

# Probabilistic Timed Automata

Simon Wimmer and Johannes Hölzl

December 14, 2021

## Abstract

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

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

## Contents

<b>1</b>	<b>Bisimulation on a Relation</b>	<b>3</b>
<b>2</b>	<b>Additional Facts on Regions</b>	<b>3</b>
2.1	Justifying Timed Until vs <i>suntil</i>	4
<b>3</b>	<b>Definition and Semantics</b>	<b>4</b>
3.1	Syntactic Definition	4
3.1.1	Collecting Information About Clocks	5
3.2	Operational Semantics as an MDP	6
3.3	Syntactic Definition	6
<b>4</b>	<b>Constructing the Corresponding Finite MDP on Regions</b>	<b>6</b>
4.1	Syntactic Definition	7
4.2	Many Closure Properties	7
4.3	The Region Graph is a Finite MDP	8
<b>5</b>	<b>Relating the MDPs</b>	<b>9</b>
5.1	Translating From $\mathcal{K}$ to $\mathcal{K}$	9
5.2	Translating Configurations	10
5.2.1	States	10
5.2.2	Intermezzo	12
5.2.3	Predicates	12
5.2.4	Distributions	12
5.2.5	Configuration	15
5.3	Equalities Between Measures of Trace Spaces	19
<b>6</b>	<b>Classifying Regions for Divergence</b>	<b>21</b>
6.1	Pairwise	21
6.2	Regions	21
6.3	Unbounded and Zero Regions	22

<b>7</b>	<b>Reachability</b>	<b>22</b>
7.1	Definitions . . . . .	22
7.2	Easier Result on All Configurations . . . . .	23
7.3	Divergent Adversaries . . . . .	24
7.4	Main Result . . . . .	29

```

theory PTA
  imports library/Lib
begin

```

## 1 Bisimulation on a Relation

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

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

**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*  
 $(\text{THE } R'. R' \neq R \wedge R' \in \text{Succ } \mathcal{R} R \wedge (\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in R' \wedge 0 \leq t')))$

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

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

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

**context** *AlphaClosure*

**begin**

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

**lemma** *succ-ex*:

**assumes**  $R \in \mathcal{R}$

**shows**  $\text{succ } \mathcal{R} R \in \mathcal{R}$  (**is** ?G1) **and**  $\text{succ } \mathcal{R} R \in \text{Succ } \mathcal{R} R$  (**is** ?G2)

**and**  $\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' \leq t. (u \oplus t') \in \text{succ } \mathcal{R} R \wedge 0 \leq t')$  (**is** ?G3)  
*<proof>*

**lemma** *region-set'-closed*:

**fixes**  $d :: \text{nat}$

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

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

*<proof>*

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

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

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

*<proof>*

**lemma** *region-reset-not-Succ*:

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

**assumes**  $R \in \mathcal{R} \ \text{set } r \subseteq X$

**shows**  $\text{region-set}' R \ r \ 0 = R \vee \text{region-set}' R \ r \ 0 \notin \text{Succ } \mathcal{R} R$  (**is** ?R =  $R \vee -$ )

*<proof>*

**end**

## 2.1 Justifying Timed Until vs *suntil*

**lemma** *guard-continuous*:

**assumes**  $u \vdash g \ u \oplus t \vdash g \ 0 \leq (t' :: t :: \text{time}) \ t' \leq t$

**shows**  $u \oplus t' \vdash g$

*<proof>*

## 3 Definition and Semantics

### 3.1 Syntactic Definition

We do not include:

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

**type-synonym**

$(\text{'c}, \text{'t}, \text{'s}) \text{ transition} = \text{'s} * (\text{'c}, \text{'t}) \text{ cconstraint} * (\text{'c} \text{ set} * \text{'s}) \text{ pmf}$

**type-synonym**

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

**definition**

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

**where**

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

**definition**

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

**definition**

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

**where**

$trans-of \equiv fst$

**definition**

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

**where**

$inv-of \equiv snd$

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

**abbreviation transition**  $::$

$(('c, 'time, 's)\ pta \Rightarrow 's \Rightarrow ('c, 'time)\ cconstraint \Rightarrow ('c\ set * 's)\ pmf \Rightarrow 'c\ set \Rightarrow 's \Rightarrow bool$   
 $(- \vdash - \longrightarrow_{\text{inv-of}} - [61,61,61,61,61,61]\ 61)$  **where**  
 $(A \vdash l \longrightarrow_{g,p,X} l') \equiv (l, g, p, X, l') \in Edges\ A$

**definition**

$locations :: ('c, 't, 's)\ pta \Rightarrow 's\ set$

**where**

$locations\ A \equiv (fst \text{ ` } Edges\ A) \cup ((snd\ o\ snd\ o\ snd\ o\ snd) \text{ ` } Edges\ A)$

### 3.1.1 Collecting Information About Clocks

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

**where**

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

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

**where**

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

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

**where**

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

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

**where**

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

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

**definition valid-abstraction**

**where**

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

**lemma valid-abstractionD[dest]:**

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

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

**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**  $valid-abstraction\ A\ X\ k$   
 $\langle proof \rangle$

### 3.2 Operational Semantics as an MDP

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

**where**

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

**term** *region-set'*

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

**where**

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

**no-notation** *zone-set* (-  $\rightarrow$  0 [71] 71)

**abbreviation** *zone-set-set* :: ('c, 't::time) zone  $\Rightarrow$  'c set  $\Rightarrow$  ('c, 't) zone  
 (-  $\rightarrow$  0 [71] 71)

**where**

Z  $\rightarrow$  0  $\equiv$  zone-set Z (SOME r. set r = X)

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

**locale** *Probabilistic-Timed-Automaton* =

**fixes** A :: ('c, 't :: time, 's) pta

**assumes** *admissible-targets*:

(l, g,  $\mu$ )  $\in$  trans-of A  $\implies$  (X, l')  $\in$   $\mu \implies$  {g}<sub>X  $\rightarrow$  0</sub>  $\subseteq$  {inv-of A l'}

(l, g,  $\mu$ )  $\in$  trans-of A  $\implies$  (X, l')  $\in$   $\mu \implies$  X  $\subseteq$  clocks A

— Not necessarily what we want to have

**begin**

### 3.3 Syntactic Definition

**definition** L = locations A

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

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

**inductive-set**

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

**where**

— Passage of time *delay*:

st  $\in$  S  $\implies$  st = (l, u)  $\implies$  t  $\geq$  0  $\implies$  u  $\oplus$  t  $\vdash$  inv-of A l  $\implies$  return-pmf (l, u  $\oplus$  t)  $\in$  K st |

— Discrete transitions *action*:

st  $\in$  S  $\implies$  st = (l, u)  $\implies$  (l, g,  $\mu$ )  $\in$  trans-of A  $\implies$  u  $\vdash$  g

