$. set $\left.r=X) 0\right) \in \mu$ by auto
from A prems $R$-G.K-closed $\langle\mu \in->$ have
$l^{\prime} \in L$ region-set $R(S O M E r$. set $r=X) 0 \subseteq\left\{\right.$ inv-of $\left.A l^{\prime}\right\}$
by (force dest: $\mathcal{S}$-L $\mathcal{S}$-inv) +
with $u$ have $[X:=0] u \vdash$ inv-of $A l^{\prime}$ unfolding region-set'-def by auto
with $\left\langle l^{\prime} \in L\right\rangle\langle[X:=0] u \in V\rangle$ have $\left(l^{\prime},[X:=0] u\right) \in S$ unfolding $S$-def $V$-def by auto
then have abss $\left(l^{\prime},[X:=0] u\right)=\left(l^{\prime},\left[[X:=0]_{\mathcal{R}}\right)\right.$ by auto
also have
$\ldots=\left(l^{\prime}\right.$, region-set $R($ SOME $r$. set $\left.r=X) 0\right)$
using region-set'-eq(1)[unfolded transition-def] prems $A u$ by force
finally show ?case.
qed
ultimately have default: ?thesis if rept s $\mu=\left(\right.$ SOME $\mu^{\prime} . \mu^{\prime} \in K s \wedge$ abst $\left.\mu^{\prime}=\mu\right)$ using that
by simp (rule someI; auto)
show ?thesis
proof (cases $\exists R . \mu=$ return-pmf $(l, R))$
case False
with $\langle s=(l, u)\rangle$ have rept s $\mu=\left(S O M E \mu^{\prime} . \mu^{\prime} \in K s \wedge\right.$ abst $\left.\mu^{\prime}=\mu\right)$ unfolding rept-def by auto
with default show ?thesis by auto
next
case True
then obtain $R^{\prime}$ where $R^{\prime}: \mu=$ return-pmf $\left(l, R^{\prime}\right)$ by auto
show ?thesis
proof (cases $R=R^{\prime}$ )
case False
from $R^{\prime} \operatorname{prems}(1)$ have
$\forall\left(X, l^{\prime}\right) \in \mu^{\prime} .\left(l^{\prime}\right.$, region-set $R(S O M E$ r. set $\left.r=X) 0\right)=\left(l, R^{\prime}\right)$
by (auto simp: map-pmf-eq-return-pmf-iff $\left[\right.$ of - $\left.\mu^{\prime}\left(l, R^{\prime}\right)\right]$ )
then obtain $X$ where
region-set ${ }^{\prime} R(S O M E$ r. set $r=X) 0=R^{\prime}(X, l) \in \mu^{\prime}$
using set-pmf-not-empty by force
with $\operatorname{prems}(4)$ have $X \subseteq \mathcal{X}$ by (simp add: admissible-targets-clocks(1))
moreover then have
set $($ SOME r. set $r=X)=X$
by $-($ rule someI-ex, metis finite-list finite(1) finite-subset)

```
        ultimately have set (SOME r. set r=X)\subseteq\mathcal{X}\mathrm{ by auto}
        with alpha-interp.region-reset-not-Succ False <- = R'> u(3,4) have R' }\not=\mathrm{ Succ }\mathcal{R}R\mathrm{ by auto
        with «s = (l, u)〉 R'u(4) False have
            rept s \mu=(SOME \mp@subsup{\mu}{}{\prime}.\mp@subsup{\mu}{}{\prime}\inKs\wedge abst \mp@subsup{\mu}{}{\prime}=\mu)
        unfolding rept-def by auto
        with default show ?thesis by auto
    next
    case T: True
    show ?thesis
    proof (cases }\forallc\in\mathcal{X}\mathrm{ . real (k c)<uc)
        case False
        with T<s = (l,u)〉 R'u(4) have
            rept s }\mu=(SOME \mp@subsup{\mu}{}{\prime}.\mp@subsup{\mu}{}{\prime}\inKs\wedge abst \mp@subsup{\mu}{}{\prime}=\mu
    unfolding rept-def by auto
    with default show ?thesis by auto
    next
        case True
        with T<s= (l,u)〉 R'u(4)<(l,u)\inS` have
            rept s }\mu=\mathrm{ return-pmf (l,u }\oplus0.5
    unfolding rept-def by auto
    from upper-right-closed[OF True] Tu\mathcal{R}-V have }u\oplus0.5\in\mp@subsup{R}{}{\prime}u\oplus0.5\inV\mathrm{ by force+
    moreover then have [u\oplus0.5\mp@subsup{]}{\mathcal{R}}{}=\mp@subsup{R}{}{\prime}
            using T alpha-interp.region-unique-spec u(3,4) by blast
            moreover note * = \langlerept - = -> R'〈abss s \in\mathcal{S}\langle\langleabss s = -> prems(5)
            ultimately have abst (rept s }\mu\mathrm{ ) = }
        apply (simp add: abst-def)
        apply (subst abss-S)
    by (auto simp: \mathcal{S-L S-def V-def T dest: \mathcal{S-inv)}}\mathbf{}\mathrm{ (a)}
    moreover from * <s= ->\langle(l,u)\inS\rangle\langle-\in R'> have
                rept s }\mu\inK\mp@code{S
        apply simp
        apply (rule K.delay)
        by (auto simp: T dest: \mathcal{S-inv)}
        ultimately show ?thesis by auto
        qed
        qed
    qed
next
    case loop
    obtain lu where s=(l,u) by force
    show ?thesis
    proof (cases s }\inS\mathrm{ )
        case T: True
        with «s=-> have *:l l\inLu\in[u\mp@subsup{]}{\mathcal{R}}{}[u\mp@subsup{]}{\mathcal{R}}{}\in\mathcal{R}\mathrm{ abss s = (l, [u] 的) by auto}
        then have abss s=(l,[u\mp@subsup{]}{\mathcal{R}}{})\mathrm{ by auto}
        with }\langles\inS\rangleS\mathrm{ -abss-S have (l, [u]}\mp@subsup{\mathcal{R}}{}{)}\in\mathcal{S}\mathrm{ by auto
        with S-inv have [u\mp@subsup{]}{\mathcal{R}}{}\subseteq{u.u\vdash inv-of A l} by auto
        show ?thesis
        proof (cases }\forallc\in\mathcal{X}\mathrm{ . real ( }kc\mathrm{ c)<uc)
        case True
        with * <\mu = -><s= ->\langles\inS\rangle have
            rept s }\mu=\mathrm{ return-pmf (l,u }\oplus0.5
    unfolding rept-def by auto
    from upper-right-closed[OF True]* have u}\oplus0.5\in[u\mp@subsup{]}{\mathcal{R}}{}\mathrm{ by auto
    moreover with *\mathcal{R}-V have }u\oplus0.5\inV\mathrm{ by auto
```



```
    moreover note * <rept - = -\rangle\langles= ->T < \mu = >><(l, -) \in\mathcal{S}\rangle\mathcal{S}\mathrm{ -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 \(*\langle s=-\rangle\langle\mu=-\rangle\) have
                rept s \(\mu=\left(S O M E \mu^{\prime} . \mu^{\prime} \in K s \wedge\right.\) abst \(\left.\mu^{\prime}=\mu\right)\)
            unfolding rept-def by auto
            with \(\langle\mu=-\rangle\) show ?thesis by simp (rule someI[where \(x=\) return-pmf s], auto simp: abst-def)
        qed
    next
        case False
        with \(\langle s=-\rangle\langle\mu=-\rangle\) have
            rept s \(\mu=\left(\right.\) SOME \(\mu^{\prime} . \mu^{\prime} \in K s \wedge\) abst \(\left.\mu^{\prime}=\mu\right)\)
    unfolding rept-def by auto
    with \(\langle\mu=-\rangle\) 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 \(\mu \in \mathcal{K} s s \in \mathcal{S}\)
    shows abst (rept (reps s) \(\mu\) ) \(=\mu\)
using assms by auto
lemma rept-K2:
    assumes \(\mu \in \mathcal{K} s s \in \mathcal{S}\)
    shows rept (reps s) \(\mu \in K\) (reps s)
using assms by auto
lemma the \(I^{\prime}\) :
    assumes \(P a\)
        and \(\bigwedge x . P x \Longrightarrow x=a\)
    shows \(P(\) THE \(x . P x) \wedge(\forall y . P y \longrightarrow y=(T H E x . P x))\)
using theI assms by metis
lemma cont-cfg-defined:
    fixes \(c f g s\)
    assumes \(c f g \in\) valid-cfg \(s \in\) abst (action cfg)
    defines \(x \equiv\) THE \(x\). abss \(x=s \wedge x \in\) action \(c f g\)
    shows (abss \(x=s \wedge x \in\) action cfg \() \wedge(\forall y\).abss \(y=s \wedge y \in\) action cfg \(\longrightarrow y=x)\)
proof -
    from \(\operatorname{assms}(2)\) obtain \(s^{\prime}\) where \(s^{\prime} \in\) action \(c f g s=\) abss \(s^{\prime}\) unfolding abst-def by auto
    with assms show ?thesis unfolding \(x\)-def
    by -(rule the \(I^{\prime}[\) of - \(s]\),auto intro: K-bisim-unique MDP.valid-cfg-state-in-S dest: MDP.valid-cfgD)
qed
definition
    \(a b s c^{\prime}::\left({ }^{\prime} s *\left({ }^{\prime} c, t\right) c v a l\right) c f g \Rightarrow\left({ }^{\prime} s *\left({ }^{\prime} c, t\right) c v a l \operatorname{set}\right) c f g\)
where
    absc' \(c f g=c f g\)-corec
        (abss (state cfg))
        (abst o action)
        ( \(\lambda\) cfg s. cont cfg (THE x. abss \(x=s \wedge x \in\) action cfg)) cfg
```


### 5.2.5 Configuration

## definition

```
absc :: ('s * ('c, t) cval) cfg \(\Rightarrow(' s *(' c, t) c v a l ~ s e t) c f g\)
```

where
absc cfg $=c f g$-corec
(abss (state cfg))
(abst o action)
( $\lambda$ cfg $s$. cont $c f g$ (THE $x$. abss $x=s \wedge x \in$ action $c f g)$ ) cfg

## definition

```
repcs :: 's * ('c, t) cval \(\Rightarrow\left({ }^{\prime} s *\left({ }^{\prime} c, t\right) c v a l ~ s e t\right) c f g \Rightarrow\left(' s *\left({ }^{\prime} c, t\right) c v a l\right) c f g\)
where
repcs s cfg \(=c f g\)-corec
\(s\)
( \(\lambda(s, c f g)\). rept \(s(\) action \(c f g))\)
\(\left(\lambda(s, c f g) s^{\prime} .\left(s^{\prime}\right.\right.\), cont \(c f g\left(\right.\) abss \(\left.\left.\left.s^{\prime}\right)\right)\right)(s, c f g)\)
```


## definition

```
repc cfg = repcs (reps (state cfg)) cfg
```

lemma $\mathcal{S}$-state-absc-repc $[$ simp $]$ :
state $c f g \in \mathcal{S} \Longrightarrow$ state $($ absc $($ repc cfg $))=$ state $c f g$
by (simp add: absc-def repc-def repcs-def)
lemma action-repc:
action $($ repc cfg $)=$ rept $($ reps $($ state $c f g))($ action cfg $)$
unfolding repc-def repcs-def by simp
lemma action-absc:
action $($ absc cfg) $=$ abst (action cfg)
unfolding absc-def by simp
lemma action-absc':
action (absc cfg) = map-pmf abss (action cfg)
unfolding absc-def unfolding abst-def by simp

```
lemma
    notes \(R\)-G.cfg-onD-state[simp del]
    assumes state \(c f g \in \mathcal{S} s^{\prime} \in\) set-pmf (action (repc cfg)) cfg \(\in R\)-G.cfg-on (state cfg)
    shows cont (repc cfg) \(s^{\prime}=\) repcs \(s^{\prime}(\) cont cfg (abss s'))
using assms by (auto simp: repc-def repcs-def abss-reps-id)
lemma cont-repcs1:
    notes \(R\)-G.cfg-onD-state[simp del]
    assumes abss \(s \in \mathcal{S} s^{\prime} \in \operatorname{set}-\mathrm{pmf}\) (action (repcs scfg)) cfg \(\in R\)-G.cfg-on (abss s)
    shows cont (repcs scfg) \(s^{\prime}=\) repcs \(s^{\prime}\left(\right.\) cont cfg (abss \(\left.\left.s^{\prime}\right)\right)\)
using assms by (auto simp: repc-def repcs-def abss-reps-id)
lemma cont-absc-1:
    notes MDP.cfg-onD-state[simp del]
    assumes \(c f g \in\) valid-cfg \(s^{\prime} \in\) set-pmf (action cfg)
    shows cont (absc cfg) (abss s') \(=a b s c(\) cont cfg s')
proof -
    define \(x\) where \(x \equiv\) THE \(x . x^{\sim} s^{\prime} \wedge x \in\) set-pmf (action cfg)
    from \(\operatorname{assms}(2)\) have abss \(s^{\prime} \in \operatorname{set}-\mathrm{pmf}\) (abst (action cfg)) unfolding abst-def by auto
    from cont-cfg-defined[OF assms(1) this] have
        \(\left(x^{\sim} s^{\prime} \wedge x \in\right.\) set-pmf \((\) action \(\left.c f g)\right) \wedge\left(\forall y . y^{\sim} s^{\prime} \wedge y \in \operatorname{set}-p m f(\right.\) action \(\left.c f g) \longrightarrow y=x\right)\)
    unfolding \(x\)-def .
    with assms have \(s^{\prime}=x\) by fastforce
    then show ?thesis
    unfolding absc-def abst-def repc-def \(x\)-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 \inR-G.valid-cfg s \in 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 $\bigwedge c f g . P c f g \Longrightarrow$ state $c f g \in S$
assumes $\bigwedge c f g . P c f g \Longrightarrow$ action $c f g \in K$ (state $c f g)$
assumes $\bigwedge c f g t . P c f g \Longrightarrow t \in$ action $c f g \Longrightarrow P($ cont $c f g t)$
shows $c f g \in$ valid-cfg
proof -
from assms have cfg $\in$ MDP.cfg-on (state cfg) by (coinduction arbitrary: cfg) auto
moreover from assms have state $c f g \in S$ by auto
ultimately show ?thesis by (intro MDP.valid-cfgI)
qed
lemma state-repcD[simp]:
assumes $c f g \in R$-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 $\mathcal{R} g R \in \mathcal{R} u \in R u \vdash g$
shows $R \subseteq\{u . u \vdash g\}$
using assms unfolding ccompatible-def by auto

```
lemma action-abscD[dest]:
    \(c f g \in M D P . c f g-o n s \Longrightarrow\) action \((a b s c c f g) \in \mathcal{K}(\) abss \(s)\)
unfolding absc-def abst-def
proof simp
    assume \(c f g: c f g \in M D P . c f g-o n s\)
    then have action \(c f g \in K s\) by auto
    then show map-pmf abss (action cfg) \(\in \mathcal{K}\) (abss s)
    proof cases
        case prems: (delay lut)
        then have \([u \oplus t]_{\mathcal{R}} \in \mathcal{R}\) by auto
        moreover with prems ccompatible-inv[of \(l\) ] have
            \([u \oplus t]_{\mathcal{R}} \subseteq\{v . v \vdash P T A . i n v-o f A l\}\)
        unfolding ccompatible-def by force
        moreover from prems have abss \((l, u \oplus t)=\left(l,[u \oplus t]_{\mathcal{R}}\right)\) by (subst abss-S) auto
        ultimately show ?thesis using prems by auto
    next
        case prems: (action l \(u g \mu\) )
            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)
                    \(=\operatorname{map-pmf}\left(\lambda(X, l) .\left(l\right.\right.\), region-set \(^{\prime}\left([u]_{\mathcal{R}}\right)(S O M E\) r. set \(\left.\left.r=X) 0\right)\right) \mu\)
            proof -
            have abss \(\left(l^{\prime},[X:=0] u\right)=\left(l^{\prime}\right.\), region-set \({ }^{\prime}\left([u]_{\mathcal{R}}\right)(\) SOME \(r\). set \(\left.r=X) 0\right)\)
                if \(\left(X, l^{\prime}\right) \in \mu\) for \(X l^{\prime}\)
```

```
        proof -
            from that prems have A\vdashl \longrightarrow}\mp@subsup{\longrightarrow}{}{g,\mu,X l}\mp@subsup{l}{}{\prime
                by auto
            from that prems MDP.action-closed [OF - cfg] have ( }\mp@subsup{l}{}{\prime},[X:=0]u)\inS by forc
            then have abss (l', [X:=0]u)=( l',[[X:=0]u\mp@subsup{]}{\mathcal{R}}{})\mathrm{ by auto}
            also have
                \ldots. = (l', region-set'}([u\mp@subsup{]}{\mathcal{R}}{})(SOME r. set r=X) 0)
                using region-set'-eq(1)[OF--\langleA\vdashl\longrightarrow\longrightarrow}\mp@subsup{}{}{g,\mu,X}\mp@subsup{l}{}{\prime}\rangle] prems by aut
            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 \inR-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'\inabst (rept s (action cfg)) unfolding repcs-def abst-def by auto
    with prems have
        abss s}\mp@subsup{s}{}{\prime}\in\mathrm{ 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\inR-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 \inR-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 \(c f g \in R\)-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 scfg) $=s$
unfolding repcs-def by auto
lemma repcs-bisim:
notes $R$-G.cfg-onD-state[simp del]
assumes $c f g \in R$-G.valid-cfg $x \in S x \sim x^{\prime}$ abss $x=$ state $c f g$
shows absc (repcs $x$ cfg) $=$ absc (repcs $x^{\prime} c f g$ )
using assms
proof -
from assms have abss $x^{\prime}=$ state cfg by auto
from assms have abss $x^{\prime} \in \mathcal{S}$ by auto
then have $x^{\prime} \in S$ by (auto intro: $\mathcal{S}$-abss- $S$ )
with assms show ?thesis
proof (coinduction arbitrary: cfg $x x^{\prime}$ )
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 scfg $x x^{\prime}$ )
define $c f g^{\prime}$ where $c f g^{\prime}=$ cont $c f g s$
define $t \quad$ where $t \equiv$ THE $y$. abss $y=s \wedge y \in \operatorname{action}$ (repcs $x$ cfg)
define $t^{\prime}$ where $t^{\prime} \equiv T H E y$.abss $y=s \wedge y \in \operatorname{action}$ (repcs $x^{\prime} c f g$ )
from prems have valid: repcs $x$ cfg $\in$ valid-cfg by (intro repcs-valid)
from prems have $*: s \in$ abst (action (repcs $x$ cfg))
unfolding $c f g^{\prime}$-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 $s \in \mathcal{S}$ by (auto intro: $R$-G.valid-cfg-action)
from cont-cfg-defined $[O F$ valid $*$ ] have $t$ :
abss $t=s t \in$ action (repcs $x c f g$ )
unfolding $t$-def by auto
have cont (absc (repcs $x c f g)$ ) $s=$ cont (absc (repcs $x c f g)$ ) (abss t) using $t$ by auto
have cont (absc (repcs $x$ cfg)) $s=a b s c($ cont (repcs xcfg) $t$ )
using $t$ valid by (auto simp: cont-absc-1)
also have $\ldots=a b s c($ 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 $x c f g)) s=a b s c(r e p c s t(c o n t c f g s))$.
from prems have valid: repcs $x^{\prime} c f g \in v a l i d-c f g$ by auto
from $\left\langle s \in\right.$ action cfg〉 prems have $s \in$ abst (action (repcs $x^{\prime}$ cfg))
by (auto dest: R-G.valid-cfgD simp: repcs-def)
from cont-cfg-defined $\left[O F\right.$ valid this] have $t^{\prime}$ :
abss $t^{\prime}=s t^{\prime} \in$ action (repcs $x^{\prime} c f g$ )
unfolding $t^{\prime}$-def by auto
have cont (absc (repcs $x^{\prime}$ cfg)) $s=$ cont (absc (repcs $x^{\prime}$ cfg)) (abss $\left.t^{\prime}\right)$ using $t^{\prime}$ by auto
have cont (absc (repcs $\left.\left.x^{\prime} c f g\right)\right) s=a b s c\left(c o n t\left(r e p c s x^{\prime} c f g\right) t^{\prime}\right)$

```
        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\inaction cfg` prems(1) t t' <s\in\mathcal{S}\rangle
    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 \(\in R\)-G.valid-cfg abss \(s=\) state \(c f g\)
    shows absc (repcs s cfg) \(=c f g\) 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 \(c f g^{\prime}\) where \(c f g^{\prime} \equiv\) repcs s \(c f g\)
    define \(t \quad\) where \(t \equiv\) THE x. abss \(x=s^{\prime} \wedge x \in\) set-pmf (action cfg')
    from prems have \(c f g \in R\)-G.cfg-on (state cfg) state cfg \(\in \mathcal{S}\) by (auto dest: \(R\) - \(G\)-I)
    then have \(*: c f g \in R\) - G.cfg-on (abss (reps (state cfg))) abss (reps (state cfg)) \(\in \mathcal{S}\) by auto
    from prems have \(s^{\prime} \in \mathcal{S}\) by (auto intro: \(R\)-G.valid-cfg-action)
    from prems have valid: cfg' \(\in\) valid-cfg unfolding \(c f g^{\prime}\)-def by (intro repcs-valid)
    from prems have \(s^{\prime} \in\) abst (action cfg') unfolding \(c f g^{\prime}\)-def by (subst action-abst-repcs)
    from cont-cfg-defined \([\) OF valid this] have \(t\) :
        abss \(t=s^{\prime} t \in\) action \(c f g^{\prime}\)
    unfolding \(t\)-def cfg'-def by auto
    with prems have \(t^{\sim}\) reps (absst)
    apply -
    apply (subst \(\mathcal{S}\)-abss-reps)
    by (auto intro: \(R\)-G.valid-cfg-action)
    have cont (absc cfg') \(s^{\prime}=\) cont (absc cfg') (abss t) using \(t\) by auto
    have cont (absc cfg') \(s^{\prime}=a b s c\) (cont cfg' \(t\) ) using \(t\) valid by (auto simp: cont-absc-1)
    also have \(\ldots=\) absc (repcs \(t\) (cont cfg \(s^{\prime}\) )) using prems \(t *\left\langle t^{\sim}\right.\)->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 \(c f g\) s \(\}\), rule exI \([\) where \(x=t]\) )
    unfolding \(c f g^{\prime}\)-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 \(c f g \in R\)-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:
    \(c f g \in v a l i d-c f g \Longrightarrow K-c f g(a b s c c f g)=m a p-p m f a b s c(K-c f g c f g)\)
```

