

Timed Automata

Simon Wimmer

September 30, 2020

Abstract

Timed automata are a widely used formalism for modeling real-time systems, which is employed in a class of successful model checkers such as UPPAAL [LPY97], HyTech [HHWt97] or Kronos [Yov97]. This work formalizes the theory for the subclass of diagonal-free timed automata, which is sufficient to model many interesting problems. We first define the basic concepts and semantics of diagonal-free timed automata. Based on this, we prove two types of decidability results for the language emptiness problem.

The first is the classic result of Alur and Dill [AD90, AD94], which uses a finite partitioning of the state space into so-called *regions*.

Our second result focuses on an approach based on *Difference Bound Matrices (DBMs)*, which is practically used by model checkers. We prove the correctness of the basic forward analysis operations on DBMs. One of these operations is the Floyd-Warshall algorithm for the all-pairs shortest paths problem. To obtain a finite search space, a widening operation has to be used for this kind of analysis. We use Patricia Bouyer's [Bou04] approach to prove that this widening operation is correct in the sense that DBM-based forward analysis in combination with the widening operation also decides language emptiness. The interesting property of this proof is that the first decidability result is reused to obtain the second one.

Contents

1	Floyd-Warshall Algorithm for the All-Pairs Shortest Paths Problem	4
1.1	Cycles in Lists	4
1.2	Definition of the Algorithm	7
1.3	Definition of Shortest Paths	12
1.4	Result Under The Absence of Negative Cycles	14
1.5	Result Under the Presence of Negative Cycles	14
2	Basic Definitions and Semantics	15
2.1	Time	15

2.2	Syntactic Definition	16
2.3	Operational Semantics	18
2.4	Zone Semantics	20
3	Difference Bound Matrices	21
3.1	Definitions	21
4	Library for Paths, Arcs and Lengths	24
4.1	Arcs	25
4.2	Length of Paths	25
4.3	Canonical Matrices	26
4.4	Cycle Rotation	26
4.5	Equivalent Characterizations of Cycle-Freeness	27
4.6	More Theorems Related to Floyd-Warshall	27
4.7	Helper Lemmas for Bouyer’s Theorem on Approximation	29
4.8	The Characteristic Property of Canonical DBMs	39
5	Forward Analysis on DBMs	42
5.1	Auxiliary	42
5.2	Time Lapse	42
5.3	From Clock Constraints to DBMs	43
5.4	Zone Intersection	45
5.5	Clock Reset	45
5.6	Misc Preservation Lemmas	50
5.7	Normalization of DBMs	56
5.8	Normalization is a Widening Operator	56
6	Refinement to β-regions	57
6.1	Definition	57
6.2	Basic Properties	59
6.3	Approximation with β -regions	62
6.4	Computing β -Approximation	64
6.5	Auxiliary β -boundedness Theorems	67
7	The Classic Construction for Decidability	69
7.1	Definition of Regions	70
7.2	Basic Properties	71
7.3	Set of Regions	74
7.4	Compatibility With Clock Constraints	77
7.5	Compatibility with Resets	78
7.6	A Semantics Based on Regions	79
7.7	Correct Approximation of Zones with α -regions	82
7.8	A New Zone Semantics Abstracting with $Closure_\alpha$	84

8	Correctness of β-approximation from α-regions	88
8.1	Preparing Bouyer's Theorem	88
8.2	Bouyer's Main Theorem	92
8.3	Nice Corollaries of Bouyer's Theorem	92
8.4	A New Zone Semantics Abstracting with <i>Approx</i> $_{\beta}$	93
9	Forward Analysis with DBMs and Widening	95
9.1	DBM-based Semantics with Normalization	95
9.2	The Final Result About Language Emptiness	100
9.3	Finiteness of the Search Space	101
9.4	Appendix: Standard Clock Numberings for Concrete Models	101

```

theory Floyd-Warshall
  imports Main
begin

```

1 Floyd-Warshall Algorithm for the All-Pairs Shortest Paths Problem

Auxiliary lemma *distinct-list-single-elem-decomp*: $\{xs. \text{set } xs \subseteq \{0\} \wedge \text{distinct } xs\} = \{\[], [0]\}$
 $\langle \text{proof} \rangle$

1.1 Cycles in Lists

abbreviation *cnt* x $xs \equiv \text{length } (\text{filter } (\lambda y. x = y) xs)$

fun *remove-cycles* :: 'a list \Rightarrow 'a \Rightarrow 'a list \Rightarrow 'a list

where

remove-cycles [] - *acc* = *rev acc* |

remove-cycles ($x\#xs$) *y acc* =

(if $x = y$ then *remove-cycles* xs *y* [x] else *remove-cycles* xs *y* ($x\#acc$))

lemma *cnt-rev*: $\text{cnt } x (\text{rev } xs) = \text{cnt } x xs$ $\langle \text{proof} \rangle$

value *as* @ [x] @ *bs* @ [x] @ *cs* @ [x] @ *ds*

lemma *remove-cycles-removes*: $\text{cnt } x (\text{remove-cycles } xs \ x \ ys) \leq \max 1 (\text{cnt } x \ ys)$

$\langle \text{proof} \rangle$

lemma *remove-cycles-id*: $x \notin \text{set } xs \implies \text{remove-cycles } xs \ x \ ys = \text{rev } ys$ @ xs

$\langle \text{proof} \rangle$

lemma *remove-cycles-cnt-id*:

$x \neq y \implies \text{cnt } y (\text{remove-cycles } xs \ x \ ys) \leq \text{cnt } y \ ys + \text{cnt } y \ xs$

$\langle \text{proof} \rangle$

lemma *remove-cycles-ends-cycle*: $\text{remove-cycles } xs \ x \ ys \neq \text{rev } ys$ @ $xs \implies x \in \text{set } xs$

$\langle \text{proof} \rangle$

lemma *remove-cycles-begins-with*: $x \in \text{set } xs \implies \exists zs. \text{remove-cycles } xs \ x \ ys = x \# zs \wedge x \notin \text{set } zs$

$\langle \text{proof} \rangle$

lemma *remove-cycles-self*:

$x \in \text{set } xs \implies \text{remove-cycles } (\text{remove-cycles } xs \ x \ ys) \ x \ zs = \text{remove-cycles } xs \ x \ ys$

$\langle \text{proof} \rangle$

lemma *remove-cycles-one*: $\text{remove-cycles } (as \ @ \ x \ \# \ xs) \ x \ ys = \text{remove-cycles } (x \ \# \ xs) \ x \ ys$

$\langle \text{proof} \rangle$

lemma *remove-cycles-cycles*:

$x \in \text{set } xs \implies \exists \ xxs \ as. \ as \ @ \ \text{concat } (\text{map } (\lambda \ xs. \ x \ \# \ xs) \ xxs) \ @ \ \text{remove-cycles } xs \ x \ ys = xs \ \wedge \ x \notin \text{set } as$

$\langle \text{proof} \rangle$

fun *start-remove* :: $'a \ \text{list} \Rightarrow 'a \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{list}$

where

$\text{start-remove } [] \ - \ \text{acc} = \text{rev } \text{acc} \ |$

$\text{start-remove } (x \ \# \ xs) \ y \ \text{acc} =$

$(\text{if } x = y \ \text{then } \text{rev } \text{acc} \ @ \ \text{remove-cycles } xs \ y \ [y] \ \text{else } \text{start-remove } xs \ y \ (x \ \# \ \text{acc}))$

lemma *start-remove-decomp*:

$x \in \text{set } xs \implies \exists \ as \ bs. \ xs = as \ @ \ x \ \# \ bs \ \wedge \ \text{start-remove } xs \ x \ ys = \text{rev } ys \ @ \ as \ @ \ \text{remove-cycles } bs \ x \ [x]$

$\langle \text{proof} \rangle$

lemma *start-remove-removes*: $\text{cnt } x \ (\text{start-remove } xs \ x \ ys) \leq \text{Suc } (\text{cnt } x \ ys)$

$\langle \text{proof} \rangle$

lemma *start-remove-id[simp]*: $x \notin \text{set } xs \implies \text{start-remove } xs \ x \ ys = \text{rev } ys \ @ \ xs$

$\langle \text{proof} \rangle$

lemma *start-remove-cnt-id*:

$x \neq y \implies \text{cnt } y \ (\text{start-remove } xs \ x \ ys) \leq \text{cnt } y \ ys + \text{cnt } y \ xs$

$\langle \text{proof} \rangle$

fun *remove-all-cycles* :: $'a \ \text{list} \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{list}$

where

$\text{remove-all-cycles } [] \ xs = xs \ |$

$\text{remove-all-cycles } (x \ \# \ xs) \ ys = \text{remove-all-cycles } xs \ (\text{start-remove } ys \ x)$

[])

lemma *cnt-remove-all-mono*: $\text{cnt } y \text{ (remove-all-cycles } xs \text{ } ys) \leq \max 1 \text{ (cnt } y \text{ } ys)$
<proof>

lemma *cnt-remove-all-cycles*: $x \in \text{set } xs \implies \text{cnt } x \text{ (remove-all-cycles } xs \text{ } ys) \leq 1$
<proof>

lemma *cnt-mono*:
 $\text{cnt } a \text{ (} b \# xs) \leq \text{cnt } a \text{ (} b \# c \# xs)$
<proof>

lemma *cnt-distinct-intro*: $\forall x \in \text{set } xs. \text{cnt } x \text{ } xs \leq 1 \implies \text{distinct } xs$
<proof>

lemma *remove-cycles-subs*:
 $\text{set (remove-cycles } xs \text{ } x \text{ } ys) \subseteq \text{set } xs \cup \text{set } ys$
<proof>

lemma *start-remove-subs*:
 $\text{set (start-remove } xs \text{ } x \text{ } ys) \subseteq \text{set } xs \cup \text{set } ys$
<proof>

lemma *remove-all-cycles-subs*:
 $\text{set (remove-all-cycles } xs \text{ } ys) \subseteq \text{set } ys$
<proof>

lemma *remove-all-cycles-distinct*: $\text{set } ys \subseteq \text{set } xs \implies \text{distinct (remove-all-cycles } xs \text{ } ys)$
<proof>

lemma *distinct-remove-cycles-inv*: $\text{distinct (} xs \text{ @ } ys) \implies \text{distinct (remove-cycles } xs \text{ } x \text{ } ys)$
<proof>

definition *remove-all* $x \text{ } xs = (\text{if } x \in \text{set } xs \text{ then } tl \text{ (remove-cycles } xs \text{ } x \text{ [])} \text{ else } xs)$

definition *remove-all-rev* $x \text{ } xs = (\text{if } x \in \text{set } xs \text{ then } rev \text{ (} tl \text{ (remove-cycles (rev } xs) \text{ } x \text{ []))} \text{ else } xs)$

lemma *remove-all-distinct*:

$distinct\ xs \implies distinct\ (x\ \# \text{remove-all}\ x\ xs)$

<proof>

lemma *remove-all-removes*:

$x \notin set\ (\text{remove-all}\ x\ xs)$

<proof>

lemma *remove-all-subsets*:

$set\ (\text{remove-all}\ x\ xs) \subseteq set\ xs$

<proof>

lemma *remove-all-rev-distinct*: $distinct\ xs \implies distinct\ (x\ \# \text{remove-all-rev}\ x\ xs)$

<proof>

lemma *remove-all-rev-removes*: $x \notin set\ (\text{remove-all-rev}\ x\ xs)$

<proof>

lemma *remove-all-rev-subsets*: $set\ (\text{remove-all-rev}\ x\ xs) \subseteq set\ xs$

<proof>

abbreviation *rem-cycles* $i\ j\ xs \equiv \text{remove-all}\ i\ (\text{remove-all-rev}\ j\ (\text{remove-all-cycles}\ xs\ xs))$

lemma *rem-cycles-distinct'*: $i \neq j \implies distinct\ (i\ \# j\ \# \text{rem-cycles}\ i\ j\ xs)$

<proof>

lemma *rem-cycles-removes-last*: $j \notin set\ (\text{rem-cycles}\ i\ j\ xs)$

<proof>

lemma *rem-cycles-distinct*: $distinct\ (\text{rem-cycles}\ i\ j\ xs)$

<proof>

lemma *rem-cycles-subsets*: $set\ (\text{rem-cycles}\ i\ j\ xs) \subseteq set\ xs$

<proof>

1.2 Definition of the Algorithm

We formalize the Floyd-Warshall algorithm on a linearly ordered abelian semigroup. However, we would not need an *abelian* monoid if we had the right type class.

class *linordered-ab-monoid-add* = *linordered-ab-semigroup-add* +

fixes *neutral* :: 'a (1)
assumes *neutl[simp]*: $\mathbf{1} + x = x$
assumes *neutr[simp]*: $x + \mathbf{1} = x$
begin

lemmas *assoc* = *add.assoc*

type-synonym 'c *mat* = *nat* \Rightarrow *nat* \Rightarrow 'c

definition (in -) *upd* :: 'c *mat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow 'c \Rightarrow 'c *mat*
where

upd *m* *x* *y* *v* = *m* (*x* := (*m* *x*) (*y* := *v*))

definition *fw-upd* :: 'a *mat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow 'a *mat* **where**
fw-upd *m* *k* *i* *j* \equiv *upd* *m* *i* *j* (*min* (*m* *i* *j*) (*m* *i* *k* + *m* *k* *j*))

lemma *fw-upd-mono*:

fw-upd *m* *k* *i* *j* *i'* *j'* \leq *m* *i'* *j'*
 \langle *proof* \rangle

fun *fw* :: 'a *mat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow 'a *mat* **where**

fw *m* *n* 0 0 0 = *fw-upd* *m* 0 0 0 |
fw *m* *n* (*Suc* *k*) 0 0 = *fw-upd* (*fw* *m* *n* *k* *n* *n*) (*Suc* *k*) 0 0 |
fw *m* *n* *k* (*Suc* *i*) 0 = *fw-upd* (*fw* *m* *n* *k* *i* *n*) *k* (*Suc* *i*) 0 |
fw *m* *n* *k* *i* (*Suc* *j*) = *fw-upd* (*fw* *m* *n* *k* *i* *j*) *k* *i* (*Suc* *j*)

lemma *fw-invariant-aux-1*:

j'' \leq *j* \implies *i* \leq *n* \implies *j* \leq *n* \implies *k* \leq *n* \implies *fw* *m* *n* *k* *i* *j* *i'* *j'* \leq *fw* *m* *n* *k* *i* *j''* *i'* *j'*
 \langle *proof* \rangle

lemma *fw-invariant-aux-2*:

i \leq *n* \implies *j* \leq *n* \implies *k* \leq *n* \implies *i''* \leq *i* \implies *j''* \leq *j*
 \implies *fw* *m* *n* *k* *i* *j* *i'* *j'* \leq *fw* *m* *n* *k* *i''* *j''* *i'* *j'*
 \langle *proof* \rangle

lemma *fw-invariant*:

k' \leq *k* \implies *i* \leq *n* \implies *j* \leq *n* \implies *k* \leq *n* \implies *j''* \leq *j* \implies *i''* \leq *i*
 \implies *fw* *m* *n* *k* *i* *j* *i'* *j'* \leq *fw* *m* *n* *k'* *i''* *j''* *i'* *j'*
 \langle *proof* \rangle

lemma *single-row-inv*:

j' $<$ *j* \implies *j* \leq *n* \implies *i'* \leq *n* \implies *fw* *m* *n* *k* *i'* *j* *i'* *j'* = *fw* *m* *n* *k* *i'* *j'* *i'* *j'*
 \langle *proof* \rangle

lemma *single-iteration-inv'*:

$$i' < i \implies j' \leq n \implies j \leq n \implies i \leq n \implies fw\ m\ n\ k\ i\ j\ i'\ j' = fw\ m\ n\ k\ i'\ j'\ i'\ j'$$

<proof>

lemma *single-iteration-inv*:

$$i' \leq i \implies j' \leq j \implies i \leq n \implies j \leq n \implies fw\ m\ n\ k\ i\ j\ i'\ j' = fw\ m\ n\ k\ i'\ j'\ i'\ j'$$

<proof>

lemma *fw-innermost-id*:

$$i \leq n \implies j \leq n \implies j' \leq n \implies i' < i \implies fw\ m\ n\ 0\ i'\ j'\ i\ j = m\ i\ j$$

<proof>

lemma *fw-middle-id*:

$$i \leq n \implies j \leq n \implies j' < j \implies i' \leq i \implies fw\ m\ n\ 0\ i'\ j'\ i\ j = m\ i\ j$$

<proof>

lemma *fw-outermost-mono*:

$$i \leq n \implies j \leq n \implies fw\ m\ n\ 0\ i\ j\ i\ j \leq m\ i\ j$$

<proof>

lemma *Suc-innermost-id1*:

$$i \leq n \implies j \leq n \implies j' \leq n \implies i' < i \implies fw\ m\ n\ (Suc\ k)\ i'\ j'\ i\ j = fw\ m\ n\ k\ i\ j\ i\ j$$

<proof>

lemma *Suc-innermost-id2*:

$$i \leq n \implies j \leq n \implies j' < j \implies i' \leq i \implies fw\ m\ n\ (Suc\ k)\ i'\ j'\ i\ j = fw\ m\ n\ k\ i\ j\ i\ j$$

<proof>

lemma *Suc-innermost-id1'*:

$$i \leq n \implies j \leq n \implies j' \leq n \implies i' < i \implies fw\ m\ n\ (Suc\ k)\ i'\ j'\ i\ j = fw\ m\ n\ k\ n\ n\ i\ j$$

<proof>

lemma *Suc-innermost-id2'*:

$$i \leq n \implies j \leq n \implies j' < j \implies i' \leq i \implies fw\ m\ n\ (Suc\ k)\ i'\ j'\ i\ j = fw\ m\ n\ k\ n\ n\ i\ j$$

<proof>

lemma *Suc-innermost-mono*:

$i \leq n \implies j \leq n \implies fw\ m\ n\ (Suc\ k)\ i\ j\ i\ j \leq fw\ m\ n\ k\ i\ j\ i\ j$
 ⟨proof⟩

lemma *fw-mono'*:

$i \leq n \implies j \leq n \implies fw\ m\ n\ k\ i\ j\ i\ j \leq m\ i\ j$
 ⟨proof⟩

lemma *fw-mono*:

$i \leq n \implies j \leq n \implies i' \leq n \implies j' \leq n \implies fw\ m\ n\ k\ i\ j\ i'\ j' \leq m\ i'\ j'$
 ⟨proof⟩

lemma *add-mono-neutr*:

assumes $1 \leq b$
shows $a \leq a + b$
 ⟨proof⟩

lemma *add-mono-neutl*:

assumes $1 \leq b$
shows $a \leq b + a$
 ⟨proof⟩

lemma *fw-step-0*:

$m\ 0\ 0 \geq 1 \implies i \leq n \implies j \leq n \implies fw\ m\ n\ 0\ i\ j\ i\ j = \min\ (m\ i\ j)\ (m\ i\ 0 + m\ 0\ j)$
 ⟨proof⟩

lemma *fw-step-Suc*:

$\forall k' \leq n. fw\ m\ n\ k\ n\ n\ k'\ k' \geq 1 \implies i \leq n \implies j \leq n \implies Suc\ k \leq n$
 $\implies fw\ m\ n\ (Suc\ k)\ i\ j\ i\ j = \min\ (fw\ m\ n\ k\ n\ n\ i\ j)\ (fw\ m\ n\ k\ n\ n\ i\ (Suc\ k) + fw\ m\ n\ k\ n\ n\ (Suc\ k)\ j)$
 ⟨proof⟩

1.2.1 Length of Paths

fun *len* :: 'a mat \Rightarrow nat \Rightarrow nat \Rightarrow nat list \Rightarrow 'a **where**

$len\ m\ u\ v\ [] = m\ u\ v$ |
 $len\ m\ u\ v\ (w\ \# ws) = m\ u\ w + len\ m\ w\ v\ ws$

lemma *len-decomp*: $xs = ys\ @\ y\ \#\ zs \implies len\ m\ x\ z\ xs = len\ m\ x\ y\ ys + len\ m\ y\ z\ zs$

⟨proof⟩

lemma *len-comp*: $len\ m\ a\ c\ (xs\ @\ b\ \#\ ys) = len\ m\ a\ b\ xs + len\ m\ b\ c\ ys$

⟨proof⟩

1.2.2 Shortening Negative Cycles

lemma *remove-cycles-neg-cycles-aux*:

fixes $i\ xs\ ys$
defines $xs' \equiv i \# ys$
assumes $i \notin \text{set } ys$
assumes $i \in \text{set } xs$
assumes $xs = as @ \text{concat} (\text{map} ((\#) i) xss) @ xs'$
assumes $\text{len } m\ i\ j\ ys > \text{len } m\ i\ j\ xs$
shows $\exists\ ys. \text{set } ys \subseteq \text{set } xs \wedge \text{len } m\ i\ i\ ys < \mathbf{1}$ *<proof>*

lemma *add-lt-neutral*: $a + b < b \implies a < \mathbf{1}$
<proof>

lemma *remove-cycles-neg-cycles-aux'*:

fixes $j\ xs\ ys$
assumes $j \notin \text{set } ys$
assumes $j \in \text{set } xs$
assumes $xs = ys @ j \# \text{concat} (\text{map} (\lambda\ xs. xs @ [j]) xss) @ as$
assumes $\text{len } m\ i\ j\ ys > \text{len } m\ i\ j\ xs$
shows $\exists\ ys. \text{set } ys \subseteq \text{set } xs \wedge \text{len } m\ j\ j\ ys < \mathbf{1}$ *<proof>*

lemma *add-le-impl*: $a + b < a + c \implies b < c$
<proof>

lemma *start-remove-neg-cycles*:

$\text{len } m\ i\ j\ (\text{start-remove } xs\ k\ []) > \text{len } m\ i\ j\ xs \implies \exists\ ys. \text{set } ys \subseteq \text{set } xs$
 $\wedge \text{len } m\ k\ k\ ys < \mathbf{1}$
<proof>

lemma *remove-all-cycles-neg-cycles*:

$\text{len } m\ i\ j\ (\text{remove-all-cycles } ys\ xs) > \text{len } m\ i\ j\ xs$
 $\implies \exists\ ys\ k. \text{set } ys \subseteq \text{set } xs \wedge k \in \text{set } xs \wedge \text{len } m\ k\ k\ ys < \mathbf{1}$
<proof>

lemma (**in** $-$) *concat-map-cons-rev*:

$\text{rev} (\text{concat} (\text{map} ((\#) j) xss)) = \text{concat} (\text{map} (\lambda\ xs. xs @ [j]) (\text{rev} (\text{map } \text{rev } xss)))$
<proof>

lemma *negative-cycle-dest*: $\text{len } m\ i\ j\ (\text{rem-cycles } i\ j\ xs) > \text{len } m\ i\ j\ xs$

$\implies \exists\ i'\ ys. \text{len } m\ i'\ i'\ ys < \mathbf{1} \wedge \text{set } ys \subseteq \text{set } xs \wedge i' \in \text{set } (i \# j \# xs)$
<proof>

1.3 Definition of Shortest Paths

definition $D :: 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ where}$

$D \ m \ i \ j \ k \equiv \text{Min} \ \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge i \notin \text{set } xs \wedge j \notin \text{set } xs \wedge \text{distinct } xs \}$

lemma (in $-$) *distinct-length-le*: $\text{finite } s \Longrightarrow \text{set } xs \subseteq s \Longrightarrow \text{distinct } xs \Longrightarrow \text{length } xs \leq \text{card } s$

$\langle \text{proof} \rangle$

lemma (in $-$) *finite-distinct*: $\text{finite } s \Longrightarrow \text{finite } \{xs \mid \text{set } xs \subseteq s \wedge \text{distinct } xs\}$

$\langle \text{proof} \rangle$

lemma *D-base-finite*:

$\text{finite } \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge \text{distinct } xs \}$

$\langle \text{proof} \rangle$

lemma *D-base-finite'*:

$\text{finite } \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge \text{distinct } (i \# j \# xs) \}$

$\langle \text{proof} \rangle$

lemma *D-base-finite''*:

$\text{finite } \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge i \notin \text{set } xs \wedge j \notin \text{set } xs \wedge \text{distinct } xs \}$

$\langle \text{proof} \rangle$

definition *cycle-free* :: $'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{bool} \text{ where}$

$\text{cycle-free } m \ n \equiv \forall \ i \ xs. \ i \leq n \wedge \text{set } xs \subseteq \{0..n\} \longrightarrow$

$(\forall \ j. \ j \leq n \longrightarrow \text{len } m \ i \ j \ (\text{rem-cycles } i \ j \ xs) \leq \text{len } m \ i \ j \ xs) \wedge \text{len } m \ i \ i \ xs \geq 1$

lemma *D-eqI*:

fixes $m \ n \ i \ j \ k$

defines $A \equiv \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \}$

defines $A\text{-distinct} \equiv \{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge i \notin \text{set } xs \wedge j \notin \text{set } xs \wedge \text{distinct } xs \}$

assumes $\text{cycle-free } m \ n \ i \leq n \ j \leq n \ k \leq n \ (\bigwedge y. y \in A\text{-distinct} \Longrightarrow x \leq y) \ x \in A$

shows $D \ m \ i \ j \ k = x \ \langle \text{proof} \rangle$

lemma *D-base-not-empty*:

$\{ \text{len } m \ i \ j \ xs \mid xs. \text{ set } xs \subseteq \{0..k\} \wedge i \notin \text{set } xs \wedge j \notin \text{set } xs \wedge \text{distinct } xs \} \neq \{ \}$

<proof>

lemma *Min-elem-dest*: $finite\ A \implies A \neq \{\} \implies x = Min\ A \implies x \in A$
<proof>

lemma *D-dest*: $x = D\ m\ i\ j\ k \implies$
 $x \in \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..Suc\ k\} \wedge i \notin set\ xs \wedge j \notin set\ xs \wedge$
distinct\ xs\}
<proof>

lemma *D-dest'*: $x = D\ m\ i\ j\ k \implies x \in \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..Suc\ k\}\}$
<proof>

lemma *D-dest''*: $x = D\ m\ i\ j\ k \implies x \in \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..k\}\}$
<proof>

lemma *cycle-free-loop-dest*: $i \leq n \implies set\ xs \subseteq \{0..n\} \implies cycle-free\ m\ n$
 $\implies len\ m\ i\ i\ xs \geq \mathbf{1}$
<proof>

lemma *cycle-free-dest*:
 $cycle-free\ m\ n \implies i \leq n \implies j \leq n \implies set\ xs \subseteq \{0..n\}$
 $\implies len\ m\ i\ j\ (rem-cycles\ i\ j\ xs) \leq len\ m\ i\ j\ xs$
<proof>

definition *cycle-free-up-to* :: $'a\ mat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
 $cycle-free-up-to\ m\ k\ n \equiv \forall\ i\ xs.\ i \leq n \wedge set\ xs \subseteq \{0..k\} \longrightarrow$
 $(\forall\ j.\ j \leq n \longrightarrow len\ m\ i\ j\ (rem-cycles\ i\ j\ xs) \leq len\ m\ i\ j\ xs) \wedge len\ m\ i\ i\ xs \geq \mathbf{1}$

lemma *cycle-free-up-to-loop-dest*:
 $i \leq n \implies set\ xs \subseteq \{0..k\} \implies cycle-free-up-to\ m\ k\ n \implies len\ m\ i\ i\ xs \geq$
 $\mathbf{1}$
<proof>

lemma *cycle-free-up-to-diag*:
assumes $cycle-free-up-to\ m\ k\ n\ i \leq n$
shows $m\ i\ i \geq \mathbf{1}$
<proof>

lemma *D-eqI2*:
fixes $m\ n\ i\ j\ k$
defines $A \equiv \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..k\}\}$

defines $A\text{-distinct} \equiv \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..k\} \wedge i \notin set\ xs \wedge j \notin set\ xs \wedge distinct\ xs\}$
assumes $cycle\text{-free}\text{-up}\text{-to}\ m\ k\ n\ i \leq n\ j \leq n\ k \leq n$
 $(\bigwedge y. y \in A\text{-distinct} \implies x \leq y)\ x \in A$
shows $D\ m\ i\ j\ k = x$ *<proof>*

1.4 Result Under The Absence of Negative Cycles

This proves that the algorithm correctly computes shortest paths under the absence of negative cycles by a standard argument.

theorem $fw\text{-shortest}\text{-path}\text{-up}\text{-to}$:

$cycle\text{-free}\text{-up}\text{-to}\ m\ k\ n \implies i' \leq i \implies j' \leq j \implies i \leq n \implies j \leq n \implies k \leq n$
 $\implies D\ m\ i'\ j'\ k = fw\ m\ n\ k\ i\ j\ i'\ j'$
<proof>

lemma $cycle\text{-free}\text{-cycle}\text{-free}\text{-up}\text{-to}$:

$cycle\text{-free}\ m\ n \implies k \leq n \implies cycle\text{-free}\text{-up}\text{-to}\ m\ k\ n$
<proof>

lemma $cycle\text{-free}\text{-diag}$:

$cycle\text{-free}\ m\ n \implies i \leq n \implies \mathbf{1} \leq m\ i\ i$
<proof>

corollary $fw\text{-shortest}\text{-path}$:

$cycle\text{-free}\ m\ n \implies i' \leq i \implies j' \leq j \implies i \leq n \implies j \leq n \implies k \leq n$
 $\implies D\ m\ i'\ j'\ k = fw\ m\ n\ k\ i\ j\ i'\ j'$
<proof>

corollary $fw\text{-shortest}$:

assumes $cycle\text{-free}\ m\ n\ i \leq n\ j \leq n\ k \leq n$
shows $fw\ m\ n\ n\ n\ n\ i\ j \leq fw\ m\ n\ n\ n\ n\ i\ k + fw\ m\ n\ n\ n\ n\ k\ j$
<proof>