$\implies$  map-pmf ( $\lambda$  (X, l). (l, ([X := 0]u)))  $\mu \in$  K st |

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

return-pmf st  $\in$  K st

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

**sublocale** MDP: Markov-Decision-Process K  $\langle$ proof $\rangle$

**end**

## 4 Constructing the Corresponding Finite MDP on Regions

**locale** *Probabilistic-Timed-Automaton-Regions* =

*Probabilistic-Timed-Automaton* A + *Regions*  $\mathcal{X}$

**for** A :: ('c, t, 's) pta +

— The following are necessary to obtain a *finite* MDP  
**assumes** *finite*:  $finite\ \mathcal{X}\ finite\ L\ finite\ (trans\text{-}of\ A)$   
**assumes** *not-trivial*:  $\exists l \in L. \exists u \in V. u \vdash inv\text{-}of\ A\ l$   
**assumes** *valid*:  $valid\text{-}abstraction\ A\ \mathcal{X}\ k$   
**begin**

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

## 4.1 Syntactic Definition

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

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

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

### inductive-set

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

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

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

## 4.2 Many Closure Properties

**lemma** *transition-def*:

$(A \vdash l \longrightarrow^{g, \mu, X} l') = ((l, g, \mu) \in trans\text{-}of\ A \wedge (X, l') \in \mu)$   
 $\langle proof \rangle$

**lemma** *transitionI*[*intro*]:

$A \vdash l \longrightarrow^{g, \mu, X} l'$  **if**  $(l, g, \mu) \in trans\text{-}of\ A\ (X, l') \in \mu$   
 $\langle proof \rangle$

**lemma** *transitionD*[*dest*]:

$(l, g, \mu) \in trans\text{-}of\ A\ (X, l') \in \mu$  **if**  $A \vdash l \longrightarrow^{g, \mu, X} l'$   
 $\langle proof \rangle$

**lemma** *bex-Edges*:

$(\exists x \in Edges\ A.\ P\ x) = (\exists l\ g\ \mu\ X\ l'. A \vdash l \longrightarrow^{g, \mu, X} l' \wedge P\ (l, g, \mu, X, l'))$   
 $\langle proof \rangle$

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

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

**lemma** *transition-X*:

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

**lemma** *admissible-targets-alt*:

$A \vdash l \xrightarrow{g, \mu, X} l' \implies \{\!\{g\}\!\}_X \rightarrow 0 \subseteq \{\!\{inv\text{-of } A \ l'\!\}$   
 $A \vdash l \xrightarrow{g, \mu, X} l' \implies X \subseteq \text{clocks } A$   
 $\langle \text{proof} \rangle$

**lemma** *V-reset-closed*[intro]:  
**assumes**  $u \in V$   
**shows**  $[r \rightarrow (d::nat)]u \in V$   
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *V-nn-closed*[intro]:  
 $u \in V \implies t \geq 0 \implies u \oplus t \in V$   
 $\langle \text{proof} \rangle$

**lemma** *K-S-closed*[intro]:  
**assumes**  $\mu \in K\ s\ s' \in \mu\ s \in S$   
**shows**  $s' \in S$   
 $\langle \text{proof} \rangle$

**lemma** *S-V*[intro]:  
 $(l, u) \in S \implies u \in V$   
 $\langle \text{proof} \rangle$

**lemma** *L-V*[intro]:  
 $(l, u) \in S \implies l \in L$   
 $\langle \text{proof} \rangle$

**lemma** *S-V*[intro]:  
 $(l, R) \in S \implies R \in \mathcal{R}$   
 $\langle \text{proof} \rangle$

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

### 4.3 The Region Graph is a Finite MDP

**lemma** *S-finite*:  
*finite*  $S$   
 $\langle \text{proof} \rangle$

**lemma** *K-finite*:  
*finite*  $(K\ st)$   
 $\langle \text{proof} \rangle$

**lemma**  *$\mathcal{R}$ -not-empty*:  
 $\mathcal{R} \neq \{\}$   
 $\langle \text{proof} \rangle$



**lemma** *S-not-empty*:

$\mathcal{S} \neq \{\}$   
*<proof>*

**lemma** *K-S-closed*:

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

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

*<proof>*

**lemmas** *K-S-closed*<sup>[intro]</sup> = *R-G.set-pmf-closed*

## 5 Relating the MDPs

### 5.1 Translating From K to K

**lemma** *ccompatible-inv*:

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

**lemma** *ccompatible-guard*:

**assumes**  $(l, g, \mu) \in \text{trans-of } A$   
**shows** *ccompatible*  $\mathcal{R} g$   
*<proof>*

**lemmas** *ccompatible-def* = *ccompatible-def*<sup>[unfolded ccval-def]</sup>

**lemma** *region-set'-eq*:

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

**lemma** *regions-part-ex-reset*:

**assumes**  $u \in V$   
**shows**  $[r \rightarrow (d::\text{nat})]u \in [[r \rightarrow d]u]_{\mathcal{R}} [[r \rightarrow d]u]_{\mathcal{R}} \in \mathcal{R}$   
*<proof>*

**lemma** *reset-sets-all-equiv*:

**assumes**  $u \in V u' \in [[r \rightarrow (d::\text{nat})]u]_{\mathcal{R}} x \in \text{set } r \text{ set } r \subseteq \mathcal{X} d \leq k x$   
**shows**  $u' x = d$   
*<proof>*

**lemma** *reset-eq*:

**assumes**  $u \in V ([[r \rightarrow 0]u]_{\mathcal{R}}) = ([[r' \rightarrow 0]u]_{\mathcal{R}}) \text{set } r \subseteq \mathcal{X} \text{set } r' \subseteq \mathcal{X}$   
**shows**  $[r \rightarrow 0]u = [r' \rightarrow 0]u$  *<proof>*

**lemma** *admissible-targets-clocks*:

**assumes**  $(l, g, \mu) \in \text{trans-of } A (X, l') \in \mu$   
**shows**  $X \subseteq \mathcal{X} \text{set } (\text{SOME } r. \text{set } r = X) \subseteq \mathcal{X}$   
*<proof>*

**lemma**

*rel-pmf*  $(\lambda a b. f a = b) \mu (\text{map-pmf } f \mu)$   
*<proof>*

**lemma** *K-pmf-rel*:

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

**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**  $rel\text{-}pmf (\lambda (l, u) st. (l, SOME u. u \in R) = st) \mu (map\text{-}pmf f \mu) \langle proof \rangle$

**lemma** *K-elem-abs-inj*:

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

**lemma** *K-elem-repr-inj*:

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

**lemma** *K-elem-pmf-map-abs*:

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

**lemma** *K-elem-pmf-map-repr*:

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

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

## 5.2 Translating Configurations

### 5.2.1 States

**definition**

$abss :: 's * ('c, t) cval \Rightarrow 's * ('c, t) cval set$

**where**

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

**definition**

$reps :: 's * ('c, t) cval set \Rightarrow 's * ('c, t) cval$

**where**

$reps \equiv \lambda (l, R). \text{if } R \in \mathcal{R} \text{ then } (l, SOME u. u \in R) \text{ else } (l, \lambda\text{-}. -1)$