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 (fog)
by (auto simp: stream.map-comp)
lemma state-abscD[simp]:
    assumes cfg\inMDP.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 Pcfg
    assumes \bigwedgecfg. P cfg \Longrightarrow state cfg \in\mathcal{S}
    assumes \bigwedgecfg. P cfg \Longrightarrowaction cfg \in\mathcal{K}(\mathrm{ state cfg)}
    assumes \bigwedgecfg t. Pcfg \Longrightarrowt\inaction cfg \LongrightarrowP(cont cfg t)
    shows cfg \inR-G.valid-cfg
proof -
    from assms have cfg \inR-G.cfg-on (state cfg) by (coinduction arbitrary:cfg) auto
    moreover from assms have state cfg \in\mathcal{S}\mathrm{ by auto}
    ultimately show ?thesis by (intro R-G.valid-cfgI)
qed
lemma absc-valid[intro]:
    assumes cfg \in valid-cfg
    shows absc cfg \inR-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: (3 s'cfg)
    define t where t\equivTHE x.abss x = s'^ x set-pmf (action cfg)
    let ?cfg= cont cfg t
    from prems obtain s}\mathrm{ where }\mp@subsup{s}{}{\prime}=\mathrm{ abss s s action cfg by (auto simp: action-absc')
    with cont-cfg-defined[OF prems(1), of s'] have
        abss t= s't\in set-pmf (action cfg)
        \forall. abss y= s'^y\in set-pmf (action cfg) \longrightarrowy=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 \in valid-cfg cfg' \inK-cfg cfg
    shows absc cfg'}\inK-cfg (absc cfg
using assms by (auto simp: K-cfg-map-absc)
lemma abst-action-repcs:
    assumes cfg \inR-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 \inR-G.valid-cfg
```

```
    shows abst (action (repc cfg)) = action cfg
using assms unfolding repc-def by (auto intro:abst-action-repcs simp: R-G-I)
lemma K-elem-abss-inj':
    assumes }\mu\inK
        and }s\in
    shows inj-on abss (set-pmf }\mu\mathrm{ )
using assms K-elem-abss-inj by (simp add: K-bisim-unique inj-onI)
lemma K-cfg-rept-aux:
    assumes cfg \inR-G.valid-cfg abss s = state cfg x\in rept s (action cfg)
    defines t\equiv\lambdacfg'.THE s'. s'\in rept s (action cfg) }\wedge\mp@subsup{s}{}{\prime}~\mp@subsup{}{}{\prime
    shows tcfg' = x
proof -
    from assms have rept s (action cfg) \inKss\inS by (auto simp: R-G-I S S-abss-S)
    from K-bisim-unique[OF this(2,1)-\operatorname{assms(3)] assms(3) show ?thesis unfolding t-def by blast}
qed
lemma K-cfg-rept-action:
    assumes cfg \inR-G.valid-cfg abss s = state cfg cfg' \in set-pmf (K-cfg cfg)
    shows abss(THE s'. s'\in rept s(action cfg) ^abss s' = state cfg') = state cfg'
proof -
    let ? }\mu=r=rept s(action cfg
    from abst-rept-id assms have action cfg = abst ? }\mu\mathrm{ by (auto simp: R-G-I)
    moreover from assms have state cfg'\in action cfg by (auto simp: set-K-cfg)
    ultimately have state cfg' \in abst ? }\mu\mathrm{ by simp
```



```
    with K-cfg-rept-aux[OF assms(1,2) this(1)] show ?thesis by auto
qed
lemma K-cfg-map-repcs:
    assumes cfg \inR-G.valid-cfg abss s = state cfg
    defines repc' \equiv(\lambdacfg'. repcs (THE s'. s' \in rept s (action cfg) ^abss s' = state cfg')cfg')
    shows K-cfg (repcs s cfg) = map-pmf repc' (K-cfg cfg)
proof -
    let ? }\mu=\mathrm{ rept s (action cfg)
```



```
    have t: t(cont cfg (abss s'))= 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\inR-G.valid-cfg
    defines
        repc' cfg' \equiv repcs (THE s. s \in rept (reps (state cfg)) (action cfg) ^ abss s = state cfg') cfg'
    shows
    K-cfg (repc cfg) = map-pmf repc' (K-cfg cfg)
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 \inR-G.valid-cfg cfg' }\inK\mathrm{ -cfg cfg
    shows cfg' = cont cfg (state cfg') state cfg' \in action cfg
proof -
    from assms(2) obtain s where s\inaction cfg cfg' = cont cfg s by (auto simp: set-K-cfg)
    with assms show
        cfg' = cont cfg (state cfg') state cfg' \in action cfg
    by (auto intro: R-G.valid-cfg-state-in-S R-G.valid-cfgD)
```

qed

```
lemma \(K\)-cfg-valid-cfgD:
    assumes \(c f g \in\) valid-cfg \(c f g^{\prime} \in K-c f g ~ c f g\)
    shows \(c f g^{\prime}=\) cont \(c f g\left(\right.\) state \(\left.c f g^{\prime}\right)\) state \(c f g^{\prime} \in\) action \(c f g\)
proof -
    from \(\operatorname{assms}\) (2) obtain \(s\) where \(s \in\) action \(c f g ~ c f g^{\prime}=c o n t c f g s\) by (auto simp: set-K-cfg)
    with assms show
        \(c f g^{\prime}=\) cont \(c f g\left(\right.\) state \(\left.c f g^{\prime}\right)\) state \(c f g^{\prime} \in\) action \(c f g\)
    by auto
qed
```

lemma absc-bisim-abss:
assumes absc $x=a b s c x^{\prime}$
shows state $x{ }^{\sim}$ state $x^{\prime}$
proof -
from assms have state (absc $x$ ) = state (absc $x^{\prime}$ ) by simp
then show ?thesis by (simp add: state-absc)
qed
lemma $K$-cfg-bisim-unique:
assumes $c f g \in$ valid-cfg and $x \in K-c f g \operatorname{cfg} x^{\prime} \in K-c f g c f g$ and state $x \sim$ state $x^{\prime}$
shows $x=x^{\prime}$
proof -
define $t$ where $t \equiv T H E x^{\prime} . x^{\prime} \sim$ state $x \wedge x^{\prime} \in \operatorname{set}-p m f($ action cfg)
from $K$-cfg-valid-cfgD assms have *:
$x=$ cont cfg (state $x)$ state $x \in$ action cfg
$x^{\prime}=$ cont cfg (state $\left.x^{\prime}\right)$ state $x^{\prime} \in$ action cfg
by auto
with assms have
$c f g \in$ valid-cfg abss (state $x) \in$ set-pmf (abst (action cfg))
unfolding abst-def by auto
with cont-cfg-defined[of cfg abss (state $x$ )] have
$\forall y . y \sim$ state $x \wedge y \in$ set-pmf (action cfg) $\longrightarrow y=t$
unfolding $t$-def by auto
with $* \operatorname{assms}(4)$ have state $x^{\prime}=t$ state $x=t$ by fastforce +
with $*$ show ?thesis by simp
qed
lemma absc-distr-self:
MDP.MC.T (absc cfg) $=\operatorname{distr}($ MDP.MC.T cfg) MDP.MC.S (smap absc) if $c f g \in$ valid-cfg
using 〈cfg $\in-\rangle$
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 $t \equiv \lambda y$.THE $x . y=a b s c x \wedge x \in K$-cfg $c f g$
define $M^{\prime}$ where $M^{\prime} \equiv \lambda c f g$. distr (MDP.MC.T ( $t c f g$ )) MDP.MC.S (smap absc)
show ?case
proof (rule exI[where $\left.x=M^{\eta}\right]$, safe, goal-cases)
case $A$ : (1 y)
from A prems obtain $x^{\prime}$ where $y=a b s c x^{\prime} x^{\prime} \in K-c f g$ cfg by (auto simp: K-cfg-map-absc)
with $K$-cfg-bisim-unique [OF prems - absc-bisim-abss] have
$y=a b s c(t y) x^{\prime}=t y$
unfolding $t$-def by (auto intro: theI2)

```
    moreover have }\mp@subsup{x}{}{\prime}\in\mathrm{ valid-cfg using <x' }\in->> prems by aut
    ultimately show ?case unfolding M'-def by auto
next
    case 5
    show ?case unfolding M'-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 }
        proof -
        have t (absc x) = x unfolding t-def
        proof (rule the-equality, goal-cases)
            case 1 with A show ?case by simp
        next
            case (2 x')
            with K-cfg-bisim-unique[OF prems - A 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'-def intro: MDP.MC.T.prob-space-distr)
qed
lemma R-G-trace-space-distr-eq:
    assumes cfg \inR-G.valid-cfg abss s = state cfg
    shows MDP.MC.T cfg = distr (MDP.MC.T (repcs s cfg)) MDP.MC.S (smap 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=\mathrm{ rept s (action cfg)
```



```
    define M' where }\mp@subsup{M}{}{\prime}\equiv\lambdacfg.distr (MDP.MC.T (repc' cfg)) MDP.MC.S (smap absc
    show ?case
    proof (intro exI[where x = M ], safe, goal-cases)
        case A:(1 cfg')
        with K-cfg-rept-action[OF prems] have
            abss(THE s.s 的 }\mu\wedge\mathrm{ ^abss s = state cfg') = state cfg'
            by auto
            moreover from A prems have cfg'}\inR\mathrm{ -G.valid-cfg by auto
            ultimately show ?case unfolding M'-def repc'-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 \circ (##) (repc' cfg')=(##) cfg'\circ smap absc
    if cfg' \in set-pmf (K-cfg cfg) for cfg'
    proof -
        from K-cfg-rept-action[OF prems that] have
                abss (THE s.s \in? }\mu\wedge\mathrm{ 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}\circ(##)(rep\mp@subsup{c}{}{\prime}cfg'))=((##)cfg'\circ smap absc) by aut
    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\inR-G.cfg-on s s\in\mathcal{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 \inX\Longrightarrow smap abss x= smap abss y \Longrightarrowy\inX
    shows (smap state xs }\inX)=(\mathrm{ smap ( }\lambdaz\mathrm{ . abss (state z)) xs }\in\mathrm{ 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 (\lambdaz.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 \inR-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: $x \in$ space (count-space UNIV)
by $\operatorname{simp}$
lemma $S$-reps-S-aux:
reps $(l, R) \in S \Longrightarrow(l, R) \in \mathcal{S}$
using ccompatible-inv unfolding reps-def ccompatible-def $\mathcal{S}$-def $S$-def
by (cases $R \in \mathcal{R}$; auto simp: non-empty)

```
lemma S-reps-\mathcal{S}[intro]:
    reps s}\inS\Longrightarrows\in\mathcal{S
    using S-reps-\mathcal{S-aux by (metis surj-pair)}
lemma absc-valid-cfg-eq:
    absc'valid-cfg=R-G.valid-cfg
    apply safe
    subgoal
        by auto
    subgoal for cfg
        using absc-repcs-id[where s=reps (state cfg)]
        by - (frule repcs-valid[where s=reps (state cfg)]; force intro: imageI)
    done
```

lemma action-repcs:
action $($ repcs $(l, u) c f g)=\operatorname{rept}(l, u)($ action $c f g)$
by (simp add: repcs-def)

### 5.3 Equalities Between Measures of Trace Spaces

lemma path-measure-eq-absc1-new:

## fixes $c f g s$

defines $c f g^{\prime} \equiv a b s c \quad c f g$
assumes valid: cfg $\in$ valid-cfg
assumes $X[$ measurable $]: X \in R-G . S t$ and $Y[$ measurable $]: Y \in M D P . S t$
assumes $P: A E x$ in ( $R$-G.T cfg $)$. $P x$ and $Q: A E x$ in (MDP.T cfg). $Q x$
assumes $P^{\prime}[$ measurable $]:$ Measurable.pred $R-G . S t P$ and $Q^{\prime}[$ measurable $]$ : Measurable.pred MDP.St $Q$
assumes $X$ - $Y$-closed: $\bigwedge x y$. $P x \Longrightarrow$ smap abss $y=x \Longrightarrow x \in X \Longrightarrow y \in Y \wedge Q y$
assumes $Y$ - $X$-closed: $\bigwedge x y . Q y \Longrightarrow$ smap abss $y=x \Longrightarrow y \in Y \Longrightarrow x \in X \wedge P x$
shows
proof -
have $*$ : stream-all2 $(\lambda s .(=)(a b s c s)) x y=$ stream-all2 $(=)($ smap absc $x) y$ for $x y$ by $\operatorname{simp}$
have $*$ : stream-all2 ( $\lambda$ st. $t=$ absc s) $x y=$ stream-all2 $(=) y($ smap absc $x)$ for $x y$ using stream.rel-conversep $[o f \lambda s t . t=a b s c s]$ by (simp add: conversep-iff [abs-def])
from $P$ have emeasure (R-G.T cfg') $X=$ emeasure ( $R$-G.T cfg') $\{x \in X . P x\}$ by (auto intro: emeasure-eq- $A E$ )
moreover from $Q$ have emeasure (MDP.T cfg) $Y=$ emeasure (MDP.T cfg) $\{y \in Y . Q y\}$ by (auto intro: emeasure-eq- $A E$ )
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 $f=a b s c$ 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=\lambdast.t=absc s])
    apply (simp; fail)
    subgoal for }x
    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 \inR-G.valid-cfg
    assumes X[measurable]: X \inR-G.St and Y[measurable]: Y\inMDP.St
    assumes P:AEx in (R-G.Tcfg). Px and Q:AEx in (MDP.Tcfg').Q x
    assumes P'[measurable]: Measurable.pred R-G.St P
        and Q'[measurable]: Measurable.pred MDP.St Q
    assumes X-Y-closed: \xy.Px\Longrightarrow smap abss }y=x\Longrightarrowx\inX\Longrightarrowy\inY\wedgeQ
    assumes Y-X-closed: \xy.Qy\Longrightarrowsmap abss y=x\Longrightarrowy\inY\Longrightarrowx\inX^Px
    shows
        emeasure (R-G.T cfg) X = emeasure (MDP.T cfg') Y
proof -
    have *: stream-all2 ( }\lambda\mathrm{ st. t=absc s) x y = stream-all2 (=) y (smap absc x) for x y
        using stream.rel-conversep[of \lambdast.t=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 和.P x}
        by (auto intro: emeasure-eq-AE)
    moreover from Q Y have
        emeasure (MDP.T cfg') Y = emeasure (MDP.T cfg') {y\inY.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 (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 (auto simp:s absc-repcs-id)
    done
qed
lemma region-compatible-suntil1:
    assumes (holds (\lambdax.\varphi (reps x)) suntil holds ( }\lambda\mathrm{ x. }\psi(\mathrm{ reps x))) (smap abss x)
        and pred-stream ( }\lambda\mathrm{ s. }\varphi(\mathrm{ reps (abss s)) }\longrightarrow\varphis)
        and pred-stream ( }\lambda\mathrm{ s. }\psi(\mathrm{ reps (abss s))}\longrightarrow\psis)
    shows (holds }\varphi\mathrm{ suntil holds }\psi\mathrm{ ) x using assms
proof (induction smap abss x arbitrary: x rule: suntil.induct)
    case base
    then show ?case by (auto intro: suntil.base simp: stream.pred-set)
next
    case step
    have
        pred-stream (\lambdas.\varphi (reps (abss s)) \longrightarrow\varphi s) (stl x)
        pred-stream (\lambdas.\psi(reps (abss s))\longrightarrow\psi s) (stl x)
        using step.prems apply (cases x; auto)
        using step.prems apply (cases x; auto)
        done
    with step.hyps(3)[of stl x] have (holds \varphi suntil holds \psi) (stl x) by auto
    with step.prems step.hyps(1-2) show ?case by (auto intro: suntil.step simp: stream.pred-set)
qed
lemma region-compatible-suntil2:
    assumes (holds \varphi suntil holds \psi) x
        and pred-stream ( }\lambda\mathrm{ s. }\varphis\longrightarrow\varphi(\mathrm{ reps (abss s))) x
        and pred-stream (\lambda s.\psis\longrightarrow\psi (reps (abss s))) x
    shows (holds (\lambdax.\varphi (reps x)) suntil holds ( }\lambdax.\psi(\mathrm{ reps x))) (smap abss x) using assms
proof (induction x rule: suntil.induct)
    case (base x)
    then show ?case by (auto intro: suntil.base simp: stream.pred-set)
next
    case (step x)
    have
        pred-stream (\lambdas.\varphis\longrightarrow\varphi(reps(abss s))) (stl x)
        pred-stream (\lambdas.\psis\longrightarrow\psi(reps (abss s))) (stl x)
        using step.prems apply (cases x; auto)
        using step.prems apply (cases x; auto)
        done
    with step show ?case by (auto intro: suntil.step simp: stream.pred-set)
qed
lemma region-compatible-suntil:
    assumes pred-stream (\lambda s.\varphi (reps (abss s))\longleftrightarrow\varphis)x
        and pred-stream (\lambda s.\psi(reps (abss s)) \longleftrightarrow\psi s)x
    shows (holds ( }\lambdax.\varphi(\mathrm{ reps x)) suntil holds ( }\lambdax.\psi(\mathrm{ reps x))) (smap abss x)
        \longleftrightarrow (holds \varphi suntil holds \psi) x using 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- \(\mathcal{S} \mathcal{S}\)-abss-S assms)
lemma measurable-sset[measurable (raw)]:
    assumes \(f[\) measurable \(]: f \in N \rightarrow_{M}\) stream-space \(M\) and \(P[\) measurable \(]\) : Measurable.pred \(M P\)
    shows Measurable.pred \(N(\lambda x . \forall s \in\) sset \((f x) . P s)\)
proof -
    have \(*:(\lambda x . \forall s \in \operatorname{sset}(f x) . P s)=(\lambda x . \forall i . P(f x!!i))\)
        by (simp add: sset-range)
    show ?thesis
        unfolding * by measurable
qed
lemma path-measure-eq-repcs" \({ }^{\prime \prime}\)-new:
    notes in-space-UNIV[measurable]
    fixes \(c f g \varphi \psi s\)
    defines \(c f g^{\prime} \equiv\) repcs s cfg
    defines \(\varphi^{\prime} \equiv a b s p \varphi\) and \(\psi^{\prime} \equiv a b s p \psi\)
    assumes \(s\) : abss \(s=\) state cfg
    assumes valid: cfg \(\in R\)-G.valid-cfg
    assumes valid': cfg' \(\in\) valid-cfg
    assumes equiv- \(\varphi: \bigwedge x\). pred-stream \((\lambda s . s \in S) x\)
                        \(\Longrightarrow\) pred-stream \((\lambda\) s. \(\varphi(\) reps \((\) abss s) \() \longleftrightarrow \varphi\) s) \((\) state cfg \(\# \# x)\)
        and equiv- \(\psi: \bigwedge x\). pred-stream \((\lambda s . s \in S) x\)
                        \(\Longrightarrow\) pred-stream \(\left(\lambda\right.\) s. \(\psi(\) reps \((\) abss \(s)) \longleftrightarrow \psi\) s) \(\left(\right.\) state \(\left.c f g^{\prime} \# \# x\right)\)
    shows
        emeasure (R-G.T cfg) \(\left\{x \in\right.\) space \(R\)-G.St. (holds \(\varphi^{\prime}\) suntil holds \(\left.\psi^{\prime}\right)(\) state cfg \(\left.\# \# x)\right\}=\)
        emeasure (MDP.T cfg') \{x space MDP.St. (holds \(\varphi\) suntil holds \(\psi\) ) (state cfg' \#\# x) \}
    unfolding \(c f g^{\prime}\)-def
    apply (rule path-measure-eq-repcs1-new \([\) where \(P=\operatorname{pred}\)-stream \((\lambda s . s \in \mathcal{S})\) and \(Q=\) 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 \(A E-m p[O F M D P . M C . A E-T\)-enabled \(A E\)-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 \(A E-m p[O F ~ M D P . M C . A E-T\)-enabled \(A E\)-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 \(\varphi^{\prime}\)-def \(\psi^{\prime}\)-def absp-def
            apply (subst region-compatible-suntil[symmetric])
            subgoal
            proof -
            from prems have pred-stream ( \(\lambda s . s \in S\) ) xs using \(\mathcal{S}\)-abss-S by (auto simp: stream.pred-set)
```

