

# Formalization of a Monitoring Algorithm for Metric First-Order Temporal Logic

Joshua Schneider      Dmitriy Traytel

February 6, 2026

## Abstract

A monitor is a runtime verification tool that solves the following problem: Given a stream of time-stamped events and a policy formulated in a specification language, decide whether the policy is satisfied at every point in the stream. We verify the correctness of an executable monitor for specifications given as formulas in metric first-order temporal logic (MFOTL) [1], an expressive extension of linear temporal logic with real-time constraints and first-order quantification. The verified monitor implements a simplified variant of the algorithm used in the efficient MonPoly monitoring tool [2]. The formalization is presented in a RV 2019 paper [4], which also compares the output of the verified monitor to that of other monitoring tools on randomly generated inputs. This case study revealed several errors in the optimized but unverified tools.

## Contents

<b>1</b>	<b>Traces and trace prefixes</b>	<b>2</b>
1.1	Infinite traces . . . . .	2
1.2	Finite trace prefixes . . . . .	4
<b>2</b>	<b>Finite tables</b>	<b>6</b>
<b>3</b>	<b>Abstract monitors and slicing</b>	<b>11</b>
3.1	First-order specifications . . . . .	11
3.2	Monitor function . . . . .	12
3.3	Slicing . . . . .	13
<b>4</b>	<b>Intervals</b>	<b>14</b>
<b>5</b>	<b>Metric first-order temporal logic</b>	<b>15</b>
5.1	Formulas and satisfiability . . . . .	15
5.2	Defined connectives . . . . .	17
5.3	Safe formulas . . . . .	18
5.4	Slicing traces . . . . .	19
<b>6</b>	<b>Monitor implementation</b>	<b>19</b>
6.1	Monitorable formulas . . . . .	19
6.2	The executable monitor . . . . .	20
6.3	Progress . . . . .	23
6.4	Specification . . . . .	24
6.5	Correctness . . . . .	24
6.5.1	Invariants . . . . .	24
6.5.2	Initialisation . . . . .	26
6.5.3	Evaluation . . . . .	27

6.5.4	Monitor step	31
6.5.5	Monitor function	31
6.6	Collected correctness results	32
<b>7</b>	<b>Slicing framework</b>	<b>33</b>
7.1	Abstract slicing	33
7.1.1	Definition 1	33
7.1.2	Definition 2	33
7.1.3	Definition 3	33
7.1.4	Lemma 1	34
7.2	Joint data slicer	34
7.2.1	Definition 4	34
7.2.2	Lemma 2	34
7.2.3	Theorem 1	34
7.2.4	Corollary 1	34
7.2.5	Definition 5	35
7.2.6	Theorem 2	35
7.2.7	Towards Theorem 3	35
7.2.8	Lemma 3	37
7.2.9	Definition of $J'$	37
7.2.10	Theorem 3	37

## 1 Traces and trace prefixes

### 1.1 Infinite traces

**coinductive**  $sorted :: 'a :: linorder \text{ stream} \Rightarrow \text{bool}$  **where**  
 $shd\ s \leq shd\ (stl\ s) \Longrightarrow sorted\ (stl\ s) \Longrightarrow sorted\ s$

**lemma**  $sorted\_siterate[simp]: (\bigwedge n. n \leq f\ n) \Longrightarrow sorted\ (siterate\ f\ n)$   
 $\langle proof \rangle$

**lemma**  $sortedD: sorted\ s \Longrightarrow s\ !!\ i \leq stl\ s\ !!\ i$   
 $\langle proof \rangle$

**lemma**  $sorted\_sdrop: sorted\ s \Longrightarrow sorted\ (sdrop\ i\ s)$   
 $\langle proof \rangle$

**lemma**  $sorted\_monoD: sorted\ s \Longrightarrow i \leq j \Longrightarrow s\ !!\ i \leq s\ !!\ j$   
 $\langle proof \rangle$

**lemma**  $sorted\_stake: sorted\ s \Longrightarrow sorted\ (stake\ i\ s)$   
 $\langle proof \rangle$

**lemma**  $sorted\_monoI: \forall i\ j. i \leq j \longrightarrow s\ !!\ i \leq s\ !!\ j \Longrightarrow sorted\ s$   
 $\langle proof \rangle$

**lemma**  $sorted\_iff\_mono: sorted\ s \longleftrightarrow (\forall i\ j. i \leq j \longrightarrow s\ !!\ i \leq s\ !!\ j)$   
 $\langle proof \rangle$

**lemma**  $sorted\_iff\_le\_Suc: sorted\ s \longleftrightarrow (\forall i. s\ !!\ i \leq s\ !!\ Suc\ i)$   
 $\langle proof \rangle$

**definition**  $sincreasing\ s = (\forall x. \exists i. x < s\ !!\ i)$

**lemma**  $sincreasingI: (\bigwedge x. \exists i. x < s\ !!\ i) \Longrightarrow sincreasing\ s$

*<proof>*

**lemma** *sincreasing\_grD*:

**fixes**  $x :: 'a :: \text{semilattice\_sup}$

**assumes** *sincreasing s*

**shows**  $\exists j > i. x < s !! j$

*<proof>*

**lemma** *sincreasing\_siterate\_nat[simp]*:

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

**assumes**  $(\bigwedge n. n < f n)$

**shows** *sincreasing (siterate f n)*

*<proof>*

**lemma** *sincreasing\_stl*: *sincreasing s*  $\implies$  *sincreasing (stl s)* **for**  $s :: 'a :: \text{semilattice\_sup}$  *stream*

*<proof>*

**typedef**  $'a$  *trace* =  $\{s :: ('a \text{ set} \times \text{nat}) \text{ stream}. \text{ssorted} (\text{smap snd } s) \wedge \text{sincreasing} (\text{smap snd } s)\}$

*<proof>*

**setup\_lifting** *type\_definition\_trace*

**lift\_definition**  $\Gamma :: 'a \text{ trace} \Rightarrow \text{nat} \Rightarrow 'a \text{ set}$  **is**

$\lambda s i. \text{fst } (s !! i)$  *<proof>*

**lift\_definition**  $\tau :: 'a \text{ trace} \Rightarrow \text{nat} \Rightarrow \text{nat}$  **is**

$\lambda s i. \text{snd } (s !! i)$  *<proof>*

**lemma** *stream\_eq\_iff*:  $s = s' \iff (\forall n. s !! n = s' !! n)$

*<proof>*

**lemma** *trace\_eqI*:  $(\bigwedge i. \Gamma \sigma i = \Gamma \sigma' i) \implies (\bigwedge i. \tau \sigma i = \tau \sigma' i) \implies \sigma = \sigma'$

*<proof>*

**lemma**  *$\tau$ \_mono[simp]*:  $i \leq j \implies \tau s i \leq \tau s j$

*<proof>*

**lemma** *ex\_le\_ $\tau$* :  $\exists j \geq i. x \leq \tau s j$

*<proof>*

**lemma** *le\_ $\tau$ \_less*:  $\tau \sigma i \leq \tau \sigma j \implies j < i \implies \tau \sigma i = \tau \sigma j$

*<proof>*

**lemma** *less\_ $\tau$ D*:  $\tau \sigma i < \tau \sigma j \implies i < j$

*<proof>*

**abbreviation**  $\Delta s i \equiv \tau s i - \tau s (i - 1)$

**lift\_definition** *map\_ $\Gamma$*  ::  $('a \text{ set} \Rightarrow 'b \text{ set}) \Rightarrow 'a \text{ trace} \Rightarrow 'b \text{ trace}$  **is**

$\lambda f s. \text{smap } (\lambda(x, i). (f x, i)) s$

*<proof>*

**lemma**  *$\Gamma$ \_map\_ $\Gamma$ [simp]*:  $\Gamma (\text{map}_\Gamma f s) i = f (\Gamma s i)$

*<proof>*

**lemma**  *$\tau$ \_map\_ $\Gamma$ [simp]*:  $\tau (\text{map}_\Gamma f s) i = \tau s i$

*<proof>*

**lemma** *map\_ $\Gamma$ \_id[simp]*: *map\_ $\Gamma$  id s = s*

*<proof>*

**lemma** *map\_Γ\_comp*:  $\text{map}_\Gamma g (\text{map}_\Gamma f s) = \text{map}_\Gamma (g \circ f) s$   
*<proof>*

**lemma** *map\_Γ\_cong*:  $\sigma_1 = \sigma_2 \implies (\bigwedge x. f_1 x = f_2 x) \implies \text{map}_\Gamma f_1 \sigma_1 = \text{map}_\Gamma f_2 \sigma_2$   
*<proof>*

## 1.2 Finite trace prefixes

**typedef** *'a prefix* =  $\{p :: ('a \text{ set} \times \text{nat}) \text{ list. sorted (map snd } p)\}$   
*<proof>*

**setup\_lifting** *type\_definition\_prefix*

**lift\_definition** *pmap\_Γ* ::  $('a \text{ set} \Rightarrow 'b \text{ set}) \Rightarrow 'a \text{ prefix} \Rightarrow 'b \text{ prefix}$  **is**  
 $\lambda f. \text{map } (\lambda(x, i). (f x, i))$   
*<proof>*

**lift\_definition** *last\_ts* ::  $'a \text{ prefix} \Rightarrow \text{nat}$  **is**  
 $\lambda p. (\text{case } p \text{ of } [] \Rightarrow 0 \mid \_ \Rightarrow \text{snd (last } p))$  *<proof>*

**lift\_definition** *first\_ts* ::  $\text{nat} \Rightarrow 'a \text{ prefix} \Rightarrow \text{nat}$  **is**  
 $\lambda n p. (\text{case } p \text{ of } [] \Rightarrow n \mid \_ \Rightarrow \text{snd (hd } p))$  *<proof>*

**lift\_definition** *pnil* ::  $'a \text{ prefix}$  **is**  $[]$  *<proof>*

**lift\_definition** *plen* ::  $'a \text{ prefix} \Rightarrow \text{nat}$  **is** *length* *<proof>*

**lift\_definition** *psnoc* ::  $'a \text{ prefix} \Rightarrow 'a \text{ set} \times \text{nat} \Rightarrow 'a \text{ prefix}$  **is**  
 $\lambda p x. \text{if } (\text{case } p \text{ of } [] \Rightarrow 0 \mid \_ \Rightarrow \text{snd (last } p)) \leq \text{snd } x \text{ then } p @ [x] \text{ else } []$   
*<proof>*

**instantiation** *prefix* ::  $(\text{type})$  **order begin**

**lift\_definition** *less\_eq\_prefix* ::  $'a \text{ prefix} \Rightarrow 'a \text{ prefix} \Rightarrow \text{bool}$  **is**  
 $\lambda p q. \exists r. q = p @ r$  *<proof>*

**definition** *less\_prefix* ::  $'a \text{ prefix} \Rightarrow 'a \text{ prefix} \Rightarrow \text{bool}$  **where**  
 $\text{less\_prefix } x y = (x \leq y \wedge \neg y \leq x)$

**instance**  
*<proof>*

**end**

**lemma** *psnoc\_inject[simp]*:  
 $\text{last\_ts } p \leq \text{snd } x \implies \text{last\_ts } q \leq \text{snd } y \implies \text{psnoc } p x = \text{psnoc } q y \iff (p = q \wedge x = y)$   
*<proof>*

**lift\_definition** *prefix\_of* ::  $'a \text{ prefix} \Rightarrow 'a \text{ trace} \Rightarrow \text{bool}$  **is**  $\lambda p s. \text{stake (length } p) s = p$  *<proof>*

**lemma** *prefix\_of\_pnil[simp]*: *prefix\_of* *pnil*  $\sigma$   
*<proof>*

**lemma** *plen\_pnil[simp]*: *plen* *pnil* = 0  
*<proof>*

**lemma** *prefix\_of\_pmap\_Γ[simp]*:  $\text{prefix\_of } \pi \sigma \implies \text{prefix\_of } (\text{pmap\_}\Gamma f \pi) (\text{map\_}\Gamma f \sigma)$   
 ⟨proof⟩

**lemma** *plen\_mono*:  $\pi \leq \pi' \implies \text{plen } \pi \leq \text{plen } \pi'$   
 ⟨proof⟩

**lemma** *prefix\_of\_psnocE*:  $\text{prefix\_of } (\text{psnoc } p x) s \implies \text{last\_ts } p \leq \text{snd } x \implies$   
 $(\text{prefix\_of } p s \implies \Gamma s (\text{plen } p) = \text{fst } x \implies \tau s (\text{plen } p) = \text{snd } x \implies P) \implies P$   
 ⟨proof⟩

**lemma** *le\_pnil[simp]*:  $\text{pnil} \leq \pi$   
 ⟨proof⟩

**lift\_definition** *take\_prefix* ::  $\text{nat} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ prefix is stake}$   
 ⟨proof⟩