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

**assumes**  $s \in \mathcal{S}$   
**shows**  $reps s \in \mathcal{S}$   
 $\langle proof \rangle$

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

**assumes**  $s \in \mathcal{S}$

**shows**  $abss\ s \in \mathcal{S}$   
 $\langle proof \rangle$

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

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

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

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

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

**lemma** *fst-abss*:  
 $fst\ (abss\ st) = fst\ st$  **for**  $st$   
 $\langle proof \rangle$

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

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

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

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

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

**lemma** *not-S-reps*:  
 $(l, R) \notin \mathcal{S} \implies reps\ (l, R) \notin S$   
 $\langle proof \rangle$

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

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

$$\text{abss } s \in \mathcal{S} \implies s \in S$$

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

**sublocale**  $MDP$ : Markov-Decision-Process-Invariant  $K S \langle \text{proof} \rangle$

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

**lemma**  $K$ -closed:

$$s \in S \implies (\bigcup D \in K. s. \text{set-pmf } D) \subseteq S$$

$\langle \text{proof} \rangle$

### 5.2.2 Intermezzo

**abbreviation**  $\text{timed-bisim}$  (**infixr**  $\sim 60$ ) **where**

$$s \sim s' \equiv \text{abss } s = \text{abss } s'$$

**lemma**  $\text{bisim-loc-id}$ [*intro*]:

$$(l, u) \sim (l', u') \implies l = l'$$

$\langle \text{proof} \rangle$

**lemma**  $\text{bisim-val-id}$ [*intro*]:

$$[u]_{\mathcal{R}} = [u']_{\mathcal{R}} \text{ if } u \in V \text{ } (l, u) \sim (l', u')$$

$\langle \text{proof} \rangle$

**lemma**  $\text{bisim-symmetric}$ :

$$(l, u) \sim (l', u') = (l', u') \sim (l, u)$$

$\langle \text{proof} \rangle$

**lemma**  $\text{bisim-val-id2}$ [*intro*]:

$$u' \in V \implies (l, u) \sim (l', u') \implies [u]_{\mathcal{R}} = [u']_{\mathcal{R}}$$

$\langle \text{proof} \rangle$

**lemma**  $K$ -bisim-unique:

$$\text{assumes } s \in S \text{ } \mu \in K \text{ } s \in \mu \text{ } x' \in \mu \text{ } x \sim x'$$

$$\text{shows } x = x'$$

$\langle \text{proof} \rangle$

### 5.2.3 Predicates

**definition**  $\text{absp}$  **where**

$$\text{absp } \varphi \equiv \varphi \circ \text{reps}$$

**definition**  $\text{repp}$  **where**

$$\text{repp } \varphi \equiv \varphi \circ \text{absp}$$

### 5.2.4 Distributions

**definition**

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

**where**

$$\text{abst} = \text{map-pmf } \text{abss}$$

**lemma**  $\text{abss-SD}$ :

$$\text{assumes } \text{abss } s \in \mathcal{S}$$

**obtains**  $l\ u$  **where**  $s = (l, u)\ u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}$   
 $\langle proof \rangle$

**lemma** *abss-SD'*:

**assumes**  $abss\ s \in \mathcal{S}\ abss\ s = (l, R)$

**obtains**  $u$  **where**  $s = (l, u)\ u \in [u]_{\mathcal{R}}\ [u]_{\mathcal{R}} \in \mathcal{R}\ R = [u]_{\mathcal{R}}$   
 $\langle proof \rangle$

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

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

**definition**  $delayedR\ R\ u \equiv$

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

**lemma** *delayedR-correct-aux-aux*:

**fixes**  $c :: nat$

**fixes**  $a\ b :: real$

**assumes**  $c < a\ a < Suc\ c\ b \geq 0\ a + b < Suc\ c$

**shows**  $frac\ (a + b) = frac\ a + b$

$\langle proof \rangle$

**lemma** *delayedR-correct-aux*:

**fixes**  $I\ r$

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

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

$\forall\ c \in \mathcal{X}.\ isIntv\ (I\ c) \longrightarrow (u \oplus t)\ c < intv-const\ (I\ c) + 1$   
 $t \geq 0$

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

**lemma** *delayedR-correct-aux'*:

**fixes**  $I\ r$

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

**assumes**  $u \oplus t1 \in R\ valid-region\ \mathcal{X}\ k\ I\ r\ \forall\ c \in \mathcal{X}.\ \neg\ isConst\ (I\ c)$

$\forall\ c \in \mathcal{X}.\ isIntv\ (I\ c) \longrightarrow (u \oplus t2)\ c < intv-const\ (I\ c) + 1$   
 $t1 \leq t2$

**shows**  $u \oplus t2 \in R$

$\langle proof \rangle$

**lemma** *valid-regions-intv-distinct*:

$valid-region\ X\ k\ I\ r \Longrightarrow valid-region\ X\ k\ I'\ r' \Longrightarrow u \in region\ X\ I\ r \Longrightarrow u \in region\ X\ I'\ r'$   
 $\Longrightarrow x \in X \Longrightarrow I\ x = I'\ x$

$\langle proof \rangle$

**lemma** *delayedR-correct*:

**fixes**  $I r$

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

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

**shows**

$\text{delayedR } R' u \in R'$

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

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

$\langle \text{proof} \rangle$

**definition**

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

**where**

$\text{rept } s \mu\text{-abs} \equiv \text{let } (l, u) = s \text{ in}$

$\text{if } (\exists R'. (l, u) \in S \wedge \mu\text{-abs} = \text{return-pmf } (l, R') \wedge$

$(([u]_{\mathcal{R}} = R' \wedge (\forall c \in \mathcal{X}. u c > k c))))$

$\text{then return-pmf } (l, u \oplus 0.5)$

$\text{else if}$

$(\exists R'. (l, u) \in S \wedge \mu\text{-abs} = \text{return-pmf } (l, R') \wedge R' \in \text{Succ } \mathcal{R} ([u]_{\mathcal{R}}) \wedge [u]_{\mathcal{R}} \neq R'$

$\wedge (\forall u \in R'. \forall c \in \mathcal{X}. \nexists d. d \leq k c \wedge u c = \text{real } d))$

$\text{then return-pmf } (l, \text{delayedR } (\text{SOME } R'. \mu\text{-abs} = \text{return-pmf } (l, R')) u)$

$\text{else SOME } \mu. \mu \in K s \wedge \text{abst } \mu = \mu\text{-abs}$

**lemma** *S-L*:

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

$\langle \text{proof} \rangle$

**lemma** *S-inv*:

$(l, R) \in \mathcal{S} \implies R \subseteq \{\text{inv-of } A l\}$

$\langle \text{proof} \rangle$

**lemma** *upper-right-closed*:

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

**shows**  $u \oplus t \in R$

$\langle \text{proof} \rangle$

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

$(l, u) \in S \text{ if } l \in L u \in V u \vdash \text{inv-of } A l$

$\langle \text{proof} \rangle$

**lemma** *rept-ex*:

**assumes**  $\mu \in \mathcal{K} (\text{abs } s)$

**shows**  $\text{rept } s \mu \in K s \wedge \text{abst } (\text{rept } s \mu) = \mu \langle \text{proof} \rangle$

**lemmas**  $\text{rept-K[intro]} = \text{rept-ex}[\text{THEN conjunct1}]$

**lemmas**  $\text{abst-rept-id[simp]} = \text{rept-ex}[\text{THEN conjunct2}]$

**lemma** *abst-rept2*:

**assumes**  $\mu \in \mathcal{K} s s \in \mathcal{S}$

**shows**  $\text{abst } (\text{rept } (\text{reps } s) \mu) = \mu$

$\langle \text{proof} \rangle$

**lemma** *rept-K2*:

**assumes**  $\mu \in \mathcal{K} s s \in \mathcal{S}$

**shows**  $\text{rept } (\text{reps } s) \mu \in K (\text{reps } s)$

$\langle \text{proof} \rangle$

**lemma** *theI'*:

**assumes**  $P a$

**and**  $\bigwedge x. P x \implies x = a$   
**shows**  $P (THE x. P x) \wedge (\forall y. P y \longrightarrow y = (THE x. P x))$   
 $\langle proof \rangle$