```
        with equiv-\varphi show ?thesis by (simp add: cfg'-def)
    qed
    subgoal
    proof -
        from prems have pred-stream (\lambdas.s\inS) xs using \mathcal{S}}\mathrm{ -abss-S by (auto simp: stream.pred-set)
        with equiv-\psi show ?thesis by (simp add: cfg'-def)
    qed
    using valid prems
        apply (auto simp:s comp-def }\mp@subsup{\varphi}{}{\prime}\mathrm{ -def }\mp@subsup{\psi}{}{\prime}\mathrm{ -def absp-def dest: R-G.valid-cfg-state-in-S)
    apply (auto simp: stream.pred-set intro: \mathcal{S}}\mathrm{ -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 \mathcal{S}}\mathrm{ -abss-S; measurable; fail)
    using prems unfolding }\mp@subsup{\varphi}{}{\prime}\mathrm{ -def }\mp@subsup{\psi}{}{\prime}\mathrm{ -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 (\lambdas.s\inS) xs using \mathcal{S}}\mathrm{ -abss-S by auto
        with equiv-\varphi show ?thesis using valid by (simp add: cfg'-def repc-def)
    qed
    subgoal
    proof -
        from prems have pred-stream (\lambdas.s\inS) xs using \mathcal{S}}\mathrm{ -abss-S by auto
        with equiv-\psi show ?thesis using valid by (simp add: cfg'-def)
    qed
    using valid prems by (auto simp: s S-abss-\mathcal{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 $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime}$ 'a stream $\Rightarrow$ bool for $P$ where $P a b \Longrightarrow$ pairwise $P(b \# \# x s) \Longrightarrow$ pairwise $P(a \# \# b \# \# x s)$

## lemma pairwise-Suc:

pairwise $P$ xs $\Longrightarrow P(x s!!i)(x s!!(S u c i))$
by (induction i arbitrary: $x s$ ) (force elim: pairwise.cases)+
lemma Suc-pairwise:

```
\forall i.P(xs !! i)(xs!! (Suc i))\Longrightarrow 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:
pairwise $P x s \longleftrightarrow(\forall$ i. $P(x s!!i)(x s!!($ Suc $i)))$
using pairwise-Suc Suc-pairwise by blast

```
lemma pairwise-stlD:
    pairwise P xs \Longrightarrow pairwise P (stl xs)
by (auto elim: pairwise.cases)
lemma pairwise-pairD:
    pairwise P xs \LongrightarrowP(shd xs) (shd (stl xs))
by (auto elim: pairwise.cases)
lemma pairwise-mp:
    assumes pairwise P xs and lift: \bigwedgexy. x\in sset xs \Longrightarrowy\in sset xs \LongrightarrowPxy\LongrightarrowQxy
    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:
```

lemma gt-GreaterD:
assumes u\in region X Ir valid-region X k Ir c\inXuc>kc
assumes u\in region X Ir valid-region X k Ir c\inXuc>kc
shows I c = Greater ( }kc\mathrm{ )
shows I c = Greater ( }kc\mathrm{ )
proof -
proof -
from assms have intv-elem c u (I c) valid-intv (kc) (I c) by auto
from assms have intv-elem c u (I c) valid-intv (kc) (I c) by auto
with assms(4) show ?thesis by (cases I c) auto
with assms(4) show ?thesis by (cases I c) auto
qed
qed
lemma const-ConstD:
lemma const-ConstD:
assumes u\in region X I r valid-region X k Ir c\inXuc=d d\leqkc
assumes u\in region X I r valid-region X k Ir c\inXuc=d d\leqkc
shows I c= Const d
shows I c= Const d
proof -
proof -
from assms have intv-elem c u (I c) valid-intv (k c) (I c) by auto
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
with assms(4,5) show ?thesis by (cases I c) auto
qed
qed
lemma not-Greater-bounded:
lemma not-Greater-bounded:
assumes I x = Greater ( }kx\mathrm{ x) x f X valid-region X k I r u f region X I r
assumes I x = Greater ( }kx\mathrm{ x) x f X valid-region X k I r u f region X I r
shows ux\leqkx
shows ux\leqkx
proof -
proof -
from assms have intv-elem x u (I x) valid-intv (kx) (I x) by auto
from assms have intv-elem x u (I x) valid-intv (kx) (I x) by auto
with assms(1) show }ux\leqkx\mathrm{ by (cases I x) auto
with assms(1) show }ux\leqkx\mathrm{ by (cases I x) auto
qed
qed
lemma Greater-closed:

```
lemma Greater-closed:
```

```
    fixes t:: real
    assumes u \in region X I r valid-region X k Ir c \in X I c = Greater (kc)t>k c
    shows u(c:=t)\in region X Ir
    using assms
    apply (intro region.intros)
        apply (auto; fail)
        apply standard
    subgoal for }
    by (cases x = c; cases I x; force intro!: intv-elem.intros)
by auto
lemma Greater-unbounded-aux:
    assumes finite X valid-region X kIr c \in X I c = Greater (k c)
    shows }\existsu\in\mathrm{ region X Ir.uc>t
using assms Greater-closed[OF - assms(2-4)]
proof -
    let ?R = region X Ir
    let ?t = if t>k c then t+1 else k c + 1
    have t: ?t > k 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 $x R \equiv \forall t . \exists u \in R . u x>t$
definition zero $x R \equiv \forall u \in R . u x=0$
lemma Greater-unbounded:
assumes finite $X$ valid-region $X k I r c \in X I c=\operatorname{Greater}(k c)$
shows unbounded $c$ (region X I r)
using Greater-unbounded-aux[OF assms] unfolding unbounded-def by blast
lemma unbounded-Greater:
assumes valid-region $X k$ Ir $c \in X$ unbounded $c$ (region X I r)
shows $I c=$ Greater ( $k c$ )
using assms unfolding unbounded-def by (auto intro: gt-GreaterD)
lemma Const-zero:
assumes $c \in X I c=$ Const 0
shows zero $c$ (region X Ir)
using assms unfolding zero-def by force
lemma zero-Const:
assumes finite $X$ valid-region $X k$ Ir $c \in X$ zero $c$ (region X Ir)
shows $I c=$ Const 0
proof -
from assms obtain $u$ where $u \in$ region $X I r$ 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 $X k$ Ir $c \in X u \in$ region X Ir u $c=0$
shows zero c (region X I r)
proof -
from assms have intv-elem cu(Ic) valid-intv $(k c)\left(\begin{array}{l}\text { c }\end{array}\right)$ by auto
then have $I c=$ Const 0 using assms(5) by cases auto
with assms have $u^{\prime} c=0$ if $u^{\prime} \in$ region $X I r$ for $u^{\prime}$ 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 $A::\left({ }^{\prime} c, t,{ }^{\prime} s\right) p t a+$
fixes $\varphi \psi::\left({ }^{\prime} s *\left({ }^{\prime} c, t\right) c v a l\right) \Rightarrow$ bool fixes $s$
assumes $\varphi: \bigwedge x y . x \in S \Longrightarrow x^{\sim} y \Longrightarrow \varphi x \longleftrightarrow \varphi y$
assumes $\psi: \bigwedge x y . x \in S \Longrightarrow x^{\sim} y \Longrightarrow \psi x \longleftrightarrow \psi y$
assumes $s[$ intro, simp $]: s \in S$
begin
definition $\varphi^{\prime} \equiv a b s p \varphi$
definition $\psi^{\prime} \equiv a b s p \psi$
definition $s^{\prime} \equiv$ abss $s$
lemma $s$ - $s^{\prime}$-cfg-on[intro]:
assumes $c f g \in M D P . c f g$-on $s$
shows absc cfg $\in R$-G.cfg-on $s^{\prime}$
proof -
from assms $s$ have $c f g \in$ 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 $s^{\prime}$-def by (auto simp: state-absc)
qed
lemma $s^{\prime}-\mathcal{S}[$ simp, intro $]$ :
$s^{\prime} \in \mathcal{S}$
unfolding $s^{\prime}$-def using $s$ by auto
lemma $s^{\prime}-s-c f g-o n[$ intro $]:$
assumes $c f g \in R$-G.cfg-on $s^{\prime}$
shows repcs $s c f g \in M D P . c f g-o n s$
proof -
from assms $s$ have $c f g \in R$-G.valid-cfg unfolding $R$-G.valid-cfg-def by auto
with assms have repcs $s c f g \in$ valid-cfg by (auto simp: $s^{\prime}$-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 $\varphi: \bigwedge x y . x \in S \Longrightarrow x^{\sim} y \Longrightarrow \varphi x \longleftrightarrow \varphi y$
assumes pred-stream $(\lambda s . s \in S)$ xs
and [intro]: $x \in S$
shows pred-stream $(\lambda s . \varphi($ reps $($ abss $s))=\varphi s)(x \# \# x s)$
unfolding stream.pred-set proof clarify
fix $l u$
assume $A:(l, u) \in \operatorname{sset}(x \# \# x s)$
from assms have pred-stream $(\lambda s . s \in S)(x \# \# x s)$ by auto
with $A$ have $(l, u) \in S$ by (fastforce simp: stream.pred-set)
then have abss $(l, u) \in \mathcal{S}$ by auto
then have reps $($ abss $(l, u)) \sim(l, u)$ by $\operatorname{simp}$
with $\varphi\langle(l, u) \in S\rangle$ show $\varphi($ reps $($ abss $(l, u)))=\varphi(l, u)$ by blast
qed
lemma $\varphi$-stream ${ }^{\prime}$ :
pred-stream $(\lambda s . \varphi($ reps $($ abss $s))=\varphi s)(x \# \# x s)$ if pred-stream $(\lambda s . s \in S) x s x \in S$ using compatible-stream [of $\varphi, O F \varphi$ that $]$.
lemma $\psi$－stream ${ }^{\prime}$ ：
pred－stream $(\lambda s . \psi($ reps $($ abss $s))=\psi s)(x \# \# x s)$ if pred－stream $(\lambda s . s \in S) x s x \in S$ using compatible－stream［of $\psi$ ，OF $\psi$ that $]$ ．
lemmas $\varphi$－stream $=$ compatible－stream $[o f \varphi, O F \varphi]$
lemmas $\psi$－stream $=$ compatible－stream $[$ of $\psi, O F \psi]$

## 7．2 Easier Result on All Configurations

```
lemma suntil-reps:
    assumes
    \foralls\insset (smap abss y). s\in\mathcal{S}
    (holds \varphi' suntil holds \psi') (s' ## smap abss y)
    shows (holds }\varphi\mathrm{ suntil holds }\psi)(s## y
    using assms
    by (subst region-compatible-suntil[symmetric]; (intro \varphi-stream \psi-stream)?)
        (auto simp: \varphi'-def \psi'-def absp-def stream.pred-set S-abss-S s'-def comp-def)
lemma suntil-abss:
    assumes
        \foralls\insset y. s\inS
        (holds }\varphi\mathrm{ suntil holds }\psi\mathrm{ ) (s## y)
    shows
        (holds \varphi' suntil holds \psi') (s'## smap abss y)
    using assms
    by (subst (asm) region-compatible-suntil[symmetric]; (intro \varphi-stream \psi-stream)?)
        (auto simp: \varphi'-def \psi'-def absp-def stream.pred-set s'-def comp-def)
```

    theorem \(P\)-sup-sunitl-eq:
    notes \([\) measurable \(]=\) in-space-UNIV and \([i f f]=\) pred-stream-iff
    shows
        (MDP.P-sup s \((\lambda x\). (holds \(\varphi\) suntil holds \(\psi) \quad(s \quad \# \# x)))\)
        \(=\left(R-G . P\right.\)-sup \(s^{\prime}\left(\lambda x .\left(\right.\right.\) holds \(\varphi^{\prime}\) suntil holds \(\left.\left.\left.\psi^{\prime}\right)\left(s^{\prime} \# \# x\right)\right)\right)\)
    unfolding MDP.P-sup-def \(R-G . P-\) sup-def
    proof (rule SUP-eq, goal-cases)
case prems: ( 1 cfg )
let ${ }^{2} c f g^{\prime}=a b s c c f g$
from prems have $c f g \in$ valid-cfg by (auto intro: MDP.valid-cfgI)
then have ? ${ }^{\prime} f g^{\prime} \in R$-G.valid-cfg by (auto intro: $R$-G.valid-cfgI)
from $\langle c f g \in$ valid-cfg〉 have alw-S: almost-everywhere (MDP.T cfg) (pred-stream ( $\lambda s . s \in S$ ))
by (rule MDP.alw-S)
from〈?cfg' $\mathcal{R}$-G.valid-cfg〉 have alw-S: almost-everywhere $\left(R-G . T\right.$ ? $\left.c f g^{\prime}\right)($ pred-stream $(\lambda s . s \in \mathcal{S}))$
by (rule $R$-G.alw-S)
have emeasure (MDP.T cfg) $\{x \in$ space MDP.St. (holds $\varphi$ suntil holds $\psi$ ) $(s \# \# x)\}$
$=$ emeasure ( $\left.R-G . T ? c f g^{\prime}\right)\left\{x \in\right.$ space $R-G . S t$. (holds $\varphi^{\prime}$ suntil holds $\left.\left.\psi^{\prime}\right)\left(s^{\prime} \# \# x\right)\right\}$
apply (rule path-measure-eq-absc1-new[symmetric, where $P=\operatorname{pred}$-stream $(\lambda s . s \in \mathcal{S})$
and $Q=$ pred-stream $(\lambda s . s \in S)]$
)
using prems alw-S alw-S apply (auto intro: MDP.valid-cfgI simp: )[7]
by (auto simp: S-abss-S intro: $\mathcal{S}$-abss-S intro!: suntil-abss suntil-reps, measurable)
with prems show ?case by (inst-existentials ?cfg') auto
next
case prems: (2 cfg)
let ? $c f g^{\prime}=$ repcs s cfg
have $s=$ state $? c f g^{\prime}$ by $\operatorname{simp}$
from prems have $s^{\prime}=$ state cfg by auto
have pred-stream $(\lambda s . \varphi($ reps $($ abss $s))=\varphi$ s) $($ state $($ repcs $s c f g) \# \# x)$
if pred-stream $(\lambda s . s \in S) x$ for $x$
using prems that by (intro $\varphi$-stream) auto

```
moreover
have pred-stream ( }\lambda\mathrm{ s. }\psi(\mathrm{ reps (abss s)) = * s) (state (repcs s cfg) ## x)
    if pred-stream (\lambdas.s\inS)x for x
    using prems that by (intro \psi-stream) auto
ultimately
have emeasure (R-G.T cfg) {x\in space R-G.St. (holds \varphi' suntil holds \psi') (s'## x)}
    = emeasure (MDP.T (repcs s cfg)) {x\in space MDP.St. (holds \varphi suntil holds \psi) (s## x)}
    apply (rewrite in s## - <s = ->)
    apply (subst «s' = ->)
    unfolding }\mp@subsup{\varphi}{}{\prime}\mathrm{ -def }\mp@subsup{\psi}{}{\prime}\mathrm{ -def }\mp@subsup{s}{}{\prime}\mathrm{ -def
    apply (rule path-measure-eq-repcs''-new)
    using prems by (auto 4 3 simp: s'-def intro:R-G.valid-cfgI MDP.valid-cfgI)
    with prems show ?case by (inst-existentials ?cfg') auto
qed
end
```