**lemma** *plen\_take\_prefix[simp]*:  $\text{plen } (\text{take\_prefix } i \sigma) = i$   
 ⟨proof⟩

**lemma** *plen\_psnoc[simp]*:  $\text{last\_ts } \pi \leq \text{snd } x \implies \text{plen } (\text{psnoc } \pi x) = \text{plen } \pi + 1$   
 ⟨proof⟩

**lemma** *prefix\_of\_take\_prefix[simp]*:  $\text{prefix\_of } (\text{take\_prefix } i \sigma) \sigma$   
 ⟨proof⟩

**lift\_definition** *pdrop* ::  $\text{nat} \Rightarrow 'a \text{ prefix} \Rightarrow 'a \text{ prefix is drop}$   
 ⟨proof⟩

**lemma** *pdrop\_0[simp]*:  $\text{pdrop } 0 \pi = \pi$   
 ⟨proof⟩

**lemma** *prefix\_of\_antimono*:  $\pi \leq \pi' \implies \text{prefix\_of } \pi' s \implies \text{prefix\_of } \pi s$   
 ⟨proof⟩

**lemma** *prefix\_of\_imp\_linear*:  $\text{prefix\_of } \pi \sigma \implies \text{prefix\_of } \pi' \sigma \implies \pi \leq \pi' \vee \pi' \leq \pi$   
 ⟨proof⟩

**lemma** *ex\_prefix\_of*:  $\exists s. \text{prefix\_of } p s$   
 ⟨proof⟩

**lemma** *τ\_prefix\_conv*:  $\text{prefix\_of } p s \implies \text{prefix\_of } p s' \implies i < \text{plen } p \implies \tau s i = \tau s' i$   
 ⟨proof⟩

**lemma** *Γ\_prefix\_conv*:  $\text{prefix\_of } p s \implies \text{prefix\_of } p s' \implies i < \text{plen } p \implies \Gamma s i = \Gamma s' i$   
 ⟨proof⟩

**lemma** *sincreasing\_sdrop*:  
**fixes**  $s :: ('a :: \text{semilattice\_sup}) \text{ stream}$   
**assumes** *sincreasing s*  
**shows** *sincreasing (sdrop n s)*  
 ⟨proof⟩

**lemma** *ssorted\_shift*:  
 $\text{ssorted } (xs @- s) = (\text{sorted } xs \wedge \text{ssorted } s \wedge (\forall x \in \text{set } xs. \forall y \in \text{sset } s. x \leq y))$   
 ⟨proof⟩

**lemma** *sincreasing\_shift*:  
**assumes** *sincreasing s*

**shows** *sincreasing* ( $xs @- s$ )  
 ⟨proof⟩

**lift\_definition** *replace\_prefix* :: 'a prefix  $\Rightarrow$  'a trace  $\Rightarrow$  'a trace **is**  
 $\lambda \pi \sigma$ . if *ssorted* ( $smap\ snd (\pi @- sdrop (length\ \pi)\ \sigma)$ ) then  
 $\pi @- sdrop (length\ \pi)\ \sigma$  else  $smap (\lambda i. (\{\}, i))\ nats$   
 ⟨proof⟩

**lemma** *prefix\_of\_replace\_prefix*:  
 $prefix\_of (pmap\_ \Gamma f \pi)\ \sigma \Longrightarrow prefix\_of\ \pi (replace\_prefix\ \pi\ \sigma)$   
 ⟨proof⟩

**lemma** *map\_Γ\_replace\_prefix*:  
 $\forall x. f (f x) = f x \Longrightarrow prefix\_of (pmap\_ \Gamma f \pi)\ \sigma \Longrightarrow map\_ \Gamma f (replace\_prefix\ \pi\ \sigma) = map\_ \Gamma f \sigma$   
 ⟨proof⟩

**lemma** *prefix\_of\_pmap\_Γ\_D*:  
**assumes**  $prefix\_of (pmap\_ \Gamma f \pi)\ \sigma$   
**shows**  $\exists \sigma'. prefix\_of\ \pi\ \sigma' \wedge prefix\_of (pmap\_ \Gamma f \pi) (map\_ \Gamma f \sigma')$   
 ⟨proof⟩

**lemma** *prefix\_of\_map\_Γ\_D*:  
**assumes**  $prefix\_of\ \pi' (map\_ \Gamma f \sigma)$   
**shows**  $\exists \pi''. \pi' = pmap\_ \Gamma f \pi'' \wedge prefix\_of\ \pi'' \sigma$   
 ⟨proof⟩

**lift\_definition** *pts* :: 'a prefix  $\Rightarrow$  nat list **is**  $map\ snd$  ⟨proof⟩

**lemma** *pts\_pmap\_Γ[simp]*:  $pts (pmap\_ \Gamma f \pi) = pts\ \pi$   
 ⟨proof⟩

## 2 Finite tables

**primrec** *tabulate* :: (nat  $\Rightarrow$  'a)  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  'a list **where**  
 $tabulate\ f\ x\ 0 = []$   
 $| tabulate\ f\ x (Suc\ n) = f\ x \# tabulate\ f (Suc\ x)\ n$

**lemma** *tabulate\_alt*:  $tabulate\ f\ x\ n = map\ f [x ..< x + n]$   
 ⟨proof⟩

**lemma** *length\_tabulate[simp]*:  $length (tabulate\ f\ x\ n) = n$   
 ⟨proof⟩

**lemma** *map\_tabulate[simp]*:  $map\ f (tabulate\ g\ x\ n) = tabulate (\lambda x. f (g\ x))\ x\ n$   
 ⟨proof⟩

**lemma** *nth\_tabulate[simp]*:  $k < n \Longrightarrow tabulate\ f\ x\ n ! k = f (x + k)$   
 ⟨proof⟩

**type\_synonym** 'a tuple = 'a option list  
**type\_synonym** 'a table = 'a tuple set

**definition** *wf\_tuple* :: nat  $\Rightarrow$  nat set  $\Rightarrow$  'a tuple  $\Rightarrow$  bool **where**  
 $wf\_tuple\ n\ V\ x \longleftrightarrow length\ x = n \wedge (\forall i < n. x ! i = None \longleftrightarrow i \notin V)$

**definition** *table* :: nat  $\Rightarrow$  nat set  $\Rightarrow$  'a table  $\Rightarrow$  bool **where**  
 $table\ n\ V\ X \longleftrightarrow (\forall x \in X. wf\_tuple\ n\ V\ x)$

**definition** *empty\_table* = {}

**definition** *unit\_table* *n* = {replicate *n* None}

**definition** *singleton\_table* *n* *i* *x* = {tabulate ( $\lambda j$ . if  $i = j$  then Some *x* else None) 0 *n*}

**lemma** *in\_empty\_table[simp]*:  $\neg x \in \text{empty\_table}$   
<proof>

**lemma** *empty\_table[simp]*: *table* *n* *V* *empty\_table*  
<proof>

**lemma** *unit\_table\_wf\_tuple[simp]*:  $V = \{\}$   $\implies x \in \text{unit\_table } n \implies \text{wf\_tuple } n \ V \ x$   
<proof>

**lemma** *unit\_table[simp]*:  $V = \{\}$   $\implies \text{table } n \ V \ (\text{unit\_table } n)$   
<proof>

**lemma** *in\_unit\_table*:  $v \in \text{unit\_table } n \iff \text{wf\_tuple } n \ \{\} \ v$   
<proof>

**lemma** *singleton\_table\_wf\_tuple[simp]*:  $V = \{i\} \implies x \in \text{singleton\_table } n \ i \ z \implies \text{wf\_tuple } n \ V \ x$   
<proof>

**lemma** *singleton\_table[simp]*:  $V = \{i\} \implies \text{table } n \ V \ (\text{singleton\_table } n \ i \ z)$   
<proof>

**lemma** *table\_Un[simp]*: *table* *n* *V* *X*  $\implies \text{table } n \ V \ Y \implies \text{table } n \ V \ (X \cup Y)$   
<proof>

**lemma** *wf\_tuple\_length*:  $\text{wf\_tuple } n \ V \ x \implies \text{length } x = n$   
<proof>

**fun** *join1* :: 'a tuple  $\times$  'a tuple  $\Rightarrow$  'a tuple option **where**

*join1* ([], []) = Some []  
| *join1* (None # *xs*, None # *ys*) = map\_option (Cons None) (*join1* (*xs*, *ys*))  
| *join1* (Some *x* # *xs*, None # *ys*) = map\_option (Cons (Some *x*)) (*join1* (*xs*, *ys*))  
| *join1* (None # *xs*, Some *y* # *ys*) = map\_option (Cons (Some *y*)) (*join1* (*xs*, *ys*))  
| *join1* (Some *x* # *xs*, Some *y* # *ys*) = (if  $x = y$   
then map\_option (Cons (Some *x*)) (*join1* (*xs*, *ys*))  
else None)  
| *join1* \_ = None

**definition** *join* :: 'a table  $\Rightarrow$  bool  $\Rightarrow$  'a table  $\Rightarrow$  'a table **where**

*join* *A* *pos* *B* = (if *pos* then Option.these (*join1* ' (*A*  $\times$  *B*))  
else *A* - Option.these (*join1* ' (*A*  $\times$  *B*)))

**lemma** *join\_True\_code[code]*: *join* *A* True *B* = ( $\bigcup a \in A. \bigcup b \in B. \text{set\_option } (\text{join1 } (a, b))$ )  
<proof>

**lemma** *join\_False\_alt*: *join* *X* False *Y* = *X* - *join* *X* True *Y*  
<proof>

**lemma** *self\_join1*: *join1* (*xs*, *ys*)  $\neq$  Some *xs*  $\implies \text{join1 } (zs, ys) \neq \text{Some } xs$   
<proof>

**lemma** *join\_False\_code[code]*: *join* *A* False *B* = { $a \in A. \forall b \in B. \text{join1 } (a, b) \neq \text{Some } a$ }

*<proof>*

**lemma** *wf\_tuple\_Nil[simp]*:  $wf\_tuple\ n\ A\ [] = (n = 0)$   
*<proof>*

**lemma** *Suc\_pred'*:  $Suc\ (x - Suc\ 0) = (case\ x\ of\ 0 \Rightarrow Suc\ 0 \mid \_ \Rightarrow x)$   
*<proof>*

**lemma** *wf\_tuple\_Cons[simp]*:  
 $wf\_tuple\ n\ A\ (x \# xs) \longleftrightarrow ((if\ x = None\ then\ 0 \notin A\ else\ 0 \in A) \wedge$   
 $(\exists m. n = Suc\ m \wedge wf\_tuple\ m\ ((\lambda x. x - 1) ' (A - \{0\}))\ xs))$   
*<proof>*

**lemma** *join1\_wf\_tuple*:  
 $join1\ (v1, v2) = Some\ v \Longrightarrow wf\_tuple\ n\ A\ v1 \Longrightarrow wf\_tuple\ n\ B\ v2 \Longrightarrow wf\_tuple\ n\ (A \cup B)\ v$   
*<proof>*

**lemma** *join\_wf\_tuple*:  $x \in join\ X\ b\ Y \Longrightarrow$   
 $\forall v \in X. wf\_tuple\ n\ A\ v \Longrightarrow \forall v \in Y. wf\_tuple\ n\ B\ v \Longrightarrow (\neg b \Longrightarrow B \subseteq A) \Longrightarrow A \cup B = C \Longrightarrow$   
 $wf\_tuple\ n\ C\ x$   
*<proof>*

**lemma** *join\_table*:  $table\ n\ A\ X \Longrightarrow table\ n\ B\ Y \Longrightarrow (\neg b \Longrightarrow B \subseteq A) \Longrightarrow A \cup B = C \Longrightarrow$   
 $table\ n\ C\ (join\ X\ b\ Y)$   
*<proof>*

**lemma** *wf\_tuple\_Suc*:  $wf\_tuple\ (Suc\ m)\ A\ a \longleftrightarrow a \neq [] \wedge$   
 $wf\_tuple\ m\ ((\lambda x. x - 1) ' (A - \{0\}))\ (tl\ a) \wedge (0 \in A \longleftrightarrow hd\ a \neq None)$   
*<proof>*

**lemma** *table\_project*:  $table\ (Suc\ n)\ A\ X \Longrightarrow table\ n\ ((\lambda x. x - Suc\ 0) ' (A - \{0\}))\ (tl\ ' X)$   
*<proof>*

**definition** *restrict where*  
 $restrict\ A\ v = map\ (\lambda i. if\ i \in A\ then\ v\ !\ i\ else\ None)\ [0 ..< length\ v]$

**lemma** *restrict\_Nil[simp]*:  $restrict\ A\ [] = []$   
*<proof>*

**lemma** *restrict\_Cons[simp]*:  $restrict\ A\ (x \# xs) =$   
 $(if\ 0 \in A\ then\ x \# restrict\ ((\lambda x. x - 1) ' (A - \{0\}))\ xs\ else\ None \# restrict\ ((\lambda x. x - 1) ' A)\ xs)$   
*<proof>*