**lemma** *cont-cfg-defined*:

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

**definition**

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

### 5.2.5 Configuration

**definition**

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

**definition**

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

**definition**

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

**lemma** *S-state-absc-repc[simp]*:

$state\ cfg \in \mathcal{S} \implies state\ (absc\ (repc\ cfg)) = state\ cfg$   
 $\langle proof \rangle$

**lemma** *action-repc*:

$action\ (repc\ cfg) = rept\ (reps\ (state\ cfg))\ (action\ cfg)$   
 $\langle proof \rangle$

**lemma** *action-absc*:

$action\ (absc\ cfg) = abst\ (action\ cfg)$   
 $\langle proof \rangle$

**lemma** *action-absc'*:

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

**lemma**

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

$\langle \text{proof} \rangle$

**lemma** *cont-repcs1*:

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

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

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

$\langle \text{proof} \rangle$

**lemma** *cont-absc-1*:

**notes**  $MDP.\text{cfg-on}D\text{-state}[\text{simp del}]$

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

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

$\langle \text{proof} \rangle$

**lemma** *state-repc*:

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

$\langle \text{proof} \rangle$

**lemma** *abss-reps-id'*:

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

**assumes**  $\text{cfg} \in R\text{-}G.\text{valid-cfg} \ s \in \text{set-pmf } (\text{action } \text{cfg})$

**shows**  $\text{abss } (\text{reps } s) = s$

$\langle \text{proof} \rangle$

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

**assumes**  $P \ \text{cfg}$

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

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

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

**shows**  $\text{cfg} \in \text{valid-cfg}$

$\langle \text{proof} \rangle$

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

**assumes**  $\text{cfg} \in R\text{-}G.\text{cfg-on } s$

**shows**  $\text{state } (\text{repc } \text{cfg}) = \text{reps } s$

$\langle \text{proof} \rangle$

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

**assumes**  $\text{ccompatible } \mathcal{R} \ g \ R \in \mathcal{R} \ u \in R \ u \vdash g$

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

$\langle \text{proof} \rangle$

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

$\text{cfg} \in MDP.\text{cfg-on } s \implies \text{action } (\text{absc } \text{cfg}) \in \mathcal{K} \ (\text{abss } s)$

$\langle \text{proof} \rangle$

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

**assumes**  $\text{cfg} \in R\text{-}G.\text{valid-cfg} \ \text{abss } s = \text{state } \text{cfg}$

**shows**  $\text{repcs } s \ \text{cfg} \in \text{valid-cfg}$

$\langle \text{proof} \rangle$

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

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

**shows**  $\text{repc } \text{cfg} \in \text{valid-cfg}$

$\langle \text{proof} \rangle$

**lemma** *action-abst-repcs*:

**assumes**  $\text{cfg} \in R\text{-}G.\text{valid-cfg} \ \text{abss } s = \text{state } \text{cfg}$

**shows**  $\text{abst } (\text{action } (\text{repcs } s \ \text{cfg})) = \text{action } \text{cfg}$



$\langle \text{proof} \rangle$

**lemma** *action-abst-repc*:

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

**shows**  $\text{abst} (\text{action} (\text{repc} \text{cfg})) = \text{action} \text{cfg}$

$\langle \text{proof} \rangle$

**lemma** *state-absc*:

$\text{state} (\text{absc} \text{cfg}) = \text{abss} (\text{state} \text{cfg})$

$\langle \text{proof} \rangle$

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

$\text{state} (\text{repcs} s \text{cfg}) = s$

$\langle \text{proof} \rangle$

**lemma** *repcs-bisim*:

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

**assumes**  $\text{cfg} \in R\text{-}G.\text{valid-cfg}$   $x \in S$   $x \sim x'$   $\text{abss } x = \text{state} \text{cfg}$

**shows**  $\text{absc} (\text{repcs } x \text{cfg}) = \text{absc} (\text{repcs } x' \text{cfg})$

$\langle \text{proof} \rangle$

**named-theorems**  $R\text{-}G\text{-}I$

**lemmas**  $R\text{-}G.\text{valid-cfg-state-in-}S[R\text{-}G\text{-}I]$   $R\text{-}G.\text{valid-cfgD}[R\text{-}G\text{-}I]$   $R\text{-}G.\text{valid-cfg-action}$

**lemma** *absc-repcs-id*:

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

**assumes**  $\text{cfg} \in R\text{-}G.\text{valid-cfg}$   $\text{abss } s = \text{state} \text{cfg}$

**shows**  $\text{absc} (\text{repcs } s \text{cfg}) = \text{cfg}$   $\langle \text{proof} \rangle$

**lemma** *absc-repc-id*:

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

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

**shows**  $\text{absc} (\text{repc} \text{cfg}) = \text{cfg}$   $\langle \text{proof} \rangle$

**lemma** *K-cfg-map-absc*:

$\text{cfg} \in \text{valid-cfg} \implies K\text{-cfg} (\text{absc} \text{cfg}) = \text{map-pmf} \text{absc} (K\text{-cfg} \text{cfg})$

$\langle \text{proof} \rangle$

**lemma** *smap-comp*:

$(\text{smap } f \circ \text{smap } g) = \text{smap} (f \circ g)$

$\langle \text{proof} \rangle$

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

**assumes**  $\text{cfg} \in \text{MDP}.\text{cfg-on } s$

**shows**  $\text{state} (\text{absc} \text{cfg}) = \text{abss } s$

$\langle \text{proof} \rangle$

**lemma**  $R\text{-}G.\text{valid-cfg-coinduct}[\text{coinduct set: valid-cfg}]$ :

**assumes**  $P \text{cfg}$

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

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

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

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

$\langle \text{proof} \rangle$

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

**assumes**  $cfg \in \text{valid-cfg}$

**shows**  $\text{absc } cfg \in R\text{-}G.\text{valid-cfg}$

$\langle \text{proof} \rangle$

**lemma** *K-cfg-set-absc*:

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

**shows**  $\text{absc } cfg' \in K\text{-cfg } (\text{absc } cfg)$

$\langle \text{proof} \rangle$

**lemma** *abst-action-repcs*:

**assumes**  $cfg \in R\text{-}G.\text{valid-cfg } \text{abss } s = \text{state } cfg$

**shows**  $\text{abst } (\text{action } (\text{repcs } s \text{ } cfg)) = \text{action } cfg$

$\langle \text{proof} \rangle$

**lemma** *abst-action-repc*:

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

**shows**  $\text{abst } (\text{action } (\text{repc } cfg)) = \text{action } cfg$

$\langle \text{proof} \rangle$

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

**assumes**  $\mu \in K \ s$

**and**  $s \in S$

**shows**  $\text{inj-on } \text{abss } (\text{set-pmf } \mu)$

$\langle \text{proof} \rangle$

**lemma** *K-cfg-rept-aux*:

**assumes**  $cfg \in R\text{-}G.\text{valid-cfg } \text{abss } s = \text{state } cfg \ x \in \text{rept } s \ (\text{action } cfg)$