1.5 Result Under the Presence of Negative Cycles

lemma $not\text{-cylce}\text{-free}\text{-dest}$: $\neg cycle\text{-free}\ m\ n \implies \exists k \leq n. \neg cycle\text{-free}\text{-up}\text{-to}\ m\ k\ n$
<proof>

lemma $D\text{-not}\text{-diag}\text{-le}$:

$(x :: 'a) \in \{len\ m\ i\ j\ xs \mid xs.\ set\ xs \subseteq \{0..k\} \wedge i \notin set\ xs \wedge j \notin set\ xs \wedge distinct\ xs\}$
 $\implies D\ m\ i\ j\ k \leq x$ *<proof>*

lemma *D-not-diag-le'*: $set\ xs \subseteq \{0..k\} \implies i \notin set\ xs \implies j \notin set\ xs \implies$
distinct xs
 $\implies D\ m\ i\ j\ k \leq len\ m\ i\ j\ xs$ *<proof>*

lemma (*in -*) *nat-upto-subst-top-removal'*:
 $S \subseteq \{0..Suc\ n\} \implies Suc\ n \notin S \implies S \subseteq \{0..n\}$
<proof>

lemma (*in -*) *nat-upto-subst-top-removal*:
 $S \subseteq \{0..n::nat\} \implies n \notin S \implies S \subseteq \{0..n - 1\}$
<proof>

lemma *fw-Suc*:
 $i \leq n \implies j \leq n \implies i' \leq n \implies j' \leq n \implies fw\ m\ n\ (Suc\ k)\ i'\ j'\ i\ j \leq$
 $fw\ m\ n\ k\ n\ n\ i\ j$
<proof>

lemma *negative-len-shortest*:
 $length\ xs = n \implies len\ m\ i\ i\ xs < \mathbf{1}$
 $\implies \exists j\ ys.\ distinct\ (j\ \#\ ys) \wedge len\ m\ j\ j\ ys < \mathbf{1} \wedge j \in set\ (i\ \#\ xs) \wedge$
 $set\ ys \subseteq set\ xs$
<proof>

theorem *FW-neg-cycle-detect*:
 $\neg\ cycle\ free\ m\ n \implies \exists i \leq n.\ fw\ m\ n\ n\ n\ n\ i\ i < \mathbf{1}$
<proof>

end

end

theory *Timed-Automata*

imports *Main*

begin

2 Basic Definitions and Semantics

2.1 Time

class *time* = *linordered-ab-group-add* +
assumes *dense*: $x < y \implies \exists z.\ x < z \wedge z < y$
assumes *non-trivial*: $\exists x.\ x \neq 0$

begin

lemma *non-trivial-neg*: $\exists x. x < 0$
 ⟨*proof*⟩

end

datatype ('c, 't :: time) cconstraint =
 AND ('c, 't) cconstraint ('c, 't) cconstraint |
 LT 'c 't |
 LE 'c 't |
 EQ 'c 't |
 GT 'c 't |
 GE 'c 't

2.2 Syntactic Definition

For an informal description of timed automata we refer to Bengtsson and Yi [BY03]. We define a timed automaton A

type-synonym

('c, 'time, 's) invassn = 's \Rightarrow ('c, 'time) cconstraint

type-synonym

('a, 'c, 'time, 's) transition = 's * ('c, 'time) cconstraint * 'a * 'c list * 's

type-synonym

('a, 'c, 'time, 's) ta = ('a, 'c, 'time, 's) transition set * ('c, 'time, 's) invassn

definition *trans-of* :: ('a, 'c, 'time, 's) ta \Rightarrow ('a, 'c, 'time, 's) transition set **where**

trans-of \equiv fst

definition *inv-of* :: ('a, 'c, 'time, 's) ta \Rightarrow ('c, 'time, 's) invassn **where**

inv-of \equiv snd

abbreviation *transition* ::

('a, 'c, 'time, 's) ta \Rightarrow 's \Rightarrow ('c, 'time) cconstraint \Rightarrow 'a \Rightarrow 'c list \Rightarrow 's \Rightarrow bool

(- \vdash - \longrightarrow τ, τ' - [61,61,61,61,61,61] 61) **where**

(A \vdash l \longrightarrow g, a, r l') \equiv (l, g, a, r, l') \in *trans-of* A

2.2.1 Collecting Information About Clocks

fun *collect-clks* :: ('c, 't :: time) cconstraint \Rightarrow 'c set

where

collect-clks (AND cc1 cc2) = *collect-clks* cc1 \cup *collect-clks* cc2 |

$collect-clks (LT c -) = \{c\} |$
 $collect-clks (LE c -) = \{c\} |$
 $collect-clks (EQ c -) = \{c\} |$
 $collect-clks (GE c -) = \{c\} |$
 $collect-clks (GT c -) = \{c\}$

fun $collect-clock-pairs :: ('c, 't :: time) cconstraint \Rightarrow ('c * 't) set$

where

$collect-clock-pairs (LT x m) = \{(x, m)\} |$
 $collect-clock-pairs (LE x m) = \{(x, m)\} |$
 $collect-clock-pairs (EQ x m) = \{(x, m)\} |$
 $collect-clock-pairs (GE x m) = \{(x, m)\} |$
 $collect-clock-pairs (GT x m) = \{(x, m)\} |$
 $collect-clock-pairs (AND cc1 cc2) = (collect-clock-pairs cc1 \cup collect-clock-pairs cc2)$

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

where

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

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

where

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

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

where

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

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

where

$collect-clkvt S = \bigcup \{set ((fst o snd o snd o snd) t) | t . t \in S\}$

abbreviation $clk-set$ **where** $clk-set A \equiv fst \text{ ` } clkp-set A \cup collect-clkvt (trans-of A)$

inductive $valid-abstraction$

where

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

2.3 Operational Semantics

type-synonym $(\prime c, \prime t) \text{ cval} = \prime c \Rightarrow \prime t$

definition $\text{cval-add} :: (\prime c, \prime t) \text{ cval} \Rightarrow \prime t :: \text{time} \Rightarrow (\prime c, \prime t) \text{ cval}$ (**infixr** \oplus 64)

where

$$u \oplus d = (\lambda x. u x + d)$$

inductive $\text{clock-val} :: (\prime c, \prime t) \text{ cval} \Rightarrow (\prime c, \prime t :: \text{time}) \text{ cconstraint} \Rightarrow \text{bool}$ ($- \vdash -$ [62, 62] 62)

where

$$\begin{aligned} \llbracket u \vdash \text{cc1}; u \vdash \text{cc2} \rrbracket &\Longrightarrow u \vdash \text{AND } \text{cc1 } \text{cc2} \mid \\ \llbracket u \text{ c} < \text{d} \rrbracket &\Longrightarrow u \vdash \text{LT } \text{c } \text{d} \mid \\ \llbracket u \text{ c} \leq \text{d} \rrbracket &\Longrightarrow u \vdash \text{LE } \text{c } \text{d} \mid \\ \llbracket u \text{ c} = \text{d} \rrbracket &\Longrightarrow u \vdash \text{EQ } \text{c } \text{d} \mid \\ \llbracket u \text{ c} \geq \text{d} \rrbracket &\Longrightarrow u \vdash \text{GE } \text{c } \text{d} \mid \\ \llbracket u \text{ c} > \text{d} \rrbracket &\Longrightarrow u \vdash \text{GT } \text{c } \text{d} \end{aligned}$$

declare clock-val.intros [intro]

inductive-cases[elim!]: $u \vdash \text{AND } \text{cc1 } \text{cc2}$

inductive-cases[elim!]: $u \vdash \text{LT } \text{c } \text{d}$

inductive-cases[elim!]: $u \vdash \text{LE } \text{c } \text{d}$

inductive-cases[elim!]: $u \vdash \text{EQ } \text{c } \text{d}$

inductive-cases[elim!]: $u \vdash \text{GE } \text{c } \text{d}$

inductive-cases[elim!]: $u \vdash \text{GT } \text{c } \text{d}$

fun $\text{clock-set} :: \prime c \text{ list} \Rightarrow \prime t :: \text{time} \Rightarrow (\prime c, \prime t) \text{ cval} \Rightarrow (\prime c, \prime t) \text{ cval}$

where

$$\begin{aligned} \text{clock-set } [] - u &= u \mid \\ \text{clock-set } (\text{c}\#\text{cs}) \text{ t } u &= (\text{clock-set } \text{cs } \text{t } u)(\text{c}:=\text{t}) \end{aligned}$$

abbreviation $\text{clock-set-abbrev} :: \prime c \text{ list} \Rightarrow \prime t :: \text{time} \Rightarrow (\prime c, \prime t) \text{ cval} \Rightarrow (\prime c, \prime t) \text{ cval}$

$([-\rightarrow]-$ [65,65,65] 65)

where

$$[r \rightarrow t]u \equiv \text{clock-set } r \text{ t } u$$

inductive $\text{step-t} ::$

$(\prime a, \prime c, \prime t, \prime s) \text{ ta} \Rightarrow \prime s \Rightarrow (\prime c, \prime t) \text{ cval} \Rightarrow (\prime t :: \text{time}) \Rightarrow \prime s \Rightarrow (\prime c, \prime t) \text{ cval} \Rightarrow \text{bool}$

$(- \vdash \langle -, - \rangle \rightarrow^{\cdot} \langle -, - \rangle$ [61,61,61] 61)

where

$$\llbracket u \vdash \text{inv-of } A \text{ l}; u \oplus d \vdash \text{inv-of } A \text{ l}; d \geq 0 \rrbracket \Longrightarrow A \vdash \langle l, u \rangle \rightarrow^d \langle l, u \oplus d \rangle$$

declare *step-t.intros*[*intro!*]

inductive-cases[*elim!*]: $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle$

lemma *step-t-determinacy1*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies A \vdash \langle l, u \rangle \rightarrow^d \langle l'', u'' \rangle \implies l' = l''$
<proof>

lemma *step-t-determinacy2*:

$A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies A \vdash \langle l, u \rangle \rightarrow^d \langle l'', u'' \rangle \implies u' = u''$
<proof>

lemma *step-t-cont1*:

$d \geq 0 \implies e \geq 0 \implies A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies A \vdash \langle l', u' \rangle \rightarrow^e \langle l'', u'' \rangle$
 $\implies A \vdash \langle l, u \rangle \rightarrow^{d+e} \langle l'', u'' \rangle$
<proof>

inductive *step-a* ::

$(\text{'a}, \text{'c}, \text{'t}, \text{'s}) \text{ta} \implies \text{'s} \implies (\text{'c}, (\text{'t}::\text{time})) \text{cval} \implies \text{'a} \implies \text{'s} \implies (\text{'c}, \text{'t}) \text{cval} \implies \text{bool}$

$(- \vdash \langle -, - \rangle \rightarrow_- \langle -, - \rangle)$ [61,61,61] 61)

where

$\llbracket A \vdash l \xrightarrow{g,a,r} l'; u \vdash g; u' \vdash \text{inv-of } A \text{ } l'; u' = [r \rightarrow 0]u \rrbracket \implies (A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle)$

inductive *step* ::

$(\text{'a}, \text{'c}, \text{'t}, \text{'s}) \text{ta} \implies \text{'s} \implies (\text{'c}, (\text{'t}::\text{time})) \text{cval} \implies \text{'s} \implies (\text{'c}, \text{'t}) \text{cval} \implies \text{bool}$
 $(- \vdash \langle -, - \rangle \rightarrow \langle -, - \rangle)$ [61,61,61] 61)

where

step-a: $A \vdash \langle l, u \rangle \rightarrow_a \langle l', u' \rangle \implies (A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle) \mid$

step-t: $A \vdash \langle l, u \rangle \rightarrow^d \langle l', u' \rangle \implies (A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle)$

inductive-cases[*elim!*]: $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$

declare *step.intros*[*intro*]

inductive

steps :: $(\text{'a}, \text{'c}, \text{'t}, \text{'s}) \text{ta} \implies \text{'s} \implies (\text{'c}, (\text{'t}::\text{time})) \text{cval} \implies \text{'s} \implies (\text{'c}, \text{'t}) \text{cval} \implies \text{bool}$

$(- \vdash \langle -, - \rangle \rightarrow_* \langle -, - \rangle)$ [61,61,61] 61)

where

refl: $A \vdash \langle l, u \rangle \rightarrow_* \langle l, u \rangle \mid$

step: $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies A \vdash \langle l', u' \rangle \rightarrow_* \langle l'', u'' \rangle \implies A \vdash \langle l, u \rangle$

$\rightarrow^* \langle l'', u' \rangle$

declare *steps.intros*[*intro*]

2.4 Zone Semantics

type-synonym (c, t) zone = (c, t) cval set

definition *zone-delay* :: $(c, (t::time))$ zone \Rightarrow (c, t) zone
(\rightarrow^* [71] 71)

where

$$Z^\uparrow = \{u \oplus d \mid u \in Z \wedge d \geq (0::t)\}$$

definition *zone-set* :: $(c, (t::time))$ zone \Rightarrow c list \Rightarrow (c, t) zone
(\rightarrow 0 [71] 71)

where

$$\text{zone-set } Z \ r = \{[r \rightarrow (0::t)]u \mid u \in Z\}$$

inductive *step-z* ::

(a, c, t, s) ta \Rightarrow $s \Rightarrow (c, (t::time))$ zone \Rightarrow $s \Rightarrow (c, t)$ zone \Rightarrow bool
($\vdash \langle -, - \rangle \rightsquigarrow \langle -, - \rangle$ [61,61,61] 61)

where

step-t-z: $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l, (Z \cap \{u. u \vdash \text{inv-of } A \ l\})^\uparrow \cap \{u. u \vdash \text{inv-of } A \ l\} \rangle$ |

step-a-z: $\llbracket A \vdash l \xrightarrow{g,a,r} l \rrbracket$
 $\implies (A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', \text{zone-set } (Z \cap \{u. u \vdash g\}) \ r \cap \{u. u \vdash \text{inv-of } A \ l'\})$

inductive-cases[*elim!*]: $A \vdash \langle l, u \rangle \rightsquigarrow \langle l', u' \rangle$

declare *step-z.intros*[*intro*]

lemma *step-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies (\forall u' \in Z'. \exists u \in Z. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle)$
<proof>

lemma *step-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \wedge u' \in Z'$
<proof>

Corresponds to version in old papers – not strong enough for inductive proof over transitive closure relation.

lemma *step-z-complete1*:

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies \exists Z. A \vdash \langle l, \{u\} \rangle \rightsquigarrow \langle l', Z \rangle \wedge u' \in Z$
 $\langle proof \rangle$

Easier proof.

lemma *step-z-complete2*:

$A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle \implies \exists Z. A \vdash \langle l, \{u\} \rangle \rightsquigarrow \langle l', Z \rangle \wedge u' \in Z$
 $\langle proof \rangle$

inductive

$steps\text{-}z :: ('a, 'c, 't, 's) ta \Rightarrow 's \Rightarrow ('c, ('t::time)) zone \Rightarrow 's \Rightarrow ('c, 't)$
 $zone \Rightarrow bool$

$(- \vdash \langle -, - \rangle \rightsquigarrow^* \langle -, - \rangle [61,61,61] 61)$

where

$refl: A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l, Z \rangle |$

$step: A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow^* \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle$
 $\rightsquigarrow^* \langle l'', Z'' \rangle$

declare *steps-z.intros*[intro]

lemma *steps-z-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \implies u' \in Z' \implies \exists u \in Z. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$
 $\langle proof \rangle$

lemma *steps-z-complete*:

$A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle \implies u \in Z \implies \exists Z'. A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \wedge u' \in Z'$
 $\langle proof \rangle$

end

theory *DBM*

imports *Floyd-Warshall Timed-Automata*

begin

3 Difference Bound Matrices

3.1 Definitions

Difference Bound Matrices (DBMs) constrain differences of clocks (or more precisely, the difference of values assigned to individual clocks by a valuation). The possible constraints are given by the following datatype:

datatype $(t::time) DBMEntry = Le\ t | Lt\ t | INF\ (\infty)$

This yields a simple definition of DBMs:

type-synonym $'t\ DBM = nat \Rightarrow nat \Rightarrow 't\ DBMEntry$

To relate clocks with rows and columns of a DBM, we use a clock numbering v of type $'c \Rightarrow nat$ to map clocks to indices. DBMs will regularly be accompanied by a natural number n , which designates the number of clocks constrained by the matrix. To be able to represent the full set of clock constraints with DBMs, we add an imaginary clock $\mathbf{0}$, which shall be assigned to 0 in every valuation. In the following predicate we explicitly keep track of $\mathbf{0}$.

inductive *dbm-entry-val* :: ($'c, 't$) *cval* $\Rightarrow 'c$ option $\Rightarrow 'c$ option $\Rightarrow ('t::time)$
DBMEntry $\Rightarrow bool$

where

$u\ r \leq d \implies dbm\text{-entry}\text{-val}\ u\ (Some\ r)\ None\ (Le\ d)\ |$
 $-u\ c \leq d \implies dbm\text{-entry}\text{-val}\ u\ None\ (Some\ c)\ (Le\ d)\ |$
 $u\ r < d \implies dbm\text{-entry}\text{-val}\ u\ (Some\ r)\ None\ (Lt\ d)\ |$
 $-u\ c < d \implies dbm\text{-entry}\text{-val}\ u\ None\ (Some\ c)\ (Lt\ d)\ |$
 $u\ r - u\ c \leq d \implies dbm\text{-entry}\text{-val}\ u\ (Some\ r)\ (Some\ c)\ (Le\ d)\ |$
 $u\ r - u\ c < d \implies dbm\text{-entry}\text{-val}\ u\ (Some\ r)\ (Some\ c)\ (Lt\ d)\ |$
 $dbm\text{-entry}\text{-val}\ -\ -\ -\ \infty$

declare *dbm-entry-val.intros*[*intro*]

inductive-cases[*elim!*]: *dbm-entry-val* $u\ None\ (Some\ c)\ (Le\ d)$
inductive-cases[*elim!*]: *dbm-entry-val* $u\ (Some\ c)\ None\ (Le\ d)$
inductive-cases[*elim!*]: *dbm-entry-val* $u\ None\ (Some\ c)\ (Lt\ d)$
inductive-cases[*elim!*]: *dbm-entry-val* $u\ (Some\ c)\ None\ (Lt\ d)$
inductive-cases[*elim!*]: *dbm-entry-val* $u\ (Some\ r)\ (Some\ c)\ (Le\ d)$
inductive-cases[*elim!*]: *dbm-entry-val* $u\ (Some\ r)\ (Some\ c)\ (Lt\ d)$

fun *dbm-entry-bound* :: ($'t::time$) *DBMEntry* $\Rightarrow 't$

where

$dbm\text{-entry}\text{-bound}\ (Le\ t) = t\ |$
 $dbm\text{-entry}\text{-bound}\ (Lt\ t) = t\ |$
 $dbm\text{-entry}\text{-bound}\ \infty = 0$

inductive *dbm-lt* :: ($'t::time$) *DBMEntry* $\Rightarrow 't$ *DBMEntry* $\Rightarrow bool$
 $(- \prec - [51, 51] 50)$

where

$dbm\text{-lt}\ (Lt\ -) \infty\ |$
 $dbm\text{-lt}\ (Le\ -) \infty\ |$
 $a < b \implies dbm\text{-lt}\ (Le\ a)\ (Le\ b)\ |$
 $a < b \implies dbm\text{-lt}\ (Le\ a)\ (Lt\ b)\ |$
 $a \leq b \implies dbm\text{-lt}\ (Lt\ a)\ (Le\ b)\ |$
 $a < b \implies dbm\text{-lt}\ (Lt\ a)\ (Lt\ b)$

declare *dbm-lt.intros*[*intro*]

definition $dbm-le :: ('t::time) DBMEntry \Rightarrow 't DBMEntry \Rightarrow bool$
 $(- \preceq - [51, 51] 50)$

where

$$dbm-le\ a\ b \equiv (a \prec b) \vee a = b$$

Now a valuation is contained in the zone represented by a DBM if it fulfills all individual constraints:

definition $DBM-val-bounded :: ('c \Rightarrow nat) \Rightarrow ('c, 't) cval \Rightarrow ('t::time) DBM \Rightarrow nat \Rightarrow bool$

where

$$\begin{aligned} DBM-val-bounded\ v\ u\ m\ n &\equiv Le\ 0\ \preceq\ m\ 0\ 0 \wedge \\ &(\forall\ c.\ v\ c \leq n \longrightarrow (dbm-entry-val\ u\ None\ (Some\ c)\ (m\ 0\ (v\ c)) \\ &\quad \wedge\ dbm-entry-val\ u\ (Some\ c)\ None\ (m\ (v\ c)\ 0))) \\ &\wedge (\forall\ c1\ c2.\ v\ c1 \leq n \wedge v\ c2 \leq n \longrightarrow dbm-entry-val\ u\ (Some\ c1)\ (Some\ c2)\ (m\ (v\ c1)\ (v\ c2))) \end{aligned}$$

abbreviation $DBM-val-bounded-abbrev ::$

$$('c, 't) cval \Rightarrow ('c \Rightarrow nat) \Rightarrow nat \Rightarrow ('t::time) DBM \Rightarrow bool$$

$(- \vdash_{v,n} -)$

where

$$u \vdash_{v,n} M \equiv DBM-val-bounded\ v\ u\ M\ n$$

abbreviation

$$dmin\ a\ b \equiv \text{if } a \prec b \text{ then } a \text{ else } b$$

lemma $dbm-le-dbm-min:$

$$a \preceq b \implies a = dmin\ a\ b \langle proof \rangle$$

lemma $dbm-lt-asymp:$

assumes $e \prec f$

shows $\sim f \prec e$

$\langle proof \rangle$

lemma $dbm-le-dbm-min2:$

$$a \preceq b \implies a = dmin\ b\ a$$

$\langle proof \rangle$

lemma $dmb-le-dbm-entry-bound-inf:$

$$a \preceq b \implies a = \infty \implies b = \infty$$

$\langle proof \rangle$

lemma $dbm-not-lt-eq: \neg a \prec b \implies \neg b \prec a \implies a = b$

$\langle proof \rangle$

lemma *dbm-not-lt-impl*: $\neg a < b \implies b < a \vee a = b$ *<proof>*

lemma *dmin a b = dmin b a*
<proof>

lemma *dbm-lt-trans*: $a < b \implies b < c \implies a < c$
<proof>

lemma *aux-3*: $\neg a < b \implies \neg b < c \implies a < c \implies c = a$
<proof>

inductive-cases[*elim!*]: $\infty < x$

lemma *dbm-lt-asymmetric[simp]*: $x < y \implies y < x \implies False$
<proof>

lemma *le-dbm-le*: $Le\ a \preceq Le\ b \implies a \leq b$ *<proof>*

lemma *le-dbm-lt*: $Le\ a \preceq Lt\ b \implies a < b$ *<proof>*

lemma *lt-dbm-le*: $Lt\ a \preceq Le\ b \implies a \leq b$ *<proof>*

lemma *lt-dbm-lt*: $Lt\ a \preceq Lt\ b \implies a \leq b$ *<proof>*

lemma *not-dbm-le-le-impl*: $\neg Le\ a < Le\ b \implies a \geq b$ *<proof>*

lemma *not-dbm-lt-le-impl*: $\neg Lt\ a < Le\ b \implies a > b$ *<proof>*

lemma *not-dbm-lt-lt-impl*: $\neg Lt\ a < Lt\ b \implies a \geq b$ *<proof>*

lemma *not-dbm-le-lt-impl*: $\neg Le\ a < Lt\ b \implies a \geq b$ *<proof><proof><proof><proof><proof><proof><proof>*

theory *Paths-Cycles*

imports *Floyd-Warshall Timed-Automata*

begin

4 Library for Paths, Arcs and Lengths

lemma *length-eq-distinct*:

assumes *set xs = set ys distinct xs length xs = length ys*

shows *distinct ys*

<proof>

4.1 Arcs

fun *arcs* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow (\text{nat} * \text{nat}) \text{ list}$ **where**
 arcs *a b* [] = [(*a,b*)] |
 arcs *a b* (*x # xs*) = (*a, x*) # *arcs* *x b xs*

definition *arcs'* :: $\text{nat list} \Rightarrow (\text{nat} * \text{nat}) \text{ set}$ **where**
 arcs' *xs* = *set* (*arcs* (*hd xs*) (*last xs*) (*butlast (tl xs)*))

lemma *arcs'-decomp*:

$\text{length } xs > 1 \implies (i, j) \in \text{arcs}' \text{ } xs \implies \exists \text{ } zs \text{ } ys. \text{ } xs = zs @ i \# j \# ys$
 <proof>

lemma *arcs-decomp-tail*:

$\text{arcs } j \text{ } l \text{ } (ys @ [i]) = \text{arcs } j \text{ } i \text{ } ys @ [(i, l)]$
 <proof>

lemma *arcs-decomp*: $xs = ys @ y \# zs \implies \text{arcs } x \text{ } z \text{ } xs = \text{arcs } x \text{ } y \text{ } ys @ \text{arcs } y \text{ } z \text{ } zs$
 <proof>

lemma *distinct-arcs-ex*:

$\text{distinct } xs \implies i \notin \text{set } xs \implies xs \neq [] \implies \exists \text{ } a \text{ } b. a \neq x \wedge (a, b) \in \text{set } (\text{arcs } i \text{ } j \text{ } xs)$
 <proof>

lemma *cycle-rotate-2-aux*:

$(i, j) \in \text{set } (\text{arcs } a \text{ } b \text{ } (xs @ [c])) \implies (i, j) \neq (c, b) \implies (i, j) \in \text{set } (\text{arcs } a \text{ } c \text{ } xs)$
 <proof>

lemma *arcs-set-elem1*:

assumes $j \neq k \text{ } k \in \text{set } (i \# xs)$
shows $\exists \text{ } l. (k, l) \in \text{set } (\text{arcs } i \text{ } j \text{ } xs)$ <proof>

lemma *arcs-set-elem2*:

assumes $i \neq k \text{ } k \in \text{set } (j \# xs)$
shows $\exists \text{ } l. (l, k) \in \text{set } (\text{arcs } i \text{ } j \text{ } xs)$ <proof>

4.2 Length of Paths

lemmas (in *linordered-ab-monoid-add*) *comm = add.commute*

lemma *len-add*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \text{ mat}$
shows $\text{len } M \ i \ j \ xs + \text{len } M \ i \ j \ xs = \text{len } (\lambda i \ j. M \ i \ j + M \ i \ j) \ i \ j \ xs$
 $\langle \text{proof} \rangle$

4.3 Canonical Matrices

abbreviation

$\text{canonical } M \ n \equiv \forall \ i \ j \ k. \ i \leq n \wedge j \leq n \wedge k \leq n \longrightarrow M \ i \ k \leq M \ i \ j + M \ j \ k$

lemma *fw-canonical*:

$\text{cycle-free } m \ n \Longrightarrow \text{canonical } (\text{fw } m \ n \ n \ n \ n) \ n$
 $\langle \text{proof} \rangle$

lemma *canonical-len*:

$\text{canonical } M \ n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow \text{set } xs \subseteq \{0..n\} \Longrightarrow M \ i \ j \leq \text{len } M \ i \ j \ xs$
 $\langle \text{proof} \rangle$

4.4 Cycle Rotation

lemma *cycle-rotate*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \text{ mat}$
assumes $\text{length } xs > 1 \ (i, j) \in \text{arcs}' \ xs$
shows $\exists \ ys \ zs. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs) \wedge xs = zs \ @ \ i \ \# \ j \ \# \ ys$ $\langle \text{proof} \rangle$

lemma *cycle-rotate-2*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \text{ mat}$
assumes $xs \neq [] \ (i, j) \in \text{set } (\text{arcs } a \ a \ xs)$
shows $\exists \ ys. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys) \wedge \text{set } ys \subseteq \text{set } (a \ \# \ xs)$
 $\wedge \text{length } ys < \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *cycle-rotate-len-arcs*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \text{ mat}$
assumes $\text{length } xs > 1 \ (i, j) \in \text{arcs}' \ xs$
shows $\exists \ ys \ zs. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)$
 $\wedge \text{set } (\text{arcs } a \ a \ xs) = \text{set } (\text{arcs } i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)) \wedge xs = zs \ @ \ i \ \# \ j \ \# \ ys$
 $\langle \text{proof} \rangle$

lemma *cycle-rotate-2'*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \text{ mat}$

assumes $xs \neq [] \wedge (i, j) \in \text{set } (\text{arcs } a \ a \ xs)$
shows $\exists ys. \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys) \wedge \text{set } (i \ \# \ j \ \# \ ys) = \text{set } (a \ \# \ xs)$
 $\wedge 1 + \text{length } ys = \text{length } xs \wedge \text{set } (\text{arcs } a \ a \ xs) = \text{set } (\text{arcs } i \ i \ (j \ \# \ ys))$
 <proof>

4.5 Equivalent Characterizations of Cycle-Freeness

lemma *negative-cycle-dest-diag*:

$\neg \text{cycle-free } M \ n \implies \exists i \ xs. i \leq n \wedge \text{set } xs \subseteq \{0..n\} \wedge \text{len } M \ i \ i \ xs < \mathbf{1}$
 <proof>

abbreviation *cyc-free* :: ('a::linordered-ab-monoid-add) mat \Rightarrow nat \Rightarrow bool
where