**lemma** *wf\_tuple\_restrict*:  $wf\_tuple\ n\ B\ v \Longrightarrow A \cap B = C \Longrightarrow wf\_tuple\ n\ C\ (restrict\ A\ v)$   
*<proof>*

**lemma** *wf\_tuple\_restrict\_simple*:  $wf\_tuple\ n\ B\ v \Longrightarrow A \subseteq B \Longrightarrow wf\_tuple\ n\ A\ (restrict\ A\ v)$   
*<proof>*

**lemma** *nth\_restrict*:  $i \in A \Longrightarrow i < length\ v \Longrightarrow restrict\ A\ v\ !\ i = v\ !\ i$   
*<proof>*

**lemma** *restrict\_eq\_Nil[simp]*:  $restrict\ A\ v = [] \longleftrightarrow v = []$   
*<proof>*

**lemma** *length\_restrict[simp]*:  $length\ (restrict\ A\ v) = length\ v$   
*<proof>*

**lemma** *join1\_Some\_restrict*:

**fixes**  $x\ y :: 'a\ tuple$

**assumes**  $wf\_tuple\ n\ A\ x\ wf\_tuple\ n\ B\ y$

**shows**  $join1\ (x,\ y) = Some\ z \longleftrightarrow wf\_tuple\ n\ (A \cup B)\ z \wedge restrict\ A\ z = x \wedge restrict\ B\ z = y$

*<proof>*

**lemma** *restrict\_idle*:  $wf\_tuple\ n\ A\ v \Longrightarrow restrict\ A\ v = v$

*<proof>*

**lemma** *map\_the\_restrict*:

$i \in A \Longrightarrow map\ the\ (restrict\ A\ v)\ !\ i = map\ the\ v\ !\ i$

*<proof>*

**lemma** *join\_restrict*:

**fixes**  $X\ Y :: 'a\ tuple\ set$

**assumes**  $\bigwedge v. v \in X \Longrightarrow wf\_tuple\ n\ A\ v \wedge \bigwedge v. v \in Y \Longrightarrow wf\_tuple\ n\ B\ v \wedge \neg b \Longrightarrow B \subseteq A$

**shows**  $v \in join\ X\ b\ Y \longleftrightarrow$

$wf\_tuple\ n\ (A \cup B)\ v \wedge restrict\ A\ v \in X \wedge (if\ b\ then\ restrict\ B\ v \in Y\ else\ restrict\ B\ v \notin Y)$

*<proof>*

**lemma** *join\_restrict\_table*:

**assumes**  $table\ n\ A\ X\ table\ n\ B\ Y \wedge \neg b \Longrightarrow B \subseteq A$

**shows**  $v \in join\ X\ b\ Y \longleftrightarrow$

$wf\_tuple\ n\ (A \cup B)\ v \wedge restrict\ A\ v \in X \wedge (if\ b\ then\ restrict\ B\ v \in Y\ else\ restrict\ B\ v \notin Y)$

*<proof>*

**lemma** *join\_restrict\_annotated*:

**fixes**  $X\ Y :: 'a\ tuple\ set$

**assumes**  $\neg b = simp \Longrightarrow B \subseteq A$

**shows**  $join\ \{v. wf\_tuple\ n\ A\ v \wedge P\ v\}\ b\ \{v. wf\_tuple\ n\ B\ v \wedge Q\ v\} =$

$\{v. wf\_tuple\ n\ (A \cup B)\ v \wedge P\ (restrict\ A\ v) \wedge (if\ b\ then\ Q\ (restrict\ B\ v)\ else\ \neg Q\ (restrict\ B\ v))\}$

*<proof>*

**lemma** *in\_joinI*:  $table\ n\ A\ X \Longrightarrow table\ n\ B\ Y \Longrightarrow (\neg b \Longrightarrow B \subseteq A) \Longrightarrow wf\_tuple\ n\ (A \cup B)\ v \Longrightarrow$

$restrict\ A\ v \in X \Longrightarrow (b \Longrightarrow restrict\ B\ v \in Y) \Longrightarrow (\neg b \Longrightarrow restrict\ B\ v \notin Y) \Longrightarrow v \in join\ X\ b\ Y$

*<proof>*

**lemma** *in\_joinE*:  $v \in join\ X\ b\ Y \Longrightarrow table\ n\ A\ X \Longrightarrow table\ n\ B\ Y \Longrightarrow (\neg b \Longrightarrow B \subseteq A) \Longrightarrow$

$(wf\_tuple\ n\ (A \cup B)\ v \Longrightarrow restrict\ A\ v \in X \Longrightarrow if\ b\ then\ restrict\ B\ v \in Y\ else\ restrict\ B\ v \notin Y \Longrightarrow$

$P) \Longrightarrow P$

*<proof>*

**definition** *qtable* ::  $nat \Rightarrow nat\ set \Rightarrow ('a\ tuple \Rightarrow bool) \Rightarrow ('a\ tuple \Rightarrow bool) \Rightarrow$

$'a\ table \Rightarrow bool$  **where**

$qtable\ n\ A\ P\ Q\ X \longleftrightarrow table\ n\ A\ X \wedge (\forall x. (x \in X \wedge P\ x \longrightarrow Q\ x) \wedge (wf\_tuple\ n\ A\ x \wedge P\ x \wedge Q\ x \longrightarrow x \in X))$

**abbreviation** *wf\_table* **where**

$wf\_table\ n\ A\ Q\ X \equiv qtable\ n\ A\ (\lambda_.\ True)\ Q\ X$

**lemma** *wf\_table\_iff*:  $wf\_table\ n\ A\ Q\ X \longleftrightarrow (\forall x. x \in X \longleftrightarrow (Q\ x \wedge wf\_tuple\ n\ A\ x))$

*<proof>*

**lemma** *table\_wf\_table*:  $table\ n\ A\ X = wf\_table\ n\ A\ (\lambda v. v \in X)\ X$

*<proof>*

**lemma** *qtableI*:  $table\ n\ A\ X \Longrightarrow$

$(\bigwedge x. x \in X \Longrightarrow wf\_tuple\ n\ A\ x \Longrightarrow P\ x \Longrightarrow Q\ x) \Longrightarrow$

$(\bigwedge x. \text{wf\_tuple } n \ A \ x \implies P \ x \implies Q \ x \implies x \in X) \implies$   
 $\text{qtable } n \ A \ P \ Q \ X$   
 $\langle \text{proof} \rangle$

**lemma** *in\_qtableI*:  $\text{qtable } n \ A \ P \ Q \ X \implies \text{wf\_tuple } n \ A \ x \implies P \ x \implies Q \ x \implies x \in X$   
 $\langle \text{proof} \rangle$

**lemma** *in\_qtableE*:  $\text{qtable } n \ A \ P \ Q \ X \implies x \in X \implies P \ x \implies (\text{wf\_tuple } n \ A \ x \implies Q \ x \implies R) \implies R$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_empty*:  $(\bigwedge x. \text{wf\_tuple } n \ A \ x \implies P \ x \implies Q \ x \implies \text{False}) \implies \text{qtable } n \ A \ P \ Q \ \text{empty\_table}$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_empty\_iff*:  $\text{qtable } n \ A \ P \ Q \ \text{empty\_table} = (\forall x. \text{wf\_tuple } n \ A \ x \longrightarrow P \ x \longrightarrow Q \ x \longrightarrow \text{False})$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_unit\_table*:  $(\bigwedge x. \text{wf\_tuple } n \ \{\} \ x \implies P \ x \implies Q \ x) \implies \text{qtable } n \ \{\} \ P \ Q \ (\text{unit\_table } n)$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_union*:  $\text{qtable } n \ A \ P \ Q1 \ X \implies \text{qtable } n \ A \ P \ Q2 \ Y \implies$   
 $(\bigwedge x. \text{wf\_tuple } n \ A \ x \implies P \ x \implies Q \ x \longleftrightarrow Q1 \ x \vee Q2 \ x) \implies \text{qtable } n \ A \ P \ Q \ (X \cup Y)$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_Union*:  $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{qtable } n \ A \ P \ (Qi \ i) \ (Xi \ i)) \implies$   
 $(\bigwedge x. \text{wf\_tuple } n \ A \ x \implies P \ x \implies Q \ x \longleftrightarrow (\exists i \in I. Qi \ i \ x)) \implies \text{qtable } n \ A \ P \ Q \ (\bigcup i \in I. Xi \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *qtable\_join*:

**assumes**  $\text{qtable } n \ A \ P \ Q1 \ X \ \text{qtable } n \ B \ P \ Q2 \ Y \ \neg b \implies B \subseteq A \ C = A \cup B$   
 $\bigwedge x. \text{wf\_tuple } n \ C \ x \implies P \ x \implies P \ (\text{restrict } A \ x) \wedge P \ (\text{restrict } B \ x)$   
 $\bigwedge x. b \implies \text{wf\_tuple } n \ C \ x \implies P \ x \implies Q \ x \longleftrightarrow Q1 \ (\text{restrict } A \ x) \wedge Q2 \ (\text{restrict } B \ x)$   
 $\bigwedge x. \neg b \implies \text{wf\_tuple } n \ C \ x \implies P \ x \implies Q \ x \longleftrightarrow Q1 \ (\text{restrict } A \ x) \wedge \neg Q2 \ (\text{restrict } B \ x)$   
**shows**  $\text{qtable } n \ C \ P \ Q \ (\text{join } X \ b \ Y)$

$\langle \text{proof} \rangle$

**lemma** *qtable\_join\_fixed*:

**assumes**  $\text{qtable } n \ A \ P \ Q1 \ X \ \text{qtable } n \ B \ P \ Q2 \ Y \ \neg b \implies B \subseteq A \ C = A \cup B$   
 $\bigwedge x. \text{wf\_tuple } n \ C \ x \implies P \ x \implies P \ (\text{restrict } A \ x) \wedge P \ (\text{restrict } B \ x)$   
**shows**  $\text{qtable } n \ C \ P \ (\lambda x. Q1 \ (\text{restrict } A \ x) \wedge (\text{if } b \ \text{then } Q2 \ (\text{restrict } B \ x) \ \text{else } \neg Q2 \ (\text{restrict } B \ x)))$   
 $(\text{join } X \ b \ Y)$

$\langle \text{proof} \rangle$

**lemma** *wf\_tuple\_cong*:

**assumes**  $\text{wf\_tuple } n \ A \ v \ \text{wf\_tuple } n \ A \ w \ \forall x \in A. \text{map the } v \ ! \ x = \text{map the } w \ ! \ x$   
**shows**  $v = w$

$\langle \text{proof} \rangle$

**definition** *mem\_restr* ::  $'a \ \text{list set} \Rightarrow 'a \ \text{tuple} \Rightarrow \text{bool}$  **where**

$\text{mem\_restr } A \ x \longleftrightarrow (\exists y \in A. \text{list\_all2 } (\lambda a \ b. a \neq \text{None} \longrightarrow a = \text{Some } b) \ x \ y)$

**lemma** *mem\_restrI*:  $y \in A \implies \text{length } y = n \implies \text{wf\_tuple } n \ V \ x \implies \forall i \in V. x \ ! \ i = \text{Some } (y \ ! \ i) \implies$   
 $\text{mem\_restr } A \ x$   
 $\langle \text{proof} \rangle$

**lemma** *mem\_restrE*:  $\text{mem\_restr } A \ x \implies \text{wf\_tuple } n \ V \ x \implies \forall i \in V. i < n \implies$   
 $(\bigwedge y. y \in A \implies \forall i \in V. x \ ! \ i = \text{Some } (y \ ! \ i) \implies P) \implies P$   
 $\langle \text{proof} \rangle$

**lemma** *mem\_restr\_IntD*:  $mem\_restr (A \cap B) v \implies mem\_restr A v \wedge mem\_restr B v$   
 ⟨proof⟩

**lemma** *mem\_restr\_Un\_iff*:  $mem\_restr (A \cup B) x \longleftrightarrow mem\_restr A x \vee mem\_restr B x$   
 ⟨proof⟩

**lemma** *mem\_restr\_UNIV [simp]*:  $mem\_restr UNIV x$   
 ⟨proof⟩

**lemma** *restrict\_mem\_restr [simp]*:  $mem\_restr A x \implies mem\_restr A (restrict V x)$   
 ⟨proof⟩

**definition** *lift\_envs* :: 'a list set  $\Rightarrow$  'a list set **where**  
*lift\_envs* R =  $(\lambda(a,b). a \# b) \text{ ' } (UNIV \times R)$

**lemma** *lift\_envs\_mem\_restr [simp]*:  $mem\_restr A x \implies mem\_restr (lift\_envs A) (a \# x)$   
 ⟨proof⟩

**lemma** *qtable\_project*:

**assumes** *qtable* (Suc n) A (mem\_restr (lift\_envs R)) P X  
**shows** *qtable* n (( $\lambda x. x - Suc 0$ ) ' (A - {0})) (mem\_restr R)  
 ( $\lambda v. \exists x. P ((if 0 \in A \text{ then } Some x \text{ else } None) \# v)$ ) (tl ' X)  
 (**is** *qtable* n ?A (mem\_restr R) ?P ?X)

⟨proof⟩

**lemma** *qtable\_cong*:  $qtable n A P Q X \implies A = B \implies (\bigwedge v. P v \implies Q v \longleftrightarrow Q' v) \implies qtable n B P Q' X$   
 ⟨proof⟩

### 3 Abstract monitors and slicing

#### 3.1 First-order specifications

We abstract from first-order trace specifications by referring only to their semantics. A first-order specification is described by a finite set of free variables and a satisfaction function that defines for every trace the pairs of valuations and time-points for which the specification is satisfied.