**defines**  $t \equiv \lambda \text{ } cfg'. \text{THE } s'. s' \in \text{rept } s \ (\text{action } cfg) \wedge s' \sim x$

**shows**  $t \text{ } cfg' = x$

$\langle \text{proof} \rangle$

**lemma** *K-cfg-rept-action*:

**assumes**  $cfg \in R\text{-}G.\text{valid-cfg } \text{abss } s = \text{state } cfg \ cfg' \in \text{set-pmf } (K\text{-cfg } cfg)$

**shows**  $\text{abss } (\text{THE } s'. s' \in \text{rept } s \ (\text{action } cfg) \wedge \text{abss } s' = \text{state } cfg') = \text{state } cfg'$

$\langle \text{proof} \rangle$

**lemma** *K-cfg-map-repcs*:

**assumes**  $cfg \in R\text{-}G.\text{valid-cfg } \text{abss } s = \text{state } cfg$

**defines**  $\text{repc}' \equiv (\lambda \text{ } cfg'. \text{repcs } (\text{THE } s'. s' \in \text{rept } s \ (\text{action } cfg) \wedge \text{abss } s' = \text{state } cfg') \text{ } cfg')$

**shows**  $K\text{-cfg } (\text{repcs } s \text{ } cfg) = \text{map-pmf } \text{repc}' \ (K\text{-cfg } cfg)$

$\langle \text{proof} \rangle$

**lemma** *K-cfg-map-repc*:

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

**defines**

$\text{repc}' \text{ } cfg' \equiv \text{repcs } (\text{THE } s. s \in \text{rept } (\text{reps } (\text{state } cfg)) \ (\text{action } cfg) \wedge \text{abss } s = \text{state } cfg') \text{ } cfg'$

**shows**

$K\text{-cfg } (\text{repc } cfg) = \text{map-pmf } \text{repc}' \ (K\text{-cfg } cfg)$

$\langle \text{proof} \rangle$

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

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

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

$\langle \text{proof} \rangle$

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

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

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

$\langle \text{proof} \rangle$

**lemma** *absc-bisim-abss*:

**assumes**  $absc\ x = absc\ x'$

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

*<proof>*

**lemma** *K-cfg-bisim-unique*:

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

**shows**  $x = x'$

*<proof>*

**lemma** *absc-distr-self*:

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

*<proof>*

**lemma** *R-G-trace-space-distr-eq*:

**assumes**  $cfg \in R\text{-}G.valid\text{-}cfg$   $abss\ s = state\ cfg$

**shows**  $MDP.MC.T\ cfg = distr\ (MDP.MC.T\ (repcs\ s\ cfg))\ MDP.MC.S\ (smap\ absc)$

*<proof>*

**lemma** *repc-inj-on-K-cfg*:

**assumes**  $cfg \in R\text{-}G.cfg\text{-}on\ s$   $s \in \mathcal{S}$

**shows**  $inj\text{-}on\ repc\ (set\text{-}pmf\ (K\text{-}cfg\ cfg))$

*<proof>*

**lemma** *smap-absc-iff*:

**assumes**  $\bigwedge x\ y.\ x \in X \implies smap\ abss\ x = smap\ abss\ y \implies y \in X$

**shows**  $(smap\ state\ xs \in X) = (smap\ (\lambda z.\ abss\ (state\ z))\ xs \in smap\ abss\ `X)$

*<proof>*

**lemma** *valid-abss-reps[simp]*:

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

**shows**  $abss\ (reps\ (state\ cfg)) = state\ cfg$

*<proof>*

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

*<proof>*

**lemma** *S-reps-S-aux*:

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

*<proof>*

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

$reps\ s \in \mathcal{S} \implies s \in \mathcal{S}$

*<proof>*

**lemma** *absc-valid-cfg-eq*:

$absc\ `valid\text{-}cfg = R\text{-}G.valid\text{-}cfg$

*<proof>*

**lemma** *action-repcs*:

$action\ (repcs\ (l,\ u)\ cfg) = rept\ (l,\ u)\ (action\ cfg)$

*<proof>*

### 5.3 Equalities Between Measures of Trace Spaces

**lemma** *path-measure-eq-absc1-new*:

**fixes**  $cfg\ s$

**defines**  $cfg' \equiv absc\ cfg$

**assumes** *valid*:  $cfg \in \text{valid-cfg}$   
**assumes**  $X[\text{measurable}]$ :  $X \in R\text{-}G.\text{St}$  **and**  $Y[\text{measurable}]$ :  $Y \in \text{MDP}.\text{St}$   
**assumes**  $P$ : *AE*  $x$  in  $(R\text{-}G.T\text{ cfg}')$ .  $P\ x$  **and**  $Q$ : *AE*  $x$  in  $(\text{MDP}.\text{T}\text{ cfg})$ .  $Q\ x$   
**assumes**  $P'[\text{measurable}]$ : *Measurable.pred*  $R\text{-}G.\text{St}\ P$   
**and**  $Q'[\text{measurable}]$ : *Measurable.pred*  $\text{MDP}.\text{St}\ Q$   
**assumes**  $X\text{-}Y\text{-closed}$ :  $\bigwedge x\ y. P\ x \implies \text{smap}\ \text{abss}\ y = x \implies x \in X \implies y \in Y \wedge Q\ y$   
**assumes**  $Y\text{-}X\text{-closed}$ :  $\bigwedge x\ y. Q\ y \implies \text{smap}\ \text{abss}\ y = x \implies y \in Y \implies x \in X \wedge P\ x$   
**shows**  
 $\text{emeasure}\ (R\text{-}G.\text{T}\text{ cfg})\ X = \text{emeasure}\ (\text{MDP}.\text{T}\text{ cfg})\ Y$   
 $\langle \text{proof} \rangle$

**lemma** *path-measure-eq-repcs1-new*:

**fixes**  $cfg\ s$   
**defines**  $cfg' \equiv \text{repcs}\ s\ cfg$   
**assumes**  $s$ : *abss*  $s = \text{state}\ cfg$   
**assumes** *valid*:  $cfg \in R\text{-}G.\text{valid-cfg}$   
**assumes**  $X[\text{measurable}]$ :  $X \in R\text{-}G.\text{St}$  **and**  $Y[\text{measurable}]$ :  $Y \in \text{MDP}.\text{St}$   
**assumes**  $P$ : *AE*  $x$  in  $(R\text{-}G.T\text{ cfg})$ .  $P\ x$  **and**  $Q$ : *AE*  $x$  in  $(\text{MDP}.\text{T}\text{ cfg}')$ .  $Q\ x$   
**assumes**  $P'[\text{measurable}]$ : *Measurable.pred*  $R\text{-}G.\text{St}\ P$   
**and**  $Q'[\text{measurable}]$ : *Measurable.pred*  $\text{MDP}.\text{St}\ Q$   
**assumes**  $X\text{-}Y\text{-closed}$ :  $\bigwedge x\ y. P\ x \implies \text{smap}\ \text{abss}\ y = x \implies x \in X \implies y \in Y \wedge Q\ y$   
**assumes**  $Y\text{-}X\text{-closed}$ :  $\bigwedge x\ y. Q\ y \implies \text{smap}\ \text{abss}\ y = x \implies y \in Y \implies x \in X \wedge P\ x$   
**shows**  
 $\text{emeasure}\ (R\text{-}G.\text{T}\text{ cfg})\ X = \text{emeasure}\ (\text{MDP}.\text{T}\text{ cfg}')$   $Y$   
 $\langle \text{proof} \rangle$