$\text{cyc-free } m \ n \equiv \forall i \ xs. i \leq n \wedge \text{set } xs \subseteq \{0..n\} \longrightarrow \text{len } m \ i \ i \ xs \geq \mathbf{1}$

lemma *cycle-free-diag-intro*:

$\text{cyc-free } M \ n \implies \text{cycle-free } M \ n$
 <proof>

lemma *cycle-free-diag-equiv*:

$\text{cyc-free } M \ n \longleftrightarrow \text{cycle-free } M \ n$ <proof>

lemma *cycle-free-diag-dest*:

$\text{cycle-free } M \ n \implies \text{cyc-free } M \ n$
 <proof>

lemma *cyc-free-diag-dest*:

assumes $\text{cyc-free } M \ n \ i \leq n \ \text{set } xs \subseteq \{0..n\}$
shows $\text{len } M \ i \ i \ xs \geq \mathbf{1}$
 <proof>

lemma *cycle-free-0-0*:

fixes $M :: ('a::linordered-ab-monoid-add) \text{ mat}$
assumes $\text{cycle-free } M \ n$
shows $M \ 0 \ 0 \geq \mathbf{1}$
 <proof>

4.6 More Theorems Related to Floyd-Warshall

lemma *D-cycle-free-len-dest*:

$\text{cycle-free } m \ n$
 $\implies \forall i \leq n. \forall j \leq n. D \ m \ i \ j \ n = m' \ i \ j \implies i \leq n \implies j \leq n \implies \text{set}$

$xs \subseteq \{0..n\}$
 $\implies \exists ys. \text{set } ys \subseteq \{0..n\} \wedge \text{len } m' \text{ } i \text{ } j \text{ } xs = \text{len } m \text{ } i \text{ } j \text{ } ys$
 <proof>

lemma *D-cyc-free-preservation:*

$\text{cyc-free } m \text{ } n \implies \forall i \leq n. \forall j \leq n. D \text{ } m \text{ } i \text{ } j \text{ } n = m' \text{ } i \text{ } j \implies \text{cyc-free } m' \text{ } n$
 <proof>

abbreviation $FW \text{ } m \text{ } n \equiv fw \text{ } m \text{ } n \text{ } n \text{ } n$

lemma *FW-cyc-free-preservation:*

$\text{cyc-free } m \text{ } n \implies \text{cyc-free } (FW \text{ } m \text{ } n) \text{ } n$
 <proof>

lemma *cyc-free-diag-dest':*

$\text{cyc-free } m \text{ } n \implies i \leq n \implies m \text{ } i \text{ } i \geq \mathbf{1}$
 <proof>

lemma *FW-diag-neutral-preservation:*

$\forall i \leq n. M \text{ } i \text{ } i = \mathbf{1} \implies \text{cyc-free } M \text{ } n \implies \forall i \leq n. (FW \text{ } M \text{ } n) \text{ } i \text{ } i = \mathbf{1}$
 <proof>

lemma *FW-fixed-preservation:*

fixes $M :: ('a::\text{linordered-ab-monoid-add}) \text{ } mat$
assumes $A: i \leq n \implies M \text{ } 0 \text{ } i + M \text{ } i \text{ } 0 = \mathbf{1}$ *canonical* $(FW \text{ } M \text{ } n) \text{ } n \text{ } \text{cyc-free}$
 $(FW \text{ } M \text{ } n) \text{ } n$
shows $FW \text{ } M \text{ } n \text{ } 0 \text{ } i + FW \text{ } M \text{ } n \text{ } i \text{ } 0 = \mathbf{1}$ <proof>

lemma *diag-cyc-free-neutral:*

$\text{cyc-free } M \text{ } n \implies \forall k \leq n. M \text{ } k \text{ } k \leq \mathbf{1} \implies \forall i \leq n. M \text{ } i \text{ } i = \mathbf{1}$
 <proof>

lemma *fw-upd-canonical-id:*

$\text{canonical } M \text{ } n \implies i \leq n \implies j \leq n \implies k \leq n \implies fw\text{-upd } M \text{ } k \text{ } i \text{ } j = M$
 <proof>

lemma *fw-canonical-id:*

$\text{canonical } M \text{ } n \implies i \leq n \implies j \leq n \implies k \leq n \implies fw \text{ } M \text{ } n \text{ } k \text{ } i \text{ } j = M$
 <proof>

lemmas $FW\text{-canonical-id} = fw\text{-canonical-id}[OF - \text{order.refl order.refl order.refl}]$

4.7 Helper Lemmas for Bouyer's Theorem on Approximation

lemma *aux1*: $i \leq n \implies j \leq n \implies \text{set } xs \subseteq \{0..n\} \implies (a,b) \in \text{set } (\text{arcs } i \ j \ xs) \implies a \leq n \wedge b \leq n$
 ⟨proof⟩

lemma *arcs-distinct1*:

$i \notin \text{set } xs \implies j \notin \text{set } xs \implies \text{distinct } xs \implies xs \neq [] \implies (a,b) \in \text{set } (\text{arcs } i \ j \ xs) \implies a \neq b$
 ⟨proof⟩

lemma *arcs-distinct2*:

$i \notin \text{set } xs \implies j \notin \text{set } xs \implies \text{distinct } xs \implies i \neq j \implies (a,b) \in \text{set } (\text{arcs } i \ j \ xs) \implies a \neq b$
 ⟨proof⟩

lemma *arcs-distinct3*: $\text{distinct } (a \# b \# c \# xs) \implies (i,j) \in \text{set } (\text{arcs } a \ b \ xs) \implies i \neq c \wedge j \neq c$
 ⟨proof⟩

lemma *arcs-elem*:

assumes $(a, b) \in \text{set } (\text{arcs } i \ j \ xs)$ **shows** $a \in \text{set } (i \# xs) \ b \in \text{set } (j \# xs)$
 ⟨proof⟩

lemma *arcs-distinct-dest1*:

$\text{distinct } (i \# a \# zs) \implies (b,c) \in \text{set } (\text{arcs } a \ j \ zs) \implies b \neq i$
 ⟨proof⟩

lemma *arcs-distinct-fix*:

$\text{distinct } (a \# x \# xs \ @ \ [b]) \implies (a,c) \in \text{set } (\text{arcs } a \ b \ (x \# xs)) \implies c = x$
 ⟨proof⟩

lemma *disjE3*: $A \vee B \vee C \implies (A \implies G) \implies (B \implies G) \implies (C \implies G) \implies G$
 ⟨proof⟩

lemma *arcs-predecessor*:

assumes $(a, b) \in \text{set } (\text{arcs } i \ j \ xs) \ a \neq i$
shows $\exists c. (c, a) \in \text{set } (\text{arcs } i \ j \ xs)$ ⟨proof⟩

lemma *arcs-successor*:

assumes $(a, b) \in \text{set } (\text{arcs } i j xs) \ b \neq j$
shows $\exists c. (b, c) \in \text{set } (\text{arcs } i j xs) \ \langle \text{proof} \rangle$

lemma arcs-predecessor':

assumes $(a, b) \in \text{set } (\text{arcs } i j (x \# xs)) \ (a, b) \neq (i, x)$
shows $\exists c. (c, a) \in \text{set } (\text{arcs } i j (x \# xs)) \ \langle \text{proof} \rangle$

lemma arcs-cases:

assumes $(a, b) \in \text{set } (\text{arcs } i j xs) \ xs \neq []$
shows $(\exists ys. xs = b \# ys \wedge a = i) \vee (\exists ys. xs = ys @ [a] \wedge b = j)$
 $\vee (\exists c d ys. (a, b) \in \text{set } (\text{arcs } c d ys) \wedge xs = c \# ys @ [d])$
 $\langle \text{proof} \rangle$

lemma arcs-cases':

assumes $(a, b) \in \text{set } (\text{arcs } i j xs) \ xs \neq []$
shows $(\exists ys. xs = b \# ys \wedge a = i) \vee (\exists ys. xs = ys @ [a] \wedge b = j) \vee$
 $(\exists ys zs. xs = ys @ a \# b \# zs)$
 $\langle \text{proof} \rangle$

lemma arcs-successor':

assumes $(a, b) \in \text{set } (\text{arcs } i j xs) \ b \neq j$
shows $\exists c. xs = [b] \wedge a = i \vee (\exists ys. xs = b \# c \# ys \wedge a = i) \vee (\exists$
 $ys. xs = ys @ [a, b] \wedge c = j)$
 $\vee (\exists ys zs. xs = ys @ a \# b \# c \# zs)$
 $\langle \text{proof} \rangle$

lemma list-last:

$xs = [] \vee (\exists y ys. xs = ys @ [y])$
 $\langle \text{proof} \rangle$

lemma arcs-predecessor'':

assumes $(a, b) \in \text{set } (\text{arcs } i j xs) \ a \neq i$
shows $\exists c. xs = [a] \vee (\exists ys. xs = a \# b \# ys) \vee (\exists ys. xs = ys @ [c, a]$
 $\wedge b = j)$
 $\vee (\exists ys zs. xs = ys @ c \# a \# b \# zs)$
 $\langle \text{proof} \rangle$

lemma arcs-ex-middle:

$\exists b. (a, b) \in \text{set } (\text{arcs } i j (ys @ a \# xs))$
 $\langle \text{proof} \rangle$

lemma arcs-ex-head:

$\exists b. (i, b) \in \text{set } (\text{arcs } i j xs)$
 $\langle \text{proof} \rangle$

4.7.1 Successive

fun *successive* **where**

$successive \ [] = True \mid$
 $successive \ P \ [-] = True \mid$
 $successive \ P \ (x \# \ y \# \ xs) \iff \neg P \ y \wedge successive \ P \ xs \vee \neg P \ x \wedge$
 $successive \ P \ (y \# \ xs)$

lemma $\neg successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0] \langle proof \rangle$

lemma $successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $\neg successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ 0, \ Suc \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $\neg successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ 0, \ 0, \ Suc \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ 0, \ Suc \ 0, \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $\neg successive \ (\lambda x. x > (0 :: nat)) \ [Suc \ 0, \ Suc \ 0, \ 0, \ Suc \ 0] \langle proof \rangle$

lemma $successive \ (\lambda x. x > (0 :: nat)) \ [0, \ 0, \ Suc \ 0, \ 0] \langle proof \rangle$

lemma *successive-step*: $successive \ P \ (x \# \ xs) \implies \neg P \ x \implies successive \ P \ xs$
 $\langle proof \rangle$

lemma *successive-step-2*: $successive \ P \ (x \# \ y \# \ xs) \implies \neg P \ y \implies successive \ P \ xs$
 $\langle proof \rangle$

lemma *successive-stepI*:

$successive \ P \ xs \implies \neg P \ x \implies successive \ P \ (x \# \ xs)$
 $\langle proof \rangle$

theorem *list-two-induct*[*case-names Nil Single Cons*]:

fixes $P :: 'a \ list \Rightarrow bool$
and $list :: 'a \ list$
assumes $Nil: P \ []$
assumes $Single: \bigwedge x. P \ [x]$
and $Cons: \bigwedge x1 \ x2 \ xs. P \ xs \implies P \ (x2 \# \ xs) \implies P \ (x1 \# \ x2 \# \ xs)$
shows $P \ xs$
 $\langle proof \rangle$

lemma *successive-end-1*:

$successive \ P \ xs \implies \neg P \ x \implies successive \ P \ (xs \ @ \ [x])$
 $\langle proof \rangle$

lemma *successive-ends-1*:

$successive\ P\ xs \implies \neg P\ x \implies successive\ P\ ys \implies successive\ P\ (xs\ @\ x\ \# \ ys)$
 ⟨proof⟩

lemma *successive-ends-1'*:

$successive\ P\ xs \implies \neg P\ x \implies P\ y \implies \neg P\ z \implies successive\ P\ ys \implies successive\ P\ (xs\ @\ x\ \# \ y\ \# \ z\ \# \ ys)$
 ⟨proof⟩

lemma *successive-end-2*:

$successive\ P\ xs \implies \neg P\ x \implies successive\ P\ (xs\ @\ [x,y])$
 ⟨proof⟩

lemma *successive-end-2'*:

$successive\ P\ (xs\ @\ [x]) \implies \neg P\ x \implies successive\ P\ (xs\ @\ [x,y])$
 ⟨proof⟩

lemma *successive-end-3*:

$successive\ P\ (xs\ @\ [x]) \implies \neg P\ x \implies P\ y \implies \neg P\ z \implies successive\ P\ (xs\ @\ [x,y,z])$
 ⟨proof⟩

lemma *successive-step-rev*:

$successive\ P\ (xs\ @\ [x]) \implies \neg P\ x \implies successive\ P\ xs$
 ⟨proof⟩

lemma *successive-glue*:

$successive\ P\ (zs\ @\ [z]) \implies successive\ P\ (x\ \# \ xs) \implies \neg P\ z \vee \neg P\ x \implies successive\ P\ (zs\ @\ [z]\ @\ x\ \# \ xs)$
 ⟨proof⟩

lemma *successive-glue'*:

$successive\ P\ (zs\ @\ [y]) \wedge \neg P\ z \vee successive\ P\ zs \wedge \neg P\ y$
 $\implies successive\ P\ (x\ \# \ xs) \wedge \neg P\ w \vee successive\ P\ xs \wedge \neg P\ x$
 $\implies \neg P\ z \vee \neg P\ w \implies successive\ P\ (zs\ @\ y\ \# \ z\ \# \ w\ \# \ x\ \# \ xs)$
 ⟨proof⟩

lemma *successive-dest-head*:

$xs = w\ \# \ x\ \# \ ys \implies successive\ P\ xs \implies successive\ P\ (x\ \# \ xs) \wedge \neg P\ w \vee successive\ P\ xs \wedge \neg P\ x$
 ⟨proof⟩

lemma *successive-dest-tail*:

$xs = zs\ @\ [y,z] \implies successive\ P\ xs \implies successive\ P\ (zs\ @\ [y]) \wedge \neg P\ z$

\vee successive P $zs \wedge \neg P$ y
 ⟨proof⟩

lemma *successive-split*:

$xs = ys @ zs \implies$ successive P $xs \implies$ successive P $ys \wedge$ successive P zs
 ⟨proof⟩

lemma *successive-decomp*:

$xs = x \# ys @ zs @ [z] \implies$ successive P $xs \implies \neg P$ $x \vee \neg P$ $z \implies$
 successive P ($zs @ [z] @ (x \# ys)$)
 ⟨proof⟩

lemma *successive-predecessor*:

assumes $(a, b) \in \text{set} (\text{arcs } i j xs)$ $a \neq i$ successive P ($\text{arcs } i j xs$) P (a, b)
 $xs \neq []$
shows $\exists c. (xs = [a] \wedge c = i \vee (\exists ys. xs = a \# b \# ys \wedge c = i) \vee (\exists$
 $ys. xs = ys @ [c, a] \wedge b = j)$
 $\vee (\exists ys zs. xs = ys @ c \# a \# b \# zs)) \wedge \neg P$ (c, a)
 ⟨proof⟩

thm *arcs-successor'*

lemma *successive-successor*:

assumes $(a, b) \in \text{set} (\text{arcs } i j xs)$ $b \neq j$ successive P ($\text{arcs } i j xs$) P (a, b)
 $xs \neq []$
shows $\exists c. (xs = [b] \wedge c = j \vee (\exists ys. xs = b \# c \# ys) \vee (\exists ys. xs =$
 $ys @ [a, b] \wedge c = j)$
 $\vee (\exists ys zs. xs = ys @ a \# b \# c \# zs)) \wedge \neg P$ (b, c)
 ⟨proof⟩

lemmas *add-mono-right* = *add-mono*[*OF order-refl*]

lemmas *add-mono-left* = *add-mono*[*OF - order-refl*]

Obtaining successive and distinct paths **lemma** *canonical-successive*:

fixes $A B$
defines $M \equiv \lambda i j. \text{min} (A i j) (B i j)$
assumes *canonical* $A n$
assumes $\text{set } xs \subseteq \{0..n\}$
assumes $i \leq n$ $j \leq n$
shows $\exists ys. \text{len } M i j ys \leq \text{len } M i j xs \wedge \text{set } ys \subseteq \{0..n\}$
 \wedge successive $(\lambda (a, b). M a b = A a b)$ ($\text{arcs } i j ys$)
 ⟨proof⟩

lemma *canonical-successive-distinct*:

fixes $A B$

defines $M \equiv \lambda i j. \min (A i j) (B i j)$

assumes *canonical* $A n$

assumes *set* $xs \subseteq \{0..n\}$

assumes $i \leq n \ j \leq n$

assumes *distinct* $xs \ i \notin set \ xs \ j \notin set \ xs$

shows $\exists ys. \text{len } M \ i \ j \ ys \leq \text{len } M \ i \ j \ xs \wedge \text{set } ys \subseteq \text{set } xs$
 $\wedge \text{successive } (\lambda (a, b). M \ a \ b = A \ a \ b) \ (\text{arcs } i \ j \ ys)$
 $\wedge \text{distinct } ys \wedge i \notin \text{set } ys \wedge j \notin \text{set } ys$

<proof>

lemma *successive-snd-last*: *successive* $P \ (xs \ @ \ [x, y]) \implies P \ y \implies \neg P \ x$

<proof>

lemma *canonical-shorten-rotate-neg-cycle*:

fixes $A B$

defines $M \equiv \lambda i j. \min (A i j) (B i j)$

assumes *canonical* $A n$

assumes *set* $xs \subseteq \{0..n\}$

assumes $i \leq n$

assumes $\text{len } M \ i \ i \ xs < \mathbf{1}$

shows $\exists j \ ys. \text{len } M \ j \ j \ ys < \mathbf{1} \wedge \text{set } (j \ # \ ys) \subseteq \text{set } (i \ # \ xs)$
 $\wedge \text{successive } (\lambda (a, b). M \ a \ b = A \ a \ b) \ (\text{arcs } j \ j \ ys)$
 $\wedge \text{distinct } ys \wedge j \notin \text{set } ys \wedge$
 $(ys \neq [] \implies M \ j \ (\text{hd } ys) \neq A \ j \ (\text{hd } ys) \vee M \ (\text{last } ys) \ j \neq A$

$(\text{last } ys) \ j)$

<proof>

lemma *successive-arcs-extend-last*:

successive $P \ (\text{arcs } i \ j \ xs) \implies \neg P \ (i, \text{hd } xs) \vee \neg P \ (\text{last } xs, j) \implies xs \neq$

$[]$

$\implies \text{successive } P \ (\text{arcs } i \ j \ xs \ @ \ [(i, \text{hd } xs)])$

<proof>

lemma *cycle-rotate-arcs*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \ \text{mat}$

assumes $\text{length } xs > 1 \ (i, j) \in \text{arcs}' \ xs$

shows $\exists ys \ zs. \text{set } (\text{arcs } a \ a \ xs) = \text{set } (\text{arcs } i \ i \ (j \ # \ ys \ @ \ a \ # \ zs)) \wedge xs$
 $= zs \ @ \ i \ # \ j \ # \ ys$ *<proof>*

lemma *cycle-rotate-len-arcs-successive*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \ \text{mat}$

assumes $\text{length } xs > 1 \ (i, j) \in \text{arcs}' \ xs \ \text{successive } P \ (\text{arcs } a \ a \ xs) \ \neg P$
 $(a, \text{hd } xs) \vee \neg P \ (\text{last } xs, a)$
shows $\exists \ ys \ zs. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)$
 $\wedge \ \text{set} \ (\text{arcs } a \ a \ xs) = \text{set} \ (\text{arcs } i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)) \wedge \ xs =$
 $zs \ @ \ i \ \# \ j \ \# \ ys$
 $\wedge \ \text{successive } P \ (\text{arcs } i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs))$
 <proof>

lemma *successive-successors*:

$xs = ys \ @ \ a \ \# \ b \ \# \ c \ \# \ zs \implies \text{successive } P \ (\text{arcs } i \ j \ xs) \implies \neg P \ (a, b)$
 $\vee \neg P \ (b, c)$
 <proof>

lemma *successive-successors'*:

$xs = ys \ @ \ a \ \# \ b \ \# \ zs \implies \text{successive } P \ xs \implies \neg P \ a \ \vee \neg P \ b$
 <proof>

lemma *cycle-rotate-len-arcs-successive'*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \ \text{mat}$
assumes $\text{length } xs > 1 \ (i, j) \in \text{arcs}' \ xs \ \text{successive } P \ (\text{arcs } a \ a \ xs)$
 $\neg P \ (a, \text{hd } xs) \vee \neg P \ (\text{last } xs, a)$
shows $\exists \ ys \ zs. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)$
 $\wedge \ \text{set} \ (\text{arcs } a \ a \ xs) = \text{set} \ (\text{arcs } i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs)) \wedge \ xs =$
 $zs \ @ \ i \ \# \ j \ \# \ ys$
 $\wedge \ \text{successive } P \ (\text{arcs } i \ i \ (j \ \# \ ys \ @ \ a \ \# \ zs) \ @ \ [(i, j)])$
 <proof>

lemma *cycle-rotate-3*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \ \text{mat}$
assumes $xs \neq [] \ (i, j) \in \text{set} \ (\text{arcs } a \ a \ xs) \ \text{successive } P \ (\text{arcs } a \ a \ xs) \ \neg P$
 $(a, \text{hd } xs) \vee \neg P \ (\text{last } xs, a)$
shows $\exists \ ys. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys) \wedge \ \text{set} \ (i \ \# \ j \ \# \ ys) = \text{set}$
 $(a \ \# \ xs) \wedge 1 + \text{length } ys = \text{length } xs$
 $\wedge \ \text{set} \ (\text{arcs } a \ a \ xs) = \text{set} \ (\text{arcs } i \ i \ (j \ \# \ ys))$
 $\wedge \ \text{successive } P \ (\text{arcs } i \ i \ (j \ \# \ ys))$
 <proof>

lemma *cycle-rotate-3'*:

fixes $M :: ('a :: \text{linordered-ab-monoid-add}) \ \text{mat}$
assumes $xs \neq [] \ (i, j) \in \text{set} \ (\text{arcs } a \ a \ xs) \ \text{successive } P \ (\text{arcs } a \ a \ xs) \ \neg P$
 $(a, \text{hd } xs) \vee \neg P \ (\text{last } xs, a)$
shows $\exists \ ys. \ \text{len } M \ a \ a \ xs = \text{len } M \ i \ i \ (j \ \# \ ys) \wedge \ \text{set} \ (i \ \# \ j \ \# \ ys) = \text{set}$
 $(a \ \# \ xs) \wedge 1 + \text{length } ys = \text{length } xs$
 $\wedge \ \text{set} \ (\text{arcs } a \ a \ xs) = \text{set} \ (\text{arcs } i \ i \ (j \ \# \ ys))$

\wedge successive P (arcs i i (j # ys) @ $[(i, j)]$)

\langle proof \rangle

end

theory *DBM-Basics*

imports *DBM Paths-Cycles*

begin

fun *get-const* **where**

get-const (*Le* c) = c |

get-const (*Lt* c) = c |

get-const ∞ = *undefined*

4.7.2 Discourse on updating DBMs

abbreviation *DBM-update* :: (' t ::time) *DBM* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow (' t *DBMEntry*) \Rightarrow (' t ::time) *DBM*

where

DBM-update M m n v \equiv (λ x y . if $m = x \wedge n = y$ then v else M x y)

fun *DBM-upd* :: (' t ::time) *DBM* \Rightarrow (*nat* \Rightarrow *nat* \Rightarrow ' t *DBMEntry*) \Rightarrow *nat* \Rightarrow *nat* \Rightarrow ' t *DBM*

where

DBM-upd M f 0 0 - = *DBM-update* M 0 0 (f 0 0) |

DBM-upd M f (*Suc* i) 0 n = *DBM-update* (*DBM-upd* M f i n) (*Suc* i) 0 (f (*Suc* i) 0) |

DBM-upd M f i (*Suc* j) n = *DBM-update* (*DBM-upd* M f i j) i (*Suc* j) (f i (*Suc* j))

lemma *upd-1*:

assumes $j \leq n$

shows *DBM-upd* $M1$ f (*Suc* m) n N (*Suc* m) j = *DBM-upd* $M1$ f (*Suc* m) j N (*Suc* m) j

\langle proof \rangle

lemma *upd-2*:

assumes $i \leq m$

shows *DBM-upd* $M1$ f (*Suc* m) n N i j = *DBM-upd* $M1$ f (*Suc* m) 0 N i j

\langle proof \rangle

lemma *upd-3*:

assumes $m \leq N$ $n \leq N$ $j \leq n$ $i \leq m$

shows (*DBM-upd* $M1$ f m n N) i j = (*DBM-upd* $M1$ f i j N) i j

\langle proof \rangle

lemma *upd-id*:

assumes $m \leq N \ n \leq N \ i \leq m \ j \leq n$
shows $(DBM\text{-}upd \ M1 \ f \ m \ n \ N) \ i \ j = f \ i \ j$
 $\langle proof \rangle$

4.7.3 Zones and DBMs

definition *DBM-zone-repr* :: $(t::time) \ DBM \Rightarrow (c \Rightarrow nat) \Rightarrow nat \Rightarrow (c,$
 $t :: time) \ zone$
 $([]_{-}, - [72, 72, 72] \ 72)$

where

$[M]_{v,n} = \{u \ . \ DBM\text{-}val\text{-}bounded \ v \ u \ M \ n\}$

lemma *dbm-entry-val-mono-1*:

$dbm\text{-}entry\text{-}val \ u \ (Some \ c) \ (Some \ c') \ b \Longrightarrow b \preceq b' \Longrightarrow dbm\text{-}entry\text{-}val \ u$
 $(Some \ c) \ (Some \ c') \ b'$
 $\langle proof \rangle$

lemma *dbm-entry-val-mono-2*:

$dbm\text{-}entry\text{-}val \ u \ None \ (Some \ c) \ b \Longrightarrow b \preceq b' \Longrightarrow dbm\text{-}entry\text{-}val \ u \ None$
 $(Some \ c) \ b'$
 $\langle proof \rangle$

lemma *dbm-entry-val-mono-3*:

$dbm\text{-}entry\text{-}val \ u \ (Some \ c) \ None \ b \Longrightarrow b \preceq b' \Longrightarrow dbm\text{-}entry\text{-}val \ u \ (Some$
 $c) \ None \ b'$
 $\langle proof \rangle$

lemma *DBM-le-subset*:

$\forall \ i \ j. \ i \leq n \longrightarrow j \leq n \longrightarrow M \ i \ j \preceq M' \ i \ j \Longrightarrow u \in [M]_{v,n} \Longrightarrow u \in$
 $[M']_{v,n}$
 $\langle proof \rangle$

4.7.4 DBMs Without Negative Cycles are Non-Empty

We need all of these assumptions for the proof that matrices without negative cycles represent non-negative zones: * Abelian (linearly ordered) monoid * Time is non-trivial * Time is dense

lemmas (in *linordered-ab-monoid-add*) $comm = add.commute$

lemma *sum-gt-neutral-dest'*:

$(a :: (t::time) \ DBMEntry) \geq \mathbf{1} \Longrightarrow a + b > \mathbf{1} \Longrightarrow \exists \ d. \ Le \ d \leq a \wedge$
 $Le \ (-d) \leq b \wedge d \geq 0$

<proof>

lemma *sum-gt-neutral-dest*:

$(a :: (('a :: time) DBMEntry)) + b > \mathbf{1} \implies \exists d. Le\ d \leq a \wedge Le\ (-d) \leq b$

<proof>

4.7.5 Negative Cycles in DBMs

lemma *DBM-val-bounded-neg-cycle1*:

fixes $i\ xs$ **assumes**

$bounded: DBM\text{-val-bounded}\ v\ u\ M\ n$ **and** $A:i \leq n$ $set\ xs \subseteq \{0..n\}$ $len\ M\ i\ i\ xs < \mathbf{1}$ **and**

$surj\text{-on}: \forall k \leq n. k > 0 \implies (\exists c. v\ c = k)$ **and** $at\text{-most}: i \neq 0\ cnt\ 0\ xs \leq 1$

shows *False*

<proof>

lemma *cnt-0-I*:

$x \notin set\ xs \implies cnt\ x\ xs = 0$

<proof>

lemma *distinct-cnt*: $distinct\ xs \implies cnt\ x\ xs \leq 1$

<proof>

lemma *DBM-val-bounded-neg-cycle*:

fixes $i\ xs$ **assumes**

$bounded: DBM\text{-val-bounded}\ v\ u\ M\ n$ **and** $A:i \leq n$ $set\ xs \subseteq \{0..n\}$ $len\ M\ i\ i\ xs < \mathbf{1}$ **and**

$surj\text{-on}: \forall k \leq n. k > 0 \implies (\exists c. v\ c = k)$

shows *False*

<proof>

4.7.6 Floyd-Warshall Algorithm Preservers Zones

lemma *D-dest*: $x = D\ m\ i\ j\ k \implies$

$x \in \{len\ m\ i\ j\ xs\ |xs. set\ xs \subseteq \{0..k\} \wedge i \notin set\ xs \wedge j \notin set\ xs \wedge distinct\ xs\}$

<proof>

lemma *FW-zone-equiv*:

$\forall k \leq n. k > 0 \implies (\exists c. v\ c = k) \implies [M]_{v,n} = [FW\ M\ n]_{v,n}$