**locale** *fo\_spec* =

**fixes**

*nfv* :: nat **and** *fv* :: nat set **and**  
*sat* :: 'a trace  $\Rightarrow$  'b list  $\Rightarrow$  nat  $\Rightarrow$  bool

**assumes**

*fv\_less\_nfv*:  $x \in fv \implies x < nfv$  **and**  
*sat\_fv\_cong*:  $(\bigwedge x. x \in fv \implies v!x = v'!x) \implies sat \sigma v i = sat \sigma v' i$

**begin**

**definition** *verdicts* :: 'a trace  $\Rightarrow$  (nat  $\times$  'b tuple) set **where**

*verdicts*  $\sigma = \{(i, v). wf\_tuple nfv fv v \wedge sat \sigma (map the v) i\}$

**end**

We usually employ a monitor to find the *violations* of a specification. That is, the monitor should output the satisfactions of its negation. Moreover, all monitor implementations must work with finite prefixes. We are therefore interested in co-safety properties, which allow us to identify all satisfactions on finite prefixes.

**locale** *cosafety\_fo\_spec* = *fo\_spec* +

**assumes** *cosafety\_br*:  $sat \sigma v i \implies \exists \pi. prefix\_of \pi \sigma \wedge (\forall \sigma'. prefix\_of \pi \sigma' \longrightarrow sat \sigma' v i)$

**begin**

**lemma** *cosafety*:  $sat \sigma v i \longleftrightarrow (\exists \pi. prefix\_of \pi \sigma \wedge (\forall \sigma'. prefix\_of \pi \sigma' \longrightarrow sat \sigma' v i))$   
 ⟨*proof*⟩

**end**

### 3.2 Monitor function

We model monitors abstractly as functions from prefixes to verdict sets. The following locale specifies a minimal set of properties that any reasonable monitor should have.

**locale** *monitor* = *fo\_spec* +

**fixes**  $M :: 'a prefix \Rightarrow (nat \times 'b tuple) set$

**assumes**

*mono\_monitor*:  $\pi \leq \pi' \Longrightarrow M \pi \subseteq M \pi'$  **and**

*wf\_monitor*:  $(i, v) \in M \pi \Longrightarrow wf\_tuple \ nfv \ fv \ v$  **and**

*sound\_monitor*:  $(i, v) \in M \pi \Longrightarrow prefix\_of \pi \sigma \Longrightarrow sat \sigma (map \ the \ v) \ i$  **and**

*complete\_monitor*:  $prefix\_of \pi \sigma \Longrightarrow wf\_tuple \ nfv \ fv \ v \Longrightarrow$

$(\bigwedge \sigma. prefix\_of \pi \sigma \Longrightarrow sat \sigma (map \ the \ v) \ i) \Longrightarrow \exists \pi'. prefix\_of \pi' \sigma \wedge (i, v) \in M \pi'$

A monitor for a co-safety specification computes precisely the set of all satisfactions in the limit:

**abbreviation** (in *monitor*)  $M\_limit \sigma \equiv \bigcup \{M \pi \mid \pi. prefix\_of \pi \sigma\}$

**locale** *cosafety\_monitor* = *cosafety\_fo\_spec* + *monitor*

**begin**

**lemma** *M\_limit\_eq*:  $M\_limit \sigma = verdicts \sigma$

⟨*proof*⟩

**end**

The monitor function  $M$  adds some information over *sat*, namely when a verdict is output. One possible behavior is that the monitor outputs its verdicts for one time-point at a time, in increasing order of time-points. Then  $M$  is uniquely defined by a progress function, which returns for every prefix the time-point up to which all verdicts are computed.

**locale** *progress* = *fo\_spec* \_\_ *sat* **for**  $sat :: 'a trace \Rightarrow 'b list \Rightarrow nat \Rightarrow bool$  +

**fixes**  $progress :: 'a prefix \Rightarrow nat$

**assumes**

*progress\_mono*:  $\pi \leq \pi' \Longrightarrow progress \pi \leq progress \pi'$  **and**

*ex\_progress\_ge*:  $\exists \pi. prefix\_of \pi \sigma \wedge x \leq progress \pi$  **and**

*progress\_sat\_cong*:  $prefix\_of \pi \sigma \Longrightarrow prefix\_of \pi \sigma' \Longrightarrow i < progress \pi \Longrightarrow$

$sat \sigma v i \longleftrightarrow sat \sigma' v i$

— The last condition is not necessary to obtain a proper monitor function. However, it corresponds to the intuitive understanding of monitor progress, and it results in a stronger characterisation. In particular, it implies that the specification is co-safety, as we will show below.

**begin**

**definition**  $M :: 'a prefix \Rightarrow (nat \times 'b tuple) set$  **where**

$M \pi = \{(i, v). i < progress \pi \wedge wf\_tuple \ nfv \ fv \ v \wedge$

$(\forall \sigma. prefix\_of \pi \sigma \longrightarrow sat \sigma (map \ the \ v) \ i)\}$

**lemma** *M\_alt*:  $M \pi = \{(i, v). i < progress \pi \wedge wf\_tuple \ nfv \ fv \ v \wedge$

$(\exists \sigma. prefix\_of \pi \sigma \wedge sat \sigma (map \ the \ v) \ i)\}$

⟨*proof*⟩

**end**

**sublocale** *progress*  $\subseteq$  *cosafety\_monitor* \_ \_ \_ *M*  
 ⟨*proof*⟩

### 3.3 Slicing

Sliceable specifications can be evaluated meaningfully on a subset of events.

**locale** *abstract\_slicer* =  
**fixes** *relevant\_events* :: 'b list set  $\Rightarrow$  'a set  
**begin**

**abbreviation** *slice* :: 'b list set  $\Rightarrow$  'a trace  $\Rightarrow$  'a trace **where**  
*slice* *S*  $\equiv$  *map\_Γ* ( $\lambda D. D \cap$  *relevant\_events* *S*)

**abbreviation** *pslice* :: 'b list set  $\Rightarrow$  'a prefix  $\Rightarrow$  'a prefix **where**  
*pslice* *S*  $\equiv$  *pmap\_Γ* ( $\lambda D. D \cap$  *relevant\_events* *S*)

**lemma** *prefix\_of\_psliceI*: *prefix\_of*  $\pi$   $\sigma \Longrightarrow$  *prefix\_of* (*pslice* *S*  $\pi$ ) (*slice* *S*  $\sigma$ )  
 ⟨*proof*⟩

**lemma** *plen\_pslice[simp]*: *plen* (*pslice* *S*  $\pi$ ) = *plen*  $\pi$   
 ⟨*proof*⟩

**lemma** *pslice\_pnil[simp]*: *pslice* *S* *pnil* = *pnil*  
 ⟨*proof*⟩

**lemma** *last\_ts\_pslice[simp]*: *last\_ts* (*pslice* *S*  $\pi$ ) = *last\_ts*  $\pi$   
 ⟨*proof*⟩

**abbreviation** *verdict\_filter* :: 'b list set  $\Rightarrow$  (*nat*  $\times$  'b tuple) set  $\Rightarrow$  (*nat*  $\times$  'b tuple) set **where**  
*verdict\_filter* *S* *V*  $\equiv$   $\{(i, v) \in V. \text{mem\_restr } S \ v\}$

**end**

**locale** *sliceable\_fo\_spec* = *fo\_spec* \_ \_ *sat* + *abstract\_slicer* *relevant\_events*  
**for** *relevant\_events* :: 'b list set  $\Rightarrow$  'a set **and** *sat* :: 'a trace  $\Rightarrow$  'b list  $\Rightarrow$  *nat*  $\Rightarrow$  *bool* +  
**assumes** *sliceable*:  $v \in S \Longrightarrow \text{sat } (\text{slice } S \ \sigma) \ v \ i \longleftrightarrow \text{sat } \sigma \ v \ i$   
**begin**

**lemma** *union\_verdicts\_slice*:  
**assumes** *part*:  $\bigcup S = UNIV$   
**shows**  $\bigcup ((\lambda S. \text{verdict\_filter } S \ (\text{verdicts } (\text{slice } S \ \sigma))) \ 'S) = \text{verdicts } \sigma$   
 ⟨*proof*⟩

**end**

We define a similar notion for monitors. It is potentially stronger because the time-point at which verdicts are output must not change.

**locale** *sliceable\_monitor* = *monitor* \_ \_ *sat* *M* + *abstract\_slicer* *relevant\_events*  
**for** *relevant\_events* :: 'b list set  $\Rightarrow$  'a set **and** *sat* :: 'a trace  $\Rightarrow$  'b list  $\Rightarrow$  *nat*  $\Rightarrow$  *bool* **and** *M* +  
**assumes** *sliceable\_M*:  $\text{mem\_restr } S \ v \Longrightarrow (i, v) \in M \ (\text{pslice } S \ \pi) \longleftrightarrow (i, v) \in M \ \pi$   
**begin**

**lemma** *union\_M\_pslice*:  
**assumes** *part*:  $\bigcup S = UNIV$   
**shows**  $\bigcup ((\lambda S. \text{verdict\_filter } S \ (M \ (\text{pslice } S \ \pi))) \ 'S) = M \ \pi$   
 ⟨*proof*⟩

**end**

If the specification is sliceable and the monitor's progress depends only on time-stamps, then also the monitor itself is sliceable.

```
locale timed_progress = progress +  
  assumes progress_time_conv: pts  $\pi = \text{pts } \pi' \implies \text{progress } \pi = \text{progress } \pi'$ 
```

```
locale sliceable_timed_progress = sliceable_fo_spec + timed_progress  
begin
```

```
lemma progress_pslice[simp]: progress (pslice S  $\pi$ ) = progress  $\pi$   
  <proof>
```

**end**

```
sublocale sliceable_timed_progress  $\subseteq$  sliceable_monitor _ _ _ _ M  
<proof>
```

## 4 Intervals

```
typedef  $\mathcal{I} = \{(i :: \text{nat}, j :: \text{enat}). i \leq j\}$   
<proof>
```

```
setup_lifting type_definition  $\mathcal{I}$ 
```

```
instantiation  $\mathcal{I} :: \text{equal}$  begin  
lift_definition equal  $\mathcal{I} :: \mathcal{I} \Rightarrow \mathcal{I} \Rightarrow \text{bool}$  is (=) <proof>  
instance <proof>  
end
```

```
instantiation  $\mathcal{I} :: \text{linorder}$  begin  
lift_definition less_eq  $\mathcal{I} :: \mathcal{I} \Rightarrow \mathcal{I} \Rightarrow \text{bool}$  is ( $\leq$ ) <proof>  
lift_definition less  $\mathcal{I} :: \mathcal{I} \Rightarrow \mathcal{I} \Rightarrow \text{bool}$  is ( $<$ ) <proof>  
instance <proof>  
end
```

```
lift_definition all ::  $\mathcal{I}$  is ( $(0, \infty)$ ) <proof>  
lift_definition left ::  $\mathcal{I} \Rightarrow \text{nat}$  is fst <proof>  
lift_definition right ::  $\mathcal{I} \Rightarrow \text{enat}$  is snd <proof>  
lift_definition point ::  $\text{nat} \Rightarrow \mathcal{I}$  is  $\lambda n. (n, \text{enat } n)$  <proof>  
lift_definition init ::  $\text{nat} \Rightarrow \mathcal{I}$  is  $\lambda n. (0, \text{enat } n)$  <proof>  
abbreviation mem where mem  $n I \equiv (\text{left } I \leq n \wedge n \leq \text{right } I)$   
lift_definition subtract ::  $\text{nat} \Rightarrow \mathcal{I} \Rightarrow \mathcal{I}$  is  
   $\lambda n (i, j). (i - n, j - \text{enat } n)$  <proof>  
lift_definition add ::  $\text{nat} \Rightarrow \mathcal{I} \Rightarrow \mathcal{I}$  is  
   $\lambda n (a, b). (a, b + \text{enat } n)$  <proof>
```

```
lemma left_right: left  $I \leq \text{right } I$   
<proof>
```

```
lemma point_simps[simp]:  
  left (point  $n$ ) =  $n$   
  right (point  $n$ ) =  $n$   
<proof>
```

```
lemma init_simps[simp]:  
  left (init  $n$ ) =  $0$ 
```