**lemma** *region-compatible-suntil1*:

**assumes**  $(\text{holds}\ (\lambda x. \varphi\ (\text{reps}\ x))\ \text{suntil}\ \text{holds}\ (\lambda x. \psi\ (\text{reps}\ x)))$   $(\text{smap}\ \text{abss}\ x)$   
**and** *pred-stream*  $(\lambda s. \varphi\ (\text{reps}\ (\text{abss}\ s)) \longrightarrow \varphi\ s)\ x$   
**and** *pred-stream*  $(\lambda s. \psi\ (\text{reps}\ (\text{abss}\ s)) \longrightarrow \psi\ s)\ x$   
**shows**  $(\text{holds}\ \varphi\ \text{suntil}\ \text{holds}\ \psi)\ x$   $\langle \text{proof} \rangle$

**lemma** *region-compatible-suntil2*:

**assumes**  $(\text{holds}\ \varphi\ \text{suntil}\ \text{holds}\ \psi)\ x$   
**and** *pred-stream*  $(\lambda s. \varphi\ s \longrightarrow \varphi\ (\text{reps}\ (\text{abss}\ s)))\ x$   
**and** *pred-stream*  $(\lambda s. \psi\ s \longrightarrow \psi\ (\text{reps}\ (\text{abss}\ s)))\ x$   
**shows**  $(\text{holds}\ (\lambda x. \varphi\ (\text{reps}\ x))\ \text{suntil}\ \text{holds}\ (\lambda x. \psi\ (\text{reps}\ x)))$   $(\text{smap}\ \text{abss}\ x)$   $\langle \text{proof} \rangle$

**lemma** *region-compatible-suntil*:

**assumes** *pred-stream*  $(\lambda s. \varphi\ (\text{reps}\ (\text{abss}\ s)) \longleftrightarrow \varphi\ s)\ x$   
**and** *pred-stream*  $(\lambda s. \psi\ (\text{reps}\ (\text{abss}\ s)) \longleftrightarrow \psi\ s)\ x$   
**shows**  $(\text{holds}\ (\lambda x. \varphi\ (\text{reps}\ x))\ \text{suntil}\ \text{holds}\ (\lambda x. \psi\ (\text{reps}\ x)))$   $(\text{smap}\ \text{abss}\ x)$   
 $\longleftrightarrow$   $(\text{holds}\ \varphi\ \text{suntil}\ \text{holds}\ \psi)\ x$   $\langle \text{proof} \rangle$

**lemma** *reps-abss-S*:

**assumes**  $\text{reps}\ (\text{abss}\ s) \in S$   
**shows**  $s \in S$   
 $\langle \text{proof} \rangle$

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

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

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

**notes** *in-space-UNIV[measurable]*  
**fixes**  $cfg\ \varphi\ \psi\ s$   
**defines**  $cfg' \equiv \text{repcs}\ s\ cfg$   
**defines**  $\varphi' \equiv \text{absp}\ \varphi$  **and**  $\psi' \equiv \text{absp}\ \psi$   
**assumes**  $s$ : *abss*  $s = \text{state}\ cfg$

**assumes** *valid*:  $cfg \in R\text{-}G.\text{valid}\text{-}cfg$   
**assumes** *valid'*:  $cfg' \in \text{valid}\text{-}cfg$   
**assumes** *equiv-φ*:  $\bigwedge x. \text{pred}\text{-}stream (\lambda s. s \in S) x$   
 $\implies \text{pred}\text{-}stream (\lambda s. \varphi (\text{reps } (abss\ s))) \longleftrightarrow \varphi\ s$  (*state*  $cfg'$   $\#\#\ x$ )  
**and** *equiv-ψ*:  $\bigwedge x. \text{pred}\text{-}stream (\lambda s. s \in S) x$   
 $\implies \text{pred}\text{-}stream (\lambda s. \psi (\text{reps } (abss\ s))) \longleftrightarrow \psi\ s$  (*state*  $cfg'$   $\#\#\ x$ )  
**shows**  
 $\text{emeasure } (R\text{-}G.T\ cfg) \{x \in \text{space } R\text{-}G.St. (\text{holds } \varphi' \text{ until holds } \psi') (\text{state } cfg\ \#\#\ x)\} =$   
 $\text{emeasure } (MDP.T\ cfg') \{x \in \text{space } MDP.St. (\text{holds } \varphi \text{ until holds } \psi) (\text{state } cfg'\ \#\#\ x)\}$   
 $\langle \text{proof} \rangle$

**end**

**end**

**theory** *PTA-Reachability*

**imports** *PTA*

**begin**

## 6 Classifying Regions for Divergence

### 6.1 Pairwise

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

**lemma** *pairwise-Suc*:

$\text{pairwise } P\ xs \implies P\ (xs\ !!\ i)\ (xs\ !!\ (\text{Suc } i))$   
 $\langle \text{proof} \rangle$

**lemma** *Suc-pairwise*:

$\forall i. P\ (xs\ !!\ i)\ (xs\ !!\ (\text{Suc } i)) \implies \text{pairwise } P\ xs$   
 $\langle \text{proof} \rangle$

**lemma** *pairwise-iff*:

$\text{pairwise } P\ xs \longleftrightarrow (\forall i. P\ (xs\ !!\ i)\ (xs\ !!\ (\text{Suc } i)))$   
 $\langle \text{proof} \rangle$

**lemma** *pairwise-stlD*:

$\text{pairwise } P\ xs \implies \text{pairwise } P\ (\text{stl } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *pairwise-pairD*:

$\text{pairwise } P\ xs \implies P\ (\text{shd } xs)\ (\text{shd } (\text{stl } xs))$   
 $\langle \text{proof} \rangle$

**lemma** *pairwise-mp*:

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

**lemma** *pairwise-sdropD*:

$\text{pairwise } P\ (\text{sdrop } i\ xs)$  **if**  $\text{pairwise } P\ xs$   
 $\langle \text{proof} \rangle$

### 6.2 Regions

**lemma** *gt-GreaterD*:

**assumes**  $u \in \text{region } X\ I\ r\ \text{valid}\text{-}\text{region } X\ k\ I\ r\ c \in X\ u\ c > k\ c$   
**shows**  $I\ c = \text{Greater } (k\ c)$   
 $\langle \text{proof} \rangle$

**lemma** *const-ConstD*:

**assumes**  $u \in \text{region } X \text{ I r valid-region } X k \text{ I r } c \in X u c = d d \leq k c$   
**shows**  $I c = \text{Const } d$