<proof>

lemma *new-negative-cycle-aux'*:

fixes $M :: ('a :: \text{time}) \text{DBM}$

fixes $i \ j \ d$

defines $M' \equiv \lambda \ i' \ j'. \text{if } (i' = i \wedge j' = j) \text{ then } Le \ d$
 else if $(i' = j \wedge j' = i) \text{ then } Le \ (-d)$
 else $M \ i' \ j'$

assumes $i \leq n \ j \leq n \ \text{set } xs \subseteq \{0..n\} \ \text{cycle-free } M \ n \ \text{length } xs = m$

assumes $\text{len } M' \ i \ i \ (j \ \# \ xs) < \mathbf{1} \vee \text{len } M' \ j \ j \ (i \ \# \ xs) < \mathbf{1}$

assumes $i \neq j$

shows $\exists xs. \text{set } xs \subseteq \{0..n\} \wedge j \notin \text{set } xs \wedge i \notin \text{set } xs$
 $\wedge (\text{len } M' \ i \ i \ (j \ \# \ xs) < \mathbf{1} \vee \text{len } M' \ j \ j \ (i \ \# \ xs) < \mathbf{1}) \langle \text{proof} \rangle$

lemma *new-negative-cycle-aux*:

fixes $M :: ('a :: \text{time}) \text{DBM}$

fixes $i \ d$

defines $M' \equiv \lambda \ i' \ j'. \text{if } (i' = i \wedge j' = 0) \text{ then } Le \ d$
 else if $(i' = 0 \wedge j' = i) \text{ then } Le \ (-d)$
 else $M \ i' \ j'$

assumes $i \leq n \ \text{set } xs \subseteq \{0..n\} \ \text{cycle-free } M \ n \ \text{length } xs = m$

assumes $\text{len } M' \ 0 \ 0 \ (i \ \# \ xs) < \mathbf{1} \vee \text{len } M' \ i \ i \ (0 \ \# \ xs) < \mathbf{1}$

assumes $i \neq 0$

shows $\exists xs. \text{set } xs \subseteq \{0..n\} \wedge 0 \notin \text{set } xs \wedge i \notin \text{set } xs$
 $\wedge (\text{len } M' \ 0 \ 0 \ (i \ \# \ xs) < \mathbf{1} \vee \text{len } M' \ i \ i \ (0 \ \# \ xs) < \mathbf{1}) \langle \text{proof} \rangle$

4.8 The Characteristic Property of Canonical DBMs

theorem *fix-index'*:

fixes $M :: (('a :: \text{time}) \text{DBMEntry}) \text{mat}$

assumes $Le \ r \leq M \ i \ j \ Le \ (-r) \leq M \ j \ i \ \text{cycle-free } M \ n \ \text{canonical } M \ n \ i \leq n \ j \leq n \ i \neq j$

defines $M' \equiv \lambda \ i' \ j'. \text{if } (i' = i \wedge j' = j) \text{ then } Le \ r$
 else if $(i' = j \wedge j' = i) \text{ then } Le \ (-r)$
 else $M \ i' \ j'$

shows $(\forall u. \text{DBM-val-bounded } v \ u \ M' \ n \longrightarrow \text{DBM-val-bounded } v \ u \ M \ n)$
 $\wedge \text{cycle-free } M' \ n$
 $\langle \text{proof} \rangle$

lemma *fix-index*:

fixes $M :: (('a :: \text{time}) \text{DBMEntry}) \text{mat}$

assumes $M \ 0 \ i + M \ i \ 0 > \mathbf{1} \ \text{cycle-free } M \ n \ \text{canonical } M \ n \ i \leq n \ i \neq 0$

shows

$\exists (M' :: ('a \ \text{DBMEntry}) \text{mat}). ((\exists u. \text{DBM-val-bounded } v \ u \ M' \ n) \longrightarrow$
 $(\exists u. \text{DBM-val-bounded } v \ u \ M \ n))$
 $\wedge M' \ 0 \ i + M' \ i \ 0 = \mathbf{1} \wedge \text{cycle-free } M' \ n$

$\wedge (\forall j. i \neq j \wedge M \ 0 \ j + M \ j \ 0 = \mathbf{1} \longrightarrow M' \ 0 \ j + M' \ j \ 0 = \mathbf{1})$
 $\wedge (\forall j. i \neq j \wedge M \ 0 \ j + M \ j \ 0 > \mathbf{1} \longrightarrow M' \ 0 \ j + M' \ j \ 0 > \mathbf{1})$
 <proof>

Putting it together lemma FW-not-empty:

DBM-val-bounded $v \ u \ (FW \ M' \ n) \ n \Longrightarrow DBM\text{-val-bounded } v \ u \ M' \ n$
 <proof>

lemma fix-indices:

fixes $M :: ('a :: time) \ DBMEntry) \ mat$
assumes $set \ xs \subseteq \{0..n\}$ *distinct xs*
assumes *cyc-free M n canonical M n*
shows
 $\exists (M' :: ('a \ DBMEntry) \ mat). ((\exists \ u. \ DBM\text{-val-bounded } v \ u \ M' \ n) \longrightarrow$
 $(\exists \ u. \ DBM\text{-val-bounded } v \ u \ M \ n))$
 $\wedge (\forall \ i \in \ set \ xs. \ i \neq 0 \longrightarrow M' \ 0 \ i + M' \ i \ 0 = \mathbf{1}) \wedge \text{cyc-free } M' \ n$
 $\wedge (\forall \ i \leq n. \ i \notin \ set \ xs \wedge M \ 0 \ i + M \ i \ 0 = \mathbf{1} \longrightarrow M' \ 0 \ i + M' \ i \ 0 = \mathbf{1})$
 <proof>

lemma cyc-free-obtains-valuation:

$\text{cyc-free } M \ n \Longrightarrow \forall \ c. \ v \ c \leq n \longrightarrow v \ c > 0 \Longrightarrow \exists \ u. \ DBM\text{-val-bounded}$
 $v \ u \ M \ n$
 <proof>

4.8.1 Floyd-Warshall and Empty DBMs

theorem FW-detects-empty-zone:

$\forall k \leq n. \ 0 < k \longrightarrow (\exists \ c. \ v \ c = k) \Longrightarrow \forall \ c. \ v \ c \leq n \longrightarrow v \ c > 0$
 $\Longrightarrow [FW \ M \ n]_{v,n} = \{\}$ $\longleftrightarrow (\exists \ i \leq n. \ (FW \ M \ n) \ i \ i < Le \ 0)$
 <proof>

hide-const D

4.8.2 Mixed Corollaries

lemma cyc-free-not-empty:

assumes $\text{cyc-free } M \ n \ \forall \ c. \ v \ c \leq n \longrightarrow 0 < v \ c$
shows $[(M :: ('a :: time) \ DBM)]_{v,n} \neq \{\}$
 <proof>

lemma empty-not-cyc-free:

assumes $\forall \ c. \ v \ c \leq n \longrightarrow 0 < v \ c \ [(M :: ('a :: time) \ DBM)]_{v,n} = \{\}$
shows $\neg \text{cyc-free } M \ n$

$\langle proof \rangle$

lemma *not-empty-cyc-free*:

assumes $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k) [(M :: ('a :: time) DBM)]_{v,n} \neq \{\}$
shows *cyc-free* $M n$ $\langle proof \rangle$

lemma *neg-cycle-empty*:

assumes $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k) \text{ set } xs \subseteq \{0..n\} \ i \leq n \ \text{len } M \ i \ i \ xs < \mathbf{1}$
shows $[(M :: ('a :: time) DBM)]_{v,n} = \{\}$ $\langle proof \rangle$

abbreviation *clock-numbering'* $:: ('c \Rightarrow nat) \Rightarrow nat \Rightarrow bool$

where

clock-numbering' $v n \equiv \forall c. v c > 0 \wedge (\forall x. \forall y. v x \leq n \wedge v y \leq n \wedge v x = v y \longrightarrow x = y)$

lemma *non-empty-dbm-diag-set*:

clock-numbering' $v n \Longrightarrow [M]_{v,n} \neq \{\} \Longrightarrow [M]_{v,n} = [(\lambda i j. \text{if } i = j \text{ then } \mathbf{1} \text{ else } M \ i \ j)]_{v,n}$
 $\langle proof \rangle$

lemma *non-empty-cycle-free*:

assumes $[M]_{v,n} \neq \{\}$
and $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k)$
shows *cycle-free* $M n$
 $\langle proof \rangle$

lemma *neg-diag-empty*:

assumes $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k) \ i \leq n \ M \ i \ i < \mathbf{1}$
shows $[M]_{v,n} = \{\}$
 $\langle proof \rangle$

lemma *canonical-empty-zone*:

assumes $\forall k \leq n. 0 < k \longrightarrow (\exists c. v c = k) \ \forall c. v c \leq n \longrightarrow 0 < v c$
and *canonical* $M n$
shows $[M]_{v,n} = \{\} \longleftrightarrow (\exists i \leq n. M \ i \ i < \mathbf{1})$
 $\langle proof \rangle$

end

5 Forward Analysis on DBMs

theory *DBM-Operations*
 imports *DBM-Basics*
begin

5.1 Auxiliary

lemma *gt-swap*:
 fixes $a\ b\ c :: 't :: time$
 assumes $c < a + b$
 shows $c < b + a$
 $\langle proof \rangle$

lemma *le-swap*:
 fixes $a\ b\ c :: 't :: time$
 assumes $c \leq a + b$
 shows $c \leq b + a$
 $\langle proof \rangle$

abbreviation *clock-numbering* $:: ('c \Rightarrow nat) \Rightarrow bool$
where
 clock-numbering $v \equiv \forall c. v\ c > 0$

5.2 Time Lapse

definition *up* $:: ('t::time)\ DBM \Rightarrow ('t::time)\ DBM$
where

$up\ M \equiv$
 $\lambda\ i\ j. \text{if } i > 0 \text{ then if } j = 0 \text{ then } \infty \text{ else } \min(\text{dbm-add } (M\ i\ 0)\ (M\ 0\ j))\ (M\ i\ j) \text{ else } M\ i\ j$

lemma *dbm-entry-dbm-lt*:
 assumes $\text{dbm-entry-val } u\ (\text{Some } c1)\ (\text{Some } c2)\ a\ a \prec b$
 shows $\text{dbm-entry-val } u\ (\text{Some } c1)\ (\text{Some } c2)\ b$
 $\langle proof \rangle$

lemma *dbm-entry-dbm-min2*:
 assumes $\text{dbm-entry-val } u\ \text{None}\ (\text{Some } c)\ (\min\ a\ b)$
 shows $\text{dbm-entry-val } u\ \text{None}\ (\text{Some } c)\ b$
 $\langle proof \rangle$

lemma *dbm-entry-dbm-min3*:
 assumes $\text{dbm-entry-val } u\ (\text{Some } c)\ \text{None}\ (\min\ a\ b)$

shows *dbm-entry-val* *u* (*Some c*) *None b*
 ⟨*proof*⟩

lemma *dbm-entry-dbm-min*:
assumes *dbm-entry-val* *u* (*Some c1*) (*Some c2*) (*min a b*)
shows *dbm-entry-val* *u* (*Some c1*) (*Some c2*) *b*
 ⟨*proof*⟩

lemma *dbm-entry-dbm-min3'*:
assumes *dbm-entry-val* *u* (*Some c*) *None* (*min a b*)
shows *dbm-entry-val* *u* (*Some c*) *None a*
 ⟨*proof*⟩

lemma *dbm-entry-dbm-min2'*:
assumes *dbm-entry-val* *u* *None* (*Some c*) (*min a b*)
shows *dbm-entry-val* *u* *None* (*Some c*) *a*
 ⟨*proof*⟩

lemma *dbm-entry-dbm-min'*:
assumes *dbm-entry-val* *u* (*Some c1*) (*Some c2*) (*min a b*)
shows *dbm-entry-val* *u* (*Some c1*) (*Some c2*) *a*
 ⟨*proof*⟩

lemma *DBM-up-complete'*: *clock-numbering* *v* $\implies u \in ([M]_{v,n})^\uparrow \implies u \in [up\ M]_{v,n}$
 ⟨*proof*⟩

fun *theLe* :: (*t*::*time*) *DBMEntry* \Rightarrow *t* **where**
theLe (*Le d*) = *d* |
theLe (*Lt d*) = *d* |
theLe ∞ = 0

lemma *DBM-up-sound'*:
assumes *clock-numbering'* *v n u* $\in [up\ M]_{v,n}$
shows *u* $\in ([M]_{v,n})^\uparrow$
 ⟨*proof*⟩

5.3 From Clock Constraints to DBMs

fun *And* :: (*t* :: *time*) *DBM* \Rightarrow *t* *DBM* \Rightarrow *t* *DBM* **where**
And *M1 M2* = (λ *i j*. *min* (*M1 i j*) (*M2 i j*))

fun *abstr* :: (*c*, *t*::*time*) *cconstraint* \Rightarrow *t* *DBM* \Rightarrow (*c* \Rightarrow *nat*) \Rightarrow *t* *DBM*
where

$$\begin{aligned}
& \text{abstr } (\text{AND } cc1 \ cc2) \ M \ v = \text{And } (\text{abstr } cc1 \ M \ v) \ (\text{abstr } cc2 \ M \ v) \ | \\
& \text{abstr } (\text{EQ } c \ d) \ M \ v = \\
& \quad (\lambda \ i \ j . \text{if } i = 0 \wedge j = v \ c \ \text{then } Le \ (-d) \ \text{else if } i = v \ c \wedge j = 0 \ \text{then } Le \\
& \quad d \ \text{else } M \ i \ j) \ | \\
& \text{abstr } (\text{LT } c \ d) \ M \ v = \\
& \quad (\lambda \ i \ j . \text{if } i = 0 \wedge j = v \ c \ \text{then } \infty \ \text{else if } i = v \ c \wedge j = 0 \ \text{then } Lt \ d \ \text{else} \\
& \quad M \ i \ j) \ | \\
& \text{abstr } (\text{LE } c \ d) \ M \ v = \\
& \quad (\lambda \ i \ j . \text{if } i = 0 \wedge j = v \ c \ \text{then } \infty \ \text{else if } i = v \ c \wedge j = 0 \ \text{then } Le \ d \ \text{else} \\
& \quad M \ i \ j) \ | \\
& \text{abstr } (\text{GT } c \ d) \ M \ v = \\
& \quad (\lambda \ i \ j . \text{if } i = 0 \wedge j = v \ c \ \text{then } Lt \ (- \ d) \ \text{else if } i = v \ c \wedge j = 0 \ \text{then } \infty \\
& \quad \text{else } M \ i \ j) \ | \\
& \text{abstr } (\text{GE } c \ d) \ M \ v = \\
& \quad (\lambda \ i \ j . \text{if } i = 0 \wedge j = v \ c \ \text{then } Le \ (- \ d) \ \text{else if } i = v \ c \wedge j = 0 \ \text{then } \infty \\
& \quad \text{else } M \ i \ j)
\end{aligned}$$

lemma *abstr-id1*:

$$\begin{aligned}
& c \notin \text{collect-clks } cc \implies \text{clock-numbering}' \ v \ n \implies \forall \ c \in \text{collect-clks } cc. \ v \ c \\
& \leq n \\
& \implies \text{abstr } cc \ M \ v \ 0 \ (v \ c) = M \ 0 \ (v \ c) \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma *abstr-id2*:

$$\begin{aligned}
& c \notin \text{collect-clks } cc \implies \text{clock-numbering}' \ v \ n \implies \forall \ c \in \text{collect-clks } cc. \ v \ c \\
& \leq n \\
& \implies \text{abstr } cc \ M \ v \ (v \ c) \ 0 = M \ (v \ c) \ 0 \\
& \langle \text{proof} \rangle
\end{aligned}$$

This lemma is trivial because we constrained our theory to difference constraints.

lemma *abstr-id3*:

$$\begin{aligned}
& \text{clock-numbering } v \implies \text{abstr } cc \ M \ v \ (v \ c1) \ (v \ c2) = M \ (v \ c1) \ (v \ c2) \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma *dbm-abstr-soundness* :

$$\begin{aligned}
& \llbracket u \vdash cc; \text{clock-numbering}' \ v \ n; \forall \ c \in \text{collect-clks } cc. \ v \ c \leq n \rrbracket \\
& \implies \text{DBM-val-bounded } v \ u \ (\text{abstr } cc \ (\lambda \ i \ j . \infty) \ v) \ n \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma *dbm-abstr-completeness*:

$$\begin{aligned}
& \llbracket \text{DBM-val-bounded } v \ u \ (\text{abstr } cc \ (\lambda \ i \ j . \infty) \ v) \ n; \forall \ c. \ v \ c > 0; \forall \ c \in \\
& \text{collect-clks } cc. \ v \ c \leq n \rrbracket \\
& \implies u \vdash cc
\end{aligned}$$

$\langle proof \rangle$

lemma *dbm-abstr-zone-eq*:

assumes *clock-numbering'* $v\ n\ \forall c \in collect-clks\ cc.\ v\ c \leq n$
shows $[abstr\ cc\ (\lambda i\ j.\ \infty)\ v]_{v,n} = \{u.\ u \vdash cc\}$

$\langle proof \rangle$

5.4 Zone Intersection

lemma *DBM-and-complete*:

assumes *DBM-val-bounded* $v\ u\ M1\ n$ *DBM-val-bounded* $v\ u\ M2\ n$
shows *DBM-val-bounded* $v\ u\ (And\ M1\ M2)\ n$

$\langle proof \rangle$

lemma *DBM-and-sound1*:

assumes *DBM-val-bounded* $v\ u\ (And\ M1\ M2)\ n$
shows *DBM-val-bounded* $v\ u\ M1\ n$

$\langle proof \rangle$

lemma *DBM-and-sound2*:

assumes *DBM-val-bounded* $v\ u\ (And\ M1\ M2)\ n$
shows *DBM-val-bounded* $v\ u\ M2\ n$

$\langle proof \rangle$

5.5 Clock Reset

definition

DBM-reset $:: ('t :: time)\ DBM \Rightarrow nat \Rightarrow nat \Rightarrow 't \Rightarrow 't\ DBM \Rightarrow bool$

where

$DBM-reset\ M\ n\ k\ d\ M' \equiv$
 $(\forall j \leq n.\ 0 < j \wedge k \neq j \longrightarrow M' k j = \infty \wedge M' j k = \infty) \wedge M' k 0$
 $= Le\ d \wedge M' 0 k = Le\ (-\ d)$
 $\wedge M' k k = M k k$
 $\wedge (\forall i \leq n.\ \forall j \leq n.$
 $i \neq k \wedge j \neq k \longrightarrow M' i j = min\ (dbm-add\ (M\ i\ k)\ (M\ k\ j))\ (M\ i\ j))$

lemma *DBM-reset-mono*:

assumes *DBM-reset* $M\ n\ k\ d\ M'\ i \leq n\ j \leq n\ i \neq k\ j \neq k$
shows $M' i j \leq M i j$

$\langle proof \rangle$

lemma *DBM-reset-len-mono*:

assumes *DBM-reset* $M\ n\ k\ d\ M'\ k \notin set\ xs\ i \neq k\ j \neq k\ set\ (i \# j \# xs)$

$\subseteq \{0..n\}$
shows $len\ M'\ i\ j\ xs \leq len\ M\ i\ j\ xs$
 $\langle proof \rangle$

lemma *DBM-reset-neg-cycle-preservation:*

assumes *DBM-reset* $M\ n\ k\ d\ M'\ len\ M\ i\ i\ xs < Le\ 0\ set\ (k\ \#\ i\ \#\ xs) \subseteq \{0..n\}$

shows $\exists j. \exists ys. set\ (j\ \#\ ys) \subseteq \{0..n\} \wedge len\ M'\ j\ j\ ys < Le\ 0$
 $\langle proof \rangle$

Implementation of DBM reset

definition *reset* $:: ('t::time)\ DBM \Rightarrow nat \Rightarrow nat \Rightarrow 't \Rightarrow 't\ DBM$

where

$reset\ M\ n\ k\ d =$
 $(\lambda\ i\ j.$
 if $i = k \wedge j = 0$ then $Le\ d$
 else if $i = 0 \wedge j = k$ then $Le\ (-d)$
 else if $i = k \wedge j \neq k$ then ∞
 else if $i \neq k \wedge j = k$ then ∞
 else if $i = k \wedge j = k$ then $M\ k\ k$
 else $min\ (dbm-add\ (M\ i\ k)\ (M\ k\ j))\ (M\ i\ j)$
 $)$

fun *reset'* $:: ('t::time)\ DBM \Rightarrow nat \Rightarrow 'c\ list \Rightarrow ('c \Rightarrow nat) \Rightarrow 't \Rightarrow 't\ DBM$

where

$reset'\ M\ n\ []\ v\ d = M\ |$
 $reset'\ M\ n\ (c\ \#\ cs)\ v\ d = reset\ (reset'\ M\ n\ cs\ v\ d)\ n\ (v\ c)\ d$

lemma *DBM-reset-reset:*

$0 < k \implies k \leq n \implies DBM-reset\ M\ n\ k\ d\ (reset\ M\ n\ k\ d)$
 $\langle proof \rangle$

lemma *DBM-reset-complete:*

assumes *clock-numbering'* $v\ n\ v\ c \leq n$ *DBM-reset* $M\ n\ (v\ c)\ d\ M'$
DBM-val-bounded $v\ u\ M\ n$

shows *DBM-val-bounded* $v\ (u(c := d))\ M'\ n$
 $\langle proof \rangle$

lemma *DBM-reset-sound-empty:*

assumes *clock-numbering'* $v\ n\ v\ c \leq n$ *DBM-reset* $M\ n\ (v\ c)\ d\ M'$
 $\forall u. \neg DBM-val-bounded\ v\ u\ M'\ n$

shows $\neg DBM-val-bounded\ v\ u\ M\ n$
 $\langle proof \rangle$

lemma *DBM-reset-diag-preservation*:

$\forall k \leq n. M \ k \ k \leq \mathbf{1} \implies \text{DBM-reset } M \ n \ i \ d \ M' \implies \forall k \leq n. M' \ k \ k \leq \mathbf{1}$
 ⟨proof⟩

lemma *FW-diag-preservation*:

$\forall k \leq n. M \ k \ k \leq \mathbf{1} \implies \forall k \leq n. (\text{FW } M \ n) \ k \ k \leq \mathbf{1}$
 ⟨proof⟩

lemma *DBM-reset-not-cyc-free-preservation*:

assumes $\neg \text{cyc-free } M \ n \ \text{DBM-reset } M \ n \ k \ d \ M' \ k \leq n$
shows $\neg \text{cyc-free } M' \ n$
 ⟨proof⟩

lemma *DBM-reset-complete-empty'*:

assumes $\forall k \leq n. k > 0 \implies (\exists c. v \ c = k) \text{ clock-numbering } v \ k \leq n$
 $\text{DBM-reset } M \ n \ k \ d \ M' \ \forall u. \neg \text{DBM-val-bounded } v \ u \ M \ n$
shows $\neg \text{DBM-val-bounded } v \ u \ M' \ n$
 ⟨proof⟩

lemma *DBM-reset-complete-empty*:

assumes $\forall k \leq n. k > 0 \implies (\exists c. v \ c = k) \text{ clock-numbering } v$
 $\text{DBM-reset } (\text{FW } M \ n) \ n \ (v \ c) \ d \ M' \ \forall u. \neg \text{DBM-val-bounded } v \ u$
 $(\text{FW } M \ n) \ n$
shows $\neg \text{DBM-val-bounded } v \ u \ M' \ n$
 ⟨proof⟩

lemma *DBM-reset-complete-empty1*:

assumes $\forall k \leq n. k > 0 \implies (\exists c. v \ c = k) \text{ clock-numbering } v$
 $\text{DBM-reset } (\text{FW } M \ n) \ n \ (v \ c) \ d \ M' \ \forall u. \neg \text{DBM-val-bounded } v \ u$
 $M \ n$
shows $\neg \text{DBM-val-bounded } v \ u \ M' \ n$
 ⟨proof⟩

Lemma *FW-canonical-id* allows us to prove correspondences between reset and canonical, like for the two below. Can be left out for the rest because of the triviality of the correspondence.

lemma *DBM-reset-empty''*:

assumes $\forall k \leq n. k > 0 \implies (\exists c. v \ c = k) \text{ clock-numbering}' \ v \ n \ v \ c \leq n$
 $\text{DBM-reset } M \ n \ (v \ c) \ d \ M'$
shows $[M]_{v,n} = \{\} \longleftrightarrow [M']_{v,n} = \{\}$
 ⟨proof⟩

lemma *DBM-reset-empty*:

assumes $\forall k \leq n. k > 0 \implies (\exists c. v \ c = k) \text{ clock-numbering}' \ v \ n \ v \ c \leq n$

DBM-reset (*FW M n*) *n* (*v c*) *d M'*
shows $[FW M n]_{v,n} = \{\} \longleftrightarrow [M']_{v,n} = \{\}$
<proof>

lemma *DBM-reset-empty'*:

assumes *canonical M n* $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'*
v n v c $\leq n$

DBM-reset (*FW M n*) *n* (*v c*) *d M'*
shows $[M]_{v,n} = \{\} \longleftrightarrow [M']_{v,n} = \{\}$
<proof>

lemma *DBM-reset-sound'*:

assumes *clock-numbering'* *v n v c* $\leq n$ *DBM-reset M n* (*v c*) *d M'*
DBM-val-bounded v u M' n

DBM-val-bounded v u'' M n
obtains *d'* **where** *DBM-val-bounded v* (*u(c := d')*) *M n*
<proof>

lemma *DBM-reset-sound2*:

assumes *v c* $\leq n$ *DBM-reset M n* (*v c*) *d M'* *DBM-val-bounded v u M' n*
shows *u c = d*

<proof>

lemma *DBM-reset-sound''*:

fixes *M v c n d*
defines $M' \equiv \text{reset } M n (v c) d$
assumes *clock-numbering'* *v n v c* $\leq n$ *DBM-val-bounded v u M' n*
DBM-val-bounded v u'' M n
obtains *d'* **where** *DBM-val-bounded v* (*u(c := d')*) *M n*
<proof>

lemma *DBM-reset-sound*:

fixes *M v c n d*
defines $M' \equiv \text{reset } M n (v c) d$
assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* *v n v c* $\leq n$
 $u \in [M']_{v,n}$
obtains *d'* **where** $u(c := d') \in [M]_{v,n}$
<proof>

lemma *DBM-reset'-complete'*:

assumes *DBM-val-bounded v u M n* *clock-numbering'* *v n* $\forall c \in \text{set } cs. v c \leq n$

shows $\exists u'. \text{DBM-val-bounded } v u' (\text{reset}' M n cs v d) n$
<proof>

lemma *DBM-reset'-complete:*

assumes *DBM-val-bounded* $v\ u\ M\ n$ *clock-numbering'* $v\ n\ \forall\ c \in\ \text{set}\ cs.\ v\ c \leq n$
shows *DBM-val-bounded* $v\ ([cs \rightarrow d]u)$ $(\text{reset}'\ M\ n\ cs\ v\ d)\ n$
 $\langle\text{proof}\rangle$

lemma *DBM-reset'-sound-empty:*

assumes *clock-numbering'* $v\ n\ \forall\ c \in\ \text{set}\ cs.\ v\ c \leq n$
 $\forall\ u.\ \neg\ \text{DBM-val-bounded}\ v\ u\ (\text{reset}'\ M\ n\ cs\ v\ d)\ n$
shows $\neg\ \text{DBM-val-bounded}\ v\ u\ M\ n$
 $\langle\text{proof}\rangle$

fun *set-clocks* $::\ 'c\ \text{list} \Rightarrow 't::\text{time}\ \text{list} \Rightarrow ('c,'t)\ \text{cval} \Rightarrow ('c,'t)\ \text{cval}$

where

set-clocks $\square - u = u \mid$
set-clocks $- \square u = u \mid$
set-clocks $(c\#\ cs)\ (t\#\ ts)\ u = (\text{set-clocks}\ cs\ ts\ (u(c:=t)))$

lemma *DBM-reset'-sound':*

fixes $M\ v\ c\ n\ d\ cs$
assumes *clock-numbering'* $v\ n\ \forall\ c \in\ \text{set}\ cs.\ v\ c \leq n$
 $\text{DBM-val-bounded}\ v\ u\ (\text{reset}'\ M\ n\ cs\ v\ d)\ n\ \text{DBM-val-bounded}\ v\ u''\ M\ n$
shows $\exists\ ts.\ \text{DBM-val-bounded}\ v\ (\text{set-clocks}\ cs\ ts\ u)\ M\ n$
 $\langle\text{proof}\rangle$

lemma *DBM-reset'-resets:*

fixes $M\ v\ c\ n\ d\ cs$
assumes $\forall\ k \leq n.\ k > 0 \longrightarrow (\exists\ c.\ v\ c = k)\ \text{clock-numbering}'\ v\ n\ \forall\ c \in\ \text{set}\ cs.\ v\ c \leq n$
 $\text{DBM-val-bounded}\ v\ u\ (\text{reset}'\ M\ n\ cs\ v\ d)\ n$
shows $\forall\ c \in\ \text{set}\ cs.\ u\ c = d$
 $\langle\text{proof}\rangle$

lemma *DBM-reset'-resets':*

fixes $M\ v\ c\ n\ d\ cs$
assumes *clock-numbering'* $v\ n\ \forall\ c \in\ \text{set}\ cs.\ v\ c \leq n\ \text{DBM-val-bounded}\ v\ u\ (\text{reset}'\ M\ n\ cs\ v\ d)\ n$
 $\text{DBM-val-bounded}\ v\ u''\ M\ n$
shows $\forall\ c \in\ \text{set}\ cs.\ u\ c = d$
 $\langle\text{proof}\rangle$

lemma *DBM-reset'-neg-diag-preservation':*

assumes $k \leq n$ M $k < \mathbf{1}$ *clock-numbering* $v \forall c \in \text{set } cs. v c \leq n$
shows $\text{reset}' M n cs v d k k < \mathbf{1}$ *<proof>*

lemma *DBM-reset'-complete-empty'*:

assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* $v n$
 $\forall c \in \text{set } cs. v c \leq n \forall u. \neg \text{DBM-val-bounded } v u M n$
shows $\forall u. \neg \text{DBM-val-bounded } v u (\text{reset}' M n cs v d) n$ *<proof>*

lemma *DBM-reset'-complete-empty*:

assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* $v n$
 $\forall c \in \text{set } cs. v c \leq n \forall u. \neg \text{DBM-val-bounded } v u M n$
shows $\forall u. \neg \text{DBM-val-bounded } v u (\text{reset}' (FW M n) n cs v d) n$ *<proof>*

lemma *DBM-reset'-empty'*:

assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* $v n \forall c \in \text{set } cs. v c \leq n$
shows $[M]_{v,n} = \{\}$ \longleftrightarrow $[\text{reset}' (FW M n) n cs v d]_{v,n} = \{\}$
<proof>

lemma *DBM-reset'-empty*:

assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* $v n \forall c \in \text{set } cs. v c \leq n$
shows $[M]_{v,n} = \{\}$ \longleftrightarrow $[\text{reset}' M n cs v d]_{v,n} = \{\}$
<proof>

lemma *DBM-reset'-sound*:

assumes $\forall k \leq n. k > 0 \longrightarrow (\exists c. v c = k)$ *clock-numbering'* $v n$
and $\forall c \in \text{set } cs. v c \leq n$
and $u \in [\text{reset}' M n cs v d]_{v,n}$
shows $\exists ts. \text{set-clocks } cs ts u \in [M]_{v,n}$
<proof>