*right* (*init* *n*) = *n*  
 ⟨*proof*⟩

**lemma** *subtract\_simps*[*simp*]:  
*left* (*subtract* *n* *I*) = *left* *I* - *n*  
*right* (*subtract* *n* *I*) = *right* *I* - *n*  
*subtract* 0 *I* = *I*  
*subtract* *x* (*point* *y*) = *point* (*y* - *x*)  
 ⟨*proof*⟩

**definition** *shifted* ::  $\mathcal{I} \Rightarrow \mathcal{I}$  **set where**  
*shifted* *I* = ( $\lambda n.$  *subtract* *n* *I*) ‘ {0 .. (case *right* *I* of  $\infty \Rightarrow$  *left* *I* | *enat* *n*  $\Rightarrow$  *n*)}

**lemma** *subtract\_too\_much*:  $i > (\text{case } \text{right } I \text{ of } \infty \Rightarrow \text{left } I \mid \text{enat } n \Rightarrow n) \implies$   
*subtract* *i* *I* = *subtract* (case *right* *I* of  $\infty \Rightarrow$  *left* *I* | *enat* *n*  $\Rightarrow$  *n*) *I*  
 ⟨*proof*⟩

**lemma** *subtract\_shifted*: *subtract* *n* *I*  $\in$  *shifted* *I*  
 ⟨*proof*⟩

**lemma** *finite\_shifted*: *finite* (*shifted* *I*)  
 ⟨*proof*⟩

**definition** *interval* ::  $\text{nat} \Rightarrow \text{enat} \Rightarrow \mathcal{I}$  **where**  
*interval* *l* *r* = (if  $l \leq r$  then *Abs* $\_I$  (*l*, *r*) else *undefined*)

**lemma** [*code abstract*]: *Rep* $\_I$  (*interval* *l* *r*) = (if  $l \leq r$  then (*l*, *r*) else *Rep* $\_I$  *undefined*)  
 ⟨*proof*⟩

## 5 Metric first-order temporal logic

context begin

### 5.1 Formulas and satisfiability

**qualified type\_synonym** *name* = *string*  
**qualified type\_synonym** 'a *event* = (*name*  $\times$  'a *list*)  
**qualified type\_synonym** 'a *database* = 'a *event* *set*  
**qualified type\_synonym** 'a *prefix* = (*name*  $\times$  'a *list*) *prefix*  
**qualified type\_synonym** 'a *trace* = (*name*  $\times$  'a *list*) *trace*

**qualified type\_synonym** 'a *env* = 'a *list*

**qualified datatype** 'a *trm* = *Var* *nat* | *is* $\_Const$ : *Const* 'a

**qualified primrec** *fvi* $\_trm$  ::  $\text{nat} \Rightarrow$  'a *trm*  $\Rightarrow$  *nat* *set* **where**  
*fvi* $\_trm$  *b* (*Var* *x*) = (if  $b \leq x$  then {*x* - *b*} else {})  
 | *fvi* $\_trm$  *b* (*Const*  $\_$ ) = {}

**abbreviation** *fv* $\_trm$   $\equiv$  *fvi* $\_trm$  0

**qualified primrec** *eval* $\_trm$  :: 'a *env*  $\Rightarrow$  'a *trm*  $\Rightarrow$  'a **where**  
*eval* $\_trm$  *v* (*Var* *x*) = *v* ! *x*  
 | *eval* $\_trm$  *v* (*Const* *x*) = *x*

**lemma** *eval\_trm\_cong*:  $\forall x \in \text{fv\_trm } t. v ! x = v' ! x \implies \text{eval\_trm } v t = \text{eval\_trm } v' t$   
 ⟨*proof*⟩ **datatype** (*discs* $\_sels$ ) 'a *formula* = *Pred* *name* 'a *trm* *list* | *Eq* 'a *trm* 'a *trm*  
 | *Neg* 'a *formula* | *Or* 'a *formula* 'a *formula* | *Exists* 'a *formula*

| *Prev*  $\mathcal{I}$  'a formula | *Next*  $\mathcal{I}$  'a formula  
| *Since* 'a formula  $\mathcal{I}$  'a formula | *Until* 'a formula  $\mathcal{I}$  'a formula

**qualified primrec** *fvi* :: nat  $\Rightarrow$  'a formula  $\Rightarrow$  nat set **where**

*fvi* b (*Pred* r ts) = ( $\bigcup$  t $\in$ set ts. *fvi\_trm* b t)  
| *fvi* b (*Eq* t1 t2) = *fvi\_trm* b t1  $\cup$  *fvi\_trm* b t2  
| *fvi* b (*Neg*  $\varphi$ ) = *fvi* b  $\varphi$   
| *fvi* b (*Or*  $\varphi$   $\psi$ ) = *fvi* b  $\varphi$   $\cup$  *fvi* b  $\psi$   
| *fvi* b (*Exists*  $\varphi$ ) = *fvi* (*Suc* b)  $\varphi$   
| *fvi* b (*Prev* I  $\varphi$ ) = *fvi* b  $\varphi$   
| *fvi* b (*Next* I  $\varphi$ ) = *fvi* b  $\varphi$   
| *fvi* b (*Since*  $\varphi$  I  $\psi$ ) = *fvi* b  $\varphi$   $\cup$  *fvi* b  $\psi$   
| *fvi* b (*Until*  $\varphi$  I  $\psi$ ) = *fvi* b  $\varphi$   $\cup$  *fvi* b  $\psi$

**abbreviation** *fv*  $\equiv$  *fvi* 0

**lemma** *finite\_fvi\_trm[simp]*: finite (*fvi\_trm* b t)  
<proof>

**lemma** *finite\_fvi[simp]*: finite (*fvi* b  $\varphi$ )  
<proof>

**lemma** *fvi\_trm\_Suc*:  $x \in$  *fvi\_trm* (*Suc* b) t  $\longleftrightarrow$  *Suc* x  $\in$  *fvi\_trm* b t  
<proof>

**lemma** *fvi\_Suc*:  $x \in$  *fvi* (*Suc* b)  $\varphi$   $\longleftrightarrow$  *Suc* x  $\in$  *fvi* b  $\varphi$   
<proof>

**lemma** *fvi\_Suc\_bound*:

**assumes**  $\forall i \in$  *fvi* (*Suc* b)  $\varphi$ .  $i < n$

**shows**  $\forall i \in$  *fvi* b  $\varphi$ .  $i <$  *Suc* n

<proof> **definition** *nfv* :: 'a formula  $\Rightarrow$  nat **where**  
*nfv*  $\varphi$  = *Max* (*insert* 0 (*Suc* ' *fv*  $\varphi$ ))

**qualified definition** *envs* :: 'a formula  $\Rightarrow$  'a env set **where**

*envs*  $\varphi$  = {v. length v = *nfv*  $\varphi$ }

**lemma** *nfv\_simps[simp]*:

*nfv* (*Neg*  $\varphi$ ) = *nfv*  $\varphi$   
*nfv* (*Or*  $\varphi$   $\psi$ ) = *max* (*nfv*  $\varphi$ ) (*nfv*  $\psi$ )  
*nfv* (*Prev* I  $\varphi$ ) = *nfv*  $\varphi$   
*nfv* (*Next* I  $\varphi$ ) = *nfv*  $\varphi$   
*nfv* (*Since*  $\varphi$  I  $\psi$ ) = *max* (*nfv*  $\varphi$ ) (*nfv*  $\psi$ )  
*nfv* (*Until*  $\varphi$  I  $\psi$ ) = *max* (*nfv*  $\varphi$ ) (*nfv*  $\psi$ )  
<proof>

**lemma** *fvi\_less\_nfv*:  $\forall i \in$  *fv*  $\varphi$ .  $i <$  *nfv*  $\varphi$

<proof> **primrec** *future\_reach* :: 'a formula  $\Rightarrow$  enat **where**

*future\_reach* (*Pred* \_\_) = 0  
| *future\_reach* (*Eq* \_\_) = 0  
| *future\_reach* (*Neg*  $\varphi$ ) = *future\_reach*  $\varphi$   
| *future\_reach* (*Or*  $\varphi$   $\psi$ ) = *max* (*future\_reach*  $\varphi$ ) (*future\_reach*  $\psi$ )  
| *future\_reach* (*Exists*  $\varphi$ ) = *future\_reach*  $\varphi$   
| *future\_reach* (*Prev* I  $\varphi$ ) = *future\_reach*  $\varphi$  - *left* I  
| *future\_reach* (*Next* I  $\varphi$ ) = *future\_reach*  $\varphi$  + *right* I + 1  
| *future\_reach* (*Since*  $\varphi$  I  $\psi$ ) = *max* (*future\_reach*  $\varphi$ ) (*future\_reach*  $\psi$  - *left* I)  
| *future\_reach* (*Until*  $\varphi$  I  $\psi$ ) = *max* (*future\_reach*  $\varphi$ ) (*future\_reach*  $\psi$ ) + *right* I + 1

**qualified primrec**  $\text{sat} :: 'a \text{ trace} \Rightarrow 'a \text{ env} \Rightarrow \text{nat} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$  where

$\text{sat } \sigma \ v \ i \ (\text{Pred } r \ ts) = ((r, \text{map } (\text{eval\_trm } v) \ ts) \in \Gamma \ \sigma \ i)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Eq } t1 \ t2) = (\text{eval\_trm } v \ t1 = \text{eval\_trm } v \ t2)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Neg } \varphi) = (\neg \text{sat } \sigma \ v \ i \ \varphi)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Or } \varphi \ \psi) = (\text{sat } \sigma \ v \ i \ \varphi \vee \text{sat } \sigma \ v \ i \ \psi)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Exists } \varphi) = (\exists z. \text{sat } \sigma \ (z \# v) \ i \ \varphi)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Prev } I \ \varphi) = (\text{case } i \ \text{of } 0 \Rightarrow \text{False} \mid \text{Suc } j \Rightarrow \text{mem } (\tau \ \sigma \ i - \tau \ \sigma \ j) \ I \wedge \text{sat } \sigma \ v \ j \ \varphi)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Next } I \ \varphi) = (\text{mem } (\tau \ \sigma \ (\text{Suc } i) - \tau \ \sigma \ i) \ I \wedge \text{sat } \sigma \ v \ (\text{Suc } i) \ \varphi)$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ I \ \psi) = (\exists j \leq i. \text{mem } (\tau \ \sigma \ i - \tau \ \sigma \ j) \ I \wedge \text{sat } \sigma \ v \ j \ \psi \wedge (\forall k \in \{j <.. i\}. \text{sat } \sigma \ v \ k \ \varphi))$   
 $|\ \text{sat } \sigma \ v \ i \ (\text{Until } \varphi \ I \ \psi) = (\exists j \geq i. \text{mem } (\tau \ \sigma \ j - \tau \ \sigma \ i) \ I \wedge \text{sat } \sigma \ v \ j \ \psi \wedge (\forall k \in \{i ..< j\}. \text{sat } \sigma \ v \ k \ \varphi))$

**lemma**  $\text{sat\_Until\_rec}$ :  $\text{sat } \sigma \ v \ i \ (\text{Until } \varphi \ I \ \psi) \longleftrightarrow$   
 $\text{mem } 0 \ I \wedge \text{sat } \sigma \ v \ i \ \psi \vee$   
 $(\Delta \ \sigma \ (i + 1) \leq \text{right } I \wedge \text{sat } \sigma \ v \ i \ \varphi \wedge \text{sat } \sigma \ v \ (i + 1) \ (\text{Until } \varphi \ (\text{subtract } (\Delta \ \sigma \ (i + 1)) \ I) \ \psi))$   
 $(\text{is } ?L \longleftrightarrow ?R)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_Since\_rec}$ :  $\text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ I \ \psi) \longleftrightarrow$   
 $\text{mem } 0 \ I \wedge \text{sat } \sigma \ v \ i \ \psi \vee$   
 $(i > 0 \wedge \Delta \ \sigma \ i \leq \text{right } I \wedge \text{sat } \sigma \ v \ i \ \varphi \wedge \text{sat } \sigma \ v \ (i - 1) \ (\text{Since } \varphi \ (\text{subtract } (\Delta \ \sigma \ i) \ I) \ \psi))$   
 $(\text{is } ?L \longleftrightarrow ?R)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_Since}_0$ :  $\text{sat } \sigma \ v \ 0 \ (\text{Since } \varphi \ I \ \psi) \longleftrightarrow \text{mem } 0 \ I \wedge \text{sat } \sigma \ v \ 0 \ \psi$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_Since\_point}$ :  $\text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ I \ \psi) \implies$   
 $(\bigwedge j. j \leq i \implies \text{mem } (\tau \ \sigma \ i - \tau \ \sigma \ j) \ I \implies \text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ (\text{point } (\tau \ \sigma \ i - \tau \ \sigma \ j)) \ \psi) \implies P) \implies P$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_Since\_pointD}$ :  $\text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ (\text{point } t) \ \psi) \implies \text{mem } t \ I \implies \text{sat } \sigma \ v \ i \ (\text{Since } \varphi \ I \ \psi)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{eval\_trm\_fvi\_cong}$ :  $\forall x \in \text{fv\_trm } t. v!x = v'!x \implies \text{eval\_trm } v \ t = \text{eval\_trm } v' \ t$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_fvi\_cong}$ :  $\forall x \in \text{fv } \varphi. v!x = v'!x \implies \text{sat } \sigma \ v \ i \ \varphi = \text{sat } \sigma \ v' \ i \ \varphi$   
 $\langle \text{proof} \rangle$