### 7.3 Divergent Adversaries

context Probabilistic-Timed-Automaton
begin
definition elapsed $u u^{\prime} \equiv \operatorname{Max}\left(\left\{u^{\prime} c-u c \mid c . c \in \mathcal{X}\right\} \cup\{0\}\right)$
definition eq-elapsed $u u^{\prime} \equiv$ elapsed $u u^{\prime}>0 \longrightarrow\left(\forall c \in \mathcal{X} . u^{\prime} c-u c=\right.$ elapsed $\left.u u^{\prime}\right)$
fun dur :: ('c, t) cval stream $\Rightarrow$ nat $\Rightarrow t$ where

```
    dur \(-0=0 \mid\)
    \(\operatorname{dur}(x \# \# y \# \# x s)(\) Suc \(i)=\) elapsed \(x y+d u r(y \# \# x s) i\)
```

definition divergent $\omega \equiv \forall t . \exists n . d u r \omega n>t$
definition div-cfg cfg $\equiv A E \omega$ in MDP.MC.T cfg. divergent (smap (snd o state) $\omega$ )
definition $\mathcal{R}$-div $\omega \equiv$
$\forall x \in \mathcal{X}$. $(\forall i .(\exists j \geq i$. zero $x(\omega!!j)) \wedge(\exists j \geq i$. $\neg$ zero $x(\omega!$ ! $j)))$
$\vee(\exists i . \forall j \geq i$. unbounded $x(\omega!!j))$
definition $R$-G-div-cfg $c f g \equiv A E \omega$ in MDP.MC.T cfg. $\mathcal{R}$-div (smap (snd o state) $\omega$ )
end
context Probabilistic-Timed-Automaton-Regions
begin
definition $c f g$-on-div st $\equiv$ MDP.cfg-on st $\cap\{c f g$. div-cfg $c f g\}$
definition $R$-G-cfg-on-div st $\equiv R$-G.cfg-on st $\cap\{c f g . R$ - $G$-div-cfg cfg $\}$
lemma measurable- $\mathcal{R}$-div[measurable]: Measurable.pred MDP.MC.S $\mathcal{R}$-div
unfolding $\mathcal{R}$-div-def
by (intro
pred-intros-finite[OF beta-interp.finite]
pred-intros-logic pred-intros-countable
measurable-count-space-const measurable-compose[OF measurable-snth]
) measurable
lemma elapsed-ge $0[$ simp $]$ : elapsed $x y \geq 0$
unfolding elapsed-def using finite(1) by auto
lemma dur-pos:

```
    dur xs i\geq0
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\leqj\Longrightarrowdur xs i\leqdur 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 \ldots\geq 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 44 dest: leI dur-mono[where xs=xs])
lemma elapsed-0D:
    assumes c}\in\mathcal{X}\mathrm{ elapsed }u\mp@subsup{u}{}{\prime}\leq
    shows }\mp@subsup{u}{}{\prime}c-uc\leq
proof -
    from assms have }\mp@subsup{u}{}{\prime}c-uc\in{\mp@subsup{u}{}{\prime}c-uc|c.c\in\mathcal{X}}\cup{0} by aut
    with finite(1) have }\mp@subsup{u}{}{\prime}c-uc\leqMax ({\mp@subsup{u}{}{\prime}c-uc|c.c\in\mathcal{X}}\cup{0}) by aut
    with assms(2) show ?thesis unfolding elapsed-def by auto
qed
lemma elapsed-ge:
    assumes eq-elapsed u u' c\in\mathcal{X}
    shows elapsed u u
    using assms unfolding eq-elapsed-def by (auto intro: elapsed-ge0 order.trans[OF elapsed-OD])
lemma elapsed-eq:
    assumes eq-elapsed u u'}c\in\mathcal{X}\mp@subsup{u}{}{\prime}c-uc\geq
    shows elapsed u \mp@subsup{u}{}{\prime}=\mp@subsup{u}{}{\prime}c-uc
    using elapsed-ge[OF assms(1,2)] assms unfolding eq-elapsed-def by auto
lemma dur-shift:
    dur \omega(i+j) = dur \omegai+dur (sdrop i \omega) j
apply (induction i arbitrary: \omega)
    apply simp
    subgoal for i }
    apply simp
    apply (subst stream.collapse[symmetric])
    apply (rewrite at stl \omega stream.collapse[symmetric])
    apply (subst dur.simps)
    apply (rewrite in dur \omega stream.collapse[symmetric])
    apply (rewrite in dur (- ## \square) (Suc -) stream.collapse[symmetric])
    apply (subst dur.simps)
    apply simp
```

```
done
done
lemma dur-zero:
    assumes
        \forall i. xs !! i\in\omega!! i\forallj\leqi.zero x ( }\omega\mathrm{ !! j) x 仪
        \forall i. eq-elapsed (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 \omega)
    let ? }x=xs!! 
    let ? y = xs !! 1
    let ?ys = stl (stl xs)
    have xs: xs = ?x ## ?y ## ?ys by auto
    from Suc.prems have
        \forall i.(?y ## ?ys) !! i \in stl \omega !! i\forallj\leqi.zero x (stl \omega !! j)
        \forall i. eq-elapsed (stl xs !! i) (stl xs !! Suc i)
        by (metis snth.simps(2)| auto)+
    from Suc.IH[OF this(1,2)<x\in ->] this(3) have [simp]: dur (stl xs) i=0 by auto
    from Suc.prems(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 ... = 0
        apply (subst elapsed-eq[OF - <x\in->])
        unfolding One-nat-def using Suc.prems(4) apply blast
        using * by auto
    finally show ?case .
qed
lemma dur-zero-tail:
    assumes }\foralli.xs!! i\in\omega!! i\forallk\geqi.k\leqj\longrightarrowzero x (\omega!!k)x\in\mathcal{X}j\geq
        \forall. eq-elapsed (xs !! i) (xs !! Suc i)
    shows dur xs j=dur xs i
proof -
    from <j\geq i`dur-shift[of xs i j - i] have
        dur xs j = dur xs i + dur (sdrop ixs) (j - i)
    by simp
    also have ... = dur xs i
        using assms
        by (rewrite in dur (sdrop - ) - dur-zero[where }\omega=\mathrm{ sdrop i w])
            (auto dest: prop-nth-sdrop-pair[of eq-elapsed] prop-nth-sdrop prop-nth-sdrop-pair[of (\epsilon)])
    finally show ?thesis.
qed
lemma elapsed-ge-pos:
    fixes u :: ('c,t) cval
    assumes eq-elapsed }u\mp@subsup{u}{}{\prime}c\in\mathcal{X}u\inV\mp@subsup{u}{}{\prime}\in
    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 }\langleu\inV\rangle\langlec\in\mathcal{X}>\mathrm{ have }uc\geq0\mathrm{ by (auto simp:V-def)
    from False assms have elapsed u u'= u'c - uc
```

```
    unfolding eq-elapsed-def by (auto simp add: less-le)
    also from }\langleuc\geq0\rangle\mathrm{ have ... }\leq\mp@subsup{u}{}{\prime}c\mathrm{ 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: $t \geq 0 \Longrightarrow u^{\prime}=u \oplus t \Longrightarrow$ trans $u u^{\prime} \mid$
reset: set $l \subseteq \mathcal{X} \Longrightarrow u^{\prime}=$ clock-set l $0 u \Longrightarrow$ trans $u u^{\prime} \mid$
$i d: u=u^{\prime} \Longrightarrow$ trans $u u^{\prime}$
abbreviation stream-trans $\equiv$ pairwise trans

```
lemma K-cfg-trans:
    assumes cfg \in MDP.cfg-on (l,R)cfg'}\inK-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=shdxs let ?y = shd (stl xs
    from MDP.pred-stream-cfg-on[OF prems] have *:
        pred-stream ( }\lambdacfg. state cfg \inS\wedgecfg\inMDP.cfg-on (state 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 }\inK-cfg ?x by (auto elim:MDP.MC.enabled.cases
    ultimately have trans R R'
        by (intro K-cfg-trans[where cfg = ?x and cfg' =?y and l=l and l'=l`])metis+
    with〈?x \in valid-cfg` 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}\mp@subsup{u}{}{\prime}u\in
    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 }\mp@subsup{u}{}{\prime}c-uc\leq0\mathrm{ if }c\in\mathcal{X}\mathrm{ for c
    using that }\langleu\inV\rangle\mathrm{ by (cases c e 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 ( }\lambdau.u\inV)x
    shows pairwise eq-elapsed xs
using trans-eq-elapsed assms by (auto intro: pairwise-mp simp: stream.pred-set)
lemma not-reset-dur:
    assumes }\forallk>i.k\leqj\longrightarrow\neg\mathrm{ zero }c([xs !!k\mp@subsup{]}{\mathcal{R}}{})j\geqic\in\mathcal{X}\mathrm{ stream-trans xs
        \forall. eq-elapsed (xs !! i) (xs !! Suc i)}\forall i. xs !! i\inV
    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 *:
        \neg zero c ([xs !! Suc j\mp@subsup{]}{\mathcal{R}}{}) eq-elapsed (xs !! j) (xs !! Suc j) if Suc j>i
        using that by auto
    from Suc.prems(6) have xs !! j G V xs !! Suc j\inV by blast+
    then have regions: [xs !! j\mp@subsup{]}{\mathcal{R}}{}\in\mathcal{R}[xs !! Suc j\mp@subsup{]}{\mathcal{R}}{}\in\mathcal{R}\mathrm{ by auto}
    from trans have (xs !! Suc j) c-(xs !! j) c\geq0 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\in[xs !! Suc j\mp@subsup{]}{\mathcal{R}}{}\mathrm{ by blast}
        ultimately have
            zero c ([xs !! Suc j] _\mathcal{R}
            using zero-all[OF finite(1) - <c\in\mathcal{X}\rangle] regions(2) by (auto simp: \mathcal{R}-def)
            with * that show ?thesis by auto
        qed
    next
        case id then show ?thesis by simp
    qed
    with * 〈c \in\mathcal{X}\rangle 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
            \ldots.=(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 }\forallj\geqi.\neg zero c([xs !! j\mp@subsup{]}{\mathcal{R}}{})j\geqic\in\mathcal{X}\mathrm{ stream-trans xs
        i. eq-elapsed (xs !! i) (xs !! Suc i) \forall j. xs !! j \inV
    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 }\forallj\geqi.\neg\mathrm{ zero c ([xs !! j] 踀) j \ic c X X stream-trans xs
        \forall i. eq-elapsed (xs !! i) (xs !! Suc i) \forall j. xs !! j \in V
        unbounded c ([xs !! i]⿻\mathcal{R}}
    shows unbounded c ([xs !! j\mp@subsup{]}{\mathcal{R}}{})
proof -
    let ?u = xs !! i let ? u' = xs !! j let ?R = [xs !! i\mp@subsup{]}{\mathcal{R}}{}
    from assms have ?u \in?R by auto
    from assms(6) have ?R \in\mathcal{R}}\mathrm{ by auto
    then obtain Ir where ?R = region \mathcal{X I r valid-region \mathcal{X k I r unfolding \mathcal{R}}\mathbf{~}\mathrm{ def by auto}}\mathbf{|}\mathrm{ b}
    with assms(3,7) unbounded-Greater <?u }\in\mathrm{ ? R> have ?u c>k c by force
    also from not-reset-dur'[OF assms(1-6)] dur-mono[OF〈j\geqi\rangle,of xs] have ?u' c\geq? ?u c by auto
    finally have ? u' c>kc by auto
    let ?R' }=[xs!! j\mp@subsup{]}{\mathcal{R}}{
    from assms have ?u' }\in
    from assms(6) have ?R' }\in\mathcal{R}\mathrm{ by auto
```



```
    moreover with }\langle?\mp@subsup{u}{}{\prime}c>-\rangle\langle?\mp@subsup{u}{}{\prime}\in-\rangle\mathrm{ gt-GreaterD }\langlec\in\mathcal{X}\rangle\mathrm{ have I c=Greater (k c) by auto
    ultimately show ?thesis using Greater-unbounded[OF finite(1)-\langlec\in\mathcal{X}\rangle] by auto
qed
```

```
lemma gt-unboundedD:
    assumes \(u \in R\)
        and \(R \in \mathcal{R}\)
        and \(c \in \mathcal{X}\)
        and real \((k c)<u c\)
    shows unbounded c \(R\)
proof -
    from assms obtain \(I r\) where \(R=\) region \(\mathcal{X}\) I r valid-region \(\mathcal{X} k I r\)
        unfolding \(\mathcal{R}\)-def by auto
    with Greater-unbounded[of \(\mathcal{X} k \operatorname{lrc}]\) gt-GreaterD[of \(u \mathcal{X} \operatorname{Irck} \mathrm{c}\) assms finite(1) show ?thesis
        by auto
qed
definition trans \({ }^{\prime}::\left({ }^{\prime} c, t\right)\) cval \(\Rightarrow\left({ }^{\prime} c, t\right)\) cval \(\Rightarrow\) bool where
    trans \({ }^{\prime}\) u \(u^{\prime} \equiv\)
        \(\left(\left(\forall c \in \mathcal{X} . u c>k c \wedge u^{\prime} c>k c \wedge u \neq u^{\prime}\right) \longrightarrow u^{\prime}=u \oplus 0.5\right) \wedge\)
        \(\left(\left(\exists c \in \mathcal{X} . u c=0 \wedge u^{\prime} c>0 \wedge\left(\forall c \in \mathcal{X} . \nexists d . d \leq k c \wedge u^{\prime} c=\right.\right.\right.\) real \(\left.\left.d\right)\right)\)
        \(\longrightarrow u^{\prime}=\) delayed \(\left.R\left([u]_{\mathcal{R}}\right) u\right)\)
lemma zeroI:
    assumes \(c \in \mathcal{X} u \in V u c=0\)
    shows zero \(c\left([u]_{\mathcal{R}}\right)\)
proof -
    from assms have \(u \in[u]_{\mathcal{R}}[u]_{\mathcal{R}} \in \mathcal{R}\) by auto
    then obtain \(I r\) where \([u]_{\mathcal{R}}=\) region \(\mathcal{X}\) Ir valid-region \(\mathcal{X} k I r\) unfolding \(\mathcal{R}\)-def by auto
    with zero-all[OF finite(1) this(2) \(\langle c \in \mathcal{X}\rangle]\left\langle u \in[u]_{\mathcal{R}}\right\rangle\langle u c=0\rangle\) show ?thesis by auto
qed
lemma zero \(D\) :
    \(u x=0\) if zero \(x\left([u]_{\mathcal{R}}\right) u \in V\)
    using that by (metis regions-part-ex(1) zero-def)
lemma not-zeroD:
    assumes \(\neg\) zero \(x\left([u]_{\mathcal{R}}\right) u \in V x \in \mathcal{X}\)
    shows \(u x>0\)
proof -
    from zeroI assms have \(u x \neq 0\) by auto
    moreover from assms have \(u x \geq 0\) unfolding \(V\)-def by auto
    ultimately show ?thesis by auto
qed
lemma not-const-intv:
    assumes \(u \in V \forall c \in \mathcal{X}\). \(\ddagger d . d \leq k c \wedge u c=\) real \(d\)
    shows \(\forall c \in \mathcal{X} . \forall u \in[u]_{\mathcal{R}} . \nexists d . \bar{d} \leq k c \wedge u c=\) real \(d\)
proof -
    from assms have \(u \in[u]_{\mathcal{R}}[u]_{\mathcal{R}} \in \mathcal{R}\) by auto
    then obtain \(\operatorname{Ir}\) where \(I:[u]_{\mathcal{R}}=\) region \(\mathcal{X}\) Ir valid-region \(\mathcal{X} k I r\) unfolding \(\mathcal{R}\)-def by auto
    have \(\nexists d . d \leq k c \wedge u^{\prime} c=\) real \(d\) if \(c \in \mathcal{X} u^{\prime} \in[u]_{\mathcal{R}}\) for \(c u^{\prime}\)
    proof safe
        fix \(d\) assume \(A: d \leq k c u^{\prime} c=\) real \(d\)
        from I that have intv-elem \(c u^{\prime}\left(\begin{array}{l}\text { c }\end{array}\right)\) valid-intv \(\left(\begin{array}{ll}k & c)\end{array}\binom{I}{c}\right.\) by auto
        then show False
            using \(A I\left\langle u \in[u]_{\mathcal{R}}\right\rangle\langle c \in \mathcal{X}\rangle\) assms(2) by (cases; fastforce)
    qed
    then show?thesis by auto
qed
```

lemma $K$-cfg-trans':
assumes repcs $(l, u) c f g \in M D P . c f g-o n(l, u) c f g^{\prime} \in K-c f g(r e p c s(l, u) c f g)$
state $c f g^{\prime}=\left(l^{\prime}, u^{\prime}\right)(l, u) \in S$ cfg $\in R$-G.valid-cfg abss $(l, u)=$ state $c f g$
shows trans ${ }^{\prime} u u^{\prime}$
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: (1lut)
from assms $\langle-=(l, u)\rangle$ have repcs $(l, u) c f g \in$ valid-cfg by (auto intro: MDP.valid-cfgI)
then have absc (repcs $(l, u) c f g) \in R$-G.valid-cfg by auto
from prems have $*$ : rept $(l, u)($ action $c f g)=$ return-pmf $(l, u \oplus t)$ unfolding repcs-def by auto
from $\langle a b s s-=-\rangle\langle-=(l, u)\rangle\langle c f g \in R-G . v a l i d-c f g\rangle$ have
action $c f g \in \mathcal{K}($ abss $(l, u))$
by (auto dest: $R-G-I$ )
from abst-rept-id[OF this] * have action cfg $=$ abst (return-pmf $(l, u \oplus t))$ by auto
with prems have $* *$ : action cfg $=$ return-pmf $\left(l,[u \oplus t]_{\mathcal{R}}\right)$ unfolding abst-def by auto
show ?thesis
proof (cases $\forall c \in \mathcal{X} . u c>k c$ )
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 action cfg $=$ return-pmf $\left(l,[u]_{\mathcal{R}}\right)$ by simp
with True have rept $(l, u)($ action $c f g)=$ 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 $u^{\prime}=u \oplus t$ by auto
moreover from prems True have $\forall c \in \mathcal{X} . u^{\prime} c>k c$ 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 $\exists c \in \mathcal{X} . u c=0 \wedge 0<u^{\prime} c \wedge\left(\forall c \in \mathcal{X}\right.$. $\exists d . d \leq k c \wedge u^{\prime} c=$ real $\left.d\right)$ )
case True
from prems have $u^{\prime} \in[u]_{\mathcal{R}}$ by auto
from prems have $[u \oplus t]_{\mathcal{R}} \in S u c c \mathcal{R}\left([u]_{\mathcal{R}}\right)$ by auto
from True obtain $c$ where $c \in \mathcal{X} u c=0 u^{\prime} c>0$ by auto
with zeroI prems have zero $c\left([u]_{\mathcal{R}}\right)$ by auto
moreover from $\left\langle u^{\prime} \in-\right\rangle\left\langle u^{\prime} c>0\right\rangle$ have $\neg$ zero $c\left([u]_{\mathcal{R}}\right)$ 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
$\forall u \in[u \oplus t]_{\mathcal{R}} . \forall c \in \mathcal{X}$. $\exists d . d \leq k c \wedge u c=$ real $d$
by auto
ultimately have $\exists R^{\prime} .(l, u) \in S \wedge$
action cfg $=$ return-pmf $\left(l, R^{\prime}\right) \wedge$
$R^{\prime} \in \operatorname{Succ} \mathcal{R}\left([u]_{\mathcal{R}}\right) \wedge[u]_{\mathcal{R}} \neq R^{\prime} \wedge\left(\forall u \in R^{\prime} . \forall c \in \mathcal{X} . \nexists d . d \leq k c \wedge u c=\right.$ real $\left.d\right)$
apply -
apply (rule exI $\left[\right.$ where $\left.\left.x=[u \oplus t]_{\mathcal{R}}\right]\right)$
apply safe
using prems ** by auto
then have
rept ( $l, u$ ) (action cfg)
$=$ return-pmf $\left(l\right.$, delayed $R\left(S O M E R^{\prime}\right.$. action cfg $=$ return-pmf $\left.\left.\left(l, R^{\prime}\right)\right) u\right)$
unfolding rept-def by auto
with $* * *$ prems have $u^{\prime}=\operatorname{delayed} R\left([u \oplus t]_{\mathcal{R}}\right) u$ by auto
with $F$ True prems show ?thesis unfolding trans'-def by auto
next
case False
with $F<-=(l, u)\rangle$ show ?thesis unfolding trans'-def by auto