5.6 Misc Preservation Lemmas

lemma *get-const-sum[simp]*:

$a \neq \infty \implies b \neq \infty \implies \text{get-const } a \in \mathbb{Z} \implies \text{get-const } b \in \mathbb{Z} \implies \text{get-const } (a + b) \in \mathbb{Z}$
<proof>

lemma *sum-not-inf-dest*:

assumes $a + b \neq \infty$
shows $a \neq \infty \wedge b \neq \infty$
<proof>

lemma *sum-not-inf-int*:

assumes $a + b \neq \infty$ *get-const* $a \in \mathbb{Z}$ *get-const* $b \in \mathbb{Z}$

shows *get-const* $(a + b) \in \mathbb{Z}$

<proof>

lemma *int-fw-upd*:

$\forall i \leq n. \forall j \leq n. m \ i \ j \neq \infty \longrightarrow \text{get-const } (m \ i \ j) \in \mathbb{Z} \implies k \leq n \implies i \leq n \implies j \leq n$

$\implies i' \leq n \implies j' \leq n \implies (\text{fw-upd } m \ k \ i \ j \ i' \ j') \neq \infty$

$\implies \text{get-const } (\text{fw-upd } m \ k \ i \ j \ i' \ j') \in \mathbb{Z}$

<proof>

lemma *fw-int-aux-c*:

assumes $\forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow \text{get-const } (M \ i \ j) \in \mathbb{Z}$ $a \leq n$ $b \leq n$ $c \leq n$

$i \leq n \ j \leq n \ ((\text{fw } M \ n) \ 0 \ 0 \ c) \ i \ j \neq \infty$

shows *get-const* $((\text{fw } M \ n) \ 0 \ 0 \ c) \ i \ j \in \mathbb{Z}$

<proof>

lemma *fw-int-aux-Suc-b*:

assumes $\forall i \leq n. \forall j \leq n. (\text{fw } M \ n) \ a \ b \ n \ i \ j \neq \infty \longrightarrow \text{get-const } ((\text{fw } M \ n) \ a \ b \ n \ i \ j) \in \mathbb{Z}$

$a \leq n$ *Suc* $b \leq n$ $c \leq n$ $i \leq n$ $j \leq n \ ((\text{fw } M \ n) \ a \ (\text{Suc } b) \ c) \ i \ j \neq \infty$

shows *get-const* $((\text{fw } M \ n) \ a \ (\text{Suc } b) \ c) \ i \ j \in \mathbb{Z}$

<proof>

lemma *fw-int-aux-b*:

assumes $\forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow \text{get-const } (M \ i \ j) \in \mathbb{Z}$ $a \leq n$ $b \leq n$ $c \leq n$

$i \leq n \ j \leq n \ ((\text{fw } M \ n) \ 0 \ b \ c) \ i \ j \neq \infty$

shows *get-const* $((\text{fw } M \ n) \ 0 \ b \ c) \ i \ j \in \mathbb{Z}$ *<proof>*

lemma *fw-int-aux-Suc-a*:

assumes $\forall i \leq n. \forall j \leq n. (\text{fw } M \ n) \ a \ n \ n \ i \ j \neq \infty \longrightarrow \text{get-const } ((\text{fw } M \ n) \ a \ n \ n \ i \ j) \in \mathbb{Z}$

$\text{Suc } a \leq n$ $b \leq n$ $c \leq n$ $i \leq n$ $j \leq n \ ((\text{fw } M \ n) \ (\text{Suc } a) \ b \ c) \ i \ j \neq \infty$

shows *get-const* $((\text{fw } M \ n) \ (\text{Suc } a) \ b \ c) \ i \ j \in \mathbb{Z}$

<proof>

lemma *fw-int-preservation*:

assumes $\forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow \text{get-const } (M \ i \ j) \in \mathbb{Z}$ $a \leq n$ $b \leq n$ $c \leq n$

$i \leq n \ j \leq n \ ((fw \ M \ n) \ a \ b \ c) \ i \ j \neq \infty$
shows $get\text{-}const \ ((fw \ M \ n) \ a \ b \ c) \ i \ j \in \mathbb{Z}$
 ⟨proof⟩

lemma *FW-int-preservation:*

assumes $\forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow get\text{-}const \ (M \ i \ j) \in \mathbb{Z}$
shows $\forall i \leq n. \forall j \leq n. FW \ M \ n \ i \ j \neq \infty \longrightarrow get\text{-}const \ (FW \ M \ n \ i \ j) \in \mathbb{Z}$
 ⟨proof⟩

abbreviation $dbm\text{-}int \ M \ n \equiv \forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow get\text{-}const \ (M \ i \ j) \in \mathbb{Z}$

lemma *And-int-preservation:*

assumes $dbm\text{-}int \ M1 \ n \ dbm\text{-}int \ M2 \ n$
shows $dbm\text{-}int \ (And \ M1 \ M2) \ n$
 ⟨proof⟩

lemma *up-int-preservation:*

$dbm\text{-}int \ M \ n \Longrightarrow dbm\text{-}int \ (up \ M) \ n$
 ⟨proof⟩

lemma *DBM-reset-int-preservation':*

assumes $dbm\text{-}int \ M \ n \ DBM\text{-}reset \ M \ n \ k \ d \ M' \ d \in \mathbb{Z} \ k \leq n$
shows $dbm\text{-}int \ M' \ n$
 ⟨proof⟩

lemma *DBM-reset-int-preservation:*

assumes $dbm\text{-}int \ M \ n \ d \in \mathbb{Z} \ 0 < k \ k \leq n$
shows $dbm\text{-}int \ (reset \ M \ n \ k \ d) \ n$
 ⟨proof⟩

lemma *DBM-reset'-int-preservation:*

assumes $dbm\text{-}int \ M \ n \ d \in \mathbb{Z} \ \forall c. v \ c > 0 \ \forall c \in set \ cs. v \ c \leq n$
shows $dbm\text{-}int \ (reset' \ M \ n \ cs \ v \ d) \ n$ ⟨proof⟩

lemma *int-zone-dbm:*

assumes $clock\text{-}numbering' \ v \ n$
 $\forall (-,d) \in collect\text{-}clock\text{-}pairs \ cc. d \in \mathbb{Z} \ \forall c \in collect\text{-}clks \ cc. v \ c \leq n$
obtains M **where** $\{u. u \vdash cc\} = [M]_{v,n}$
and $\forall i \leq n. \forall j \leq n. M \ i \ j \neq \infty \longrightarrow get\text{-}const \ (M \ i \ j) \in \mathbb{Z}$
 ⟨proof⟩

lemma *reset-set1*:

$\forall c \in \text{set } cs. ([cs \rightarrow d]u) c = d$
 $\langle \text{proof} \rangle$

lemma *reset-set11*:

$\forall c. c \notin \text{set } cs \longrightarrow ([cs \rightarrow d]u) c = u c$
 $\langle \text{proof} \rangle$

lemma *reset-set2*:

$\forall c. c \notin \text{set } cs \longrightarrow (\text{set-clocks } cs \text{ } ts \text{ } u) c = u c$
 $\langle \text{proof} \rangle$

lemma *reset-set*:

assumes $\forall c \in \text{set } cs. u c = d$
shows $[cs \rightarrow d](\text{set-clocks } cs \text{ } ts \text{ } u) = u$
 $\langle \text{proof} \rangle$

abbreviation *global-clock-numbering* ::

$(\text{'a}, \text{'c}, \text{'t} :: \text{time}, \text{'s}) \text{ta} \Rightarrow (\text{'c} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{bool}$

where

$\text{global-clock-numbering } A \text{ } v \text{ } n \equiv$
 $\text{clock-numbering}' v \text{ } n \wedge (\forall c \in \text{clk-set } A. v c \leq n) \wedge (\forall k \leq n. k > 0 \longrightarrow$
 $(\exists c. v c = k))$

lemma *dbm-int-abstr*:

assumes $\forall (x, m) \in \text{collect-clock-pairs } g. m \in \mathbb{Z}$
shows $\text{dbm-int } (\text{abstr } g \text{ } (\lambda i j. \infty) v) n$
 $\langle \text{proof} \rangle$

lemma *dbm-int-inv-abstr*:

assumes $\forall (x, m) \in \text{clkp-set } A. m \in \mathbb{N}$
shows $\text{dbm-int } (\text{abstr } (\text{inv-of } A \text{ } l) \text{ } (\lambda i j. \infty) v) n$
 $\langle \text{proof} \rangle$

lemma *dbm-int-guard-abstr*:

assumes $\text{valid-abstraction } A \text{ } X \text{ } k \text{ } A \vdash l \longrightarrow^{g, a, r} l'$
shows $\text{dbm-int } (\text{abstr } g \text{ } (\lambda i j. \infty) v) n$
 $\langle \text{proof} \rangle$

lemma *collect-clks-id*: $\text{collect-clks } cc = \text{fst } \text{' collect-clock-pairs } cc \langle \text{proof} \rangle$

5.6.1 Unused theorems

lemma *canonical-cyc-free*:

canonical $M\ n \implies \forall i \leq n. M\ i\ i \geq \mathbf{1} \implies \text{cyc-free } M\ n$
 ⟨proof⟩

lemma *canonical-cyc-free2*:

canonical $M\ n \implies \text{cyc-free } M\ n \iff (\forall i \leq n. M\ i\ i \geq \mathbf{1})$

⟨proof⟩

lemma *DBM-reset'-diag-preservation*:

assumes $\forall k \leq n. M\ k\ k \leq \mathbf{1}$ *clock-numbering* $v\ \forall\ c \in \text{set } cs. v\ c \leq n$

shows $\forall k \leq n. \text{reset}'\ M\ n\ cs\ v\ d\ k\ k \leq \mathbf{1}$ ⟨proof⟩

end

5.6.2 Semantics Based on DBMs

theory *DBM-Zone-Semantics*

imports *DBM-Operations*

begin

5.6.3 Single Step

inductive *step-z-dbm* ::

$(\ 'a, 'c, 't, 's) \text{ ta} \Rightarrow 's \Rightarrow ('t::\text{time})\ \text{DBM}$

$\Rightarrow ('c \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow 's \Rightarrow ('t::\text{time})\ \text{DBM} \Rightarrow \text{bool}$

$(- \vdash \langle -, - \rangle \rightsquigarrow_{-, -} \langle -, - \rangle [61, 61, 61] 61)$

where

step-t-z-dbm:

$D\text{-inv} = \text{abstr } (\text{inv-of } A\ l) (\lambda i\ j. \infty)\ v \implies A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l, \text{And } (\text{up } (\text{And } D\ D\text{-inv}))\ D\text{-inv} \rangle \mid$

step-a-z-dbm:

$A \vdash l \xrightarrow{g, a, r} l'$

$\implies A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', \text{And } (\text{reset}'\ (\text{And } D\ (\text{abstr } g\ (\lambda i\ j. \infty)\ v))\ n\ r\ v$

$0) \rangle$

$(\text{abstr } (\text{inv-of } A\ l') (\lambda i\ j. \infty)\ v) \rangle$

inductive-cases *step-z-cases*: $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', D' \rangle$

declare *step-z-dbm.intros*[intro]

lemma *step-z-dbm-preserves-int*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', D' \rangle$ *global-clock-numbering* $A\ v\ n$ *valid-abstraction*

$A\ X\ k$

dbm-int $D\ n$

shows *dbm-int* $D'\ n$

⟨proof⟩

lemma *And-correct:*

shows $[M1]_{v,n} \cap [M2]_{v,n} = [And\ M1\ M2]_{v,n}$
 $\langle proof \rangle$

lemma *up-correct:*

assumes *clock-numbering' v n*
shows $[up\ M]_{v,n} = [M]_{v,n}^\uparrow$
 $\langle proof \rangle$

lemma *step-z-dbm-sound:*

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n} \langle l', D' \rangle$ *global-clock-numbering A v n*
shows $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow \langle l', [D']_{v,n} \rangle$
 $\langle proof \rangle$

lemma *step-z-dbm-DBM:*

assumes $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow \langle l', Z \rangle$ *global-clock-numbering A v n*
obtains D' **where** $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n} \langle l', D' \rangle$ $Z = [D']_{v,n}$
 $\langle proof \rangle$

lemma *step-z-computable:*

assumes $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow \langle l', Z \rangle$ *global-clock-numbering A v n*
obtains D' **where** $Z = [D']_{v,n}$
 $\langle proof \rangle$

lemma *step-z-dbm-complete:*

assumes *global-clock-numbering A v n* $A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle$
and $u \in [(D)]_{v,n}$
shows $\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{v,n} \langle l', D' \rangle \wedge u' \in [D']_{v,n}$
 $\langle proof \rangle$

5.6.4 Multi Step

inductive *steps-z-dbm ::*

$('a, 'c, 't, 's) ta \Rightarrow 's \Rightarrow ('t::time) DBM$
 $\Rightarrow ('c \Rightarrow nat) \Rightarrow nat \Rightarrow 's \Rightarrow ('t::time) DBM \Rightarrow bool$
 $(- \vdash \langle -, - \rangle \rightsquigarrow_{*, -} \langle -, - \rangle [61, 61, 61] 61)$

where

refl: $A \vdash \langle l, D \rangle \rightsquigarrow_{*, v, n} \langle l, D \rangle$ |
step: $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', D' \rangle \Longrightarrow A \vdash \langle l', D' \rangle \rightsquigarrow_{*, v, n} \langle l'', D'' \rangle \Longrightarrow$
 $A \vdash \langle l, D \rangle \rightsquigarrow_{*, v, n} \langle l'', D'' \rangle$

declare *steps-z-dbm.intros[intro]*

lemma *steps-z-dbm-sound*:
assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{*v,n} \langle l', D' \rangle$
and *global-clock-numbering* $A \ v \ n$
and $u' \in [D']_{v,n}$
shows $\exists u \in [D]_{v,n}. A \vdash \langle l, u \rangle \rightarrow_* \langle l', u' \rangle \langle \text{proof} \rangle$

lemma *steps-z-dbm-complete*:
assumes $A \vdash \langle l, u \rangle \rightarrow_* \langle l', u' \rangle$
and *global-clock-numbering* $A \ v \ n$
and $u \in [D]_{v,n}$
shows $\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{*v,n} \langle l', D' \rangle \wedge u' \in [D']_{v,n} \langle \text{proof} \rangle$

end

$\langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle$

5.7 Normalization of DBMs

theory *DBM-Normalization*
imports *DBM-Basics*
begin

This is the implementation of the common approximation operation.

fun *norm-upper* :: $(t::\text{time}) \text{DBMEntry} \Rightarrow 't \Rightarrow (t::\text{time}) \text{DBMEntry}$
where
norm-upper $e \ t = (\text{if } L_e \ t \prec e \text{ then } \infty \text{ else } e)$

fun *norm-lower* :: $(t::\text{time}) \text{DBMEntry} \Rightarrow 't \Rightarrow (t::\text{time}) \text{DBMEntry}$
where
norm-lower $e \ t = (\text{if } e \prec L_t \ t \text{ then } L_t \ t \text{ else } e)$

Note that literature pretends that $\mathbf{0}$ would have some (presumably infinite bound) in k and thus defines normalization uniformly. The easiest way to get around this seems to explicate this in the definition as below.

definition *norm* :: $(t::\text{time}) \text{DBM} \Rightarrow (\text{nat} \Rightarrow 't) \Rightarrow \text{nat} \Rightarrow 't \text{DBM}$
where

norm $M \ k \ n \equiv \lambda \ i \ j.$
let $ub = \text{if } i > 0 \text{ then } (k \ i) \text{ else } 0 \text{ in}$
let $lb = \text{if } j > 0 \text{ then } (- \ k \ j) \text{ else } 0 \text{ in}$
if $i \leq n \wedge j \leq n \text{ then } \text{norm-lower } (\text{norm-upper } (M \ i \ j) \ ub) \ lb \text{ else } M \ i \ j$

5.8 Normalization is a Widening Operator

lemma *norm-mono*:


```

assumes  $\forall c. v\ c > 0\ u \in [M]_{v,n}$ 
shows  $u \in [norm\ M\ k\ n]_{v,n}$  (is  $u \in [?M2]_{v,n}$ )
 $\langle proof \rangle$ 

```

end

theory *Regions-Beta*

imports *Misc DBM-Normalization DBM-Operations*

begin

6 Refinement to β -regions

6.1 Definition

type-synonym $'c\ ceiling = ('c \Rightarrow nat)$

datatype *intv* =

Const nat |

Intv nat |

Greater nat

datatype *intv'* =

Const' int |

Intv' int |

Greater' int |

Smaller' int

type-synonym *t* = *real*

instantiation *real* :: *time*

begin

instance $\langle proof \rangle$

end

inductive *valid-intv* :: *nat* \Rightarrow *intv* \Rightarrow *bool*

where

$0 \leq d \Longrightarrow d \leq c \Longrightarrow valid-intv\ c\ (Const\ d)$ |

$0 \leq d \Longrightarrow d < c \Longrightarrow valid-intv\ c\ (Intv\ d)$ |

valid-intv c (Greater c)

inductive *valid-intv'* :: *int* \Rightarrow *int* \Rightarrow *intv'* \Rightarrow *bool*

where

valid-intv' l - (Smaller' (-l)) |

$-l \leq d \Longrightarrow d \leq u \Longrightarrow valid-intv'\ l\ u\ (Const'\ d)$ |

$-l \leq d \Longrightarrow d < u \Longrightarrow valid-intv'\ l\ u\ (Intv'\ d)$ |

valid-intv' - *u* (*Greater'* *u*)

inductive *intv-elem* :: '*c* ⇒ (*'c,t*) *cval* ⇒ *intv* ⇒ *bool*

where

$u\ x = d \implies \text{intv-elem}\ x\ u\ (\text{Const}\ d) \mid$
 $d < u\ x \implies u\ x < d + 1 \implies \text{intv-elem}\ x\ u\ (\text{Intv}\ d) \mid$
 $c < u\ x \implies \text{intv-elem}\ x\ u\ (\text{Greater}\ c)$

inductive *intv'-elem* :: '*c* ⇒ '*c* ⇒ (*'c,t*) *cval* ⇒ *intv'* ⇒ *bool*

where

$u\ x - u\ y < c \implies \text{intv'-elem}\ x\ y\ u\ (\text{Smaller}'\ c) \mid$
 $u\ x - u\ y = d \implies \text{intv'-elem}\ x\ y\ u\ (\text{Const}'\ d) \mid$
 $d < u\ x - u\ y \implies u\ x - u\ y < d + 1 \implies \text{intv'-elem}\ x\ y\ u\ (\text{Intv}'\ d) \mid$
 $c < u\ x - u\ y \implies \text{intv'-elem}\ x\ y\ u\ (\text{Greater}'\ c)$

abbreviation *total-preorder* *r* ≡ *refl* *r* ∧ *trans* *r*

inductive *isConst* :: *intv* ⇒ *bool*

where

isConst (*Const* -)

inductive *isIntv* :: *intv* ⇒ *bool*

where

isIntv (*Intv* -)

inductive *isGreater* :: *intv* ⇒ *bool*

where

isGreater (*Greater* -)

declare *isIntv.intros*[*intro!*] *isConst.intros*[*intro!*] *isGreater.intros*[*intro!*]

declare *isIntv.cases*[*elim!*] *isConst.cases*[*elim!*] *isGreater.cases*[*elim!*]

inductive *valid-region* :: '*c* *set* ⇒ (*'c* ⇒ *nat*) ⇒ (*'c* ⇒ *intv*) ⇒ (*'c* ⇒ '*c* ⇒ *intv'*) ⇒ '*c* *rel* ⇒ *bool*

where

$\llbracket X_0 = \{x \in X. \exists d. I\ x = \text{Intv}\ d\}; \text{refl-on}\ X_0\ r; \text{trans}\ r; \text{total-on}\ X_0\ r;$
 $\forall x \in X. \text{valid-intv}\ (k\ x)\ (I\ x);$
 $\forall x \in X. \forall y \in X. \text{isGreater}\ (I\ x) \vee \text{isGreater}\ (I\ y) \longrightarrow \text{valid-intv}'\ (k\ y)\ (k\ x)\ (J\ x\ y)\rrbracket$
 $\implies \text{valid-region}\ X\ k\ I\ J\ r$

inductive-set *region* **for** *X* *I* *J* *r*

where

$$\begin{aligned}
& \forall x \in X. u x \geq 0 \implies \forall x \in X. \text{intv-elem } x u (I x) \implies X_0 = \{x \in X. \\
& \exists d. I x = \text{Intv } d\} \implies \\
& \forall x \in X_0. \forall y \in X_0. (x, y) \in r \iff \text{frac } (u x) \leq \text{frac } (u y) \implies \\
& \forall x \in X. \forall y \in X. \text{isGreater } (I x) \vee \text{isGreater } (I y) \longrightarrow \text{intv}'\text{-elem } x y \\
& u (J x y) \\
& \implies u \in \text{region } X I J r
\end{aligned}$$

Defining the unique element of a partition that contains a valuation

definition *part* ($[-]$ - [61,61] 61) **where** *part* $v \mathcal{R} \equiv \text{THE } R. R \in \mathcal{R} \wedge v \in R$

First we need to show that the set of regions is a partition of the set of all clock assignments. This property is only claimed by P. Bouyer.

inductive-cases $[\text{elim!}]$: *intv-elem* $x u (Const d)$
inductive-cases $[\text{elim!}]$: *intv-elem* $x u (Intv d)$
inductive-cases $[\text{elim!}]$: *intv-elem* $x u (Greater d)$
inductive-cases $[\text{elim!}]$: *valid-intv* $c (Greater d)$
inductive-cases $[\text{elim!}]$: *valid-intv* $c (Const d)$
inductive-cases $[\text{elim!}]$: *valid-intv* $c (Intv d)$
inductive-cases $[\text{elim!}]$: *intv'*-elem $x y u (Const' d)$
inductive-cases $[\text{elim!}]$: *intv'*-elem $x y u (Intv' d)$
inductive-cases $[\text{elim!}]$: *intv'*-elem $x y u (Greater' d)$
inductive-cases $[\text{elim!}]$: *intv'*-elem $x y u (Smaller' d)$
inductive-cases $[\text{elim!}]$: *valid-intv'* $l u (Greater' d)$
inductive-cases $[\text{elim!}]$: *valid-intv'* $l u (Smaller' d)$
inductive-cases $[\text{elim!}]$: *valid-intv'* $l u (Const' d)$
inductive-cases $[\text{elim!}]$: *valid-intv'* $l u (Intv' d)$

declare *valid-intv.intros* $[\text{intro}]$
declare *valid-intv'.intros* $[\text{intro}]$
declare *intv-elem.intros* $[\text{intro}]$
declare *intv'-elem.intros* $[\text{intro}]$

declare *region.cases* $[\text{elim}]$
declare *valid-region.cases* $[\text{elim}]$

6.2 Basic Properties

First we show that all valid intervals are distinct

lemma *valid-intv-distinct*:

$$\begin{aligned}
& \text{valid-intv } c I \implies \text{valid-intv } c I' \implies \text{intv-elem } x u I \implies \text{intv-elem } x u I' \\
& \implies I = I'
\end{aligned}$$

$\langle \text{proof} \rangle$

lemma *valid-intv'-distinct:*

$-c \leq d \implies \text{valid-intv}' c d I \implies \text{valid-intv}' c d I' \implies \text{intv}'\text{-elem } x y u I$
 $\implies \text{intv}'\text{-elem } x y u I'$
 $\implies I = I'$
 ⟨proof⟩

From this we show that all valid regions are distinct

lemma *valid-regions-distinct:*

$\text{valid-region } X k I J r \implies \text{valid-region } X k I' J' r' \implies v \in \text{region } X I J$
 $r \implies v \in \text{region } X I' J' r'$
 $\implies \text{region } X I J r = \text{region } X I' J' r'$
 ⟨proof⟩

locale *Beta-Regions =*

fixes $X k \mathcal{R}$ **and** $V :: ('c, t)$ *cval set*
defines $\mathcal{R} \equiv \{\text{region } X I J r \mid I J r. \text{valid-region } X k I J r\}$
defines $V \equiv \{v . \forall x \in X. v x \geq 0\}$
assumes *finite: finite X*
assumes *non-empty: X ≠ {}*

begin

lemma *\mathcal{R} -regions-distinct:*

$\llbracket R \in \mathcal{R}; v \in R; R' \in \mathcal{R}; R \neq R' \rrbracket \implies v \notin R'$
 ⟨proof⟩

Secondly, we also need to show that every valuations belongs to a region which is part of the partition.

definition *intv-of* $:: \text{nat} \Rightarrow t \Rightarrow \text{intv}$ **where**

intv-of $c v \equiv$
 if $(v > c)$ then *Greater* c
 else if $(\exists x :: \text{nat}. x = v)$ then $(\text{Const } (\text{nat } (\text{floor } v)))$
 else $(\text{Intv } (\text{nat } (\text{floor } v)))$

definition *intv'-of* $:: \text{int} \Rightarrow \text{int} \Rightarrow t \Rightarrow \text{intv}'$ **where**

intv'-of $l u v \equiv$
 if $(v > u)$ then *Greater'* u
 else if $(v < l)$ then *Smaller'* l
 else if $(\exists x :: \text{int}. x = v)$ then $(\text{Const}' (\text{floor } v))$
 else $(\text{Intv}' (\text{floor } v))$

lemma *region-cover:*

$\forall x \in X. v x \geq 0 \implies \exists R. R \in \mathcal{R} \wedge v \in R$
 ⟨proof⟩

lemma *region-cover-V*: $v \in V \implies \exists R. R \in \mathcal{R} \wedge v \in R$ *<proof>*

Note that we cannot show that every region is non-empty anymore. The problem are regions fixing differences between an 'infeasible' constant.