$\langle \text{proof} \rangle$

**lemma** *not-Greater-bounded*:

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

$\langle \text{proof} \rangle$

**lemma** *Greater-closed*:

**fixes**  $t :: \text{real}$

**assumes**  $u \in \text{region } X \text{ I r valid-region } X k \text{ I r } c \in X I c = \text{Greater } (k c) t > k c$

**shows**  $u(c := t) \in \text{region } X \text{ I r}$

$\langle \text{proof} \rangle$

**lemma** *Greater-unbounded-aux*:

**assumes**  $\text{finite } X \text{ valid-region } X k \text{ I r } c \in X I c = \text{Greater } (k c)$

**shows**  $\exists u \in \text{region } X \text{ I r}. u c > t$

$\langle \text{proof} \rangle$

## 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**  $\text{finite } X \text{ valid-region } X k \text{ I r } c \in X I c = \text{Greater } (k c)$

**shows**  $\text{unbounded } c (\text{region } X \text{ I r})$

$\langle \text{proof} \rangle$

**lemma** *unbounded-Greater*:

**assumes**  $\text{valid-region } X k \text{ I r } c \in X \text{ unbounded } c (\text{region } X \text{ I r})$

**shows**  $I c = \text{Greater } (k c)$

$\langle \text{proof} \rangle$

**lemma** *Const-zero*:

**assumes**  $c \in X I c = \text{Const } 0$

**shows**  $\text{zero } c (\text{region } X \text{ I r})$

$\langle \text{proof} \rangle$

**lemma** *zero-Const*:

**assumes**  $\text{finite } X \text{ valid-region } X k \text{ I r } c \in X \text{ zero } c (\text{region } X \text{ I r})$

**shows**  $I c = \text{Const } 0$

$\langle \text{proof} \rangle$

**lemma** *zero-all*:

**assumes**  $\text{finite } X \text{ valid-region } X k \text{ I r } c \in X u \in \text{region } X \text{ I r } u c = 0$

**shows**  $\text{zero } c (\text{region } X \text{ I r})$

$\langle \text{proof} \rangle$

## 7 Reachability

### 7.1 Definitions

**locale** *Probabilistic-Timed-Automaton-Regions-Reachability* =

*Probabilistic-Timed-Automaton-Regions*  $k v n \text{ not-in-} X A$

**for**  $k v n \text{ not-in-} X$  **and**  $A :: ('c, t, 's) \text{ pta} +$

**fixes**  $\varphi \psi :: ('s * ('c, t) \text{ cval}) \Rightarrow \text{bool}$  **fixes**  $s$

**assumes**  $\varphi: \bigwedge x y. x \in S \implies x \sim y \implies \varphi x \longleftrightarrow \varphi y$   
**assumes**  $\psi: \bigwedge x y. x \in S \implies x \sim y \implies \psi x \longleftrightarrow \psi y$   
**assumes**  $s[\text{intro}, \text{simp}]: s \in S$   
**begin**

**definition**  $\varphi' \equiv \text{absp } \varphi$   
**definition**  $\psi' \equiv \text{absp } \psi$   
**definition**  $s' \equiv \text{abss } s$

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

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

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

**lemma** (in *Probabilistic-Timed-Automaton-Regions*) *compatible-stream*:  
**assumes**  $\varphi: \bigwedge x y. x \in S \implies x \sim y \implies \varphi x \longleftrightarrow \varphi y$   
**assumes**  $\text{pred-stream } (\lambda s. s \in S) xs$   
**and**  $[\text{intro}]: x \in S$   
**shows**  $\text{pred-stream } (\lambda s. \varphi (\text{reps } (\text{abss } s)) = \varphi s) (x \#\# xs)$   
 $\langle \text{proof} \rangle$

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

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

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

## 7.2 Easier Result on All Configurations

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

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

**theorem**  $P\text{-sup-suntil-eq}$ :

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

**end**

### 7.3 Divergent Adversaries

**context** *Probabilistic-Timed-Automaton*  
**begin**

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

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

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

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

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

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

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

**end**

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

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

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

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

**lemma**  $elapsed\text{-}ge0[simp]: elapsed\ x\ y \geq 0$   
 $\langle proof \rangle$

**lemma**  $dur\text{-}pos:$   
 $dur\ xs\ i \geq 0$   
 $\langle proof \rangle$

**lemma**  $dur\text{-}mono:$   
 $i \leq j \Longrightarrow dur\ xs\ i \leq dur\ xs\ j$   
 $\langle proof \rangle$

**lemma**  $dur\text{-}monoD:$   
**assumes**  $dur\ xs\ i < dur\ xs\ j$   
**shows**  $i < j$   $\langle proof \rangle$

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



$\langle \text{proof} \rangle$

**lemma** *elapsed-ge*:

**assumes**  $\text{eq-elapsed } u \ u' \ c \in \mathcal{X}$   
**shows**  $\text{elapsed } u \ u' \geq u' \ c - u \ c$   
 $\langle \text{proof} \rangle$

**lemma** *elapsed-eq*:

**assumes**  $\text{eq-elapsed } u \ u' \ c \in \mathcal{X} \ u' \ c - u \ c \geq 0$   
**shows**  $\text{elapsed } u \ u' = u' \ c - u \ c$   
 $\langle \text{proof} \rangle$

**lemma** *dur-shift*:

$\text{dur } \omega \ (i + j) = \text{dur } \omega \ i + \text{dur} \ (\text{sdrop } i \ \omega) \ j$   
 $\langle \text{proof} \rangle$

**lemma** *dur-zero*:

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

**lemma** *dur-zero-tail*:

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

**lemma** *elapsed-ge-pos*:

**fixes**  $u :: ('c, t) \text{ cval}$   
**assumes**  $\text{eq-elapsed } u \ u' \ c \in \mathcal{X} \ u \in V \ u' \in V$   
**shows**  $\text{elapsed } u \ u' \leq u' \ c$   
 $\langle \text{proof} \rangle$

**lemma** *dur-Suc*:

$\text{dur } \text{xs} \ (\text{Suc } i) - \text{dur } \text{xs} \ i = \text{elapsed} \ (\text{xs} \ !! \ i) \ (\text{xs} \ !! \ \text{Suc } i)$   
 $\langle \text{proof} \rangle$

**inductive trans where**

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

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

**lemma** *K-cfg-trans*:

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

**lemma** *enabled-stream-trans*:

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

**lemma** *stream-trans-trans*:

**assumes**  $\text{stream-trans } \text{xs}$   
**shows**  $\text{trans} \ (\text{xs} \ !! \ i) \ (\text{stl } \text{xs} \ !! \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *trans-eq-elapsed*:

**assumes**  $\text{trans } u \ u' \ u \in V$   
**shows**  $\text{eq-elapsed } u \ u'$   
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

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

**lemma** *not-const-intv*:  
**assumes**  $u \in V \ \forall c \in \mathcal{X}. \nexists d. d \leq k \ c \wedge u \ c = \text{real } d$

**shows**  $\forall c \in \mathcal{X}. \forall u \in [u]_{\mathcal{R}}. \nexists d. d \leq k c \wedge u c = \text{real } d$   
 ⟨proof⟩