## 5.2 Defined connectives

**qualified definition**  $\text{And } \varphi \ \psi = \text{Neg } (\text{Or } (\text{Neg } \varphi) \ (\text{Neg } \psi))$

**lemma**  $\text{fvi\_And}$ :  $\text{fvi } b \ (\text{And } \varphi \ \psi) = \text{fvi } b \ \varphi \cup \text{fvi } b \ \psi$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{nfv\_And[simp]}$ :  $\text{nfv } (\text{And } \varphi \ \psi) = \max (\text{nfv } \varphi) \ (\text{nfv } \psi)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{future\_reach\_And}$ :  $\text{future\_reach } (\text{And } \varphi \ \psi) = \max (\text{future\_reach } \varphi) \ (\text{future\_reach } \psi)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sat\_And}$ :  $\text{sat } \sigma \ v \ i \ (\text{And } \varphi \ \psi) = (\text{sat } \sigma \ v \ i \ \varphi \wedge \text{sat } \sigma \ v \ i \ \psi)$   
 $\langle \text{proof} \rangle$  **definition**  $\text{And\_Not } \varphi \ \psi = \text{Neg } (\text{Or } (\text{Neg } \varphi) \ \psi)$

**lemma**  $\text{fvi\_And\_Not}$ :  $\text{fvi } b \ (\text{And\_Not } \varphi \ \psi) = \text{fvi } b \ \varphi \cup \text{fvi } b \ \psi$   
 $\langle \text{proof} \rangle$

**lemma** *nfv\_And\_Not[simp]*:  $nfv (And\_Not \varphi \psi) = max (nfv \varphi) (nfv \psi)$   
 ⟨proof⟩

**lemma** *future\_reach\_And\_Not*:  $future\_reach (And\_Not \varphi \psi) = max (future\_reach \varphi) (future\_reach \psi)$   
 ⟨proof⟩

**lemma** *sat\_And\_Not*:  $sat \sigma v i (And\_Not \varphi \psi) = (sat \sigma v i \varphi \wedge \neg sat \sigma v i \psi)$   
 ⟨proof⟩

### 5.3 Safe formulas

**fun** *safe\_formula* :: 'a MFOTL.formula  $\Rightarrow$  bool **where**

*safe\_formula* (MFOTL.Eq t1 t2) = (MFOTL.is\_Const t1  $\vee$  MFOTL.is\_Const t2)  
 | *safe\_formula* (MFOTL.Neg (MFOTL.Eq (MFOTL.Const x) (MFOTL.Const y))) = True  
 | *safe\_formula* (MFOTL.Neg (MFOTL.Eq (MFOTL.Var x) (MFOTL.Var y))) = (x = y)  
 | *safe\_formula* (MFOTL.Pred e ts) = True  
 | *safe\_formula* (MFOTL.Neg (MFOTL.Or (MFOTL.Neg  $\varphi$ )  $\psi$ )) = (*safe\_formula*  $\varphi$   $\wedge$   
 ( *safe\_formula*  $\psi$   $\wedge$  MFOTL.fv  $\psi$   $\subseteq$  MFOTL.fv  $\varphi$   $\vee$  (case  $\psi$  of MFOTL.Neg  $\psi'$   $\Rightarrow$  *safe\_formula*  $\psi'$  |  
 \_  $\Rightarrow$  False)))  
 | *safe\_formula* (MFOTL.Or  $\varphi$   $\psi$ ) = (MFOTL.fv  $\psi$  = MFOTL.fv  $\varphi$   $\wedge$  *safe\_formula*  $\varphi$   $\wedge$  *safe\_formula*  
 $\psi$ )  
 | *safe\_formula* (MFOTL.Exists  $\varphi$ ) = (*safe\_formula*  $\varphi$ )  
 | *safe\_formula* (MFOTL.Prev I  $\varphi$ ) = (*safe\_formula*  $\varphi$ )  
 | *safe\_formula* (MFOTL.Next I  $\varphi$ ) = (*safe\_formula*  $\varphi$ )  
 | *safe\_formula* (MFOTL.Since  $\varphi$  I  $\psi$ ) = (MFOTL.fv  $\varphi$   $\subseteq$  MFOTL.fv  $\psi$   $\wedge$   
 (*safe\_formula*  $\varphi$   $\vee$  (case  $\varphi$  of MFOTL.Neg  $\varphi'$   $\Rightarrow$  *safe\_formula*  $\varphi'$  | \_  $\Rightarrow$  False))  $\wedge$  *safe\_formula*  $\psi$ )  
 | *safe\_formula* (MFOTL.Until  $\varphi$  I  $\psi$ ) = (MFOTL.fv  $\varphi$   $\subseteq$  MFOTL.fv  $\psi$   $\wedge$   
 (*safe\_formula*  $\varphi$   $\vee$  (case  $\varphi$  of MFOTL.Neg  $\varphi'$   $\Rightarrow$  *safe\_formula*  $\varphi'$  | \_  $\Rightarrow$  False))  $\wedge$  *safe\_formula*  $\psi$ )  
 | *safe\_formula* \_ = False

**lemma** *disjE\_Not2*:  $P \vee Q \Longrightarrow (P \Longrightarrow R) \Longrightarrow (\neg P \Longrightarrow Q \Longrightarrow R) \Longrightarrow R$   
 ⟨proof⟩

**lemma** *safe\_formula\_induct[consumes 1]*:

**assumes** *safe\_formula*  $\varphi$   
**and**  $\bigwedge t1 t2. MFOTL.is\_Const t1 \Longrightarrow P (MFOTL.Eq t1 t2)$   
**and**  $\bigwedge t1 t2. MFOTL.is\_Const t2 \Longrightarrow P (MFOTL.Eq t1 t2)$   
**and**  $\bigwedge x y. P (MFOTL.Neg (MFOTL.Eq (MFOTL.Const x) (MFOTL.Const y)))$   
**and**  $\bigwedge x y. x = y \Longrightarrow P (MFOTL.Neg (MFOTL.Eq (MFOTL.Var x) (MFOTL.Var y)))$   
**and**  $\bigwedge e ts. P (MFOTL.Pred e ts)$   
**and**  $\bigwedge \varphi \psi. \neg (safe\_formula (MFOTL.Neg \psi) \wedge MFOTL.fv \psi \subseteq MFOTL.fv \varphi) \Longrightarrow P \varphi \Longrightarrow P \psi$   
 $\Longrightarrow P (MFOTL.And \varphi \psi)$   
**and**  $\bigwedge \varphi \psi. safe\_formula \psi \Longrightarrow MFOTL.fv \psi \subseteq MFOTL.fv \varphi \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.And\_Not$   
 $\varphi \psi)$   
**and**  $\bigwedge \varphi \psi. MFOTL.fv \psi = MFOTL.fv \varphi \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.Or \varphi \psi)$   
**and**  $\bigwedge \varphi. P \varphi \Longrightarrow P (MFOTL.Exists \varphi)$   
**and**  $\bigwedge I \varphi. P \varphi \Longrightarrow P (MFOTL.Prev I \varphi)$   
**and**  $\bigwedge I \varphi. P \varphi \Longrightarrow P (MFOTL.Next I \varphi)$   
**and**  $\bigwedge \varphi I \psi. MFOTL.fv \varphi \subseteq MFOTL.fv \psi \Longrightarrow safe\_formula \varphi \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.Since$   
 $\varphi I \psi)$   
**and**  $\bigwedge \varphi I \psi. MFOTL.fv (MFOTL.Neg \varphi) \subseteq MFOTL.fv \psi \Longrightarrow$   
 $\neg safe\_formula (MFOTL.Neg \varphi) \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.Since (MFOTL.Neg \varphi) I \psi)$   
**and**  $\bigwedge \varphi I \psi. MFOTL.fv \varphi \subseteq MFOTL.fv \psi \Longrightarrow safe\_formula \varphi \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.Until$   
 $\varphi I \psi)$   
**and**  $\bigwedge \varphi I \psi. MFOTL.fv (MFOTL.Neg \varphi) \subseteq MFOTL.fv \psi \Longrightarrow$   
 $\neg safe\_formula (MFOTL.Neg \varphi) \Longrightarrow P \varphi \Longrightarrow P \psi \Longrightarrow P (MFOTL.Until (MFOTL.Neg \varphi) I \psi)$

**shows**  $P \varphi$   
 ⟨proof⟩

## 5.4 Slicing traces

**qualified primrec**  $\text{matches} :: 'a \text{ env} \Rightarrow 'a \text{ formula} \Rightarrow \text{name} \times 'a \text{ list} \Rightarrow \text{bool}$  **where**  
 $\text{matches } v \text{ (Pred } r \text{ ts)} e = (r = \text{fst } e \wedge \text{map } (\text{eval\_trm } v) \text{ ts} = \text{snd } e)$   
 $\text{matches } v \text{ (Eq } \_ \_ \_) e = \text{False}$   
 $\text{matches } v \text{ (Neg } \varphi \_) e = \text{matches } v \varphi e$   
 $\text{matches } v \text{ (Or } \varphi \psi \_) e = (\text{matches } v \varphi e \vee \text{matches } v \psi e)$   
 $\text{matches } v \text{ (Exists } \_ \_) e = (\exists z. \text{matches } (z \# v) \varphi e)$   
 $\text{matches } v \text{ (Prev } I \varphi \_) e = \text{matches } v \varphi e$   
 $\text{matches } v \text{ (Next } I \varphi \_) e = \text{matches } v \varphi e$   
 $\text{matches } v \text{ (Since } \varphi I \psi \_) e = (\text{matches } v \varphi e \vee \text{matches } v \psi e)$   
 $\text{matches } v \text{ (Until } \varphi I \psi \_) e = (\text{matches } v \varphi e \vee \text{matches } v \psi e)$

**lemma**  $\text{matches\_fvi\_cong}: \forall x \in \text{fv } \varphi. v!x = v'!x \implies \text{matches } v \varphi e = \text{matches } v' \varphi e$   
 ⟨proof⟩

**abbreviation**  $\text{relevant\_events where relevant\_events } \varphi S \equiv \{e. S \cap \{v. \text{matches } v \varphi e\} \neq \{\}\}$

**lemma**  $\text{sat\_slice\_strong}: \text{relevant\_events } \varphi S \subseteq E \implies v \in S \implies$   
 $\text{sat } \sigma v i \varphi \longleftrightarrow \text{sat } (\text{map\_}\Gamma (\lambda D. D \cap E) \sigma) v i \varphi$   
 ⟨proof⟩

**end**

**interpretation**  $\text{MFOTL\_slicer}: \text{abstract\_slicer relevant\_events } \varphi$  **for**  $\varphi$  ⟨proof⟩

**lemma**  $\text{sat\_slice\_iff}: \text{assumes } v \in S$   
**shows**  $\text{MFOTL.sat } \sigma v i \varphi \longleftrightarrow \text{MFOTL.sat } (\text{MFOTL\_slicer.slice } \varphi S \sigma) v i \varphi$   
 ⟨proof⟩

**lemma**  $\text{slice\_replace\_prefix}: \text{prefix\_of } (\text{MFOTL\_slicer.pslic} \varphi R \pi) \sigma \implies$   
 $\text{MFOTL\_slicer.slice } \varphi R (\text{replace\_prefix } \pi \sigma) = \text{MFOTL\_slicer.slice } \varphi R \sigma$   
 ⟨proof⟩

## 6 Monitor implementation

### 6.1 Monitorable formulas

**definition**  $\text{mmonitorable } \varphi \longleftrightarrow \text{safe\_formula } \varphi \wedge \text{MFOTL.future\_reach } \varphi \neq \infty$