We can show that there is always exactly one region a valid valuation belongs to. Note that we do not need non-emptiness for that.

lemma *regions-partition*:

$\forall x \in X. 0 \leq v x \implies \exists! R \in \mathcal{R}. v \in R$
<proof>

lemma *region-unique*:

$v \in R \implies R \in \mathcal{R} \implies [v]_{\mathcal{R}} = R$
<proof>

lemma *regions-partition'*:

$\forall x \in X. 0 \leq v x \implies \forall x \in X. 0 \leq v' x \implies v' \in [v]_{\mathcal{R}} \implies [v']_{\mathcal{R}} = [v]_{\mathcal{R}}$
<proof>

lemma *regions-closed*:

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies [v \oplus t]_{\mathcal{R}} \in \mathcal{R}$
<proof>

lemma *regions-closed'*:

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies (v \oplus t) \in [v \oplus t]_{\mathcal{R}}$
<proof>

lemma *valid-regions-I-cong*:

valid-region $X k I J r \implies \forall x \in X. I x = I' x$
 $\implies \forall x \in X. \forall y \in X. (isGreater (I x) \vee isGreater (I y)) \longrightarrow J x y = J' x y$
 $\implies region X I J r = region X I' J' r \wedge valid-region X k I' J' r$
<proof>

fun *intv-const* :: *intv* \Rightarrow *nat*

where

intv-const (*Const* d) = d |
intv-const (*Intv* d) = d |
intv-const (*Greater* d) = d

fun *intv'-const* :: *intv'* \Rightarrow *int*

where

intv'-const (*Smaller'* d) = d |
intv'-const (*Const'* d) = d |

$intv'\text{-const } (Intv' d) = d \mid$
 $intv'\text{-const } (Greater' d) = d$

lemma *finite- \mathcal{R} -aux*:

fixes $P A B$ **assumes** $finite \{x. A x\} \text{ finite } \{x. B x\}$

shows $finite \{(I, J) \mid I J. P I J r \wedge A I \wedge B J\}$

$\langle proof \rangle$

lemma *finite- \mathcal{R}* :

notes $[[simproc \text{ add: } finite\text{-Collect}]]$

shows $finite \mathcal{R}$

$\langle proof \rangle$

end

6.3 Approximation with β -regions

locale *Beta-Regions'* = *Beta-Regions* +

fixes $v n \text{ not-in-}X$

assumes *clock-numbering*: $\forall c. v c > 0 \wedge (\forall x. \forall y. v x \leq n \wedge v y \leq n \wedge v x = v y \longrightarrow x = y)$

$\forall k :: nat \leq n. k > 0 \longrightarrow (\exists c \in X. v c = k) \forall c \in X. v c \leq n$

assumes *not-in- X* : $\text{not-in-}X \notin X$

begin

definition $v' \equiv \lambda i. \text{if } i \leq n \text{ then } (THE c. c \in X \wedge v c = i) \text{ else not-in-}X$

lemma *v-v'*:

$\forall c \in X. v' (v c) = c$

$\langle proof \rangle$

abbreviation

$vabstr (S :: ('a, t) \text{ zone}) M \equiv S = [M]_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z})$

definition *normalized*:

normalized $M \equiv$

$(\forall i j. 0 < i \wedge i \leq n \wedge 0 < j \wedge j \leq n \wedge M i j \neq \infty \longrightarrow$

$Lt (- ((k o v') j)) \leq M i j \wedge M i j \leq Le ((k o v') i))$

$\wedge (\forall i \leq n. i > 0 \longrightarrow (M i 0 \leq Le ((k o v') i) \vee M i 0 = \infty) \wedge Lt (- ((k o v') i)) \leq M 0 i)$

definition *apx-def*:

$Approx_{\beta} Z \equiv \bigcap \{S. \exists U M. S = \bigcup U \wedge U \subseteq \mathcal{R} \wedge Z \subseteq S \wedge vabstr S M \wedge \text{normalized } M\}$

lemma *apx-min*:

$S = \bigcup U \implies U \subseteq \mathcal{R} \implies S = [M]_{v,n} \implies \forall i \leq n. \forall j \leq n. M i j \neq \infty$
 $\implies \text{get-const } (M i j) \in \mathbb{Z}$
 $\implies \text{normalized } M \implies Z \subseteq S \implies Approx_{\beta} Z \subseteq S$
 ⟨proof⟩

lemma $U \neq \{\} \implies x \in \bigcap U \implies \exists S \in U. x \in S$ ⟨proof⟩

lemma \mathcal{R} -union: $\bigcup \mathcal{R} = V$ ⟨proof⟩

lemma *all-dbm*: $\exists M. vabstr (\bigcup \mathcal{R}) M \wedge \text{normalized } M$
 ⟨proof⟩

lemma \mathcal{R} -int:

$R \in \mathcal{R} \implies R' \in \mathcal{R} \implies R \neq R' \implies R \cap R' = \{\}$ ⟨proof⟩

lemma *aux1*:

$u \in R \implies R \in \mathcal{R} \implies U \subseteq \mathcal{R} \implies u \in \bigcup U \implies R \subseteq \bigcup U$ ⟨proof⟩

lemma *aux2*: $x \in \bigcap U \implies U \neq \{\} \implies \exists S \in U. x \in S$ ⟨proof⟩

lemma *aux2'*: $x \in \bigcap U \implies U \neq \{\} \implies \forall S \in U. x \in S$ ⟨proof⟩

lemma *apx-subset*: $Z \subseteq Approx_{\beta} Z$ ⟨proof⟩

lemma *aux3*:

$\forall X \in U. \forall Y \in U. X \cap Y \in U \implies S \subseteq U \implies S \neq \{\} \implies \text{finite } S$
 $\implies \bigcap S \in U$
 ⟨proof⟩

lemma *empty-zone-dbm*:

$\exists M :: t \text{ DBM}. vabstr \{\} M \wedge \text{normalized } M \wedge (\forall k \leq n. M k k \leq Le 0)$
 ⟨proof⟩

lemma *valid-dbms-int*:

$\forall X \in \{S. \exists M. vabstr S M\}. \forall Y \in \{S. \exists M. vabstr S M\}. X \cap Y \in \{S. \exists M. vabstr S M\}$
 ⟨proof⟩

print-statement *split-min*

lemma *split-min'*:

$P (\min i j) = ((\min i j = i \longrightarrow P i) \wedge (\min i j = j \longrightarrow P j))$
 $\langle \text{proof} \rangle$

lemma *normalized-and-preservation*:

$\text{normalized } M1 \Longrightarrow \text{normalized } M2 \Longrightarrow \text{normalized } (\text{And } M1 M2)$
 $\langle \text{proof} \rangle$

lemma *valid-dbms-int'*:

$\forall X \in \{S. \exists M. \text{vabstr } S M \wedge \text{normalized } M\}. \forall Y \in \{S. \exists M. \text{vabstr } S M \wedge \text{normalized } M\}.$
 $X \cap Y \in \{S. \exists M. \text{vabstr } S M \wedge \text{normalized } M\}$
 $\langle \text{proof} \rangle$

lemma *apx-in*:

$Z \subseteq V \Longrightarrow \text{Approx}_\beta Z \in \{S. \exists U M. S = \bigcup U \wedge U \subseteq \mathcal{R} \wedge Z \subseteq S \wedge \text{vabstr } S M \wedge \text{normalized } M\}$
 $\langle \text{proof} \rangle$

lemma *apx-empty*:

$\text{Approx}_\beta \{\} = \{\}$
 $\langle \text{proof} \rangle$

end

6.4 Computing β -Approximation

context *Beta-Regions'*

begin

lemma *dbm-regions*:

$\text{vabstr } S M \Longrightarrow \text{normalized } M \Longrightarrow [M]_{v,n} \neq \{\} \Longrightarrow [M]_{v,n} \subseteq V \Longrightarrow \exists U \subseteq \mathcal{R}. S = \bigcup U$
 $\langle \text{proof} \rangle$

lemma *dbm-regions'*:

$\text{vabstr } S M \Longrightarrow \text{normalized } M \Longrightarrow S \subseteq V \Longrightarrow \exists U \subseteq \mathcal{R}. S = \bigcup U$
 $\langle \text{proof} \rangle$

lemma *dbm-regions''*:

$\text{dbm-int } M n \Longrightarrow \text{normalized } M \Longrightarrow [M]_{v,n} \subseteq V \Longrightarrow \exists U \subseteq \mathcal{R}. [M]_{v,n} = \bigcup U$
 $\langle \text{proof} \rangle$

lemma *canonical-saturated-1*:
assumes $Le\ r \leq M\ (v\ c1)\ 0$
and $Le\ (-\ r) \leq M\ 0\ (v\ c1)$
and *cycle-free* $M\ n$
and *canonical* $M\ n$
and $v\ c1 \leq n$
and $v\ c1 > 0$
and $\forall c. v\ c \leq n \longrightarrow 0 < v\ c$
obtains u **where** $u \in [M]_{v,n}\ u\ c1 = r$
 $\langle proof \rangle$

lemma *canonical-saturated-2*:
assumes $Le\ r \leq M\ 0\ (v\ c2)$
and $Le\ (-\ r) \leq M\ (v\ c2)\ 0$
and *cycle-free* $M\ n$
and *canonical* $M\ n$
and $v\ c2 \leq n$
and $v\ c2 > 0$
and $\forall c. v\ c \leq n \longrightarrow 0 < v\ c$
obtains u **where** $u \in [M]_{v,n}\ u\ c2 = -\ r$
 $\langle proof \rangle$

lemma *canonical-saturated-3*:
assumes $Le\ r \leq M\ (v\ c1)\ (v\ c2)$
and $Le\ (-\ r) \leq M\ (v\ c2)\ (v\ c1)$
and *cycle-free* $M\ n$
and *canonical* $M\ n$
and $v\ c1 \leq n\ v\ c2 \leq n$
and $v\ c1 \neq v\ c2$
and $\forall c. v\ c \leq n \longrightarrow 0 < v\ c$
obtains u **where** $u \in [M]_{v,n}\ u\ c1 - u\ c2 = r$
 $\langle proof \rangle$

lemma *DBM-canonical-subset-le*:
notes *any-le-inf* $[intro]$
fixes $M :: t\ DBM$
assumes *canonical* $M\ n\ [M]_{v,n} \subseteq [M]_{v,n}\ [M]_{v,n} \neq \{\}\ i \leq n\ j \leq n\ i \neq j$
shows $M\ i\ j \leq M'\ i\ j$
 $\langle proof \rangle$

lemma *DBM-set-diag*:
assumes $[M]_{v,n} \neq \{\}$
shows $[M]_{v,n} = [(\lambda\ i\ j. \text{if } i = j \text{ then } Le\ 0 \text{ else } M\ i\ j)]_{v,n}$
 $\langle proof \rangle$

lemma *DBM-le-subset'*:

assumes $\forall i \leq n. \forall j \leq n. i \neq j \longrightarrow M i j \leq M' i j$
and $\forall i \leq n. M' i i \geq Le 0$
and $u \in [M]_{v,n}$
shows $u \in [M']_{v,n}$

<proof>

lemma *neg-diag-empty-spec*:

assumes $i \leq n \ M i i < \mathbf{1}$
shows $[M]_{v,n} = \{\}$

<proof>

lemma *canonical-empty-zone-spec*:

assumes *canonical* $M n$
shows $[M]_{v,n} = \{\} \longleftrightarrow (\exists i \leq n. M i i < \mathbf{1})$

<proof>

lemma *norm-set-diag*:

assumes *canonical* $M n$ $[M]_{v,n} \neq \{\}$
obtains M' **where** $[M]_{v,n} = [M']_{v,n}$ $[norm M (k o v') n]_{v,n} = [norm M' (k o v') n]_{v,n}$
 $\forall i \leq n. M' i i = \mathbf{1}$ *canonical* $M' n$

<proof>

lemma *norm-normalizes*:

notes *any-le-inf* [intro]
shows *normalized* $(norm M (k o v') n)$

<proof>

lemma *norm-int-preservation*:

assumes *dbm-int* $M n$ $i \leq n$ $j \leq n$ $norm M (k o v') n i j \neq \infty$
shows *get-const* $(norm M (k o v') n i j) \in \mathbb{Z}$

<proof>

lemma *norm-V-preservation'*:

notes *any-le-inf* [intro]
assumes $[M]_{v,n} \subseteq V$ *canonical* $M n$ $[M]_{v,n} \neq \{\}$
shows $[norm M (k o v') n]_{v,n} \subseteq V$

<proof>

lemma *norm-V-preservation*:

assumes $[M]_{v,n} \subseteq V$ *canonical* $M n$
shows $[norm M (k o v') n]_{v,n} \subseteq V$ (**is** $[?M]_{v,n} \subseteq V$)

<proof>

lemma *norm-min*:

assumes *normalized* $M1$ $[M]_{v,n} \subseteq [M1]_{v,n}$
canonical M n $[M]_{v,n} \neq \{\}$ $[M]_{v,n} \subseteq V$

shows $[norm\ M\ (k\ o\ v')\ n]_{v,n} \subseteq [M1]_{v,n}$ (**is** $[?M2]_{v,n} \subseteq [M1]_{v,n}$)
<proof>

lemma *apx-norm-eq*:

assumes *canonical* M n $[M]_{v,n} \subseteq V$ *dbm-int* M n

shows $Approx_\beta ([M]_{v,n}) = [norm\ M\ (k\ o\ v')\ n]_{v,n}$
<proof>

end

6.5 Auxiliary β -boundedness Theorems

context *Beta-Regions'*

begin

lemma *β -boundedness-diag-lt*:

fixes $m :: int$

assumes $-k\ y \leq m$ $m \leq k\ x$ $x \in X$ $y \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u\ x - u\ y < m\}$
<proof>

lemma *β -boundedness-diag-eq*:

fixes $m :: int$

assumes $-k\ y \leq m$ $m \leq k\ x$ $x \in X$ $y \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u\ x - u\ y = m\}$
<proof>

lemma *β -boundedness-lt*:

fixes $m :: int$

assumes $m \leq k\ x$ $x \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u\ x < m\}$
<proof>

lemma *β -boundedness-gt*:

fixes $m :: int$

assumes $m \leq k\ x$ $x \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u\ x > m\}$
<proof>

lemma β -boundedness-eq:

fixes $m :: int$

assumes $m \leq k \ x \ x \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u \ x = m\}$

$\langle proof \rangle$

lemma β -boundedness-diag-le:

fixes $m :: int$

assumes $- k \ y \leq m \ m \leq k \ x \ x \in X \ y \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u \ x - u \ y \leq m\}$

$\langle proof \rangle$

lemma β -boundedness-le:

fixes $m :: int$

assumes $m \leq k \ x \ x \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u \ x \leq m\}$

$\langle proof \rangle$

lemma β -boundedness-ge:

fixes $m :: int$

assumes $m \leq k \ x \ x \in X$

shows $\exists U \subseteq \mathcal{R}. \bigcup U = \{u \in V. u \ x \geq m\}$

$\langle proof \rangle$

lemma β -boundedness-diag-lt':

fixes $m :: int$

shows

$- k \ y \leq (m :: int) \implies m \leq k \ x \implies x \in X \implies y \in X \implies Z \subseteq \{u \in V. u \ x - u \ y < m\}$

$\implies Approx_\beta Z \subseteq \{u \in V. u \ x - u \ y < m\}$

$\langle proof \rangle$

lemma β -boundedness-diag-le':

fixes $m :: int$

shows

$- k \ y \leq (m :: int) \implies m \leq k \ x \implies x \in X \implies y \in X \implies Z \subseteq \{u \in V. u \ x - u \ y \leq m\}$

$\implies Approx_\beta Z \subseteq \{u \in V. u \ x - u \ y \leq m\}$

$\langle proof \rangle$

lemma β -boundedness-lt':

fixes $m :: int$

shows

$m \leq k \ x \implies x \in X \implies Z \subseteq \{u \in V. u \ x < m\} \implies Approx_\beta Z \subseteq \{u$

$\in V. u x < m$
 $\langle proof \rangle$

lemma β -boundedness-gt':

fixes $m :: int$

shows

$m \leq k x \implies x \in X \implies Z \subseteq \{u \in V. u x > m\} \implies Approx_\beta Z \subseteq \{u$
 $\in V. u x > m\}$

$\langle proof \rangle$

lemma obtains-dbm-le:

fixes $m :: int$

assumes $x \in X m \leq k x$

obtains M **where** $vabstr \{u \in V. u x \leq m\} M$ *normalized* M
 $\langle proof \rangle$

lemma β -boundedness-le':

fixes $m :: int$

shows

$m \leq k x \implies x \in X \implies Z \subseteq \{u \in V. u x \leq m\} \implies Approx_\beta Z \subseteq \{u$
 $\in V. u x \leq m\}$

$\langle proof \rangle$

lemma obtains-dbm-ge:

fixes $m :: int$

assumes $x \in X m \leq k x$

obtains M **where** $vabstr \{u \in V. u x \geq m\} M$ *normalized* M
 $\langle proof \rangle$

lemma β -boundedness-ge':

fixes $m :: int$

shows $m \leq k x \implies x \in X \implies Z \subseteq \{u \in V. u x \geq m\} \implies Approx_\beta Z$
 $\subseteq \{u \in V. u x \geq m\}$

$\langle proof \rangle$

end

end

7 The Classic Construction for Decidability

theory *Regions*

imports *Timed-Automata Misc*
begin

The following is a formalization of regions in the correct version of Patricia Bouyer et al.

7.1 Definition of Regions

type-synonym $'c$ *ceiling* = ($'c \Rightarrow nat$)

datatype *intv* =
Const nat |
Intv nat |
Greater nat

type-synonym $t = real$

instantiation *real* :: *time*
begin
 instance $\langle proof \rangle$
end

inductive *valid-intv* :: $nat \Rightarrow intv \Rightarrow bool$
where
 $0 \leq d \Longrightarrow d \leq c \Longrightarrow valid-intv\ c\ (Const\ d)$ |
 $0 \leq d \Longrightarrow d < c \Longrightarrow valid-intv\ c\ (Intv\ d)$ |
 $valid-intv\ c\ (Greater\ c)$

inductive *intv-elem* :: $'c \Rightarrow ('c, t)\ cval \Rightarrow intv \Rightarrow bool$
where
 $u\ x = d \Longrightarrow intv-elem\ x\ u\ (Const\ d)$ |
 $d < u\ x \Longrightarrow u\ x < d + 1 \Longrightarrow intv-elem\ x\ u\ (Intv\ d)$ |
 $c < u\ x \Longrightarrow intv-elem\ x\ u\ (Greater\ c)$

abbreviation *total-preorder* $r \equiv refl\ r \wedge trans\ r$

inductive *valid-region* :: $'c\ set \Rightarrow ('c \Rightarrow nat) \Rightarrow ('c \Rightarrow intv) \Rightarrow 'c\ rel \Rightarrow bool$
where
 $\llbracket X_0 = \{x \in X. \exists d. I\ x = Intv\ d\};\ refl-on\ X_0\ r;\ trans\ r;\ total-on\ X_0\ r;$
 $\forall x \in X. valid-intv\ (k\ x)\ (I\ x) \rrbracket$
 $\Longrightarrow valid-region\ X\ k\ I\ r$

inductive-set *region* **for** $X\ I\ r$

where

$$\begin{aligned} & \forall x \in X. u x \geq 0 \implies \forall x \in X. \text{intv-elem } x u (I x) \implies X_0 = \{x \in X. \\ & \exists d. I x = \text{Intv } d\} \implies \\ & \forall x \in X_0. \forall y \in X_0. (x, y) \in r \iff \text{frac } (u x) \leq \text{frac } (u y) \\ & \implies u \in \text{region } X I r \end{aligned}$$

Defining the unique element of a partition that contains a valuation

definition *part* ([-]- [61,61] 61) **where** *part* $v \mathcal{R} \equiv \text{THE } R. R \in \mathcal{R} \wedge v \in R$

inductive-set *Succ* **for** $\mathcal{R} R$ **where**

$$u \in R \implies R \in \mathcal{R} \implies R' \in \mathcal{R} \implies t \geq 0 \implies R' = [u \oplus t]_{\mathcal{R}} \implies R' \in \text{Succ } \mathcal{R} R$$

First we need to show that the set of regions is a partition of the set of all clock assignments. This property is only claimed by P. Bouyer.

inductive-cases^[elim!]: *intv-elem* $x u (Const d)$
inductive-cases^[elim!]: *intv-elem* $x u (Intv d)$
inductive-cases^[elim!]: *intv-elem* $x u (Greater d)$
inductive-cases^[elim!]: *valid-intv* $c (Greater d)$
inductive-cases^[elim!]: *valid-intv* $c (Const d)$
inductive-cases^[elim!]: *valid-intv* $c (Intv d)$

declare *valid-intv.intros*^[intro]

declare *intv-elem.intros*^[intro]

declare *Succ.intros*^[intro]

declare *Succ.cases*^[elim]

declare *region.cases*^[elim]

declare *valid-region.cases*^[elim]

7.2 Basic Properties

First we show that all valid intervals are distinct.

lemma *valid-intv-distinct*:

$$\begin{aligned} & \text{valid-intv } c I \implies \text{valid-intv } c I' \implies \text{intv-elem } x u I \implies \text{intv-elem } x u I' \\ & \implies I = I' \\ & \langle \text{proof} \rangle \end{aligned}$$

From this we show that all valid regions are distinct.

lemma *valid-regions-distinct*:

$$\begin{aligned} & \text{valid-region } X k I r \implies \text{valid-region } X k I' r' \implies v \in \text{region } X I r \implies \\ & v \in \text{region } X I' r' \end{aligned}$$

$\implies \text{region } X I r = \text{region } X I' r'$
 $\langle \text{proof} \rangle$

lemma *\mathcal{R} -regions-distinct:*

$\llbracket \mathcal{R} = \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}; R \in \mathcal{R}; v \in R; R' \in \mathcal{R}; R \neq R' \rrbracket \implies v \notin R'$
 $\langle \text{proof} \rangle$

Secondly, we also need to show that every valuations belongs to a region which is part of the partition.

definition *intv-of* :: $\text{nat} \Rightarrow t \Rightarrow \text{intv}$ **where**

intv-of $k c \equiv$
 if $(c > k)$ then *Greater* k
 else if $(\exists x :: \text{nat}. x = c)$ then $(\text{Const } (\text{nat } (\text{floor } c)))$
 else $(\text{Intv } (\text{nat } (\text{floor } c)))$

lemma *region-cover:*

$\forall x \in X. u \geq 0 \implies \exists R. R \in \{\text{region } X I r \mid I r. \text{valid-region } X k I r\} \wedge u \in R$
 $\langle \text{proof} \rangle$

lemma *intv-not-empty:*

obtains d **where** *intv-elem* $x (v(x := d)) (I x)$
 $\langle \text{proof} \rangle$

fun *get-intv-val* :: $\text{intv} \Rightarrow \text{real} \Rightarrow \text{real}$

where

get-intv-val $(\text{Const } d) \quad - = d \mid$
get-intv-val $(\text{Intv } d) \quad f = d + f \mid$
get-intv-val $(\text{Greater } d) \quad - = d + 1$

lemma *region-not-empty-aux:*

assumes $0 < f < 1 \quad 0 < g < 1$
shows $\text{frac } (\text{get-intv-val } (\text{Intv } d) f) \leq \text{frac } (\text{get-intv-val } (\text{Intv } d') g) \longleftrightarrow f \leq g$
 $\langle \text{proof} \rangle$

lemma *region-not-empty:*

assumes *finite* X *valid-region* $X k I r$
shows $\exists u. u \in \text{region } X I r$
 $\langle \text{proof} \rangle$

Now we can show that there is always exactly one region a valid valuation belongs to.

lemma *regions-partition*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies \forall x \in X. 0 \leq u \ x \implies \exists! R \in \mathcal{R}. u \in R$
<proof>

lemma *region-unique*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies u \in R \implies R \in \mathcal{R} \implies [u]_{\mathcal{R}} = R$
<proof>

lemma *regions-partition'*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies \forall x \in X. 0 \leq v \ x \implies \forall x \in X. 0 \leq v' \ x \implies v' \in [v]_{\mathcal{R}} \implies [v']_{\mathcal{R}} = [v]_{\mathcal{R}}$
<proof>

lemma *regions-closed*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies [v \oplus t]_{\mathcal{R}} \in \mathcal{R}$
<proof>

lemma *regions-closed'*:

$\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\} \implies R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies (v \oplus t) \in [v \oplus t]_{\mathcal{R}}$
<proof>

lemma *valid-regions-I-cong*:

$\text{valid-region } X \ k \ I \ r \implies \forall x \in X. I \ x = I' \ x \implies \text{region } X \ I \ r = \text{region } X \ I' \ r \wedge \text{valid-region } X \ k \ I' \ r$
<proof>

fun *intv-const* :: *intv* \Rightarrow *nat*

where

intv-const (*Const* *d*) = *d* |
intv-const (*Intv* *d*) = *d* |
intv-const (*Greater* *d*) = *d*

lemma *finite- \mathcal{R}* :

notes *[[simproc add: finite-Collect]] finite-subset[intro]*

fixes *X k*

defines $\mathcal{R} \equiv \{\text{region } X \ I \ r \mid I \ r. \text{ valid-region } X \ k \ I \ r\}$

assumes *finite X*

shows *finite \mathcal{R}*

<proof>

lemma *SuccI2*:

$$\begin{aligned} \mathcal{R} = \{ \text{region } X I r \mid I r. \text{ valid-region } X k I r \} &\implies v \in R \implies R \in \mathcal{R} \implies \\ t \geq 0 &\implies R' = [v \oplus t]_{\mathcal{R}} \\ &\implies R' \in \text{Succ } \mathcal{R} R \\ \langle \text{proof} \rangle \end{aligned}$$

7.3 Set of Regions

The first property Bouyer shows is that these regions form a 'set of regions'.

For the unbounded region in the upper right corner, the set of successors only contains itself.

lemma *Succ-refl*:

$$\begin{aligned} \mathcal{R} = \{ \text{region } X I r \mid I r. \text{ valid-region } X k I r \} &\implies \text{finite } X \implies R \in \mathcal{R} \implies \\ R &\in \text{Succ } \mathcal{R} R \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *Succ-refl'*:

$$\begin{aligned} \mathcal{R} = \{ \text{region } X I r \mid I r. \text{ valid-region } X k I r \} &\implies \text{finite } X \implies \forall x \in X. \\ \exists c. I x = \text{Greater } c & \\ \implies \text{region } X I r \in \mathcal{R} &\implies \text{Succ } \mathcal{R} (\text{region } X I r) = \{ \text{region } X I r \} \\ \langle \text{proof} \rangle \end{aligned}$$

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

definition

$$\begin{aligned} \text{succ } \mathcal{R} R &= \\ (\text{SOME } R'. R' \in \text{Succ } \mathcal{R} R \wedge (\forall u \in R. \forall t \geq 0. (u \oplus t) \notin R \longrightarrow (\exists t' & \\ \leq t. (u \oplus t') \in R' \wedge 0 \leq t'))) & \end{aligned}$$

inductive *isConst* :: *intv* \Rightarrow *bool*

where

isConst (*Const* -)

inductive *isIntv* :: *intv* \Rightarrow *bool*

where

isIntv (*Intv* -)

inductive *isGreater* :: *intv* \Rightarrow *bool*

where

isGreater (*Greater* -)

declare *isIntv.intros*[*intro!*] *isConst.intros*[*intro!*] *isGreater.intros*[*intro!*]

declare *isIntv.cases*[elim!] *isConst.cases*[elim!] *isGreater.cases*[elim!]