**lemma** *K-cfg-trans'*:

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

**coinductive** *enabled-repcs where*

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

**lemma** *K-cfg-rept-in*:

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

**lemma** *enabled-repcsI*:

**assumes**  $\text{cfg} \in \text{R-G.valid-cfg } \text{abss } st = \text{state } \text{cfg } \text{MDP.MC.enabled } (\text{repcs } st \text{ cfg}) xs$   
**shows**  $\text{enabled-repcs } (\text{repcs } st \text{ cfg}) xs$  ⟨proof⟩

**lemma** *repcs-eq-rept*:

$\text{rept } st \text{ (action } \text{cfg}) = \text{rept } st'' \text{ (action } \text{cfg}'')$  **if**  $\text{repcs } st \text{ cfg} = \text{repcs } st'' \text{ cfg}''$   
 ⟨proof⟩

**lemma** *enabled-stream-trans'*:

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

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

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

**lemma** *(in -)*

**fixes**  $f :: \text{nat} \Rightarrow \text{real}$   
**assumes**  $\forall i. f i \geq 0 \forall i. \exists j \geq i. f j > d \ d > 0$   
**shows**  $\exists n. (\sum_{i \leq n} f i) > t$   
 ⟨proof⟩

**lemma** *dur-ev-exceedsI*:

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

**lemma** *not-reset-mono*:

**assumes**  $\text{stream-trans } xs \text{ shd } xs \ c1 \geq \text{shd } xs \ c2 \text{ stream-all } (\lambda u. u \in V) xs \ c2 \in \mathcal{X}$   
**shows**  $(\text{holds } (\lambda u. u \ c1 \geq u \ c2) \text{ until holds } (\lambda u. u \ c1 = 0)) xs$  ⟨proof⟩

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

**fixes**  $xs :: ('c, t) \text{ cval stream}$   
**assumes**  $\text{stream-trans } xs \text{ stream-all } (\lambda u. u \in V) xs$   
 $(xs !! i) c1 = 0 \exists k > i. k \leq j \wedge (xs !! k) c2 = 0$   
 $\forall k > i. k \leq j \longrightarrow (xs !! k) c1 \neq 0$   
 $c1 \in \mathcal{X} \ c2 \in \mathcal{X}$   
**shows**  $(xs !! j) c1 \geq (xs !! j) c2$   
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

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

**definition**

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

**lemma** K-cfg-same-loc-iff:

( $\forall$  cfg'  $\in$  K-cfg cfg. fst (state cfg') = fst (state cfg))  
 $\longleftrightarrow$  ( $\forall$  cfg'  $\in$  K-cfg (absc cfg). fst (state cfg') = fst (state (absc cfg)))  
 if cfg  $\in$  valid-cfg  
 <proof>

**lemma** (in -) stream-all2-flip:

stream-all2 ( $\lambda$ a b. R b a) xs ys = stream-all2 R ys xs  
 <proof>

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

assumes cfg  $\in$  valid-cfg  
 shows alternating cfg  $\longleftrightarrow$  alternating (absc cfg)  
 <proof>

**lemma** AE-alw-ev-same-loc-iff':

assumes cfg  $\in$  R-G.cfg-on (abss st) st  $\in$  S  
 shows alternating cfg  $\longleftrightarrow$  alternating (repcs st cfg)  
 <proof>

**lemma** (in -) cval-add-non-id:

False if  $b \oplus d = b d > 0$  for  $d :: \text{real}$   
 <proof>

**lemma** repcs-unbounded-AE-non-loop-end-strong:

assumes cfg  $\in$  R-G.cfg-on (abss st) st  $\in$  S  
 and alternating cfg  
 shows AE  $\omega$  in MDP.MC.T (repcs st cfg).  
 ( $\forall$  u :: ('c  $\Rightarrow$  real). ( $\forall$  c  $\in$   $\mathcal{X}$ . u c > real (k c))  $\longrightarrow$   
 $\neg$  (ev (alw ( $\lambda$  xs. shd xs = u))) (smap (snd o state)  $\omega$ )) (is AE  $\omega$  in ?M. ?P  $\omega$ )  
 <proof>

**lemma** cfg-on-div-repcs-strong:

notes in-space-UNIV[measurable]  
 assumes cfg  $\in$  R-G.cfg-on-div (abss st) st  $\in$  S and alternating cfg  
 shows repcs st cfg  $\in$  cfg-on-div st  
 <proof>

**lemma** repcs-unbounded-AE-non-loop-end:

assumes cfg  $\in$  R-G.cfg-on (abss st) st  $\in$  S  
 shows AE  $\omega$  in MDP.MC.T (repcs st cfg).  
 ( $\forall$  s :: ('s  $\times$  ('c  $\Rightarrow$  real)). ( $\forall$  c  $\in$   $\mathcal{X}$ . snd s c > k c)  $\longrightarrow$   
 $\neg$  (ev (alw ( $\lambda$  xs. shd xs = s))) (smap state  $\omega$ )) (is AE  $\omega$  in ?M. ?P  $\omega$ )  
 <proof>

end

## 7.4 Main Result

context Probabilistic-Timed-Automaton-Regions-Reachability

begin

**lemma** R-G.cfg-on-valid:

cfg  $\in$  R-G.valid-cfg if cfg  $\in$  R-G.cfg-on-div s'  
 <proof>

**lemma** *cfg-on-valid*:

*cfg* ∈ *valid-cfg* **if** *cfg* ∈ *cfg-on-div* *s*

⟨*proof*⟩

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

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

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

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

Summary of our results on divergent configurations:

**lemma** *absc-valid-cfg-eq*:

*absc* ‘*progressive* *s* = *R-G-progressive* *s*’

⟨*proof*⟩

Main theorem:

**theorem** *Min-Max-reachability*:

**notes** *in-space-UNIV*[*measurable*] **and** [*iff*] = *pred-stream-iff*

**shows**

( $\bigwedge$  *cfg* ∈ *progressive* *s.* *path-measure* ( $\lambda$  *x.* (*holds*  $\varphi$  *suntil* *holds*  $\psi$ ) (*s* ## *x*)) *cfg*)  
= ( $\bigwedge$  *cfg* ∈ *R-G-progressive* *s*’. *R-G-path-measure* ( $\lambda$  *x.* (*holds*  $\varphi$ ’ *suntil* *holds*  $\psi$ ’) (*s*’ ## *x*)) *cfg*)  
∧ ( $\bigwedge$  *cfg* ∈ *progressive* *s.* *path-measure* ( $\lambda$  *x.* (*holds*  $\varphi$  *suntil* *holds*  $\psi$ ) (*s* ## *x*)) *cfg*)  
= ( $\bigwedge$  *cfg* ∈ *R-G-progressive* *s*’. *R-G-path-measure* ( $\lambda$  *x.* (*holds*  $\varphi$ ’ *suntil* *holds*  $\psi$ ’) (*s*’ ## *x*)) *cfg*)  
⟨*proof*⟩

**end**

**end**

## References

- [1] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Th. Comp. Sci.*, 282(1).
- [2] S. Wimmer and J. Hölzl. MDP + TA = PTA: Probabilistic timed automata, formalized. In J. Avigad and A. Mahboubi, editors, *ITP 2018, Proceedings*, Lecture Notes in Computer Science. Springer, 2018.