**fun**  $\text{mmonitorable\_exec} :: 'a \text{ MFOTL.formula} \Rightarrow \text{bool}$  **where**  
 $\text{mmonitorable\_exec } (\text{MFOTL.Eq } t1 \ t2) = (\text{MFOTL.is\_Const } t1 \vee \text{MFOTL.is\_Const } t2)$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Neg } (\text{MFOTL.Eq } (\text{MFOTL.Const } x) (\text{MFOTL.Const } y))) = \text{True}$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Neg } (\text{MFOTL.Eq } (\text{MFOTL.Var } x) (\text{MFOTL.Var } y))) = (x = y)$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Pred } e \text{ ts}) = \text{True}$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Neg } (\text{MFOTL.Or } (\text{MFOTL.Neg } \varphi) \psi)) = (\text{mmonitorable\_exec } \varphi \wedge$   
 $(\text{mmonitorable\_exec } \psi \wedge \text{MFOTL.fv } \psi \subseteq \text{MFOTL.fv } \varphi \vee (\text{case } \psi \text{ of MFOTL.Neg } \psi' \Rightarrow \text{mmonitorable\_exec } \psi' \mid \_ \Rightarrow \text{False})))$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Or } \varphi \psi) = (\text{MFOTL.fv } \psi = \text{MFOTL.fv } \varphi \wedge \text{mmonitorable\_exec } \varphi \wedge \text{mmonitorable\_exec } \psi)$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Exists } \varphi) = (\text{mmonitorable\_exec } \varphi)$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Prev } I \varphi) = (\text{mmonitorable\_exec } \varphi)$   
 $\text{mmonitorable\_exec } (\text{MFOTL.Next } I \varphi) = (\text{mmonitorable\_exec } \varphi \wedge \text{right } I \neq \infty)$

```

| mmonitorable_exec (MFOTL.Since  $\varphi$  I  $\psi$ ) = (MFOTL.fv  $\varphi$   $\subseteq$  MFOTL.fv  $\psi$   $\wedge$ 
  (mmonitorable_exec  $\varphi$   $\vee$  (case  $\varphi$  of MFOTL.Neg  $\varphi'$   $\Rightarrow$  mmonitorable_exec  $\varphi'$  |  $\_ \Rightarrow$  False))  $\wedge$ 
  mmonitorable_exec  $\psi$ )
| mmonitorable_exec (MFOTL.Until  $\varphi$  I  $\psi$ ) = (MFOTL.fv  $\varphi$   $\subseteq$  MFOTL.fv  $\psi$   $\wedge$  right I  $\neq$   $\infty$   $\wedge$ 
  (mmonitorable_exec  $\varphi$   $\vee$  (case  $\varphi$  of MFOTL.Neg  $\varphi'$   $\Rightarrow$  mmonitorable_exec  $\varphi'$  |  $\_ \Rightarrow$  False))  $\wedge$ 
  mmonitorable_exec  $\psi$ )
| mmonitorable_exec  $\_ =$  False

```

**lemma plus\_eq\_enat\_iff**:  $a + b = \text{enat } i \iff (\exists j k. a = \text{enat } j \wedge b = \text{enat } k \wedge j + k = i)$   
 <proof>

**lemma minus\_eq\_enat\_iff**:  $a - \text{enat } k = \text{enat } i \iff (\exists j. a = \text{enat } j \wedge j - k = i)$   
 <proof>

**lemma safe\_formula\_mmonitorable\_exec**:  $\text{safe\_formula } \varphi \implies \text{MFOTL.future\_reach } \varphi \neq \infty \implies \text{mmonitorable\_exec } \varphi$   
 <proof>

**lemma mmonitorable\_exec\_mmonitorable**:  $\text{mmonitorable\_exec } \varphi \implies \text{mmonitorable } \varphi$   
 <proof>

**lemma monitorable\_formula\_code**[code]:  $\text{mmonitorable } \varphi = \text{mmonitorable\_exec } \varphi$   
 <proof>

## 6.2 The executable monitor

**type\_synonym**  $ts = \text{nat}$

**type\_synonym**  $'a \text{ mbuf2} = 'a \text{ table list} \times 'a \text{ table list}$   
**type\_synonym**  $'a \text{ msaux} = (ts \times 'a \text{ table}) \text{ list}$   
**type\_synonym**  $'a \text{ muaux} = (ts \times 'a \text{ table} \times 'a \text{ table}) \text{ list}$

**datatype**  $'a \text{ mformula} =$   
 MRel  $'a \text{ table}$   
 | MPred MFOTL.name  $'a \text{ MFOTL.trm list}$   
 | MAnd  $'a \text{ mformula bool 'a mformula 'a mbuf2}$   
 | MOr  $'a \text{ mformula 'a mformula 'a mbuf2}$   
 | MExists  $'a \text{ mformula}$   
 | MPrev  $\mathcal{I} 'a \text{ mformula bool 'a table list ts list}$   
 | MNext  $\mathcal{I} 'a \text{ mformula bool ts list}$   
 | MSince  $\text{bool 'a mformula } \mathcal{I} 'a \text{ mformula 'a mbuf2 ts list 'a msaux}$   
 | MUntil  $\text{bool 'a mformula } \mathcal{I} 'a \text{ mformula 'a mbuf2 ts list 'a muaux}$

**record**  $'a \text{ mstate} =$   
 mstate\_i ::  $\text{nat}$   
 mstate\_m ::  $'a \text{ mformula}$   
 mstate\_n ::  $\text{nat}$

**fun**  $\text{eq\_rel} :: \text{nat} \Rightarrow 'a \text{ MFOTL.trm} \Rightarrow 'a \text{ MFOTL.trm} \Rightarrow 'a \text{ table}$  **where**  
 eq\_rel  $n$  (MFOTL.Const  $x$ ) (MFOTL.Const  $y$ ) = (if  $x = y$  then unit\_table  $n$  else empty\_table)  
 | eq\_rel  $n$  (MFOTL.Var  $x$ ) (MFOTL.Const  $y$ ) = singleton\_table  $n$   $x$   $y$   
 | eq\_rel  $n$  (MFOTL.Const  $x$ ) (MFOTL.Var  $y$ ) = singleton\_table  $n$   $y$   $x$   
 | eq\_rel  $n$  (MFOTL.Var  $x$ ) (MFOTL.Var  $y$ ) = undefined

**fun**  $\text{neq\_rel} :: \text{nat} \Rightarrow 'a \text{ MFOTL.trm} \Rightarrow 'a \text{ MFOTL.trm} \Rightarrow 'a \text{ table}$  **where**  
 neq\_rel  $n$  (MFOTL.Const  $x$ ) (MFOTL.Const  $y$ ) = (if  $x = y$  then empty\_table else unit\_table  $n$ )  
 | neq\_rel  $n$  (MFOTL.Var  $x$ ) (MFOTL.Var  $y$ ) = (if  $x = y$  then empty\_table else undefined)  
 | neq\_rel  $\_ \_ \_ =$  undefined

```

fun minit0 :: nat ⇒ 'a MFOTL.formula ⇒ 'a mformula where
  minit0 n (MFOTL.Neg φ) = (case φ of
    MFOTL.Eq t1 t2 ⇒ MRel (neq_rel n t1 t2)
  | MFOTL.Or (MFOTL.Neg φ) ψ ⇒ (if safe_formula ψ ∧ MFOTL.fv ψ ⊆ MFOTL.fv φ
    then MAnd (minit0 n φ) False (minit0 n ψ) ([], []))
    else (case ψ of MFOTL.Neg ψ ⇒ MAnd (minit0 n φ) True (minit0 n ψ) ([], []) | _ ⇒ undefined))
  | _ ⇒ undefined)
| minit0 n (MFOTL.Eq t1 t2) = MRel (eq_rel n t1 t2)
| minit0 n (MFOTL.Pred e ts) = MPred e ts
| minit0 n (MFOTL.Or φ ψ) = MOr (minit0 n φ) (minit0 n ψ) ([], [])
| minit0 n (MFOTL.Exists φ) = MExists (minit0 (Suc n) φ)
| minit0 n (MFOTL.Prev I φ) = MPrev I (minit0 n φ) True [] []
| minit0 n (MFOTL.Next I φ) = MNext I (minit0 n φ) True [] []
| minit0 n (MFOTL.Since φ I ψ) = (if safe_formula φ
  then MSince True (minit0 n φ) I (minit0 n ψ) ([], []) [] []
  else (case φ of
    MFOTL.Neg φ ⇒ MSince False (minit0 n φ) I (minit0 n ψ) ([], []) [] []
  | _ ⇒ undefined))
| minit0 n (MFOTL.Until φ I ψ) = (if safe_formula φ
  then MUntil True (minit0 n φ) I (minit0 n ψ) ([], []) [] []
  else (case φ of
    MFOTL.Neg φ ⇒ MUntil False (minit0 n φ) I (minit0 n ψ) ([], []) [] []
  | _ ⇒ undefined))

```

```

definition minit :: 'a MFOTL.formula ⇒ 'a mstate where
  minit φ = (let n = MFOTL.nfv φ in (mstate_i = 0, mstate_m = minit0 n φ, mstate_n = n))

```

```

fun mprev_next :: ℐ ⇒ 'a table list ⇒ ts list ⇒ 'a table list × 'a table list × ts list where
  mprev_next I [] ts = ([], [], ts)
| mprev_next I xs [] = ([], xs, [])
| mprev_next I xs [t] = ([], xs, [t])
| mprev_next I (x # xs) (t # t' # ts) = (let (ys, zs) = mprev_next I xs (t' # ts)
  in ((if mem (t' - t) I then x else empty_table) # ys, zs))

```

```

fun mbuf2_add :: 'a table list ⇒ 'a table list ⇒ 'a mbuf2 ⇒ 'a mbuf2 where
  mbuf2_add xs' ys' (xs, ys) = (xs @ xs', ys @ ys')

```

```

fun mbuf2_take :: ('a table ⇒ 'a table ⇒ 'b) ⇒ 'a mbuf2 ⇒ 'b list × 'a mbuf2 where
  mbuf2_take f (x # xs, y # ys) = (let (zs, buf) = mbuf2_take f (xs, ys) in (f x y # zs, buf))
| mbuf2_take f (xs, ys) = ([], (xs, ys))

```

```

fun mbuf2t_take :: ('a table ⇒ 'a table ⇒ ts ⇒ 'b ⇒ 'b) ⇒ 'b ⇒
  'a mbuf2 ⇒ ts list ⇒ 'b × 'a mbuf2 × ts list where
  mbuf2t_take f z (x # xs, y # ys) (t # ts) = mbuf2t_take f (f x y t z) (xs, ys) ts
| mbuf2t_take f z (xs, ys) ts = (z, (xs, ys), ts)

```

```

fun match :: 'a MFOTL.trm list ⇒ 'a list ⇒ (nat → 'a) option where
  match [] [] = Some Map.empty
| match (MFOTL.Const x # ts) (y # ys) = (if x = y then match ts ys else None)
| match (MFOTL.Var x # ts) (y # ys) = (case match ts ys of
  None ⇒ None
| Some f ⇒ (case f x of
  None ⇒ Some (f(x ↦ y))
  | Some z ⇒ if y = z then Some f else None))
| match _ _ = None

```

```

definition update_since :: ℐ ⇒ bool ⇒ 'a table ⇒ 'a table ⇒ ts ⇒

```

'a msaux  $\Rightarrow$  'a table  $\times$  'a msaux **where**  
 update\_since I pos rel1 rel2 nt aux =  
 (let aux = (case [(t, join rel pos rel1). (t, rel)  $\leftarrow$  aux, nt - t  $\leq$  right I] of  
 []  $\Rightarrow$  [(nt, rel2)]  
 | x # aux'  $\Rightarrow$  (if fst x = nt then (fst x, snd x  $\cup$  rel2) # aux' else (nt, rel2) # x # aux'))  
 in (foldr ( $\cup$ ) [rel. (t, rel)  $\leftarrow$  aux, left I  $\leq$  nt - t] {}, aux))

**definition** update\_until ::  $\mathcal{I} \Rightarrow \text{bool} \Rightarrow$  'a table  $\Rightarrow$  'a table  $\Rightarrow$  ts  $\Rightarrow$  'a muaux  $\Rightarrow$  'a muaux **where**  
 update\_until I pos rel1 rel2 nt aux =  
 (map ( $\lambda x$ . case x of (t, a1, a2)  $\Rightarrow$  (t, if pos then join a1 True rel1 else a1  $\cup$  rel1,  
 if mem (nt - t) I then a2  $\cup$  join rel2 pos a1 else a2)) aux) @  
 [(nt, rel1, if left I = 0 then rel2 else empty\_table)]

**fun** eval\_until ::  $\mathcal{I} \Rightarrow$  ts  $\Rightarrow$  'a muaux  $\Rightarrow$  'a table list  $\times$  'a muaux **where**  
 eval\_until I nt [] = ([], [])  
 | eval\_until I nt ((t, a1, a2) # aux) = (if t + right I < nt then  
 (let (xs, aux) = eval\_until I nt aux in (a2 # xs, aux)) else ([], (t, a1, a2) # aux))