```
        qed
    qed
next
    case prems:(2--\tau \mu)
    then obtain X where X: u' = ([X:=0]u) (X, l')\in set-pmf }\mu\mathrm{ by auto
    from }\langle-\inS\rangle\mathrm{ have }u\inV\mathrm{ by auto
    let ?r = SOME r. set r = X
    show ?case
    proof (cases X={})
        case True
        with }X\mathrm{ have }u=\mp@subsup{u}{}{\prime}\mathrm{ by auto
        with non-empty show ?thesis unfolding trans'-def by auto
    next
        case False
        then obtain x where }x\inX\mathrm{ by auto
        moreover have X\subseteq\mathcal{X}\mathrm{ using admissible-targets-clocks(1)[OF prems(10)X(2)] by auto}
        ultimately have }x\in\mathcal{X}\mathrm{ by auto
        from}\langleX\subseteq\mathcal{X}\rangle\mathrm{ finite(1) obtain r where set r = X using finite-list finite-subset by blast
        then have r: set ?r = X by (rule someI)
        with }\langlex\inX\rangleX\mathrm{ have }\mp@subsup{u}{}{\prime}x=0\mathrm{ by auto
        from X r <u\inV V\langleX\subseteq\mathcal{X}> have }\mp@subsup{u}{}{\prime}x\lequx\mathrm{ for }
            by (cases }x\inX\mathrm{ ; auto simp: V-def)
        have False if }\mp@subsup{u}{}{\prime}x>0\wedgeux=0\mathrm{ for }
            using < 'u
```



```
    qed
next
    case 3
    with non-empty show ?case unfolding trans'-def by auto
qed
coinductive enabled-repcs where
    enabled-repcs (shd xs) (stl xs) \Longrightarrow shd xs = repcs st' cfg' \Longrightarrow st' }\in\mathrm{ rept st (action cfg)
    "abss st' = state cfg'
    \Longrightarrow c f g ' \in R - G . v a l i d - c f g ~
     enabled-repcs (repcs st cfg) xs
lemma K-cfg-rept-in:
assumes cfg \inR-G.valid-cfg
    and abss st = state cfg
    and cfg'\inK-cfg cfg
    shows (THE s'. s'\in set-pmf (rept st (action cfg)) ^abss s'= state cfg')
            set-pmf (rept st (action cfg))
proof -
    from assms(1,2) have action cfg \in\mathcal{K}(abss st) by (auto simp: R-G-I)
    from }\langlecf\mp@subsup{g}{}{\prime}\in >> hav
        cfg' = cont cfg (state cfg') state cfg' }\in\mathrm{ action cfg
    by (auto simp: set-K-cfg)
    with abst-rept-id[OF <action - \epsilon ->] pmf.set-map have
        state cfg'\in abss ' set-pmf (rept st (action cfg)) unfolding abst-def by metis
    then obtain st' where
        st'}\in\mathrm{ rept st (action cfg) abss st' = 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 cfg \inR-G.valid-cfg abss st = state cfg MDP.MC.enabled (repcs st cfg) xs
    shows enabled-repcs (repcs st cfg) xs using assms
proof (coinduction arbitrary: cfg xs st)
```

case prems：（enabled－repcs cfg xs st）
let $? x=\operatorname{shd} x s$ and $? y=\operatorname{shd}(s t l x s)$
let ？st $=T H E s^{\prime} . s^{\prime} \in \operatorname{set}-\mathrm{pmf}($ rept st $($ action cfg $)) \wedge$ abss $s^{\prime}=$ state $($ absc ？$x)$
from $\operatorname{prems}(3)$ have $? x \in K$－cfg（repcs st cfg）by cases
with $K$－cfg－map－repcs $[O F \operatorname{prems}(1,2)]$ obtain $c f g^{\prime}$ where
$c f g^{\prime} \in K-c f g c f g ? x=$ repcs $\left(T H E s^{\prime} . s^{\prime} \in\right.$ rept st $($ action $c f g) \wedge$ abss $s^{\prime}=$ state $\left.c f g^{\prime}\right) c f g^{\prime}$
by auto
let ？st $=$ THE $s^{\prime} . s^{\prime} \in$ rept st $($ action cfg $) \wedge$ abss $s^{\prime}=$ state $c f g^{\prime}$
from $K$－cfg－rept－action $\left[O F \operatorname{prems}(1, \mathcal{Q})\left\langle c f g^{\prime} \in-\right\rangle\right]$ have abss ？st $=$ state $c f g^{\prime}$ ．
moreover from $K$－cfg－rept－in $\left[O F \operatorname{prems}(1,2)\left\langle c f g^{\prime} \in-\right\rangle\right]$ have ？st $\in$ rept st（action cfg）．
moreover have $c f g^{\prime} \in R$－G．valid－cfg using $\left\langle c f g^{\prime} \in K\right.$－cfg $\left.\operatorname{cfg}\right\rangle$ prems（1）by blast
moreover from absc－repcs－id［OF this «abss ？st $=$ state $\left.\left.\left.c f g^{\prime}\right\rangle\right] \prec ? x=-\right\rangle$ have $a b s c ? x=c f g^{\prime}$
by auto
moreover from $\operatorname{prems(3)}$ have MDP．MC．enabled（shd xs）（stl xs）by cases
ultimately show ？case
using 〈？$x=->$ by（inst－existentials $x s$ ？st absc ？$x$ st cfg）fastforce＋
qed
lemma repcs－eq－rept：
rept st（action cfg）$=$ rept st $t^{\prime \prime}\left(\right.$ action cfg＇） $\mathbf{i f}$ repcs st cfg $=$ repcs st $t^{\prime \prime} c f g^{\prime \prime}$
by（metis（mono－tags，lifting）action－cfg－corec old．prod．case repcs－def that）
lemma enabled－stream－trans＇：
assumes $c f g \in R$－G．valid－cfg abss st $=$ state $c f g$ 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 ？$x s=s t l x s$
from prems have $A$ ：enabled－repcs（repcs st cfg）xs by（auto intro：enabled－repcsI）
then obtain $s t^{\prime} c f g^{\prime}$ where enabled－repcs（shd xs）（stl xs）shd xs $=$ repcs $s t^{\prime} c f^{\prime}{ }^{\prime}$ st＇$\in$ rept st（action cfg）
abss st ${ }^{\prime}=$ state $c f g^{\prime} c f g^{\prime} \in R$－G．valid－cfg
apply atomize－elim
apply（cases rule：enabled－repcs．cases）
apply assumption
subgoal for $s t^{\prime} c f g^{\prime} s t^{\prime \prime} c f g^{\prime \prime}$
by（inst－existentials st ${ }^{\prime}$ cfg＇）（auto dest：repcs－eq－rept）
done
then obtain $s t^{\prime \prime} c f g^{\prime \prime}$ where
enabled－repcs（shd ？$x s$ ）（stl ？$x s$ ）
shd ？$x s=$ repcs $s t^{\prime \prime} c f g^{\prime \prime} s t^{\prime \prime} \in$ rept st＇$\left(\right.$ action cfg＇）abss st ${ }^{\prime \prime}=$ state cfg＂
by atomize－elim（subst（asm）enabled－repcs．simps，fastforce dest：repcs－eq－rept）
let $? x=s h d x s$ let $? y=s h d(s t l x s)$
let ？cfg $=$ repcs st $c f g$
from prems have ？cfg $\in$ valid－cfg by auto
from MDP．pred－stream－cfg－on $[O F\langle ? c f g \in$ valid－cfg〉prems（3）］have $*$ ： pred－stream（ $\lambda c f g$ ．state $c f g \in S \wedge c f g \in M D P . c f g-o n($ state $c f g))$ xs．
obtain $l u l^{\prime} u^{\prime}$ where eq：st $=(l, u) s t^{\prime \prime}=\left(l^{\prime}, u^{\prime}\right)$
by force
moreover from $*$ have
$? x \in M D P . c f g$－on（state $? x$ ）$? x \in$ valid－cfg
by（auto intro：MDP．valid－cfgI simp：stream．pred－set）
moreover from prems（3）have ？y $\in K-c f g ? x$ by（auto elim：MDP．MC．enabled．cases）
ultimately have trans ${ }^{\prime} u u^{\prime}$
using 〈？$x=-\rangle\langle ? y=-\rangle\left\langle c f g^{\prime} \in-\right\rangle\left\langle a b s s t^{\prime}=-\right\rangle$
by（intro K－cfg－trans＇）（auto dest：MDP．valid－cfg－state－in－S）
with $\langle ? x \in$ valid－cfg〉〈cfg＇$\in R$－G．valid－cfg〉prems（3）«abss－＝state cfg $\rangle$ show ？case
apply（inst－existentials u $u^{\prime}$ smap（snd o state）（stl ？$x s$ ））
apply（simp add：eq〈？$x=-\rangle\langle ? y=-\rangle ;$ fail $)+$
by（（intro disjI1 exI）？；auto simp：＜？$x=-\rangle\langle ? y=-\rangle$ 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 \(\left(\lambda u .[u]_{\mathcal{R}}\right)\) xs ) (is \(\mathcal{R}\)-div ? \(\omega\) )
unfolding \(\mathcal{R}\)-div-def proof (safe, simp-all)
    fix \(x i\)
    assume \(x: x \in \mathcal{X}\) and bounded: \(\forall i . \exists j \geq i\). \(\neg\) unbounded \(x\left([x s!!j]_{\mathcal{R}}\right)\)
    from \(i n-S\) have \(x s-\omega\) : \(\forall i . x s!!i \in ? \omega!!i\) by (auto simp: stream.pred-set)
    from trans in-S have elapsed:
        \(\forall\) i. eq-elapsed (xs !! i) (xs !! Suc i)
        by (fastforce intro: pairwise-trans-eq-elapsed pairwise-Suc[where \(P=\) eq-elapsed \(]\) )
    \{ assume \(A: \forall j \geq i\). \(\neg\) zero \(x\left([x s!!j]_{\mathcal{R}}\right)\)
        let \(? t=d u r x s i+k x\)
        from div obtain \(j\) where \(j\) : dur xs \(j>d u r x s i+k x\) unfolding divergent-def by auto
        then have \(k x<d u r\) xs \(j-d u r x s i\) by auto
        also with not-reset-dur' \([\) OF A less-imp-le \([\) OF dur-monoD], of \(x s]\langle x \in \mathcal{X}\rangle\) assms elapsed have
            \(\ldots=(x s!!j) x-(x s!!i) x\)
            by (auto simp: stream.pred-set)
        also have \(\ldots \leq(x s!!j) x\)
            using assms(1) \(\langle x \in \mathcal{X}\rangle\) unfolding \(V\)-def by (auto simp: stream.pred-set)
            finally have unbounded \(x\left([x s!!j]_{\mathcal{R}}\right)\)
            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 \(\forall j^{\prime} \geq j\). \(\neg\) zero \(x\left([x s!!j]_{\mathcal{R}}\right)\) by auto
            ultimately have \(\forall i \geq j\). unbounded \(x\left([x s!!i]_{\mathcal{R}}\right)\)
            using elapsed assms \(x\) by (auto intro: not-reset-unbounded simp: stream.pred-set)
            with bounded have False by auto
    \}
    then show \(\exists j \geq i\). zero \(x\left([x s!!j]_{\mathcal{R}}\right)\) by auto
    \{ assume \(A: \forall j \geq i\). zero \(x\left([x s!!j]_{\mathcal{R}}\right)\)
        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 \(\forall j \geq i\). zero \(x(? \omega!\) ! \(j\) ) by auto
        with dur-zero-tail[OF xs- \(\omega-x\langle i \leq j\rangle\) elapsed \(] j\) have False by simp
    \}
    then show \(\exists j \geq i\). \(\neg\) zero \(x\left([x s!!j]_{\mathcal{R}}\right)\) by auto
qed
lemma (in -)
    fixes \(f::\) nat \(\Rightarrow\) real
    assumes \(\forall i . f i \geq 0 \forall i . \exists j \geq i . f j>d d>0\)
    shows \(\exists n\). \(\left(\sum i \leq n . f i\right)>t\)
    oops
lemma dur-ev-exceedsI:
    assumes \(\forall i\). \(\exists j \geq i\). dur xs \(j-d u r x s i \geq d\) and \(d>0\)
    obtains \(i\) where dur xs \(i>t\)
proof -
    have base: \(\exists i\). dur xs \(i>t\) if \(t<d\) for \(t\)
    proof -
        from assms obtain \(j\) where dur xs \(j-d u r x s 0 \geq d\) 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. 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 \langled> 0\rangle obtain i where dur xs i>0 by auto
    moreover from assms obtain j where dur xs j - dur xs i\geqd by auto
    ultimately have dur xs j>d by auto
    with }\langlet=d\rangle\mathrm{ show ?thesis by auto
qed
show ?thesis
proof (cases t\geq0)
    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 \lceilt / d\rceil
    from True have }\exists\mathrm{ i. dur xs i> ?m*d
    proof (induction ?m arbitrary: t)
        case 0
        with base[OF<0<d\rangle] show ?case by simp
    next
        case (Suc n t)
        let ?t = t-d
        show ?case
        proof (cases t \geqd)
            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\rceil= [t/d\rceil-1 by simp
            with <Suc n = -> have n= nat \lceil?t / d\rceil by simp
            with Suc <t \geqd\rangle obtain i where nat \lceil?t / d\rceil*d<dur xs i by fastforce
            from assms obtain }j\mathrm{ where dur xs j - dur xs i }\geqdj\geqi by aut
            with <dur xs i> -> have nat \lceil?t / d\rceil*d +d< <ur xs j by simp
            with True have dur xs j> nat \lceilt / d\rceil*d
            by (metis Suc.hyps(2) <n= nat \lceil(t-d) / d\rceil` add.commute distrib-left mult.commute
                    mult.right-neutral of-nat-Suc)
            then show ?thesis by blast
        next
            case False
            with }\langlet\geq0\rangle\langled>0\rangle\mathrm{ have nat }\lceilt/d\rceil\leq1 by sim
            then have nat \lceilt / d\rceil*d\leqd
            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-refl 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\geq0\rangle\langled> 0\rangle have ?m*d\geqt
        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 \geq shd xs c2 stream-all ( }\lambda\mathrm{ u.u }\inV=V) xs c\mathcal{Z}\in\mathcal{X
    shows (holds ( }\lambdau.uc1\gequc2) until holds (\lambdau.u c1 = 0)) xs using assm
proof (coinduction arbitrary: xs)
    case prems: (UNTIL xs)
    let ?xs = stl xs
    let ? }x=shdx
    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 \geq?y cz
            proof cases
                case A: (reset t)
                show ?thesis
                proof (cases c1 \in set t)
                case True
                    with A False show ?thesis by auto
                next
                    case False
                    from prems have ?x c2 \geq0 by (auto simp:V-def)
                    with A have ?y c\mathcal{L}\leq?x c2 by (cases c\mathcal{L }\in\mathrm{ set t) auto}
                    with A False〈?x c1 \geq?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 (\lambdau.u\inV) ?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 \mathcal{R}
    fixes xs :: ('c,t) cval stream
    assumes stream-trans xs stream-all ( }\lambdau.u\inV)x
                (xs !! i) c1 = 0 \exists k>i.k\leqj^(xs !! k) c2 = 0
                \forall > i. k\leqj\longrightarrow(xs !! k) c1 \not=0
                c1\in\mathcal{X}c\mathcal{L}\in\mathcal{X}
    shows (xs !! j) c1 \geq (xs !! j) c2
proof -
    from assms obtain k where k: k>ik\leqj(xs !! k)c2 = 0 by auto
    with assms(5)<k\leqj\rangle have (xs !! k) c1 \not=0 by auto
    moreover from assms(2) <c1 \in\mathcal{X}\rangle have (xs !! k) c1 \geq0 by (auto simp:V-def)
    ultimately have (xs !! k) c1 > 0 by auto
    with «(xs !! k) c\mathcal{L = 0 > have shd (sdrop k xs) c1 \geq shd (sdrop k xs) c\mathcal{L}}\mathrm{ by auto}
    from not-reset-mono[OF - this] assms have
        (holds (\lambdau.u c2 \lequ c1) until holds ( }\lambdau.u c1 = 0)) (sdrop k xs)
    by (auto intro: sset-sdrop pairwise-sdropD)
    from assms(5) k(2)<k> i\rangle have }\forallm\leqj-k.(sdrop kxs !! m) c1 =0 by simp
    with holds-untilD[OF «(- until -) ->, of j - k] have
        (sdrop k xs !! (j - k)) c2 \leq (sdrop k xs !! (j - k)) c1 .
    then show (xs !! j) c\mathcal{L}\leq(xs !! j) c1 using k(1,2) by simp
qed
lemma unbounded-all:
```



```
    shows ux>kx
proof -
    from assms obtain I r where R: R = region \mathcal{X I r valid-region \mathcal{X k I r unfolding \mathcal{R}}\mathbf{\mathcal{R}}\mathrm{ def by auto}}\mathbf{~}\mathrm{ b}
    with unbounded-Greater }\langlex\in\mathcal{X}\rangle\operatorname{assms(3) have I }x=Greater (kx) by sim
    with }\langleu\inR\rangleR<x\in\mathcal{X}\rangle\mathrm{ show ?thesis by force
qed
lemma trans-not-delay-mono:
    u}
    using <trans u u'>
proof (cases)
    case (reset l)
    with that show ?thesis by (cases c \in set l) (auto simp:V-def)
qed (use that in <auto simp: cval-add-def V-def add-nonneg-eq-0-iff>)
lemma dur-reset:
    assumes pairwise eq-elapsed xs pred-stream (\lambdau.u\inV) xs zero x ([xs !! Suc i\mp@subsup{]}{\mathcal{R}}{})x\in\mathcal{X}
    shows dur xs (Suc i) - dur xs i=0
proof -
    from assms(2) have in-V: xs !! Suc i }\in
        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
        elapsed (xs !! i) (xs !! Suc i) \leq (xs !! Suc i) x
        unfolding stream.pred-set by auto
    with in-V assms(3) have elapsed (xs !! i) (xs !! Suc i) \leq 0 by (auto simp:zeroD)
    with elapsed-ge0[of xs !! i xs !! Suc i] have elapsed (xs !! i) (xs !! Suc i)=0
        by linarith
    then show ?thesis by (subst dur-Suc)
qed
lemma resets-mono-0':
    assumes pairwise eq-elapsed xs stream-all (\lambdau.u\inV) xs stream-trans xs
                \forallj\leqi. 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 (Suc i)
    from Suc.prems have *:(xs !! Suc i) x=0 (xs !! i) x=0
        by (blast intro: zeroD snth-sset, force intro: zeroD snth-sset)
    from pairwise-Suc[OF Suc.prems(3)] have trans (xs !! i) (xs !! 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 !! 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 !! Suc i) c=0 \vee (xs !! 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\mathrm{ u. u GV) xs stream-trans xs
                \forallk\geqi.k\leqj\longrightarrowzero x ([xs !! k\mp@subsup{]}{\mathcal{R}}{})x\in\mathcal{X}c\in\mathcal{X}i\leqj
```

```
    shows (xs !! j) c=(xs !! i) c\vee(xs !! j) c=0 using assms
proof -
    from assms have 1: stream-all ( }\lambdau.u\inV) (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:
        \forallk\leqj - 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)]<i\leqj\rangle show ?thesis by simp
qed
lemma resets-mono:
    assumes pairwise eq-elapsed xs pred-stream ( }\lambda\mathrm{ u. u G V) xs stream-trans xs
        \forallk\geqi.k\leqj\longrightarrowzero x ([xs !! k\mp@subsup{]}{\mathcal{R}}{})x\in\mathcal{X}c\in\mathcal{X}i\leqj
    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 }\mathcal{R}\mathrm{ -divergent-divergent-aux2:
    fixes M :: (nat }=>\mathrm{ bool) set
    assumes }\foralli.\forallP\inM.\existsj\geqi.PjM\not={} finite 
    shows }\foralli.\existsj\geqi.\existsk>j.\existsP\inM.Pj\wedgePk\wedge(\forallm<k.j<m\longrightarrow\negPm
        \wedge(\forallQ\inM.\existsm\leqk.j<m^Qm)
proof
    fix i
    let ?j1 = Max {LEAST m. m>i^Pm|P.P\inM}
    from <M\not={}` obtain P where P\inM by auto
    let ?m = LEAST m. m> i^Pm
    from assms(1)<P\inM\rangle obtain j where j\geqSuc i P j by auto
    then have j>i Pj by auto
    with }\langleP\inM\rangle\mathrm{ have ? m > i}^PP?m by - (rule LeastI; auto
    moreover with < finite M\rangle\langleP\inM\rangle have ?j1 \geq?m by - (rule Max-ge; auto)
    ultimately have ? j1 }\geqi\mathrm{ by simp
    moreover have }\existsm>i.m\leq? j1^Pm\mathrm{ if }P\inM\mathrm{ for P
    proof -
        let ?m=LEAST m. m> i^Pm
        from assms(1)\langleP\inM\rangle obtain j where j\geqSuc i P j by auto
        then have j>iPj by auto
        with }\langleP\inM\rangle\mathrm{ have ?m > i^P ?m by - (rule LeastI; auto)
        moreover with <finite M\rangle\langleP\inM\rangle have ?j1 \geq?m by - (rule Max-ge; auto)
        ultimately show ?thesis by auto
    qed
    ultimately obtain j1 where j1:j1\geqi\forallP\inM.\existsm>i.j1\geqm\wedgePm}\mathrm{ by auto
    define k where k Q = (LEAST k. k>j1^Qk) for Q
    let ?k = Max {kQ|Q.Q\inM}
    let ?P =SOME P. P\inM\wedgekP=?k
    let ?j }=\operatorname{Max}{j.i\leqj\wedgej\leqj1\wedge?P j
    have ? }k\in{kQ|Q.Q\inM} using assms by - (rule Max-in; auto
    then obtain P where P:kP=?k P\inM by auto
    have ?k \geqk Q if Q GM for Q using assms that by - (rule Max-ge; auto)
    have *:?P }\inM\wedgek?P=?k\mathrm{ using P by - (rule someI[where x = P]; auto)
    with j1 have \existsm>i.j1\geqm^?P m by auto
    with〈finite -> have ? j }\in{j.i\leqj^j\leqj1\wedge?P j} by - (rule Max-in; auto
    have k:kQ>j1^Q(kQ) if Q}\inM\mathrm{ for }
    proof -
    from assms(1)<Q\inM\rangle obtain m}\mathrm{ where m}\geq\mathrm{ Suc j1 Q m by auto
    then have m>j1Q m by auto
    then show }kQ>j1\wedgeQ(kQ)\mathrm{ unfolding k-def by - (rule LeastI; blast)
    qed
    with * <?j \in -> have ?P ? k ?j < ?k by fastforce+
    have \neg?P m if ?j<mm<? f for m
```

```
    proof (rule ccontr, simp)
    assume ?P m
    have m>j1
    proof (rule ccontr)
        assume }\negj1<
        with <?j < m><?j }\in-> have i\leqmm\leqj1 by aut
        with <?P m>< finite -> have ?j \geqm by - (rule Max-ge; auto)
        with <?j < m〉 show False by simp
    qed
    with〈?P m><finite -> have k?P \leqm unfolding k-def by (auto intro: Least-le)
    with * <m<? ?k\rangle show False by auto
qed
moreover have \existsm\leq? ?. ?j < m ^Q m if Q\inM for Q
proof -
    from k[OF<Q < M>] have k Q>j1^Q (kQ).
    moreover with <finite ->\langleQ\inM\rangle have k Q\leq? ?k by - (rule Max-ge; auto)
    moreover with <?j }\in->\langlekQ>-\wedge >> have ?j < k Q by aut
    ultimately show ?thesis by auto
qed
ultimately show
    \exists \geqi.\existsk>j.\exists P\inM.Pj^Pk^(\forallm<k.j<m\longrightarrow\negPm)
        \wedge(\forallQ\inM.\existsm\leqk.j<m^Qm)
    using\langle?j < ?k\rangle\langle?j \in >\langle?P ?k\rangle* by (inst-existentials ?j ?k ?P; blast)
qed
lemma \mathcal{R}}\mathrm{ -divergent-divergent:
    assumes in-S: pred-stream ( }\lambdau.u\inV)x
        and div: \mathcal{R-div (smap ( }\lambdau.[u\mp@subsup{]}{\mathcal{R}}{})xs)
        and trans: stream-trans xs
        and trans': pairwise trans' xs
    and unbounded-not-const:
    \forallu.(\forallc\in\mathcal{X}.real (kc)<uc)\longrightarrow\negev(alw (\lambdaxs. shd xs=u)) xs
shows divergent xs
unfolding divergent-def proof
fix }
from pairwise-trans-eq-elapsed[OF trans in-S] have eq-elapsed: pairwise eq-elapsed xs .
define X1 where X1 = {x. x \in\mathcal{X}\wedge(\existsi.\forallj\geqi. unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{}))}
let ?i = Max {(SOME i. }\forallj\geqi.unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{}))|x.x\in\mathcal{X}
from finite(1) non-empty have
```



```
    by (intro Max-in) auto
    have unbounded x ([xs !! j] \mathcal{R}})\mathrm{ if }x\inX1j\geq?i for x 
    proof -
        have X1\subseteq\mathcal{X}\mathrm{ unfolding X1-def by auto}
        with finite(1) non-empty <x \in X1` have *:
        ?i}\geq(SOME i.\forallj\geqi. unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{}))(\mathrm{ is ? i }\geq? ?k
        by (intro Max-ge) auto
    from <x \inX1> have }\existsk.\forallj\geqk. unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{})\mathrm{ by (auto simp: X1-def)
    then have }\forallj\geq?}. unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{})\mathrm{ by (rule someI-ex)
    moreover from <j\geq? ?i><? i\geq -> have j\geq? }k\mathrm{ by auto
    ultimately show ?thesis by blast
    qed
    then obtain }i\mathrm{ where unbounded: }\forallx\inX1.\forallj\geqi.unbounded x ([xs !! j\mp@subsup{]}{\mathcal{R}}{}
        using finite by auto
    show }\existsn.t<dur xs 
    proof (cases }\forallx\in\mathcal{X}.(\existsi.\forallj\geqi.unbounded x ([xs !! j] 踀))
    case True
    then have X1 = \mathcal{X unfolding X1-def by auto}
    have }\existsk\geqj.0.5\leqdur xs k-dur xs j for j
    proof -
        let ?u = xs !! 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 $\langle X 1=\mathcal{X}\rangle$ have
$\forall x \in \mathcal{X}$. unbounded $x\left([? u]_{\mathcal{R}}\right)$
by force
ultimately have $\forall x \in \mathcal{X}$. ? $u x>k x$
by (auto intro: unbounded-all)
with unbounded-not-const have $\neg e v(a l w(H L D\{? u\})) x s$
unfolding $H L D$-iff by simp
then obtain $r$ where
$r \geq \max i j x s!!r \neq x s$ !! 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 ( $\max i j$ ) xs] by (auto 43 elim: ev-sdropD)
let $? k=$ Suc $r$
from in-S have $x s!!? k \in V$ using snth-sset unfolding stream.pred-set by blast
with $i n-S$ have $*$ :
$x s!!r \in[x s!!r]_{\mathcal{R}}[x s!!r]_{\mathcal{R}} \in \mathcal{R}$
xs !! ? $k \in[x s!!? k]_{\mathcal{R}}[x s!!? k]_{\mathcal{R}} \in \mathcal{R}$
by (auto simp: stream.pred-set)
from $\langle r \geq$-〉 have $r \geq i ? k \geq i$ by auto
with unbounded $\langle X 1=\mathcal{X}\rangle$ have $\forall x \in \mathcal{X}$. unbounded $x\left([x s!!r]_{\mathcal{R}}\right) \forall x \in \mathcal{X}$. unbounded $x\left([x s!!? k]_{\mathcal{R}}\right)$ by (auto simp del: snth.simps(2))
with $i n-S$ have $\forall x \in \mathcal{X} .(x s!!r) x>k x \forall x \in \mathcal{X} .(x s!!? k) x>k x$ using $*$ by (auto intro: unbounded-all)
moreover from trans ${ }^{\prime}$ have trans ${ }^{\prime}(x s!!r)(x s!!? k)$
using pairwise-Suc by auto
ultimately have $(x s!!? k)=(x s!!r) \oplus 0.5$
unfolding trans ${ }^{\prime}$-def using «xs !! $r \neq->$ by auto
moreover from pairwise-Suc $[$ OF eq-elapsed $]$ have eq-elapsed (xs !! r) (xs !! ?k)
by auto
ultimately have
dur xs ? $k-$ dur xs $r=0.5$
using non-empty by (auto simp: cval-add-def dur-Suc elapsed-eq)
with dur-mono[of $j r x s]\langle r \geq \max i j\rangle$ have dur $x s ? k-d u r x s j \geq 0.5$
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 $X 2$ where $X 2=\mathcal{X}-X 1$
from False have $X 2 \neq\{ \}$ unfolding $X 1$-def X2-def by fastforce
have inf-resets:
$\forall i$. $\left(\exists j \geq i\right.$. zero $\left.x\left([x s!!j]_{\mathcal{R}}\right)\right) \wedge\left(\exists j \geq i\right.$. $\neg$ zero $\left.x\left([x s!!j]_{\mathcal{R}}\right)\right)$ if $x \in X 2$ for $x$
using that div unfolding X1-def X2-def $\mathcal{R}$-div-def by fastforce
have $\exists j \geq i$. $\exists k>j$. $\exists x \in$ X2. zero $x\left([x s!!j]_{\mathcal{R}}\right) \wedge$ zero $x\left([x s!!k]_{\mathcal{R}}\right)$
$\wedge\left(\forall m . j<m \wedge m<k \longrightarrow \neg\right.$ zero $\left.x\left([x s!!m]_{\mathcal{R}}\right)\right)$
$\wedge\left(\forall x \in\right.$ X2. $\exists m . j<m \wedge m \leq k \wedge$ zero $\left.x\left([x s!!m]_{\mathcal{R}}\right)\right)$
$\wedge\left(\forall x \in X 1 . \forall m \geq j\right.$. unbounded $\left.x\left([x s!!m]_{\mathcal{R}}\right)\right)$ for $i$
proof -
from unbounded obtain $i^{\prime}$ where $i^{\prime}: \forall x \in X 1 . \forall m \geq i^{\prime}$. unbounded $x\left([x s!!m]_{\mathcal{R}}\right)$ by auto then obtain $i^{\prime}$ where $i^{\prime}$ :
$i^{\prime} \geq i \forall x \in X 1 . \forall m \geq i^{\prime}$. unbounded $x\left([x s!!m]_{\mathcal{R}}\right)$
by (cases $i^{\prime} \geq i$; auto)
from finite（1）have finite X2 unfolding X2－def by auto
with $\langle X 2 \neq\{ \}\rangle \mathcal{R}$－divergent－divergent－aux2 $\left[\right.$ where $M=\left\{\lambda\right.$ i．zero $\left.\left.x\left([x s!!i]_{\mathcal{R}}\right) \mid x . x \in X \mathcal{Z}\right\}\right]$ inf－resets
have $\exists j \geq i^{\prime}$ ．$\exists k>j$ ．$\exists P \in\left\{\lambda i\right.$ ．zero $\left.x\left([x s!!i]_{\mathcal{R}}\right) \mid x . x \in X 2\right\} . P j \wedge P k$
$\wedge(\forall m<k . j<m \longrightarrow \neg P m) \wedge\left(\forall Q \in\left\{\lambda i\right.\right.$ ．zero $\left.\left.x\left([x s!!i]_{\mathcal{R}}\right) \mid x . x \in X 2\right\} . \exists m \leq k . j<m \wedge Q m\right)$
by force
then obtain $j k x$ where
$j \geq i^{\prime} k>j x \in$ X2 zero $x\left([x s!!j]_{\mathcal{R}}\right)$ zero $x\left([x s!!k]_{\mathcal{R}}\right)$
$\forall m . j<m \wedge m<k \longrightarrow \neg$ zero $x\left([x s!!m]_{\mathcal{R}}\right)$
$\forall Q \in\left\{\lambda i\right.$ ．zero $x\left([x s!!i]_{\mathcal{R}}\right) \mid x . x \in X$ 2 $\} . \exists m \leq k . j<m \wedge Q m$
by auto
moreover from this（7）have $\forall x \in X 2 . \exists m \leq k . j<m \wedge$ zero $x\left([x s!!m]_{\mathcal{R}}\right)$ by auto
ultimately show ？thesis using $i^{\prime}$
by（inst－existentials $j k x$ ）auto
qed
moreover have $\exists j^{\prime} \geq j$ ．dur xs $j^{\prime}-$ dur xs $i \geq 0.5$
if $x: x \in$ X2 $i<j$ zero $x\left([x s!!i]_{\mathcal{R}}\right)$ zero $x\left([x s!!j]_{\mathcal{R}}\right)$ and not－reset：$\forall m . i<m \wedge m<j \longrightarrow \neg$ zero $x\left([x s!!m]_{\mathcal{R}}\right)$ and X2：$\forall x \in X 2 . \exists m . i<m \wedge m \leq j \wedge z \operatorname{zero} x\left([x s!!m]_{\mathcal{R}}\right)$ and $X 1: \forall x \in X 1 . \forall m \geq i$ ．unbounded $x\left([x s!!m]_{\mathcal{R}}\right)$
for $x i j$
proof－
have $\exists j^{\prime}>j$ ．$\neg$ zero $x\left(\left[x s!!j^{\prime}\right]_{\mathcal{R}}\right)$
proof－
from inf－resets $[$ OF $x(1)]$ obtain $j^{\prime}$ where $j^{\prime} \geq$ Suc $j \neg$ zero $x\left([x s!!j]_{\mathcal{R}}\right)$ by auto then show ？thesis by－（rule exI［where $x=j\rceil$ ；auto）
qed
from inf－resets $[$ OF $x(1)]$ obtain $j^{\prime}$ where $j^{\prime} \geq$ Suc $j \neg$ zero $x\left(\left[x s!!j^{\prime}\right]_{\mathcal{R}}\right)$ by auto
with nat－eventually－critical－path［OF $x$（4）this（2）］
obtain $j^{\prime}$ where $j^{\prime}$ ：