What Bouyer states at the end. However, we have to be a bit more precise than in her statement.

lemma *closest-prestable-1*:

fixes $I X k r$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$
defines $R \equiv region\ X\ I\ r$
defines $Z \equiv \{x \in X . \exists c. I\ x = Const\ c\}$
assumes $Z \neq \{\}$
defines $I' \equiv \lambda x. \text{if } x \notin Z \text{ then } I\ x \text{ else if } intv\text{-const } (I\ x) = k\ x \text{ then } Greater\ (k\ x) \text{ else } Intv\ (intv\text{-const } (I\ x))$
defines $r' \equiv r \cup \{(x,y) . x \in Z \wedge y \in X \wedge intv\text{-const } (I\ x) < k\ x \wedge isIntv\ (I'\ y)\}$
assumes *finite* X
assumes *valid-region* $X\ k\ I\ r$
shows $\forall v \in R. \forall t > 0. \exists t' \leq t. (v \oplus t') \in region\ X\ I'\ r' \wedge t' \geq 0$
and $\forall v \in region\ X\ I'\ r'. \forall t \geq 0. (v \oplus t) \notin R$
and $\forall x \in X. \neg isConst\ (I'\ x)$
and $\forall v \in R. \forall t < 1. \forall t' \geq 0. (v \oplus t') \in region\ X\ I'\ r' \longrightarrow \{x. x \in X \wedge (\exists c. I\ x = Intv\ c \wedge v\ x + t \geq c + 1)\} = \{x. x \in X \wedge (\exists c. I'\ x = Intv\ c \wedge (v \oplus t')\ x + (t - t') \geq c + 1)\}$
 $\langle proof \rangle$

lemma *closest-valid-1*:

fixes $I X k r$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$
defines $R \equiv region\ X\ I\ r$
defines $Z \equiv \{x \in X . \exists c. I\ x = Const\ c\}$
assumes $Z \neq \{\}$
defines $I' \equiv \lambda x. \text{if } x \notin Z \text{ then } I\ x \text{ else if } intv\text{-const } (I\ x) = k\ x \text{ then } Greater\ (k\ x) \text{ else } Intv\ (intv\text{-const } (I\ x))$
defines $r' \equiv r \cup \{(x,y) . x \in Z \wedge y \in X \wedge intv\text{-const } (I\ x) < k\ x \wedge isIntv\ (I'\ y)\}$
assumes *finite* X
assumes *valid-region* $X\ k\ I\ r$
shows *valid-region* $X\ k\ I'\ r'$
 $\langle proof \rangle$

lemma *closest-prestable-2*:

fixes $I X k r$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$

defines $R \equiv \text{region } X \ I \ r$
assumes $\forall x \in X. \neg \text{isConst } (I \ x)$
defines $X_0 \equiv \{x \in X. \text{isIntv } (I \ x)\}$
defines $M \equiv \{x \in X_0. \forall y \in X_0. (x, y) \in r \longrightarrow (y, x) \in r\}$
defines $I' \equiv \lambda x. \text{if } x \notin M \text{ then } I \ x \text{ else } \text{Const } (\text{intv-const } (I \ x) + 1)$
defines $r' \equiv \{(x, y) \in r. x \notin M \wedge y \notin M\}$
assumes *finite* X
assumes *valid-region* $X \ k \ I \ r$
assumes $M \neq \{\}$
shows $\forall v \in R. \forall t \geq 0. (v \oplus t) \notin R \longrightarrow (\exists t' \leq t. (v \oplus t') \in \text{region } X \ I' \ r' \wedge t' \geq 0)$
and $\forall v \in \text{region } X \ I' \ r'. \forall t \geq 0. (v \oplus t) \notin R$
and $\forall v \in R. \forall t'. \{x. x \in X \wedge (\exists c. I' \ x = \text{Intv } c \wedge (v \oplus t') \ x + (t - t') \geq \text{real } (c + 1))\}$
 $\quad = \{x. x \in X \wedge (\exists c. I \ x = \text{Intv } c \wedge v \ x + t \geq \text{real } (c + 1))\} - M$
and $\exists x \in X. \text{isConst } (I' \ x)$
 $\langle \text{proof} \rangle$

lemma *closest-valid-2:*

fixes $I \ X \ k \ r$
defines $\mathcal{R} \equiv \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$
defines $R \equiv \text{region } X \ I \ r$
assumes $\forall x \in X. \neg \text{isConst } (I \ x)$
defines $X_0 \equiv \{x \in X. \text{isIntv } (I \ x)\}$
defines $M \equiv \{x \in X_0. \forall y \in X_0. (x, y) \in r \longrightarrow (y, x) \in r\}$
defines $I' \equiv \lambda x. \text{if } x \notin M \text{ then } I \ x \text{ else } \text{Const } (\text{intv-const } (I \ x) + 1)$
defines $r' \equiv \{(x, y) \in r. x \notin M \wedge y \notin M\}$
assumes *finite* X
assumes *valid-region* $X \ k \ I \ r$
assumes $M \neq \{\}$
shows *valid-region* $X \ k \ I' \ r'$
 $\langle \text{proof} \rangle$

7.3.1 Putting the Proof for the 'Set of Regions' Property Together

Misc lemma *total-finite-trans-max:*

$X \neq \{\} \implies \text{finite } X \implies \text{total-on } X \ r \implies \text{trans } r \implies \exists x \in X. \forall y \in X. x \neq y \longrightarrow (y, x) \in r$
 $\langle \text{proof} \rangle$

lemma *card-mono-strict-subset:*

$\text{finite } A \implies \text{finite } B \implies \text{finite } C \implies A \cap B \neq \{\} \implies C = A - B \implies$

$card\ C < card\ A$
 $\langle proof \rangle$

Proof First we show that a shift by a non-negative integer constant means that any two valuations from the same region are being shifted to the same region.

lemma *int-shift-equiv*:

fixes $X\ k$ **fixes** $t :: int$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$
assumes $v \in R\ v' \in R\ R \in \mathcal{R}\ t \geq 0$
shows $(v' \oplus t) \in [v \oplus t]_{\mathcal{R}} \langle proof \rangle$

Now, we can use the 'immediate' induction proposed by P. Bouyer for shifts smaller than one. The induction principle is not at all obvious: the induction is over the set of clocks for which the valuation is shifted beyond the current interval boundaries. Using the two successor operations, we can see that either the set of these clocks remains the same ($Z =$) or strictly decreases ($Z =$).

lemma *set-of-regions-lt-1*:

fixes $X\ k\ I\ r\ t\ v$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$
defines $C \equiv \{x.\ x \in X \wedge (\exists\ c.\ I\ x = Intv\ c \wedge v\ x + t \geq c + 1)\}$
assumes $valid-region\ X\ k\ I\ r\ v \in region\ X\ I\ r\ v' \in region\ X\ I\ r\ finite\ X$
 $0 \leq t < 1$
shows $\exists\ t' \geq 0.\ (v' \oplus t') \in [v \oplus t]_{\mathcal{R}} \langle proof \rangle$

Finally, we can put the two pieces together: for a non-negative shift t , we first shift $\lfloor t \rfloor$ and then *frac* t .

lemma *set-of-regions*:

fixes $X\ k$
defines $\mathcal{R} \equiv \{region\ X\ I\ r \mid I\ r.\ valid-region\ X\ k\ I\ r\}$
assumes $R \in \mathcal{R}\ v \in R\ R' \in Succ\ \mathcal{R}\ R\ finite\ X$
shows $\exists\ t \geq 0.\ [v \oplus t]_{\mathcal{R}} = R' \langle proof \rangle$

7.4 Compability With Clock Constraints

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

definition *ccompatible*

where

$ccompatible\ \mathcal{R}\ cc \equiv \forall\ R \in \mathcal{R}.\ R \subseteq ccval\ cc \vee ccval\ cc \cap R = \{\}$

lemma *ccompatible1*:

fixes $X k$ **fixes** $c :: \text{real}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $c \leq k \ x \ c \in \mathbb{N} \ x \in X$
shows $\text{compatible } \mathcal{R} \ (\text{EQ } x \ c) \ \langle \text{proof} \rangle$

lemma *compatible2*:

fixes $X k$ **fixes** $c :: \text{real}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $c \leq k \ x \ c \in \mathbb{N} \ x \in X$
shows $\text{compatible } \mathcal{R} \ (\text{LT } x \ c) \ \langle \text{proof} \rangle$

lemma *compatible3*:

fixes $X k$ **fixes** $c :: \text{real}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $c \leq k \ x \ c \in \mathbb{N} \ x \in X$
shows $\text{compatible } \mathcal{R} \ (\text{LE } x \ c) \ \langle \text{proof} \rangle$

lemma *compatible4*:

fixes $X k$ **fixes** $c :: \text{real}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $c \leq k \ x \ c \in \mathbb{N} \ x \in X$
shows $\text{compatible } \mathcal{R} \ (\text{GT } x \ c) \ \langle \text{proof} \rangle$

lemma *compatible5*:

fixes $X k$ **fixes** $c :: \text{real}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $c \leq k \ x \ c \in \mathbb{N} \ x \in X$
shows $\text{compatible } \mathcal{R} \ (\text{GE } x \ c) \ \langle \text{proof} \rangle$

lemma *compatible*:

fixes $X k$ **fixes** $c :: \text{nat}$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$
assumes $\forall (x, m) \in \text{collect-clock-pairs} \ cc. \ m \leq k \ x \wedge x \in X \wedge m \in \mathbb{N}$
shows $\text{compatible } \mathcal{R} \ cc \ \langle \text{proof} \rangle$

7.5 Compability with Resets

definition *region-set*

where

$\text{region-set } R \ x \ c = \{v(x := c) \mid v. v \in R\}$

lemma *region-set-id*:

fixes $X k$
defines $\mathcal{R} \equiv \{\text{region } X I r \mid I r. \text{valid-region } X k I r\}$

assumes $R \in \mathcal{R} \ v \in R \ \text{finite } X \ 0 \leq c \ c \leq k \ x \ x \in X$
shows $[v(x := c)]_{\mathcal{R}} = \text{region-set } R \ x \ c \ [v(x := c)]_{\mathcal{R}} \in \mathcal{R} \ v(x := c) \in [v(x := c)]_{\mathcal{R}}$
 $\langle \text{proof} \rangle$

definition *region-set'*

where

region-set' $R \ r \ c = \{[r \rightarrow c]v \mid v. v \in R\}$

lemma *region-set'-id*:

fixes $X \ k$ **and** $c :: \text{nat}$

defines $\mathcal{R} \equiv \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$

assumes $R \in \mathcal{R} \ v \in R \ \text{finite } X \ 0 \leq c \ \forall \ x \in \text{set } r. \ c \leq k \ x \ \text{set } r \subseteq X$

shows $[[r \rightarrow c]v]_{\mathcal{R}} = \text{region-set}' \ R \ r \ c \ \wedge \ [[r \rightarrow c]v]_{\mathcal{R}} \in \mathcal{R} \ \wedge \ [r \rightarrow c]v \in [[r \rightarrow c]v]_{\mathcal{R}}$ $\langle \text{proof} \rangle$

7.6 A Semantics Based on Regions

7.6.1 Single step

inductive *step-r* ::

$(\ 'a, \ 'c, \ t, \ 's) \ \text{ta} \Rightarrow (\ 'c, \ t) \ \text{zone set} \Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ \text{zone} \Rightarrow \ 's \Rightarrow (\ 'c, \ t) \ \text{zone} \Rightarrow \text{bool}$

$(-, - \vdash \langle -, - \rangle \rightsquigarrow \langle -, - \rangle \ [61, 61, 61, 61] \ 61)$

where

step-t-r:

$\llbracket \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}; \text{valid-abstraction } A \ X \ k; R \in \mathcal{R}; R' \in \text{Succ } \mathcal{R} \ R;$

$R \subseteq \{\text{inv-of } A \ l\}; R' \subseteq \{\text{inv-of } A \ l\} \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l, R' \rangle \mid$

step-a-r:

$\llbracket \mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}; \text{valid-abstraction } A \ X \ k; A \vdash l \xrightarrow{g, a, r} l'; R \in \mathcal{R} \rrbracket$

$\implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', \text{region-set}' (R \cap \{u. u \vdash g\}) \ r \ 0 \cap \{u. u \vdash \text{inv-of } A \ l'\} \rangle$

inductive-cases[*elim!*]: $A, \mathcal{R} \vdash \langle l, u \rangle \rightsquigarrow \langle l', u' \rangle$

declare *step-r.intros*[*intro*]

lemma *region-cover'*:

assumes $\mathcal{R} = \{\text{region } X \ I \ r \mid I \ r. \text{valid-region } X \ k \ I \ r\}$ **and** $\forall x \in X. \ 0 \leq v \ x$

shows $v \in [v]_{\mathcal{R}} \ [v]_{\mathcal{R}} \in \mathcal{R}$

$\langle \text{proof} \rangle$

lemma *step-r-complete-aux*:

fixes $R\ r\ A\ l'\ g$
defines $R' \equiv \text{region-set}' (R \cap \{u. u \vdash g\})\ r\ 0 \cap \{u. u \vdash \text{inv-of } A\ l'\}$
assumes $\mathcal{R} = \{\text{region } X\ I\ r \mid I\ r. \text{valid-region } X\ k\ I\ r\}$
and *valid-abstraction* $A\ X\ k$
and $u \in R$
and $R \in \mathcal{R}$
and $A \vdash l \xrightarrow{g, a, r} l'$
and $u \vdash g$
and $[r \rightarrow 0]u \vdash \text{inv-of } A\ l'$
shows $R = R \cap \{u. u \vdash g\} \wedge R' = \text{region-set}'\ R\ r\ 0 \wedge R' \in \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *step-r-complete*:

$\llbracket A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle; \mathcal{R} = \{\text{region } X\ I\ r \mid I\ r. \text{valid-region } X\ k\ I\ r\};$
valid-abstraction $A\ X\ k;$
 $\forall x \in X. u\ x \geq 0 \rrbracket \implies \exists R'. A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow \langle l', R' \rangle \wedge u' \in R' \wedge R' \in \mathcal{R}$
 $\langle \text{proof} \rangle$

Compare this to lemma *step-z-sound*. This version is weaker because for regions we may very well arrive at a successor for which not every valuation can be reached by the predecessor. This is the case for e.g. the region with only Greater (k x) bounds.

lemma *step-r-sound*:

$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies \mathcal{R} = \{\text{region } X\ I\ r \mid I\ r. \text{valid-region } X\ k\ I\ r\}$
 $\implies R' \neq \{\}$ $\implies (\forall u \in R. \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow \langle l', u' \rangle)$
 $\langle \text{proof} \rangle$

7.6.2 Multi Step

inductive

steps-r :: $(l'a, l'c, t, l's) \text{ ta} \Rightarrow (l'c, t) \text{ zone set} \Rightarrow l's \Rightarrow (l'c, t) \text{ zone} \Rightarrow l's \Rightarrow$
 $(l'c, t) \text{ zone} \Rightarrow \text{bool}$
 $(-, - \vdash \langle -, - \rangle \rightsquigarrow^* \langle -, - \rangle [61, 61, 61, 61, 61, 61] 61)$

where

refl: $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l, R \rangle \mid$
step: $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \implies A, \mathcal{R} \vdash \langle l', R' \rangle \rightsquigarrow \langle l'', R'' \rangle \implies A, \mathcal{R} \vdash$
 $\langle l, R \rangle \rightsquigarrow^* \langle l'', R'' \rangle$

declare *steps-r.intros*[intro]

lemma *steps-alt*:

$$A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle \implies A \vdash \langle l', u' \rangle \rightarrow \langle l'', u'' \rangle \implies A \vdash \langle l, u \rangle \rightarrow^* \langle l'', u'' \rangle$$

<proof>

lemma emptiness-preservation: $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies R = \{\} \implies R' = \{\}$

<proof>

lemma emptiness-preservation-steps: $A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \implies R = \{\} \implies R' = \{\}$

<proof>

Note how it is important to define the multi-step semantics "the right way round". This also the direction Bouyer implies for her implicit induction.

lemma steps-r-sound:

$$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \implies \mathcal{R} = \{\text{region } X \text{ I } r \mid \text{I } r. \text{ valid-region } X \text{ k I } r\}$$

$$\implies R' \neq \{\} \implies u \in R \implies \exists u' \in R'. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$$

<proof>

lemma steps-r-sound':

$$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle \implies \mathcal{R} = \{\text{region } X \text{ I } r \mid \text{I } r. \text{ valid-region } X \text{ k I } r\}$$

$$\implies R' \neq \{\} \implies (\exists u' \in R'. \exists u \in R. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle)$$

<proof>

lemma single-step-r:

$$A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l', R' \rangle$$

<proof>

lemma steps-r-alt:

$$A, \mathcal{R} \vdash \langle l', R' \rangle \rightsquigarrow^* \langle l'', R'' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow \langle l', R' \rangle \implies A, \mathcal{R} \vdash \langle l, R \rangle \rightsquigarrow^* \langle l'', R'' \rangle$$

<proof>

lemma single-step:

$$x1 \vdash \langle x2, x3 \rangle \rightarrow \langle x4, x5 \rangle \implies x1 \vdash \langle x2, x3 \rangle \rightarrow^* \langle x4, x5 \rangle$$

<proof>

lemma steps-r-complete:

$$\llbracket A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle; \mathcal{R} = \{\text{region } X \text{ I } r \mid \text{I } r. \text{ valid-region } X \text{ k I } r\}; \text{valid-abstraction } A \text{ X k};$$

$$\forall x \in X. u \ x \geq 0 \rrbracket \implies \exists R'. A, \mathcal{R} \vdash \langle l, ([u]_{\mathcal{R}}) \rangle \rightsquigarrow^* \langle l', R' \rangle \wedge u' \in R'$$

<proof>

```

end
theory Closure
  imports Regions
begin

```

7.7 Correct Approximation of Zones with α -regions

```

locale AlphaClosure =
  fixes X k  $\mathcal{R}$  and V :: ('c, t) cval set
  defines  $\mathcal{R} \equiv \{region\ X\ I\ r\ \mid\ I\ r.\ valid\_region\ X\ k\ I\ r\}$ 
  defines  $V \equiv \{v.\ \forall\ x \in X.\ v\ x \geq 0\}$ 
  assumes finite: finite X
begin

lemmas set-of-regions-spec = set-of-regions[OF - - - finite, of - k, folded
 $\mathcal{R}$ -def]
lemmas region-cover-spec = region-cover[of X - k, folded  $\mathcal{R}$ -def]
lemmas region-unique-spec = region-unique[of  $\mathcal{R}$  X k, folded  $\mathcal{R}$ -def, sim-
simplified]
lemmas regions-closed'-spec = regions-closed'[of  $\mathcal{R}$  X k, folded  $\mathcal{R}$ -def, sim-
simplified]

lemma valid-regions-distinct-spec:
   $R \in \mathcal{R} \implies R' \in \mathcal{R} \implies v \in R \implies v \in R' \implies R = R'$ 
  <proof>

definition cla ( $Closure_\alpha$  - [71] 71)
where
   $cla\ Z = \bigcup \{R \in \mathcal{R}.\ R \cap Z \neq \{\}\}$ 

The nice and easy properties proved by Bouyer lemma closure-constraint-id:
   $\forall (x, m) \in collect\_clock\_pairs\ g.\ m \leq real\ (k\ x) \wedge x \in X \wedge m \in \mathbf{N} \implies$ 
 $Closure_\alpha\ \{g\} = \{g\} \cap V$ 
  <proof>

lemma closure-id':
   $Z \neq \{\} \implies Z \subseteq R \implies R \in \mathcal{R} \implies Closure_\alpha\ Z = R$ 
  <proof>

lemma closure-id:
   $Closure_\alpha\ Z \neq \{\} \implies Z \subseteq R \implies R \in \mathcal{R} \implies Closure_\alpha\ Z = R$ 
  <proof>

lemma closure-update-mono:

```

$Z \subseteq V \implies \text{set } r \subseteq X \implies \text{zone-set } (\text{Closure}_\alpha Z) r \subseteq \text{Closure}_\alpha(\text{zone-set } Z r)$
 ⟨proof⟩

lemma *SuccI3*:

$R \in \mathcal{R} \implies v \in R \implies t \geq 0 \implies (v \oplus t) \in R' \implies R' \in \mathcal{R} \implies R' \in \text{Succ } \mathcal{R} R$
 ⟨proof⟩

lemma *closure-delay-mono*:

$Z \subseteq V \implies (\text{Closure}_\alpha Z)^\dagger \subseteq \text{Closure}_\alpha (Z^\dagger)$
 ⟨proof⟩

lemma *region-V*: $R \in \mathcal{R} \implies R \subseteq V$ ⟨proof⟩

lemma *closure-V*:

$\text{Closure}_\alpha Z \subseteq V$
 ⟨proof⟩

lemma *closure-V-int*:

$\text{Closure}_\alpha Z = \text{Closure}_\alpha (Z \cap V)$
 ⟨proof⟩

lemma *closure-constraint-mono*:

$\text{Closure}_\alpha g = g \implies g \cap (\text{Closure}_\alpha Z) \subseteq \text{Closure}_\alpha (g \cap Z)$
 ⟨proof⟩

lemma *closure-constraint-mono'*:

assumes $\text{Closure}_\alpha g = g \cap V$
shows $g \cap (\text{Closure}_\alpha Z) \subseteq \text{Closure}_\alpha (g \cap Z)$
 ⟨proof⟩

lemma *cla-empty-iff*:

$Z \subseteq V \implies Z = \{\} \iff \text{Closure}_\alpha Z = \{\}$
 ⟨proof⟩

lemma *closure-involutive-aux*:

$U \subseteq \mathcal{R} \implies \text{Closure}_\alpha \bigcup U = \bigcup U$
 ⟨proof⟩

lemma *closure-involutive-aux'*:

$\exists U. U \subseteq \mathcal{R} \wedge \text{Closure}_\alpha Z = \bigcup U$
 ⟨proof⟩

lemma *closure-involutive*:

$$\text{Closure}_\alpha \text{ Closure}_\alpha Z = \text{Closure}_\alpha Z$$

<proof>

lemma *closure-involutive'*:

$$Z \subseteq \text{Closure}_\alpha W \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha W$$

<proof>

lemma *closure-subst*:

$$Z \subseteq V \implies Z \subseteq \text{Closure}_\alpha Z$$

<proof>

lemma *cla-mono'*:

$$Z' \subseteq V \implies Z \subseteq Z' \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha Z'$$

<proof>

lemma *cla-mono*:

$$Z \subseteq Z' \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha Z'$$

<proof>

7.8 A New Zone Semantics Abstracting with Closure_α

7.8.1 Single step

inductive *step-z-alpha* ::

$$('a, 'c, t, 's) \text{ ta} \implies 's \implies ('c, t) \text{ zone} \implies 's \implies ('c, t) \text{ zone} \implies \text{bool}$$
$$(- \vdash \langle -, - \rangle \rightsquigarrow_\alpha \langle -, - \rangle) \text{ [61,61,61] 61}$$

where

$$\text{step-alpha}: A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_\alpha \langle l', \text{Closure}_\alpha Z \rangle$$

$$\text{inductive-cases}[\text{elim!}]: A \vdash \langle l, u \rangle \rightsquigarrow_\alpha \langle l', u \rangle$$

declare *step-z-alpha.intros*[*intro*]

lemma *up-V*: $Z \subseteq V \implies Z^\uparrow \subseteq V$

<proof>

lemma *reset-V*: $Z \subseteq V \implies (\text{zone-set } Z \text{ } r) \subseteq V$

<proof>

lemma *step-z-V*: $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$

<proof>

Single-step soundness and completeness follows trivially from *cla-empty-iff*.

lemma *step-z-alpha-sound*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \neq \{\} \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle \wedge Z'' \neq \{\}$$

<proof>

lemma *step-z-alpha-complete*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \neq \{\} \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$$

<proof>

lemma *zone-set-mono*:

$$A \subseteq B \implies \text{zone-set } A \text{ } r \subseteq \text{zone-set } B \text{ } r$$

<proof>

lemma *zone-delay-mono*:

$$A \subseteq B \implies A^{\uparrow} \subseteq B^{\uparrow}$$

<proof>

7.8.2 Multi step

inductive

$$\text{steps-z-alpha} :: ('a, 'c, t, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow \text{bool}$$

$$(- \vdash \langle -, - \rangle \rightsquigarrow_{\alpha^*} \langle -, - \rangle) [61, 61, 61] \text{ 61}$$

where

$$\text{refl: } A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l, Z \rangle \mid$$

$$\text{step: } A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow_{\alpha} \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l'', Z'' \rangle$$

declare *steps-z-alpha.intros*[intro]

lemma *subset-int-mono*: $A \subseteq B \implies A \cap C \subseteq B \cap C$ *<proof>*

P. Bouyer's calculation for $\text{Post} (\text{Closure}_{\alpha} Z, e) \subseteq \text{Closure}_{\alpha} \text{Post} (Z, e)$

This is now obsolete as we argue solely with monotonicity of *steps-z* w.r.t Closure_{α}

lemma *calc*:

$$\text{valid-abstraction } A \text{ } X \text{ } k \implies Z \subseteq V \implies A \vdash \langle l, \text{Closure}_{\alpha} Z \rangle \rightsquigarrow \langle l', Z' \rangle$$

$$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$$

<proof>

Turning P. Bouyer's argument for multiple steps into an inductive proof is not direct. With this initial argument we can get to a point where the induction hypothesis is applicable. This breaks the "information hiding" induced by the different variants of steps.

lemma *steps-z-alpha-closure-involutive'-aux*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha W \implies \text{valid-abstraction}$
 $A \ X \ k \implies Z \subseteq V$
 $\implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow \langle l', W' \rangle \wedge \text{Closure}_\alpha Z' \subseteq \text{Closure}_\alpha W'$
<proof>

lemma *steps-z-alpha-closure-involutive'-aux'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies \text{Closure}_\alpha Z \subseteq \text{Closure}_\alpha W \implies \text{valid-abstraction}$
 $A \ X \ k \implies Z \subseteq V \implies W \subseteq Z$
 $\implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow \langle l', W' \rangle \wedge \text{Closure}_\alpha Z' \subseteq \text{Closure}_\alpha W' \wedge W' \subseteq Z'$
<proof>

lemma *steps-z-alt*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l'', Z'' \rangle$
<proof>

lemma *steps-z-alpha-V*: $A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha*} \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$
<proof>

lemma *steps-z-alpha-closure-involutive'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha*} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow \langle l'', Z'' \rangle \implies \text{valid-abstraction}$
 $A \ X \ k \implies Z \subseteq V$
 $\implies \exists Z'''. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l'', Z''' \rangle \wedge \text{Closure}_\alpha Z'' \subseteq \text{Closure}_\alpha Z''' \wedge Z''' \subseteq Z''$
<proof>

Old proof using Bouyer's calculation

lemma *steps-z-alpha-closure-involutive''*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha*} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow \langle l'', Z'' \rangle \implies \text{valid-abstraction}$
 $A \ X \ k \implies Z \subseteq V$
 $\implies \exists Z'''. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l'', Z''' \rangle \wedge \text{Closure}_\alpha Z'' \subseteq \text{Closure}_\alpha Z'''$
<proof>

lemma *steps-z-alpha-closure-involutive*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha*} \langle l', Z' \rangle \implies \text{valid-abstraction}$ $A \ X \ k \implies Z \subseteq V$
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z'' \rangle \wedge \text{Closure}_\alpha Z' \subseteq \text{Closure}_\alpha Z'' \wedge Z'' \subseteq Z'$
<proof>

lemma *steps-z-V*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_* \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$
<proof>

lemma *steps-z-alpha-sound*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies Z' \neq \{\}$$

$$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z'' \rangle \wedge Z'' \neq \{\} \wedge Z'' \subseteq Z'$$

<proof>

lemma *step-z-mono*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies Z \subseteq W \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow \langle l', W' \rangle \wedge Z' \subseteq W'$$

<proof>

lemma *step-z-alpha-mono*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\alpha} \langle l', W' \rangle \wedge Z' \subseteq W'$$

<proof>

lemma *steps-z-alpha-mono*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\alpha^*} \langle l', W' \rangle \wedge Z' \subseteq W'$$

<proof>

lemma *steps-z-alpha-alt*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow_{\alpha^*} \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l'', Z'' \rangle$$

<proof>

lemma *steps-z-alpha-complete*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies Z' \neq \{\}$$

$$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$$

<proof>

lemma *steps-z-alpha-complete'*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies Z' \neq \{\}$$

$$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$$

<proof>

end

end

theory *Approx-Beta*

imports *DBM-Zone-Semantics Regions-Beta Closure*

begin

8 Correctness of β -approximation from α -regions

Instantiating real

instantiation *real* :: *linordered-ab-monoid-add*
begin

definition

neutral-real: $\mathbf{1} = (0 :: \text{real})$

instance *<proof>*

end

Merging the locales for the two types of regions

locale *Regions* =

fixes *X* **and** *k* :: '*c* \Rightarrow *nat* **and** *v* :: '*c* \Rightarrow *nat* **and** *n* :: *nat* **and** *not-in-X*
assumes *finite*: *finite X*

assumes *clock-numbering*: *clock-numbering' v n $\forall k \leq n. k > 0 \longrightarrow (\exists c \in X. v c = k)$*

$\forall c \in X. v c \leq n$

assumes *not-in-X*: *not-in-X $\notin X$*

assumes *non-empty*: *X $\neq \{\}$*

begin

definition *R-def*: $\mathcal{R} \equiv \{\text{Regions.region } X \ I \ r \mid I \ r. \text{Regions.valid-region } X \ k \ I \ r\}$

definition *R _{β}* -def: $\mathcal{R}_\beta \equiv \{\text{Regions-Beta.region } X \ I \ J \ r \mid I \ J \ r. \text{Regions-Beta.valid-region } X \ k \ I \ J \ r\}$

definition *V-def*: $V \equiv \{v . \forall x \in X. v x \geq 0\}$

sublocale *alpha-interp*: *AlphaClosure X k \mathcal{R} V <proof>*

sublocale *beta-interp*: *Beta-Regions' X k \mathcal{R}_β V v n not-in-X <proof>*

abbreviation *Approx _{β}* \equiv *beta-interp.Approx _{β}*

8.1 Preparing Bouyer's Theorem

lemma *region-dbm*:

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

defines $v' \equiv \lambda i. \text{THE } c. c \in X \wedge v c = i$
obtains M
where $[M]_{v,n} = R$
and $\forall i \leq n. \forall j \leq n. M i 0 = \infty \wedge j > 0 \wedge i \neq j \longrightarrow M i j = \infty \wedge M j i = \infty$
and $\forall i \leq n. M i i = \text{Le } 0$
and $\forall i \leq n. \forall j \leq n. i > 0 \wedge j > 0 \wedge M i 0 \neq \infty \wedge M j 0 \neq \infty \longrightarrow$
 $(\exists d :: \text{int.}$
 $\quad (-k(v' j) \leq d \wedge d \leq k(v' i) \wedge M i j = \text{Le } d \wedge M j i = \text{Le } (-d))$
 $\quad \vee (-k(v' j) \leq d - 1 \wedge d \leq k(v' i) \wedge M i j = \text{Lt } d \wedge M j i = \text{Lt } (-d + 1)))$
and $\forall i \leq n. i > 0 \wedge M i 0 \neq \infty \longrightarrow$
 $(\exists d :: \text{int. } d \leq k(v' i) \wedge d \geq 0$
 $\quad \wedge (M i 0 = \text{Le } d \wedge M 0 i = \text{Le } (-d) \vee M i 0 = \text{Lt } d \wedge M 0 i = \text{Lt } (-d + 1)))$
and $\forall i \leq n. i > 0 \longrightarrow (\exists d :: \text{int. } -k(v' i) \leq d \wedge d \leq 0 \wedge (M 0 i = \text{Le } d \vee M 0 i = \text{Lt } d))$
and $\forall i. \forall j. M i j \neq \infty \longrightarrow \text{get-const } (M i j) \in \mathbb{Z}$
and $\forall i \leq n. \forall j \leq n. M i j \neq \infty \wedge i > 0 \wedge j > 0 \longrightarrow$
 $(\exists d :: \text{int. } (M i j = \text{Le } d \vee M i j = \text{Lt } d) \wedge (-k(v' j)) \leq d \wedge d \leq k(v' i))$
 $\langle \text{proof} \rangle$

lemma *len-inf-elem*:

$(a, b) \in \text{set } (\text{arcs } i j \text{ xs}) \implies M a b = \infty \implies \text{len } M i j \text{ xs} = \infty$
 $\langle \text{proof} \rangle$

lemma *dbm-add-strict-right-mono-neutral*: $a < \text{Le } d \implies a + \text{Le } (-d) < \text{Le } 0$

$\langle \text{proof} \rangle$

lemma *dbm-lt-not-inf-less[intro]*: $A \neq \infty \implies A < \infty$ $\langle \text{proof} \rangle$

lemma *add-inf[simp]*:

$a + \infty = \infty \quad \infty + a = \infty$
 $\langle \text{proof} \rangle$

lemma *inf-lt[simp,dest!]*:

$\infty < x \implies \text{False}$
 $\langle \text{proof} \rangle$

lemma *zone-diag-lt*:

assumes $a \leq n \quad b \leq n$ **and** $C: v c1 = a \quad v c2 = b$ **and** $\text{not } 0: a > 0 \quad b > 0$
shows $[(\lambda i j. \text{if } i = a \wedge j = b \text{ then } \text{Lt } d \text{ else } \infty)]_{v,n} = \{u. u c1 - u c2$

$< d\}$
 $\langle proof \rangle$

lemma zone-diag-le:

assumes $a \leq n$ **and** $b \leq n$ **and** $C: v\ c1 = a\ v\ c2 = b$ **and** $not0: a > 0\ b > 0$
shows $[(\lambda\ i\ j. \text{if } i = a \wedge j = b \text{ then } Le\ d \text{ else } \infty)]_{v,n} = \{u. u\ c1 - u\ c2 \leq d\}$
 $\langle proof \rangle$

lemma zone-diag-lt-2:

assumes $a \leq n$ **and** $C: v\ c = a$ **and** $not0: a > 0$
shows $[(\lambda\ i\ j. \text{if } i = a \wedge j = 0 \text{ then } Lt\ d \text{ else } \infty)]_{v,n} = \{u. u\ c < d\}$
 $\langle proof \rangle$

lemma zone-diag-le-2:

assumes $a \leq n$ **and** $C: v\ c = a$ **and** $not0: a > 0$
shows $[(\lambda\ i\ j. \text{if } i = a \wedge j = 0 \text{ then } Le\ d \text{ else } \infty)]_{v,n} = \{u. u\ c \leq d\}$
 $\langle proof \rangle$

lemma zone-diag-lt-3:

assumes $a \leq n$ **and** $C: v\ c = a$ **and** $not0: a > 0$
shows $[(\lambda\ i\ j. \text{if } i = 0 \wedge j = a \text{ then } Lt\ d \text{ else } \infty)]_{v,n} = \{u. -u\ c < d\}$
 $\langle proof \rangle$

lemma len-int-closed:

$\forall\ i\ j. (M\ i\ j :: real) \in \mathbb{Z} \implies len\ M\ i\ j\ xs \in \mathbb{Z}$
 $\langle proof \rangle$

lemma get-const-distr:

$a \neq \infty \implies b \neq \infty \implies get\ const\ (a + b) = get\ const\ a + get\ const\ b$
 $\langle proof \rangle$

lemma len-int-dbm-closed:

$\forall\ (i, j) \in set\ (arcs\ i\ j\ xs). (get\ const\ (M\ i\ j) :: real) \in \mathbb{Z} \wedge M\ i\ j \neq \infty$
 $\implies get\ const\ (len\ M\ i\ j\ xs) \in \mathbb{Z} \wedge len\ M\ i\ j\ xs \neq \infty$
 $\langle proof \rangle$

lemma zone-diag-le-3:

assumes $a \leq n$ **and** $C: v\ c = a$ **and** $not0: a > 0$
shows $[(\lambda\ i\ j. \text{if } i = 0 \wedge j = a \text{ then } Le\ d \text{ else } \infty)]_{v,n} = \{u. -u\ c \leq d\}$
 $\langle proof \rangle$

lemma dbm-lt':

assumes $[M]_{v,n} \subseteq V\ M\ a\ b \leq Lt\ d\ a \leq n\ b \leq n\ v\ c1 = a\ v\ c2 = b\ a >$

$0 b > 0$

shows $[M]_{v,n} \subseteq \{u \in V. u c1 - u c2 < d\}$
<proof>

lemma *dbm-lt'2*:

assumes $[M]_{v,n} \subseteq V M a 0 \leq Lt d a \leq n v c1 = a a > 0$
shows $[M]_{v,n} \subseteq \{u \in V. u c1 < d\}$
<proof>

lemma *dbm-lt'3*:

assumes $[M]_{v,n} \subseteq V M 0 a \leq Lt d a \leq n v c1 = a a > 0$
shows $[M]_{v,n} \subseteq \{u \in V. - u c1 < d\}$
<proof>

lemma *dbm-le'*:

assumes $[M]_{v,n} \subseteq V M a b \leq Le d a \leq n b \leq n v c1 = a v c2 = b a > 0 b > 0$
shows $[M]_{v,n} \subseteq \{u \in V. u c1 - u c2 \leq d\}$
<proof>

lemma *dbm-le'2*:

assumes $[M]_{v,n} \subseteq V M a 0 \leq Le d a \leq n v c1 = a a > 0$
shows $[M]_{v,n} \subseteq \{u \in V. u c1 \leq d\}$
<proof>

lemma *dbm-le'3*:

assumes $[M]_{v,n} \subseteq V M 0 a \leq Le d a \leq n v c1 = a a > 0$
shows $[M]_{v,n} \subseteq \{u \in V. - u c1 \leq d\}$
<proof>

lemma *int-zone-dbm*:

assumes $\forall (-,d) \in collect-clock-pairs cc. d \in \mathbf{Z} \forall c \in collect-clks cc. v c \leq n$
obtains M **where** $\{u. u \vdash cc\} = [M]_{v,n}$ **and** *dbm-int* $M n$
<proof>

lemma *non-empty-dbm-diag-set'*:

assumes *clock-numbering'* $v n \forall i \leq n. \forall j \leq n. M i j \neq \infty \longrightarrow get-const (M i j) \in \mathbf{Z}$
 $[M]_{v,n} \neq \{\}$
obtains M' **where** $[M]_{v,n} = [M']_{v,n} \wedge (\forall i \leq n. \forall j \leq n. M' i j \neq \infty \longrightarrow get-const (M' i j) \in \mathbf{Z})$
 $\wedge (\forall i \leq n. M' i i = \mathbf{1})$
<proof>

lemma *dbm-entry-int*:

$x \neq \infty \implies \text{get-const } x \in \mathbf{Z} \implies \exists d :: \text{int. } x = \text{Le } d \vee x = \text{Lt } d$
<proof>

abbreviation $\text{vabstr} \equiv \text{beta-interp.vabstr}$

8.2 Bouyer's Main Theorem

theorem *region-zone-intersect-empty-approx-correct*:

assumes $R \in \mathcal{R} \ Z \subseteq V \ R \cap Z = \{\}$ $\text{vabstr } Z \ M$
shows $R \cap \text{Approx}_\beta Z = \{\}$
<proof>

8.3 Nice Corollaries of Bouyer's Theorem

lemma *\mathcal{R} -V*: $\bigcup \mathcal{R} = V$ *<proof>*

lemma *regions-beta-V*: $R \in \mathcal{R}_\beta \implies R \subseteq V$ *<proof>*

lemma *apx-V*: $Z \subseteq V \implies \text{Approx}_\beta Z \subseteq V$
<proof>

corollary *approx-beta-closure-alpha*:

assumes $Z \subseteq V \ \text{vabstr } Z \ M$
shows $\text{Approx}_\beta Z \subseteq \text{Closure}_\alpha Z$
<proof>

definition $V' \equiv \{Z. Z \subseteq V \wedge (\exists M. \text{vabstr } Z \ M)\}$

corollary *approx-beta-closure-alpha'*: $Z \in V' \implies \text{Approx}_\beta Z \subseteq \text{Closure}_\alpha Z$
<proof>

We could prove this more directly too (without using $\text{Closure}_\alpha Z$), obviously

lemma *apx-empty-iff*:

assumes $Z \subseteq V \ \text{vabstr } Z \ M$
shows $Z = \{\} \longleftrightarrow \text{Approx}_\beta Z = \{\}$
<proof>

lemma *apx-empty-iff'*:

assumes $Z \in V'$ **shows** $Z = \{\} \longleftrightarrow \text{Approx}_\beta Z = \{\}$
<proof>

lemma *apx-V'*:

assumes $Z \subseteq V$ **shows** $\text{Approx}_\beta Z \in V'$
 $\langle \text{proof} \rangle$

8.4 A New Zone Semantics Abstracting with Approx_β

lemma *step-z-V'*:

assumes $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle$ *valid-abstraction* $A \ X \ k \ \forall c \in \text{clk-set } A. v \ c$
 $\leq n \ Z \in V'$

shows $Z' \in V'$

$\langle \text{proof} \rangle$

lemma *steps-z-V'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow^* \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A. v \ c$
 $\leq n \implies Z \in V' \implies Z' \in V'$

$\langle \text{proof} \rangle$

8.4.1 Single Step

inductive *step-z-beta* ::

$(\ 'a, 'c, t, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow \text{bool}$
 $(- \vdash \langle -, - \rangle \rightsquigarrow_\beta \langle -, - \rangle) \ [61, 61, 61] \ 61)$

where

step-beta: $A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', \text{Approx}_\beta Z' \rangle$

inductive-cases[*elim!*]: $A \vdash \langle l, u \rangle \rightsquigarrow_\beta \langle l', u' \rangle$

declare *step-z-beta.intros*[*intro*]

lemma *step-z-alpha-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A. v \ c$
 $\leq n \implies Z \in V' \implies Z' \neq \{\}$

$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

$\langle \text{proof} \rangle$

lemma *step-z-alpha-complete*:

$A \vdash \langle l, Z \rangle \rightsquigarrow \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies \forall c \in \text{clk-set } A. v \ c$
 $\leq n \implies Z \in V' \implies Z' \neq \{\}$

$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_\beta \langle l', Z'' \rangle \wedge Z'' \neq \{\}$

$\langle \text{proof} \rangle$

8.4.2 Multi step

inductive

steps-z-beta :: $(\ 'a, 'c, t, 's) \text{ ta} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow 's \Rightarrow ('c, t) \text{ zone} \Rightarrow$
 bool

$(- \vdash \langle -, - \rangle \rightsquigarrow_{\beta^*} \langle -, - \rangle)$ [61,61,61] 61)

where

$refl: A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l, Z \rangle$ |
 $step: A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow_{\beta} \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l'', Z'' \rangle$

declare *steps-z-beta.intros*[intro]

lemma *V'-V*: $Z \in V' \implies Z \subseteq V$ <proof>

lemma *steps-z-beta-V'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies valid-abstraction A X k \implies \forall c \in clk-set A. v c \leq n \implies Z \in V' \implies Z' \in V'$
 <proof>

lemma *alpha-beta-step*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \implies valid-abstraction A X k \implies \forall c \in clk-set A. v c \leq n \implies Z \in V'$
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$
 <proof>

Soundness lemma *alpha-beta-step'*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \implies valid-abstraction A X k \implies \forall c \in clk-set A. v c \leq n \implies Z \in V' \implies W \subseteq V$
 $\implies Z \subseteq W \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\alpha} \langle l', W' \rangle \wedge Z' \subseteq W'$
 <proof>

lemma *alpha-beta-steps*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies valid-abstraction A X k \implies \forall c \in clk-set A. v c \leq n \implies Z \in V'$
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\alpha^*} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$
 <proof>

corollary *steps-z-beta-sound*:

$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies \forall c \in clk-set A. v c \leq n \implies valid-abstraction A X k \implies Z \in V' \implies Z' \neq \{\}$
 $\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$
 <proof>

Completeness lemma *apx-mono*:

$Z' \subseteq V \implies Z \subseteq Z' \implies Approx_{\beta} Z \subseteq Approx_{\beta} Z'$
 <proof>

lemma *step-z-beta-mono*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\beta} \langle l', W' \rangle \wedge Z' \subseteq W'$$

<proof>

lemma *steps-z-beta-V*: $A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies Z \subseteq V \implies Z' \subseteq V$
<proof>

lemma *steps-z-beta-mono*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies Z \subseteq W \implies W \subseteq V \implies \exists W'. A \vdash \langle l, W \rangle \rightsquigarrow_{\beta^*} \langle l', W' \rangle \wedge Z' \subseteq W'$$

<proof>

lemma *steps-z-beta-alt*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta} \langle l', Z' \rangle \implies A \vdash \langle l', Z' \rangle \rightsquigarrow_{\beta^*} \langle l'', Z'' \rangle \implies A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l'', Z'' \rangle$$

<proof>

lemma *steps-z-beta-complete*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z'' \rangle \wedge Z' \subseteq Z''$$

<proof>

lemma *steps-z-beta-complete'*:

$$A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z' \rangle \implies \text{valid-abstraction } A \ X \ k \implies Z \subseteq V \implies Z' \neq \{\}$$

$$\implies \exists Z''. A \vdash \langle l, Z \rangle \rightsquigarrow_{\beta^*} \langle l', Z'' \rangle \wedge Z'' \neq \{\}$$

<proof>

end

end

9 Forward Analysis with DBMs and Widening

theory *Normalized-Zone-Semantics*

imports *DBM-Zone-Semantics Approx-Beta*

begin

9.1 DBM-based Semantics with Normalization

9.1.1 Single Step

inductive *step-z-norm* ::

$(\prime a, \prime c, t, \prime s) ta \Rightarrow \prime s \Rightarrow t DBM \Rightarrow (nat \Rightarrow nat) \Rightarrow (\prime c \Rightarrow nat) \Rightarrow nat \Rightarrow \prime s \Rightarrow t DBM \Rightarrow bool$
 $(- \vdash \langle -, - \rangle \rightsquigarrow_{-, -, -} \langle -, - \rangle [61, 61, 61, 61, 61] 61)$
where *step-z-norm*:
 $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', D' \rangle \Longrightarrow A \vdash \langle l, D \rangle \rightsquigarrow_{k, v, n} \langle l', norm (FW D' n) k n \rangle$

inductive *steps-z-norm* ::

$(\prime a, \prime c, t, \prime s) ta \Rightarrow \prime s \Rightarrow t DBM \Rightarrow (nat \Rightarrow nat) \Rightarrow (\prime c \Rightarrow nat) \Rightarrow nat \Rightarrow \prime s \Rightarrow t DBM \Rightarrow bool$
 $(- \vdash \langle -, - \rangle \rightsquigarrow_{-, -, -*} \langle -, - \rangle [61, 61, 61, 61, 61] 61)$

where

$refl: A \vdash \langle l, Z \rangle \rightsquigarrow_{k, v, n*} \langle l, Z \rangle \mid$
 $step: A \vdash \langle l, Z \rangle \rightsquigarrow_{k, v, n*} \langle l', Z' \rangle \Longrightarrow A \vdash \langle l', Z' \rangle \rightsquigarrow_{k, v, n} \langle l'', Z'' \rangle$
 $\Longrightarrow A \vdash \langle l, Z \rangle \rightsquigarrow_{k, v, n*} \langle l'', Z'' \rangle$

context *Regions*

begin

abbreviation $v' \equiv beta\text{-interp}.v'$

abbreviation *step-z-norm'* $(- \vdash \langle -, - \rangle \rightsquigarrow_{\mathcal{N}} \langle -, - \rangle [61, 61, 61] 61)$

where

$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}} \langle l', D' \rangle \equiv A \vdash \langle l, D \rangle \rightsquigarrow_{(k \circ v'), v, n} \langle l', D' \rangle$

abbreviation *steps-z-norm'* $(- \vdash \langle -, - \rangle \rightsquigarrow_{\mathcal{N}*} \langle -, - \rangle [61, 61, 61] 61)$

where

$A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}*} \langle l', D' \rangle \equiv A \vdash \langle l, D \rangle \rightsquigarrow_{(k \circ v'), v, n*} \langle l', D' \rangle$

inductive-cases[*elim!*]: $A \vdash \langle l, u \rangle \rightsquigarrow_{\mathcal{N}} \langle l', u' \rangle$

declare *step-z-norm.intros*[*intro*]

lemma *apx-empty-iff''*:

assumes *canonical* $M1\ n\ [M1]_{v, n} \subseteq V\ dbm\text{-int}\ M1\ n$

shows $[M1]_{v, n} = \{\} \longleftrightarrow [norm\ M1\ (k \circ v')\ n]_{v, n} = \{\}$

<proof>

inductive *valid-dbm* **where**

$[M]_{v, n} \subseteq V \Longrightarrow dbm\text{-int}\ M\ n \Longrightarrow valid\text{-dbm}\ M$

inductive-cases *valid-dbm-cases*[*elim*]: *valid-dbm* M

declare *valid-dbm.intros*[*intro*]

lemma *step-z-valid-dbm*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{v,n} \langle l', D' \rangle$

and *global-clock-numbering* A v n *valid-abstraction* A X k *valid-dbm* D

shows *valid-dbm* D'

<proof>

lemma *FW-zone-equiv-spec*:

shows $[M]_{v,n} = [FW\ M\ n]_{v,n}$

<proof>

lemma *cn-weak*: $\forall k \leq n. 0 < k \longrightarrow (\exists c. v\ c = k)$ *<proof>*

lemma *valid-dbm-non-empty-diag*:

assumes *valid-dbm* M $[M]_{v,n} \neq \{\}$

shows $\forall k \leq n. M\ k\ k \geq 1$

<proof>

lemma *non-empty-cycle-free*:

assumes $[M]_{v,n} \neq \{\}$

shows *cycle-free* $M\ n$

<proof>

lemma *norm-empty-diag-preservation*:

assumes $i \leq n$

assumes $M\ i\ i < Le\ 0$

shows *norm* M $(k\ o\ v')$ $n\ i\ i < Le\ 0$

<proof>

lemma *norm-FW-empty*:

assumes *valid-dbm* M

assumes $[M]_{v,n} = \{\}$

shows $[norm\ (FW\ M\ n)\ (k\ o\ v')\ n]_{v,n} = \{\}$ (**is** $[?M]_{v,n} = \{\}$)

<proof>

lemma *apx-norm-eq-spec*:

assumes *valid-dbm* M

and $[M]_{v,n} \neq \{\}$

shows *beta-interp.Approx* β $([M]_{v,n}) = [norm\ (FW\ M\ n)\ (k\ o\ v')\ n]_{v,n}$

<proof>

print-statement *step-z-norm.inducts*

lemma *step-z-norm-induct*[*case-names - step-z-norm step-z-refl*]:

assumes $x1 \vdash \langle x2, x3 \rangle \rightsquigarrow_{(k \ o \ v'), v, n} \langle x7, x8 \rangle$
and *step-z-norm*:
 $\bigwedge A \ l \ D \ l' \ D'$
 $A \vdash \langle l, D \rangle \rightsquigarrow_{v, n} \langle l', D' \rangle \implies$
 $P \ A \ l \ D \ l' \ (\text{norm } (FW \ D' \ n) \ (k \ o \ v') \ n)$
shows $P \ x1 \ x2 \ x3 \ x7 \ x8$
 $\langle \text{proof} \rangle$

lemma *FW-valid-preservation*:

assumes *valid-dbm* M
shows *valid-dbm* $(FW \ M \ n)$
 $\langle \text{proof} \rangle$

Obsolete

lemma *norm-diag-preservation*:

assumes $\forall l \leq n. \ M1 \ l \ l \leq \mathbf{1}$
shows $\forall l \leq n. \ (\text{norm } M1 \ (k \ o \ v') \ n) \ l \ l \leq \mathbf{1}$ (**is** $\forall l \leq n. \ ?M \ l \ l \leq \mathbf{1}$)
 $\langle \text{proof} \rangle$

lemma *norm-FW-valid-preservation-non-empty*:

assumes *valid-dbm* $M \ [M]_{v, n} \neq \{\}$
shows *valid-dbm* $(\text{norm } (FW \ M \ n) \ (k \ o \ v') \ n)$ (**is** *valid-dbm* $?M$)
 $\langle \text{proof} \rangle$

lemma *norm-FW-valid-preservation-empty*:

assumes *valid-dbm* $M \ [M]_{v, n} = \{\}$
shows *valid-dbm* $(\text{norm } (FW \ M \ n) \ (k \ o \ v') \ n)$ (**is** *valid-dbm* $?M$)
 $\langle \text{proof} \rangle$

lemma *norm-FW-valid-preservation*:

assumes *valid-dbm* M
shows *valid-dbm* $(\text{norm } (FW \ M \ n) \ (k \ o \ v') \ n)$
 $\langle \text{proof} \rangle$

lemma *norm-beta-sound*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}} \langle l', D' \rangle$ *global-clock-numbering* $A \ v \ n$ *valid-abstraction*
 $A \ X \ k$
and *valid-dbm* D
shows $A \vdash \langle l, [D]_{v, n} \rangle \rightsquigarrow_{\beta} \langle l', [D']_{v, n} \rangle$ $\langle \text{proof} \rangle$

lemma *step-z-norm-valid-dbm*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}} \langle l', D' \rangle$ *global-clock-numbering* $A \ v \ n$ *valid-abstraction*
 $A \ X \ k$ *valid-dbm* D
shows *valid-dbm* D' $\langle \text{proof} \rangle$

lemma *norm-beta-complete*:

assumes $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta} \langle l', Z \rangle$ *global-clock-numbering* A v n *valid-abstraction*
 A X k
and *valid-dbm* D
obtains D' **where** $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}} \langle l', D' \rangle$ $[D']_{v,n} = Z$ *valid-dbm* D'
 \langle *proof* \rangle

9.1.2 Multi step

declare *steps-z-norm.intros*[*intro*]

lemma *steps-z-norm-induct*[*case-names - refl step*]:

assumes $x1 \vdash \langle x2, x3 \rangle \rightsquigarrow_{(k \ o \ v'), v, n^*} \langle x7, x8 \rangle$
and $\bigwedge A \ l \ Z. P \ A \ l \ Z \ l \ Z$
and
 $\bigwedge A \ l \ Z \ l' \ Z' \ l'' \ Z''.$
 $A \vdash \langle l, Z \rangle \rightsquigarrow_{(k \ o \ v'), v, n^*} \langle l', Z' \rangle \implies$
 $P \ A \ l \ Z \ l' \ Z' \implies$
 $A \vdash \langle l', Z' \rangle \rightsquigarrow_{(k \ o \ v'), v, n} \langle l'', Z'' \rangle \implies P \ A \ l \ Z \ l'' \ Z''$
shows $P \ x1 \ x2 \ x3 \ x7 \ x8$
 \langle *proof* \rangle

lemma *norm-beta-sound-multi*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle$ *global-clock-numbering* A v n *valid-abstraction*
 A X k *valid-dbm* D
shows $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta^*} \langle l', [D']_{v,n} \rangle \wedge$ *valid-dbm* D' \langle *proof* \rangle

lemma *norm-beta-complete-multi*:

assumes $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow_{\beta^*} \langle l', Z \rangle$ *global-clock-numbering* A v n *valid-abstraction*
 A X k
and *valid-dbm* D
obtains D' **where** $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle$ $[D']_{v,n} = Z$ *valid-dbm* D'
 \langle *proof* \rangle

lemma *norm-beta-equiv-multi*:

assumes *global-clock-numbering* A v n *valid-abstraction* A X k
and *valid-dbm* D
shows $(\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge Z = [D']_{v,n}) \iff A \vdash \langle l, [D]_{v,n} \rangle$
 $\rightsquigarrow_{\beta^*} \langle l', Z \rangle$
 \langle *proof* \rangle

9.1.3 Connecting with Correctness Results for Approximating Semantics

lemma *steps-z-norm-complete'*:

assumes $A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow^* \langle l', Z \rangle$ *global-clock-numbering A v n valid-abstraction A X k*
and *valid-dbm D*
shows $\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge Z \subseteq [D']_{v,n}$
<proof>

lemma *valid-dbm-V'*:

assumes *valid-dbm M*
shows $[M]_{v,n} \in V'$
<proof>

lemma *steps-z-norm-sound'*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle$
and *global-clock-numbering A v n*
and *valid-abstraction A X k*
and *valid-dbm D*
and $[D]_{v,n} \neq \{\}$
shows $\exists Z. A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow^* \langle l', Z \rangle \wedge Z \neq \{\}$
<proof>

9.2 The Final Result About Language Emptiness

lemma *steps-z-norm-complete*:

assumes $A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$ $u \in [D]_{v,n}$
and *global-clock-numbering A v n valid-abstraction A X k valid-dbm D*
shows $\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge u' \in [D']_{v,n}$
<proof>

lemma *steps-z-norm-sound*:

assumes $A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle$
and *global-clock-numbering A v n valid-abstraction A X k valid-dbm D*
and $[D']_{v,n} \neq \{\}$
shows $\exists u \in [D]_{v,n}. \exists u'. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle$
<proof>

theorem *steps-z-norm-decides-emptiness*:

assumes *global-clock-numbering A v n valid-abstraction A X k valid-dbm D*
shows $(\exists D'. A \vdash \langle l, D \rangle \rightsquigarrow_{\mathcal{N}^*} \langle l', D' \rangle \wedge [D']_{v,n} \neq \{\})$
 $\iff (\exists u \in [D]_{v,n}. \exists u'. A \vdash \langle l, u \rangle \rightarrow^* \langle l', u' \rangle)$

<proof>

9.3 Finiteness of the Search Space

abbreviation *dbm-default* $M \equiv (\forall i > n. \forall j. M\ i\ j = \mathbf{1}) \wedge (\forall j > n. \forall i. M\ i\ j = \mathbf{1})$

lemma $a \in \mathbb{Z} \implies \exists b. a = \text{real-of-int } b$ *<proof>*

lemma *norm-default-preservation:*

$\text{dbm-default } M \implies \text{dbm-default } (\text{norm } M\ (k\ o\ v')\ n)$
<proof>

lemma *normalized-integral-dbms-finite:*

$\text{finite } \{\text{norm } M\ (k\ o\ v')\ n \mid M. \text{dbm-int } M\ n \wedge \text{dbm-default } M\}$
<proof>

end

9.4 Appendix: Standard Clock Numberings for Concrete Models

locale *Regions'* =

fixes X **and** $k :: 'c \Rightarrow \text{nat}$ **and** $v :: 'c \Rightarrow \text{nat}$ **and** $n :: \text{nat}$ **and** *not-in-X*
assumes *finite:* $\text{finite } X$
assumes *clock-numbering':* $\forall c \in X. v\ c > 0 \wedge \forall c. c \notin X \longrightarrow v\ c > n$
assumes *bij:* $\text{bij-betw } v\ X\ \{1..n\}$
assumes *non-empty:* $X \neq \{\}$
assumes *not-in-X:* $\text{not-in-}X \notin X$

begin

lemma *inj:* $\text{inj-on } v\ X$ *<proof>*

lemma *cn-weak:* $\forall c. v\ c > 0$ *<proof>*

lemma *in-X:* **assumes** $v\ x \leq n$ **shows** $x \in X$ *<proof>*

end

sublocale *Regions' ⊆ Regions*

<proof>

lemma *standard-abstraction*:

assumes *finite* (*clkp-set* *A*) *finite* (*collect-clkvt* (*trans-of* *A*)) $\forall (-, m :: \text{real})$
 $\in \text{clkp-set } A. m \in \mathbb{N}$
obtains $k :: 'c \Rightarrow \text{nat}$ **where** *valid-abstraction* *A* (*clk-set* *A*) *k*
 $\langle \text{proof} \rangle$

definition

finite-ta *A* \equiv *finite* (*clkp-set* *A*) \wedge *finite* (*collect-clkvt* (*trans-of* *A*))
 $\wedge (\forall (-, m :: \text{real}) \in \text{clkp-set } A. m \in \mathbb{N}) \wedge \text{clk-set } A \neq \{\} \wedge$
 $-\text{clk-set } A \neq \{\}$

lemma *finite-ta-Regions'*:

fixes *A* :: (*'a*, *'c*, *real*, *'s*) *ta*
assumes *finite-ta* *A*
obtains $v \ n \ x$ **where** *Regions'* (*clk-set* *A*) $v \ n \ x$
 $\langle \text{proof} \rangle$

lemma *finite-ta-RegionsD*:

assumes *finite-ta* *A*
obtains $k :: 'b \Rightarrow \text{nat}$ **and** $v \ n \ x$ **where**
Regions' (*clk-set* *A*) $v \ n \ x$ *valid-abstraction* *A* (*clk-set* *A*) *k* *global-clock-numbering*
 $A \ v \ n$
 $\langle \text{proof} \rangle$

definition *valid-dbm* **where** *valid-dbm* *M* *n* \equiv *dbm-int* *M* *n* $\wedge (\forall i \leq n. M \ 0 \ i \leq 1)$

lemma *dbm-positive*:

assumes $M \ 0 \ (v \ c) \leq 1$ $v \ c \leq n$ *DBM-val-bounded* $v \ u \ M \ n$
shows $u \ c \geq 0$
 $\langle \text{proof} \rangle$

lemma *valid-dbm-pos*:

assumes *valid-dbm* *M* *n*
shows $[M]_{v, n} \subseteq \{u. \forall c. v \ c \leq n \longrightarrow u \ c \geq 0\}$
 $\langle \text{proof} \rangle$

lemma (**in** *Regions'*) *V-alt-def*:

shows $\{u. \forall c. v \ c > 0 \wedge v \ c \leq n \longrightarrow u \ c \geq 0\} = V$
 $\langle \text{proof} \rangle$

An example of obtaining concrete models from our formalizations.

lemma *steps-z-norm-sound-spec*:

assumes *finite-ta* A
obtains $k\ v\ n$ **where**
 $A \vdash \langle l, D \rangle \rightsquigarrow_{k,v,n}^* \langle l', D' \rangle \wedge \text{valid-dbm } D\ n \wedge [D]_{v,n} \neq \{\}$
 $\longrightarrow (\exists Z. A \vdash \langle l, [D]_{v,n} \rangle \rightsquigarrow^* \langle l', Z \rangle \wedge Z \neq \{\})$
 $\langle \text{proof} \rangle$
end

References

- [AD90] Rajeev Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
- [BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, pages 87–124, 2003.
- [HHWt97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.
- [LPY97] G. Kim Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.
- [Yov97] Sergio Yovine. KRONOS: A verification tool for real-time systems. *STTT*, 1(1-2):123–133, 1997.