```
        j'>j\neg zero x ([xs !! j'\mp@subsup{]}{\mathcal{R}}{})\forallm\geqj.m<\mp@subsup{j}{}{\prime}\longrightarrow\mathrm{ zero x }([xs !!m\mp@subsup{]}{\mathcal{R}}{})
```

by auto
from $\langle x \in$ X2 $\rangle$ have $x \in \mathcal{X}$ unfolding X2－def by simp
with $\langle i<j\rangle$ not－reset not－reset－dur $\langle$ stream－trans－〉 in－S pairwise－Suc［OF eq－elapsed］have dur xs $(j-1)-$ dur xs $i=(x s!!(j-1)) x-(x s!!i) x($ is ？$d 1=? d 2)$
by（auto simp：stream．pred－set）
moreover from «zero $x\left([x s!!i]_{\mathcal{R}}\right)$ 〉 in－S have $(x s!!i) x=0$
by（auto intro：zeroD simp：stream．pred－set）
ultimately have
dur xs $(j-1)-d u r x s i=(x s!!(j-1)) x($ is ？$d 1=? d 2)$
by $\operatorname{simp}$
show ？thesis
proof（cases ？$d 1 \geq 0.5$ ）
case True
with dur－mono［of j－1 jxs］have $5 / 10 \leq$ dur xs $j-$ dur xs $i$
by $\operatorname{simp}$
then show ？thesis by blast
next
case False
have $j$－c－bound：$(x s!!j) c \leq$ ？$d 2$ if $c \in X 2$ for $c$
proof（cases（xs ！！j）c＝0）
case True
from $i n-S\langle j>-\rangle$ True $\langle x \in \mathcal{X}\rangle$ show ？thesis by（auto simp：V－def stream．pred－set）
next
case False
from X2 $\langle c \in X$ 2 $〉 i n-S$ have $\exists k>i . k \leq j \wedge(x s!!k) c=0$
by（force simp：zeroD stream．pred－set）
with False have
$\exists k>i . k \leq j-S u c 0 \wedge(x s!!k) c=0$
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
$\forall k>i . k \leq j-1 \longrightarrow(x s!!k) x \neq 0$
by (auto simp: zeroI stream.pred-set)
ultimately have

$$
(x s!!(j-1)) c \leq ? d 2
$$

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 $j-1]$
$\langle x \in \mathcal{X}\rangle\langle c \in \mathcal{X}\rangle\langle j>->$ in-S $x(4)$
have $(x s!!j) c \leq(x s!!(j-1)) c$ by (auto simp: zeroD stream.pred-set)
ultimately show ?thesis by auto
qed
moreover from False $\langle ? d 1=? d 2\rangle$ have ? $d 2<1$ by auto
moreover from in-S have $(x s!!j) c \geq 0$ if $c \in \mathcal{X}$ for $c$
using that by (auto simp: V-def stream.pred-set)
ultimately have frac-bound: frac $((x s!!j) c) \leq$ ? $d 2$ if $c \in X 2$ for $c$
using that frac-le-1I by (force simp: X2-def)
let $? u=(x s!!j)$
from $i n-S$ have $[x s!!j]_{\mathcal{R}} \in \mathcal{R}$ by (auto simp: stream.pred-set)
then obtain $I r$ where region:
$[x s!!j]_{\mathcal{R}}=$ region $\mathcal{X}$ Ir valid-region $\mathcal{X} k$ Ir
unfolding $\mathcal{R}$-def by auto
let $? S=\{$ frac $(? u c) \mid c . c \in \mathcal{X} \wedge$ isIntv $(I c)\}$
have $\mathcal{X}$-X2: $c \in X 2$ if $c \in \mathcal{X}$ isIntv $(I c)$ for $c$
proof -
from $X 1\langle j>i\rangle$ have $\forall x \in X 1$. unbounded $x\left([x s!!j]_{\mathcal{R}}\right)$ by auto
with unbounded-Greater [OF region(2) $\langle c \in \mathcal{X}\rangle] \operatorname{region}(1)$ that(2) have $c \notin X 1$ by auto
with $\langle c \in \mathcal{X}\rangle$ show $c \in X 2$ unfolding X2-def by auto
qed
have frac-bound: frac $((x s!!j) c) \leq$ ? d2 if $c \in \mathcal{X}$ isIntv $(I c)$ for $c$
using frac-bound [OF $\mathcal{X}$-X2] that.
have dur xs $\left(j^{\prime}-1\right)=$ dur xs $j$ using $j^{\prime}\langle x \in \mathcal{X}\rangle$ in-S eq-elapsed
by (subst dur-zero-tail $\left[\right.$ where $\left.\left.\omega=\operatorname{smap}\left(\lambda u .[u]_{\mathcal{R}}\right) x s\right]\right)$
(auto dest: pairwise-Suc simp: stream.pred-set)
moreover from dur-reset $[O F$ eq-elapsed in-S, of $x j-1]\langle x \in \mathcal{X}\rangle x(4)\langle j>-\rangle$ have
dur xs $j=$ dur xs $(j-1)$
by (auto simp: stream.pred-set)
ultimately have dur xs $\left(j^{\prime}-1\right)=$ dur $x s(j-1)$ by auto
moreover have dur xs $j^{\prime}-\operatorname{dur} x s\left(j^{\prime}-1\right) \geq(1-$ ?d2 $) / 2$
proof -
from $\left\langle j^{\prime}>-\right\rangle$ have $j^{\prime}>0$ by auto
with pairwise-Suc[OF trans ${ }^{\prime}$, of $\left.j^{\prime}-1\right]$ have
trans $^{\prime}\left(x s!!\left(j^{\prime}-1\right)\right)\left(x s!!j^{\prime}\right)$
by auto
moreover from $j^{\prime}$ have
$\left(x s!!\left(j^{\prime}-1\right)\right) x=0\left(x s!!j^{\prime}\right) x>0$
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
$\left(x s!!j^{\prime}\right)=\left(x s!!\left(j^{\prime}-1\right)\right) \oplus t t \geq(1-$ ? $d 2) / 2 t \geq 0$
proof -
from in-S have $[x s!!j]_{\mathcal{R}} \in \mathcal{R}$ by (auto simp: stream.pred-set)
then obtain $I^{\prime} r^{\prime}$ where region':
$\left[x s!!j^{\prime}\right]_{\mathcal{R}}=$ region $\mathcal{X} I^{\prime} r^{\prime}$ valid-region $\mathcal{X} k I^{\prime} r^{\prime}$
unfolding $\mathcal{R}$-def by auto
let $? S^{\prime}=\left\{\operatorname{frac}\left(\left(x s!!\left(j^{\prime}-1\right)\right) c\right) \mid c . c \in \mathcal{X} \wedge\right.$ Regions.isIntv $\left.\left(I^{\prime} c\right)\right\}$

```
from finite(1) have ? \(d 2 \geq \operatorname{Max}\left(? S^{\prime} \cup\{0\}\right)\)
    apply -
    apply (rule Max.boundedI)
    apply fastforce
    apply fastforce
    apply safe
    subgoal premises prems for - \(c d\)
    proof -
        from \(j^{\prime}\) have \(\left(x s!!\left(j^{\prime}-1\right)\right) c=? u c \vee\left(x s!!\left(j^{\prime}-1\right)\right) c=0\)
        by (intro resets-mono \({ }^{\prime}[\) OF eq-elapsed in-S trans \(-\langle x \in \mathcal{X}\rangle\langle c \in \mathcal{X}\rangle] ;\) auto)
    then show? ?hesis
    proof (standard, goal-cases)
        case \(A: 1\)
        show ?thesis
        proof (cases \(c \in X 1\) )
            case True
            with \(X 1\left\langle j^{\prime}>j\right\rangle\langle j>i\rangle\) have unbounded \(c\left(\left[x s!!j^{\prime}\right]_{\mathcal{R}}\right)\) by auto
            with region' \(\langle c \in \mathcal{X}\rangle\) have \(I^{\prime} c=\) Greater \((k c)\)
                by (auto intro: unbounded-Greater)
            with prems show ?thesis by auto
        next
            case False
            with \(\langle c \in \mathcal{X}\rangle\) have \(c \in X 2\) unfolding X2-def by auto
            with \(j\)-c-bound have mono: \((x s!!j) c \leq(x s!!(j-1)) x\).
            from in-S \(\langle c \in \mathcal{X}\rangle\) have \(\left(x s!!\left(j^{\prime}-1\right)\right) c \geq 0\)
                unfolding \(V\)-def stream.pred-set by auto
            then have
                frac \(\left(\left(x s!!\left(j^{\prime}-1\right)\right) c\right) \leq\left(x s!!\left(j^{\prime}-1\right)\right) c\)
                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 \() \leq(0\) :: real \()\) by linarith
            moreover from \(i n-S\langle x \in \mathcal{X}\rangle\) have \((x s!!(j-1)) x \geq 0\)
            unfolding \(V\)-def stream.pred-set by auto
            ultimately show ?thesis using prems by auto
        qed
    qed
    using in-S \(\langle x \in \mathcal{X}\rangle\) by (auto simp: \(V\)-def stream.pred-set)
    then have \(l e:(1-? d 2) / 2 \leq\left(1-\operatorname{Max}\left(? S^{\prime} \cup\{0\}\right)\right) / 2\) by simp
let ? \(u=x s!!j^{\prime}\)
let \(? u^{\prime}=x s!!\left(j^{\prime}-1\right)\)
from in-S have \(*: ? u^{\prime} \in V\left[?{ }^{\prime}\right]_{\mathcal{R}} \in \mathcal{R} ? u \in V[?]_{\mathcal{R}} \in \mathcal{R}\)
    by (auto simp: stream.pred-set)
from pairwise-Suc \(\left[\right.\) OF trans, of \(\left.j^{\prime}-1\right]\left\langle j^{\prime}>j\right\rangle\) have
    trans \(\left(x s!!\left(j^{\prime}-1\right)\right)\left(x s!!j^{\prime}\right)\)
    by auto
then have Succ:
    \(\left[x s!!j^{\prime}\right]_{\mathcal{R}} \in \operatorname{Succ} \mathcal{R}\left(\left[x s!!\left(j^{\prime}-1\right)\right]_{\mathcal{R}}\right) \wedge\left(\exists t \geq 0 . ? u=? u^{\prime} \oplus t\right)\)
proof cases
    case prems: (succ t)
    from \(*\) have \(? u^{\prime} \in\left[? u^{\prime}\right]_{\mathcal{R}}\) by auto
    with prems \(*\) show ?thesis by auto
next
    case (reset \(l\) )
    with \(\left\langle ? u^{\prime} \in V\right\rangle\) have ? \(u x \leq ? u^{\prime} x\) by (cases \(x \in\) set \(l\) ) (auto simp: \(V\)-def)
    from \(j^{\prime}\) have zero \(x\left([? u]_{\mathcal{R}}\right)\) by auto
```

with $\left\langle ? u^{\prime} \in V\right\rangle$ have $? u^{\prime} x=0$ unfolding zero-def by auto
with $\langle ? u x \leq-\rangle\langle ? u x>0\rangle$ show ?thesis by auto
next
case $i d$
with $*$ Succ-refl[of $\mathcal{R} \mathcal{X} k$, folded $\mathcal{R}$-def, OF - finite(1)] show ?thesis unfolding cval-add-def by auto
qed
then obtain $t$ where $t: ? u=x s!!\left(j^{\prime}-1\right) \oplus t t \geq 0$ by auto
note Succ $=$ Succ[THEN conjunct1]
show ?thesis
proof (cases $\exists c \in$ X2. $\exists d::$ nat. ? $u c=d$ )
case True
from True obtain $c$ and $d::$ nat where $c$ :
$c \in \mathcal{X} \quad c \in X$ 2 ? $u c=d$
by (auto simp: X2-def)
have ? $u x>0$ by fact
from pairwise-Suc $\left[O F\right.$ eq-elapsed, of $\left.j^{\prime}-1\right]\left\langle j^{\prime}>j\right\rangle$ have
eq-elapsed $\left(x s!!\left(j^{\prime}-1\right)\right)$ ?u
by auto
moreover from
elapsed-eq[OF this $\langle x \in \mathcal{X}\rangle]\left\langle\left(x s!!\left(j^{\prime}-1\right)\right) x=0\right\rangle\left\langle\left(x s!!j^{\prime}\right) x>0\right\rangle$
have elapsed $\left(x s!!\left(j^{\prime}-1\right)\right)\left(x s!!j^{\prime}\right)>0$
by auto
ultimately have
? $u c-\left(x s!!\left(j^{\prime}-1\right)\right) c>0$
using $\langle c \in \mathcal{X}\rangle$ unfolding eq-elapsed-def by auto
moreover from in-S have xs !! $\left(j^{\prime}-1\right) \in V$ 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 intv-elem $c$ ? $u\left(I^{\prime} c\right)$
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 $\left(x s!!\left(j^{\prime}-1\right)\right) c \leq 0.5$
proof -
have $\left(x s!!\left(j^{\prime}-1\right)\right) c \leq(x s!!j) c$
using $j^{\prime}(1,3)$
by (auto intro: resets-mono[OF eq-elapsed in-S trans $-\langle x \in \mathcal{X}\rangle\langle c \in \mathcal{X}\rangle]$ )
also have $\ldots \leq$ ? d2 using $j$-c-bound $[O F\langle c \in X 2\rangle]$.
also from $\langle ? d 1=? d 2\rangle\langle\neg 5 / 10 \leq-\rangle$ have $\ldots \leq 0.5$ by simp
finally show ?thesis.
qed
moreover have ? $d 2 \geq 0$ using in- $S\langle x \in \mathcal{X}\rangle$ by (auto simp: $V$-def stream.pred-set)
ultimately have ? u $c-\left(x s\right.$ !! $\left.\left(j^{\prime}-1\right)\right) c \geq(1-$ ?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: $\neg$ isConst $\left(I^{\prime} c\right)$ if $c \in \mathcal{X}$ for $c$
proof (rule ccontr, simp)
assume $A$ : isConst ( $I^{\prime} c$ )
show False
proof (cases $c \in X 1$ )
case True
with $X 1\left\langle j^{\prime}>j\right\rangle\langle j>-\rangle$ have unbounded $c\left(\left[x s!!j_{\mathcal{R}}\right)\right.$ by auto
with unbounded-Greater $\langle c \in \mathcal{X}\rangle$ region' have isGreater $\left(I^{\prime} c\right)$ by force
with $A$ show False by auto
next
case False
with $\langle c \in \mathcal{X}\rangle$ have $c \in X 2$ unfolding X2-def by auto
from region' in-S $\langle c \in \mathcal{X}\rangle$ have intv-elem $c$ ? $u\left(I^{\prime} c\right)$

```
                    unfolding stream.pred-set by force
                    with }<c\inX2\rangleA False F show False by aut
                qed
            qed
            have }\not\existsx.x\leqkc\wedge(xs!! j')c=real x if c\in\mathcal{X}\mathrm{ for c
            proof (cases c \in X2; safe)
                fix d
                assume c \in X2 (xs !! j') c = real d
                with F}\mathrm{ show False by auto
            next
                fix }
                    assume c& X2
                with that have c\inX1 unfolding X2-def by auto
                with X1 〈j'> j\rangle\langlej> i\rangle have unbounded c ([?u\mp@subsup{]}{\mathcal{R}}{})\mathrm{ by auto}
                from unbounded-all[OF - this]\langlec\in\mathcal{X}\ranglein-S have ?u c>kc
                    by (force simp: stream.pred-set)
            moreover assume ?u c= real d d \leqkc
            ultimately show False by auto
            qed
            with delayedR-aux have
                (xs !! j') = delayedR ([xs !! j j ]\mathcal{R})(xs !! ( }\mp@subsup{j}{}{\prime}-1)
                using }\langlex\in\mathcal{X}\rangle\mathrm{ unfolding trans'-def by auto
            from not-const region'(1) in-S Succ(1) have
                \existst\geq0.delayedR ([xs !! j` \}\mathcal{R})(xs !! (j' - 1)) = xs !! ( j' - 1)\oplust
                    (1-Max (?S'\cup{0})) / 2 \leqt
            apply simp
            apply (rule delayedR-correct(2)[OF - region'(2), simplified])
            by (auto simp: stream.pred-set)
            with le <- = delayedR --> show ?thesis by (auto intro: that)
            qed
            qed
            moreover from pairwise-Suc[OF eq-elapsed, of j' - 1]< \'> 0\rangle have
            eq-elapsed (xs !! (j' - 1)) (xs !! j')
            by auto
            ultimately show dur xs j' - dur xs (j' - 1) \geq (1 - ?d2) / 2
            using \langlej'> 0\rangle dur-Suc[of- j' - 1] <x \in\mathcal{X}\rangle by (auto simp: cval-add-def elapsed-eq)
        qed
            moreover from dur-mono[of i j - 1 xs] <i< j> have dur xs i\leqdur xs (j - 1) by simp
            ultimately have dur xs j' - dur xs i\geq 0.5 unfolding <?d1 = ?d2`[symmetric] by auto
            then show ?thesis using <j< j'> by - (rule exI[where x = j]; auto)
        qed
    qed
    moreover
    have }\exists\mp@subsup{j}{}{\prime}\geqi.dur xs \mp@subsup{j}{}{\prime}-dur xs i\geq0.5 for 
    proof -
        from calculation(1)[of i] obtain jkx where
        j\geqik>j x\inX2 zero x ([xs !! j] _\mathcal{R}
        zero x ([xs !! k\mp@subsup{]}{\mathcal{R}}{})
        \forallm.j<m\wedgem<k\longrightarrow\neg zero x ([xs !! m\mp@subsup{]}{\mathcal{R}}{})
        \forallx\inX2. \existsm>j.m\leqk^ zero x ([xs !! m\mp@subsup{]}{\mathcal{R}}{})
        \forallx\inX1.\forallm\geqj. unbounded x ([xs !! m\mp@subsup{]}{\mathcal{R}}{})
        by auto
        from calculation(2)[OF this(3,2,4-8)] obtain j' where
            j}\geqk5/10\leqdur xs j' - dur xs j
            by auto
        with dur-mono[of ijxs]<j\geqi\rangle\langlek> 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-absc:
    notes in-space-UNIV[measurable]
    assumes cfg \incfg-on-div st st }\in
    shows absc cfg \inR-G-cfg-on-div (abss st)
proof -
    from assms have *: cfg \in MDP.cfg-on st state cfg = st div-cfg cfg
        unfolding cfg-on-div-def by auto
    with assms have cfg \in valid-cfg by (auto intro: MDP.valid-cfgI)
    have almost-everywhere (MDP.MC.T cfg) (MDP.MC.enabled cfg)
        by (rule MDP.MC.AE-T-enabled)
    moreover from * have AE x in MDP.MC.T cfg. divergent (smap (snd\circ state) x)
        by (simp add: div-cfg-def)
    ultimately have AE x in MDP.MC.T cfg. \mathcal{R-div (smap (snd o state) (smap absc x))}
    proof eventually-elim
        case (elim \omega)
        let ?xs = smap (snd o state) \omega
        from MDP.pred-stream-cfg-on[OF <- \in valid-cfg〉<MDP.MC.enabled -->] have *:
            pred-stream (\lambda x. x \in S) (smap state \omega)
            by (auto simp: stream.pred-set)
        have [snd (state x)]}\mp@subsup{\mathcal{R}}{}{=}\mathrm{ snd (abss (state x)) if x sset }\omega\mathrm{ for x
        proof -
                from * that have state x 
                then have snd (abss (state x)) =[snd (state x) ]}\mp@subsup{]}{\mathcal{R}}{}\mathrm{ by (metis abss-S snd-conv surj-pair)
                then show ?thesis ..
        qed
        then have smap (\lambdaz.[snd (state z)\mp@subsup{]}{\mathcal{R}}{})\omega=(\operatorname{smap}(\lambdaz. snd (abss (state z))) \omega) by auto
        from * have pred-stream ( }\lambdau.u\inV)\mathrm{ ?xs
            apply (simp add: map-def stream.pred-set)
            apply (subst (asm) surjective-pairing)
            using S-V by blast
            moreover have stream-trans ?xs
                by (rule enabled-stream-trans <-\in valid-cfg\rangle\langleMDP.MC.enabled -->)+
            ultimately show ?case using <divergent -\rangle\langlesmap - \omega = -〉
                by - (drule divergent-\mathcal{R}\mathrm{ -divergent, auto simp add: stream.map-comp state-absc)}
    qed
    with <cfg \in valid-cfg> have R-G-div-cfg (absc cfg) unfolding R-G-div-cfg-def
        by (subst absc-distr-self) (auto intro: MDP.valid-cfgI simp: AE-distr-iff)
    with R-G.valid-cfgD <cfg \in valid-cfg>* show ?thesis unfolding R-G-cfg-on-div-def by auto force
qed
```


## definition

```
    alternating cfg = (AE \omega in MDP.MC.T cfg.
        alw (ev (HLD {cfg.\forallcfg'\inK-cfg cfg.fst (state cfg')=fst (state cfg)})) \omega)
lemma \(K\)-cfg-same-loc-iff:
\(\left(\forall c f g^{\prime} \in K-c f g c f g . f s t\left(\right.\right.\) state \(\left.c f g^{\prime}\right)=f s t(\) state \(\left.c f g)\right)\)
\(\longleftrightarrow\left(\forall c f g^{\prime} \in K-c f g(a b s c c f g) . f s t\left(\right.\right.\) state \(\left.c f g^{\prime}\right)=f s t(\) state \(\left.(a b s c c f g))\right)\)
if \(c f g \in\) valid-cfg
using that by (auto simp: state-absc fst-abss \(K\)-cfg-map-absc)
lemma (in -) stream-all2-flip:
stream-all2 ( \(\lambda a b . R \quad b a)\) xs ys \(=\) stream-all2 \(R\) ys xs
by (standard; coinduction arbitrary: xs ys; auto dest: sym)
```

```
lemma AE-alw-ev-same-loc-iff:
```

lemma AE-alw-ev-same-loc-iff:
assumes cfg \in valid-cfg
assumes cfg \in valid-cfg
shows alternating cfg \longleftrightarrow alternating (absc cfg)
shows alternating cfg \longleftrightarrow alternating (absc cfg)
unfolding alternating-def
unfolding alternating-def
apply (simp add: MDP.MC.T.AE-iff-emeasure-eq-1)
apply (simp add: MDP.MC.T.AE-iff-emeasure-eq-1)
subgoal

```
    subgoal
```

```
    proof -
    show ?thesis (is (?x=1)=(?y=1))
    proof -
        have *: stream-all2 ( }\lambdast.t=absc s) x y = stream-all2 (=) y (smap absc x) for x y
            by (subst stream-all2-flip) simp
        have ?}x=\mathrm{ ? y
            apply (rule T-eq-rel-half[where f=absc and S=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=\lambdast. t=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 \in valid-cfg> by simp+
        then show ?thesis
            by simp
        qed
    qed
    done
lemma AE-alw-ev-same-loc-iff':
    assumes cfg \inR-G.cfg-on (abss st) st \inS
    shows alternating cfg \longleftrightarrow alternating (repcs st cfg)
proof -
    from assms have cfg \inR-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\oplusd=bd>0\mathrm{ for d :: real
proof -
    from that(1) have ( }b\oplusd)x=b
        by (rule fun-cong)
    with <d> 0\rangle show False
        unfolding cval-add-def by simp
qed
lemma repcs-unbounded-AE-non-loop-end-strong:
    assumes cfg}\inR-G.cfg-on (abss st) st \in
        and alternating cfg
    shows AE \omega in MDP.MC.T (repcs st cfg).
        (\forallu:: ('c m real). (\forallc\in\mathcal{X.u c> real (kc))\longrightarrow}
        \neg ( e v ( a l w ~ ( \lambda x s . ~ s h d ~ x s = u ) ) ) ( s m a p ~ ( s n d ~ o ~ s t a t e ) ~ \omega ) ) ( i s ~ A E ~ \omega ~ i n ~ ? M . ~ ? P ~ \omega )
proof -
    from assms have cfg \inR-G.valid-cfg
        by (auto intro: R-G.valid-cfgI)
    with assms(1) have repcs st cfg \in valid-cfg
        by auto
    from R-G.valid-cfgD[OF <cfg \inR-G.valid-cfg>] have cfg \inR-G.cfg-on (state cfg).
    let ? U = \lambda u. \bigcupl\inL. {\mu\inK (l,u). \mu\not= return-pmf (l, u)^(\forallx\in\mu.fst x=l)}
    let ?r = \lambdau.Sup ({0}\cup (\lambda . measure-pmf }\mu{x.\mathrm{ snd }x=u})'? ? U u
    have lt-1: ?r }u<1\mathrm{ for u
    proof -
        have *: emeasure (measure-pmf }\mu\mathrm{ ) {x. snd x=u} < < 
            if }\mu\not=\mathrm{ return-pmf (l,u)}\forallx\in\mathrm{ set-pmf }\mu\mathrm{ . fst x=l for }\mu\mathrm{ and l :: 's
        proof (rule ccontr)
            assume }\neg\mathrm{ emeasure (measure-pmf }\mu\mathrm{ ) {x. 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| |lg. (l,g, \mu)\in trans-of A}
have (\lambda \mu. measure-pmf }\mu{x.\mathrm{ snd }x=u})'? U
    \subseteq{0,1}\cup(\lambda \mu. measure-pmf }\mu{x.\mathrm{ snd }x=u})'?
    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\mathrm{ . measure-pmf }\mu{x.\mathrm{ 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 m real and cfg :: ('s > ('c m real) set) cfg
    assume unbounded: }\forallc\in\mathcal{X}.uc>kc and cfg\inR-G.cfg-on (abss (l,u)) abss (l,u)\in\mathcal{S
        and same-loc: }\forall cfg'\inK-cfg cfg. fst (state cfg')=
    then have cfg}\inR\mathrm{ -G.valid-cfg repcs (l,u) cfg }\in\mathrm{ 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 <cfg \inR-G.cfg-on >> 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\inL
    using MDP.valid-cfg-state-in-S <repcs (l,u) cfg \in MDP.valid-cfg> by fastforce
    moreover have rept (l,u) (action cfg) = 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 <action cfg \in ->]
        by (simp add: abst-def)
    moreover have (l,u)\inS
        using <- }\in\mathcal{S}\rangle\mathrm{ by (auto dest: S-abss-S)
    moreover have abss (l,u)=(l, [u\mp@subsup{]}{\mathcal{R}}{})
        by (metis abss-S calculation(2))
    ultimately show False
        using <rept (l,u) - = -> unbounded unfolding rept-def by (auto dest: cval-add-non-id)
    qed
    moreover have rept (l,u) (action cfg) \inK (l,u)
proof -
    have action (repcs (l,u) cfg)\inK (l,u)
        using cfg-on by blast
    then show ?thesis
        by (simp add: repcs-def)
    qed
    moreover have }\forallx\inset-pmf (rept (l,u) (action cfg)). fst x = l
    using same-loc K-cfg-same-loc-iff[of repcs (l,u) cfg]
        <repcs (l,u)-\in valid-cfg\rangle\langlecfg \inR-G.valid-cfg\rangle\langlecfg \inR-G.cfg-on ->
```

```
    by (simp add: absc-repcs-id fst-abss K-cfg-rept[symmetric])
    ultimately have rept (l,u) (action cfg) \in?U u
        by blast
    then have measure-pmf (rept (l,u) (action cfg)) {x. snd x=u}\leq ?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}\leq ?r u
        by auto
}
note * = this
let ?S = {cfg. \exists cfg' s.cfg' \inR-G.valid-cfg ^ cfg = repcs s cfg'^ abss s = state cfg'}
have start: repcs st cfg \in?S
    using <cfg \inR-G.valid-cfg> assms unfolding R-G-cfg-on-div-def
    by clarsimp (inst-existentials cfg fst st snd st, auto)
have step: y\in?S if y\inK-cfg x x \in?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 \in?S if (repcs st cfg, x)\inMDP.MC.acc for x
proof -
    from MDP.MC.acc-relfunD[OF that] obtain n where ((\lambdaab.b 位-cfg a) ^~ 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 (\lambda\omega. HLD (state -' snd -'{u}) \omega\longrightarrow
        ev (HLD {cfg.\forallcfg'\inset-pmf (K-cfg cfg). fst (state cfg') = fst (state cfg)}) \omega))
    for u unfolding alternating-def by (auto elim: alw-mono)
let ?X = {cfg:: ('s > ('c m real)) cfg. \forallc\in\mathcal{X}. snd (state cfg) c>kc}
let ?Y ={cfg.\forallcfg'\inK-cfg cfg.fst (state cfg')= fst (state cfg)}
have }(AE\omega\mathrm{ in ?M. ?P }\omega)
```

```
    (AE\omega in ?M. \forall u :: ('c m real).
    (\forallc\in\mathcal{X.u c>kc)^u\in snd'state '(MDP.MC.acc " {repcs st cfg})}\longrightarrow
    \neg ( e v ( a l w ~ ( \lambda ~ x s . ~ s h d ~ x s = u ) ) ) ~ ( s m a p ~ ( s n d ~ o ~ s t a t e ) ~ \omega ) ) ~ ( i s ~ ? 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 \omegau)
        then show ?case
            by - (intro alw-HLD-smap alw-disjoint-ccontr[where
                    S = (snd o state)' MDP.MC.acc " {repcs st cfg}
                    and }R={u}\mathrm{ and }\omega=\operatorname{smap}\mathrm{ (snd o state) }
                        ]; auto simp: HLD-iff)
    qed
qed
also have ... \longleftrightarrow
        (\forallu :: ('c = real).
            (\forallc\in\mathcal{X.u c >k c)^u\in snd'state '(MDP.MC.acc " {repcs st cfg}) }\longrightarrow
            (AE\omega in ?M. \neg(ev (alw (\lambdaxs. shd xs = u))) (smap (snd o 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.acc " {repcs st cfg}])
also show ?thesis
    unfolding calculation
    apply clarsimp
    subgoal for l ux
        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)=\mathrm{ state x><snd (l1,u1) = snd (state x)><u2 = u1〉 snd-conv)
            have [simp]: snd -'{snd (state x)}={y. snd y = snd (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 \in 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 \inR-G-cfg-on-div (abss st) st \inS and alternating cfg
    shows repcs st cfg}\incfg-on-div s
proof -
    let ?st = abss st
    let ?cfg = repcs st cfg
    from assms have *:
        cfg\inR-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 \inR-G.valid-cfg by (auto intro: R-G.valid-cfgI)
    with «st \inS\rangle\langle- = ?st\rangle have ?cfg \in valid-cfg by auto
    from *(1)\langlest \inS\rangle\langlealternating cfg\rangle have
        AE \omega in MDP.MC.T ?cfg. }\forallu.(\forallc\in\mathcal{X}.real (kc)<uc)
                        \neg e v ( a l w ~ ( \lambda x s . ~ s h d ~ x s ~ = u ) ) ( s m a p ~ ( s n d \circ ~ s t a t e ) ~ \omega )
        by (rule repcs-unbounded-AE-non-loop-end-strong)
    - Move to lower level
```



```
        unfolding R-G-div-cfg-def
        by (subst (asm) R-G-trace-space-distr-eq[OF <cfg \in 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 \omega)
            let ?xs = smap (snd o state) }
            from MDP.pred-stream-cfg-on[OF <- \in valid-cfg> <MDP.MC.enabled -->] have *:
                pred-stream (\lambdax.x\inS) (smap state \omega)
                by (auto simp: stream.pred-set)
            have [snd (state x)] ]
            proof -
                from * that have state x\inS by (auto simp: stream.pred-set)
                then have snd (abss (state x)) = [snd (state x) ]}\mp@subsup{]}{\mathcal{R}}{}\mathrm{ by (metis abss-S snd-conv surj-pair)
                then show ?thesis ..
            qed
            then have smap (\lambdaz.[snd (state z)\mp@subsup{]}{\mathcal{R}}{})\omega=(\operatorname{smap}(\lambdaz.snd (abss (state z))) \omega) by auto
            from * have pred-stream ( }\lambdau.u\inV)\mathrm{ ?xs
                by (simp add: map-def stream.pred-set, subst (asm) surjective-pairing, blast)
            moreover have stream-trans ?xs
                by (rule enabled-stream-trans <-\in valid-cfg\rangle\langleMDP.MC.enabled -->)+
            moreover have pairwise trans' ?xs
                using <-\inR-G.valid-cfg\rangle<state cfg = ->[symmetric]<MDP.MC.enabled - ->
                by (rule enabled-stream-trans')
            moreover from prems(1) have
                \forallu.(\forallc\in\mathcal{X}.real (kc)<uc)\longrightarrow\negev(alw (\lambdaxs. snd (shd xs)=u))(smap state }\omega
                by simp
            ultimately show ?case using <\mathcal{R}
                by (simp add: stream.map-comp state-absc «smap - \omega = -> \mathcal{R}
        qed
    with MDP.valid-cfgD <cfg \inR-G.valid-cfg〉* show ?thesis unfolding cfg-on-div-def by auto force
qed
```

```
lemma repcs-unbounded-AE-non-loop-end:
    assumes cfg}\inR\mathrm{ -G.cfg-on (abss st) st }\in
    shows AE \omega in MDP.MC.T (repcs st cfg).
        (}\foralls::('s\times('c=> real)). (\forallc\in\mathcal{X}. snd s c>kc)
        \neg ( e v ( a l w ~ ( \lambda ~ x s . ~ s h d ~ x s = s ) ) ) ( s m a p ~ s t a t e ~ \omega ) ) ~ ( i s ~ A E ~ \omega ~ i n ~ ? M . ~ ? P ~ \omega )
proof -
    from assms have cfg}\inR\mathrm{ -G.valid-cfg
        by (auto intro: R-G.valid-cfgI)
    with assms(1) have repcs st cfg \in valid-cfg
        by auto
    from R-G.valid-cfgD[OF<cfg\inR-G.valid-cfg>] have cfg \inR-G.cfg-on (state cfg).
    let ?K = \lambda x. { }\mu\inKx.\mu\not= return-pmf x
    let ?r = \lambda x. Sup ((\lambda \mu. measure-pmf }\mu{x})'?K x
    have lt-1: ?r }x<1\mathrm{ if }\mu\in\mathrm{ ? K }x\mathrm{ for }\mu
    proof -
        have *: emeasure (measure-pmf }\mu\mathrm{ ) {x}<1 if }\mu\not=\mathrm{ return-pmf x for }
        proof (rule ccontr)
            assume \neg emeasure (measure-pmf }\mu\mathrm{ ) {x}<1
            then have emeasure (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 =
            {map-pmf (\lambda (X,l).(l,([X := 0]u))) \mu| \mulug.
                x=(l,u)\wedge(l,g, \mu)\in trans-of A}
        have ( }\lambda\mu\mathrm{ . measure-pmf }\mu{x})\mathrm{ '? }K
            \subseteq \{ 0 , 1 \} \cup ( \lambda \mu . ~ m e a s u r e - p m f ~ \mu \{ x \} ) ' ? 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](snd x)))) \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})'?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=> real) and cfg :: ('s }\times(\mp@subsup{}{}{\prime}c=>\mathrm{ real ) set) cfg
        assume unbounded: }\forallc\in\mathcal{X}\mathrm{ . snd s c>kc and cfg }\inR\mathrm{ -G.cfg-on (abss s) abss s }\in\mathcal{S
        then have repcs s cfg \in valid-cfg
            by (auto intro: R-G.valid-cfgI)
        then have cfg-on: repcs s cfg \in MDP.cfg-on s
            by (auto dest: MDP.valid-cfgD)
        from <cfg \in -> have action cfg \in\mathcal{K}(abss s)
            by (rule R-G.cfg-onD-action)
        have rept s (action cfg) \not= return-pmf s
        proof (rule ccontr, simp)
            assume rept s (action cfg) = return-pmf s
            then have action cfg = return-pmf (abss s)
                using abst-rept-id[OF<action cfg \in ->]
                by (simp add: abst-def)
            moreover have (fst s, snd s)\inS
                using <- \in\mathcal{S}\rangle by (auto dest: \mathcal{S}
            moreover have abss s}=(fsts,[snd s\mp@subsup{]}{\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) \inKs
    proof -
        have action (repcs s cfg) \inKs
        using cfg-on by blast
    then show ?thesis
        by (simp add: repcs-def)
    qed
    ultimately have rept s(action cfg) \in?K s
        by blast
    then have measure-pmf (rept s (action cfg)) {s}\leq?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 scfg)) {s}\leq?r s
        by auto
    note this <rept s (action cfg) \in?K s\rangle
}
note * = this
let ?S = {cfg. \exists cfg' s.cfg'\inR-G.valid-cfg ^cfg= repcs s cfg'^ abss s= state cfg'}
have start: repcs st cfg \in?S
    using <cfg \inR-G.valid-cfg> assms unfolding R-G-cfg-on-div-def
    by clarsimp (inst-existentials cfg fst st snd st, auto)
have step: y\in?S if y\inK-cfg x x \in?S for x y
    using that apply safe
    subgoal for cfg' lu
        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 \in?S if (repcs st cfg, x)\inMDP.MC.acc for x
proof -
```



```
    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\omega\mathrm{ in ?M. ?P }\omega)
    (AE\omega in ?M. }\forall\textrm{s}::('('s\times('c=>real))
        (\forallc\in\mathcal{X}. snd sc>kc)\wedges\in state '(MDP.MC.acc " {repcs st cfg})\longrightarrow
        \neg ( e v ( a l w ~ ( \lambda ~ x s . ~ s h d ~ x s = s ) ) ) ( s m a p ~ s t a t e ~ \omega ) ) ~ ( i s ~ ? 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 \omegas s)
        from this(1,2,5,6) show ?case
            by (intro alw-HLD-smap alw-disjoint-ccontr[where
                S = state 'MDP.MC.acc " {repcs st cfg} and R}={s}\mathrm{ and }\omega=\mathrm{ smap state }
                ]; simp add: HLD-iff; blast)
    qed
qed
also have ... \longleftrightarrow
    (}\foralls::('s\times('c=> real))
            (\forallc\in\mathcal{X. snd s c>k c)^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 ux
        apply (rule MDP.non-loop-tail'}[\mathrm{ 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',}\mp@subsup{u}{}{\prime}
                        by (metis MDP.cfg-onD-state state-repcs)
            with <- = state x> have [simp]:l= l' u= u'
                by auto
            show ?thesis
                unfolding <state x = -> 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 <repcs st cfg \in valid-cfg>]; 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:
\(c f g \in R\)-G.valid-cfg if \(c f g \in R\) - \(G\)-cfg-on-div \(s^{\prime}\)
using that unfolding \(R\)-G-cfg-on-div-def \(R\)-G.valid-cfg-def by auto
```

lemma cfg－on－valid：
$c f g \in$ valid－cfg if $c f g \in c f g$－on－div $s$
using that unfolding $c f g$－on－div－def MDP．valid－cfg－def by auto
abbreviation path－measure $P c f g \equiv$ emeasure（MDP．T cfg）$\{x \in$ space MDP．St．$P x\}$
abbreviation $R$－G－path－measure $P c f g \equiv$ emeasure（ $R$－G．T cfg）$\{x \in$ space $R$－G．St．$P x\}$
abbreviation progressive st $\equiv c f g$－on－div st $\cap\{c f g$ ．alternating cfg $\}$
abbreviation $R$－$G$－progressive st $\equiv R$－$G$－cfg－on－div st $\cap\{c f g$ ．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-alw-ev-same-loc-iff cfg-on-valid)
    subgoal for cfg
    unfolding s'-def
    by (frule cfg-on-div-repcs-strong)
        (auto 44
            simp: s'-def R-G-cfg-on-div-def AE-alw-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 $[i f f]=$ pred-stream-iff
shows
$(\bigsqcup c f g \in$ progressive $s . \quad$ path-measure $\quad(\lambda x$. (holds $\varphi$ suntil holds $\psi)(s \# \# x)) c f g)$
$=\left(\bigsqcup c f g \in R\right.$ - $G$-progressive $s^{\prime}$. $R$ - $G$-path-measure $\left(\lambda x\right.$. (holds $\varphi^{\prime}$ suntil holds $\left.\left.\psi^{\prime}\right)\left(s^{\prime} \# \# x\right)\right) c f g$ )
$\wedge(\Pi c f g \in$ progressive s. path-measure $\quad(\lambda x$. (holds $\varphi$ suntil holds $\psi)(s \# \# x)) c f g)$
$=\left(\Pi c f g \in R\right.$ - $G$-progressive $s^{\prime}$. $R$ - $G$-path-measure $\left(\lambda x\right.$. (holds $\varphi^{\prime}$ suntil holds $\left.\left.\left.\psi^{\prime}\right)\left(s^{\prime} \# \# x\right)\right) c f g\right)$
proof (rule $S U P$-eq-and-INF-eq; rule bexI[rotated]; erule IntE)
fix $c f g$ assume $c f g$-div: $c f g \in R-G$-cfg-on-div $s^{\prime}$ and $c f g \in$ Collect alternating
then have alternating cfg
by auto
let $? c f g^{\prime}=$ repcs $s c f g$
from 〈alternating cfg〉cfg-div have alternating ?cfg'
by (simp add: R-G-cfg-on-div-def $s^{\prime}$-def AE-alw-ev-same-loc-iff' $\left.[o f-s]\right)$
with $c f g$-div 〈alternating $c f g\rangle$ show ? ${ }^{c} f g^{\prime} \in c f g$-on-div $s \cap$ Collect alternating
by (auto intro: cfg-on-div-repcs-strong simp: $s^{\prime}$-def)
show emeasure ( $R$-G.T cfg) $\quad\left\{x \in\right.$ space $R$-G.St. (holds $\varphi^{\prime}$ suntil holds $\left.\left.\psi^{\prime}\right)\left(s^{\prime} \# \# x\right)\right\}$
$=$ emeasure (MDP.T?cfg') $\{x \in$ space MDP.St. (holds $\varphi$ suntil holds $\psi$ ) $(s \# \# x)\}$
(is ? $a=? b$ )
proof -
from $c f g$-div have $c f g \in R$-G.valid-cfg
by (rule $R$ - $G$-cfg-on-valid)
from $c f g$-div have $c f g \in R$-G.cfg-on $s^{\prime}$
unfolding $R$-G-cfg-on-div-def by auto
then have state $c f g=s^{\prime}$
by auto
have ? $a=? b$
apply (rule
path-measure-eq-repcs" ${ }^{\prime \prime}$-new[
of $s$ cfg $\varphi \psi$, folded $\varphi^{\prime}$-def $\psi^{\prime}$-def, unfolded $\left\langle-=s^{\prime}\right\rangle$ state-repcs
]
)

```
        subgoal
            unfolding s'-def ..
        subgoal
            by fact
        subgoal
            using <?cfg' \in cfg-on-div s \cap -> by (blast intro:cfg-on-valid)
        subgoal premises prems for xs
            using prems s by (intro \varphi-stream)
        subgoal premises prems
            using prems s by (intro \psi-stream)
        done
    then show ?thesis
        by simp
    qed
next
    fix cfg assume cfg-div: cfg \incfg-on-div s and cfg \in Collect alternating
    with absc-valid-cfg-eq show absc cfg \inR-G-cfg-on-div s'\cap Collect alternating
        by auto
    show emeasure (MDP.T cfg) {x { space MDP.St. (holds \varphi suntil holds \psi) (s## x)}
        = emeasure (R-G.T (absc cfg)) {x\in space R-G.St. (holds \varphi' suntil holds \psi') (s'## x)}
        (is ?a = ?b)
    proof -
        have absc cfg \inR-G.valid-cfg
            using R-G-cfg-on-valid «absc cfg \inR-G-cfg-on-div s'\cap -> by blast
        from cfg-div have cfg \in valid-cfg
            by (simp add: cfg-on-valid)
        with <absc cfg\inR-G.valid-cfg> have ?b = ?a
            by (intro MDP.alw-S R-G.alw-S path-measure-eq-absc1-new
                    [where P=pred-stream ( }\lambdas.s\in\mathcal{S})\mathrm{ and }Q=\operatorname{pred-stream (\lambdas.s\inS)]
            )
            (auto simp:S-abss-\mathcal{S intro: S-abss-S intro!: suntil-abss suntil-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.