**primrec** meval :: nat  $\Rightarrow$  ts  $\Rightarrow$  'a MFOTL.database  $\Rightarrow$  'a mformula  $\Rightarrow$  'a table list  $\times$  'a mformula **where**  
 meval n t db (MRel rel) = ([rel], MRel rel)  
 | meval n t db (MPred e ts) = (( $\lambda f$ . tabulate f 0 n) ' Option.these  
 (match ts ' ( $\bigcup$  (e', x)  $\in$  db. if e = e' then {x} else {})), MPred e ts)  
 | meval n t db (MAnd  $\varphi$  pos  $\psi$  buf) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ; (ys,  $\psi$ ) = meval n t db  $\psi$ ;  
 (zs, buf) = mbuf2\_take ( $\lambda r1 r2$ . join r1 pos r2) (mbuf2\_add xs ys buf)  
 in (zs, MAnd  $\varphi$  pos  $\psi$  buf))  
 | meval n t db (MOr  $\varphi$   $\psi$  buf) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ; (ys,  $\psi$ ) = meval n t db  $\psi$ ;  
 (zs, buf) = mbuf2\_take ( $\lambda r1 r2$ . r1  $\cup$  r2) (mbuf2\_add xs ys buf)  
 in (zs, MOr  $\varphi$   $\psi$  buf))  
 | meval n t db (MExists  $\varphi$ ) =  
 (let (xs,  $\varphi$ ) = meval (Suc n) t db  $\varphi$  in (map ( $\lambda r$ . tl ' r) xs, MExists  $\varphi$ ))  
 | meval n t db (MPrev I  $\varphi$  first buf nts) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ;  
 (zs, buf, nts) = mprev\_next I (buf @ xs) (nts @ [t])  
 in (if first then empty\_table # zs else zs, MPrev I  $\varphi$  False buf nts))  
 | meval n t db (MNext I  $\varphi$  first nts) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ;  
 (xs, first) = (case (xs, first) of (\_ # xs, True)  $\Rightarrow$  (xs, False) | a  $\Rightarrow$  a);  
 (zs, \_, nts) = mprev\_next I xs (nts @ [t])  
 in (zs, MNext I  $\varphi$  first nts))  
 | meval n t db (MSince pos  $\varphi$  I  $\psi$  buf nts aux) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ; (ys,  $\psi$ ) = meval n t db  $\psi$ ;  
 ((zs, aux), buf, nts) = mbuf2t\_take ( $\lambda r1 r2 t$  (zs, aux).  
 let (z, aux) = update\_since I pos r1 r2 t aux  
 in (zs @ [z], aux)) ([], aux) (mbuf2\_add xs ys buf) (nts @ [t])  
 in (zs, MSince pos  $\varphi$  I  $\psi$  buf nts aux))  
 | meval n t db (MUntil pos  $\varphi$  I  $\psi$  buf nts aux) =  
 (let (xs,  $\varphi$ ) = meval n t db  $\varphi$ ; (ys,  $\psi$ ) = meval n t db  $\psi$ ;  
 (aux, buf, nts) = mbuf2t\_take (update\_until I pos) aux (mbuf2\_add xs ys buf) (nts @ [t]);  
 (zs, aux) = eval\_until I (case nts of []  $\Rightarrow$  t | nt # \_  $\Rightarrow$  nt) aux  
 in (zs, MUntil pos  $\varphi$  I  $\psi$  buf nts aux))

**definition** mstep :: 'a MFOTL.database  $\times$  ts  $\Rightarrow$  'a mstate  $\Rightarrow$  (nat  $\times$  'a tuple) set  $\times$  'a mstate **where**  
 mstep tdb st =  
 (let (xs, m) = meval (mstate\_n st) (snd tdb) (fst tdb) (mstate\_m st)  
 in ( $\bigcup$  (set (map ( $\lambda(i, X)$ . ( $\lambda v$ . (i, v)) ' X) (List.enumerate (mstate\_i st) xs))),  
 (mstate\_i = mstate\_i st + length xs, mstate\_m = m, mstate\_n = mstate\_n st)))

**lemma** *mstep\_alt*:  $mstep\ tdb\ st =$   
 $(let\ (xs,\ m) = meval\ (mstate\_n\ st)\ (snd\ tdb)\ (fst\ tdb)\ (mstate\_m\ st)$   
 $in\ (\bigcup\ (i,\ X) \in set\ (List.enumerate\ (mstate\_i\ st)\ xs). \bigcup\ v \in X. \{(i,v)\},$   
 $(mstate\_i = mstate\_i\ st + length\ xs,\ mstate\_m = m,\ mstate\_n = mstate\_n\ st)))$   
 $\langle proof \rangle$

### 6.3 Progress

**primrec** *progress* :: 'a MFOTL.trace  $\Rightarrow$  'a MFOTL.formula  $\Rightarrow$  nat  $\Rightarrow$  nat **where**  
 $progress\ \sigma\ (MFOTL.Pred\ e\ ts)\ j = j$   
 $| progress\ \sigma\ (MFOTL.Eq\ t1\ t2)\ j = j$   
 $| progress\ \sigma\ (MFOTL.Neg\ \varphi)\ j = progress\ \sigma\ \varphi\ j$   
 $| progress\ \sigma\ (MFOTL.Or\ \varphi\ \psi)\ j = \min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)$   
 $| progress\ \sigma\ (MFOTL.Exists\ \varphi)\ j = progress\ \sigma\ \varphi\ j$   
 $| progress\ \sigma\ (MFOTL.Prev\ I\ \varphi)\ j = (if\ j = 0\ then\ 0\ else\ \min\ (Suc\ (progress\ \sigma\ \varphi\ j))\ j)$   
 $| progress\ \sigma\ (MFOTL.Next\ I\ \varphi)\ j = progress\ \sigma\ \varphi\ j - 1$   
 $| progress\ \sigma\ (MFOTL.Since\ \varphi\ I\ \psi)\ j = \min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)$   
 $| progress\ \sigma\ (MFOTL.Until\ \varphi\ I\ \psi)\ j =$   
 $Inf\ \{i.\ \forall k. k < j \wedge k \leq \min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j) \longrightarrow \tau\ \sigma\ i + right\ I \geq \tau\ \sigma\ k\}$

**lemma** *progress\_And[simp]*:  $progress\ \sigma\ (MFOTL.And\ \varphi\ \psi)\ j = \min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)$   
 $\langle proof \rangle$

**lemma** *progress\_And\_Not[simp]*:  $progress\ \sigma\ (MFOTL.And\_Not\ \varphi\ \psi)\ j = \min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)$   
 $\langle proof \rangle$

**lemma** *progress\_mono*:  $j \leq j' \implies progress\ \sigma\ \varphi\ j \leq progress\ \sigma\ \varphi\ j'$   
 $\langle proof \rangle$

**lemma** *progress\_le*:  $progress\ \sigma\ \varphi\ j \leq j$   
 $\langle proof \rangle$

**lemma** *progress\_0[simp]*:  $progress\ \sigma\ \varphi\ 0 = 0$   
 $\langle proof \rangle$

**lemma** *progress\_ge*:  $MFOTL.future\_reach\ \varphi \neq \infty \implies \exists j. i \leq progress\ \sigma\ \varphi\ j$   
 $\langle proof \rangle$

**lemma** *cInf\_restrict\_nat*:  
**fixes**  $x :: nat$   
**assumes**  $x \in A$   
**shows**  $Inf\ A = Inf\ \{y \in A. y \leq x\}$   
 $\langle proof \rangle$

**lemma** *progress\_time\_conv*:  
**assumes**  $\forall i < j. \tau\ \sigma\ i = \tau\ \sigma'\ i$   
**shows**  $progress\ \sigma\ \varphi\ j = progress\ \sigma'\ \varphi\ j$   
 $\langle proof \rangle$

**lemma** *Inf\_UNIV\_nat*:  $(Inf\ UNIV :: nat) = 0$   
 $\langle proof \rangle$

**lemma** *progress\_prefix\_conv*:  
**assumes** *prefix\_of*  $\pi\ \sigma$  **and** *prefix\_of*  $\pi\ \sigma'$   
**shows**  $progress\ \sigma\ \varphi\ (plen\ \pi) = progress\ \sigma'\ \varphi\ (plen\ \pi)$   
 $\langle proof \rangle$

**lemma** *sat\_prefix\_conv*:  
**assumes** *prefix\_of*  $\pi$   $\sigma$  **and** *prefix\_of*  $\pi$   $\sigma'$  **and**  $i < \text{progress } \sigma \ \varphi$  (*plen*  $\pi$ )  
**shows**  $\text{MFOTL.sat } \sigma \ v \ i \ \varphi \longleftrightarrow \text{MFOTL.sat } \sigma' \ v \ i \ \varphi$   
*<proof>*

## 6.4 Specification

**definition** *pprogress* ::  $'a \text{ MFOTL.formula} \Rightarrow 'a \text{ MFOTL.prefix} \Rightarrow \text{nat}$  **where**  
*pprogress*  $\varphi \ \pi = (\text{THE } n. \forall \sigma. \text{prefix\_of } \pi \ \sigma \longrightarrow \text{progress } \sigma \ \varphi \ (\text{plen } \pi) = n)$

**lemma** *pprogress\_eq*: *prefix\_of*  $\pi \ \sigma \Longrightarrow \text{pprogress } \varphi \ \pi = \text{progress } \sigma \ \varphi \ (\text{plen } \pi)$   
*<proof>*

**locale** *future\_bounded\_mfotl* =  
**fixes**  $\varphi :: 'a \text{ MFOTL.formula}$   
**assumes** *future\_bounded*:  $\text{MFOTL.future\_reach } \varphi \neq \infty$

**sublocale** *future\_bounded\_mfotl*  $\subseteq$  *sliceable\_timed\_progress*  $\text{MFOTL.nfv } \varphi \ \text{MFOTL.fv } \varphi \ \text{relevant\_events } \varphi$   
 $\lambda \sigma \ v \ i. \ \text{MFOTL.sat } \sigma \ v \ i \ \varphi \ \text{pprogress } \varphi$   
*<proof>*

**locale** *monitorable\_mfotl* =  
**fixes**  $\varphi :: 'a \text{ MFOTL.formula}$   
**assumes** *monitorable*: *mmonitorable*  $\varphi$

**sublocale** *monitorable\_mfotl*  $\subseteq$  *future\_bounded\_mfotl*  
*<proof>*

## 6.5 Correctness

### 6.5.1 Invariants

**definition** *wf\_mbuf2* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a \text{ table} \Rightarrow \text{bool}) \Rightarrow (\text{nat} \Rightarrow 'a \text{ table} \Rightarrow \text{bool}) \Rightarrow 'a \text{ mbuf2} \Rightarrow \text{bool}$  **where**  
*wf\_mbuf2*  $i \ j_a \ j_b \ P \ Q \ \text{buf} \longleftrightarrow i \leq j_a \wedge i \leq j_b \wedge (\text{case } \text{buf} \text{ of } (xs, ys) \Rightarrow \text{list\_all2 } P \ [i..<j_a] \ xs \wedge \text{list\_all2 } Q \ [i..<j_b] \ ys)$

**definition** *wf\_mbuf2'* ::  $'a \text{ MFOTL.trace} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list set} \Rightarrow 'a \text{ MFOTL.formula} \Rightarrow 'a \text{ MFOTL.formula} \Rightarrow 'a \text{ mbuf2} \Rightarrow \text{bool}$  **where**  
*wf\_mbuf2'*  $\sigma \ j \ n \ R \ \varphi \ \psi \ \text{buf} \longleftrightarrow \text{wf\_mbuf2} \ (\min \ (\text{progress } \sigma \ \varphi \ j) \ (\text{progress } \sigma \ \psi \ j)) \ (\text{progress } \sigma \ \varphi \ j) \ (\text{progress } \sigma \ \psi \ j)$   
 $(\lambda i. \ \text{qtable } n \ (\text{MFOTL.fv } \varphi) \ (\text{mem\_restr } R) \ (\lambda v. \ \text{MFOTL.sat } \sigma \ (\text{map the } v) \ i \ \varphi))$   
 $(\lambda i. \ \text{qtable } n \ (\text{MFOTL.fv } \psi) \ (\text{mem\_restr } R) \ (\lambda v. \ \text{MFOTL.sat } \sigma \ (\text{map the } v) \ i \ \psi)) \ \text{buf}$

**lemma** *wf\_mbuf2'\_UNIV\_alt*: *wf\_mbuf2'*  $\sigma \ j \ n \ \text{UNIV } \varphi \ \psi \ \text{buf} \longleftrightarrow (\text{case } \text{buf} \text{ of } (xs, ys) \Rightarrow \text{list\_all2} \ (\lambda i. \ \text{wf\_table } n \ (\text{MFOTL.fv } \varphi) \ (\lambda v. \ \text{MFOTL.sat } \sigma \ (\text{map the } v) \ i \ \varphi)) \ [\min \ (\text{progress } \sigma \ \varphi \ j) \ (\text{progress } \sigma \ \psi \ j)] \ ..< \ (\text{progress } \sigma \ \varphi \ j)] \ xs \wedge \text{list\_all2} \ (\lambda i. \ \text{wf\_table } n \ (\text{MFOTL.fv } \psi) \ (\lambda v. \ \text{MFOTL.sat } \sigma \ (\text{map the } v) \ i \ \psi)) \ [\min \ (\text{progress } \sigma \ \varphi \ j) \ (\text{progress } \sigma \ \psi \ j)] \ ..< \ (\text{progress } \sigma \ \psi \ j)] \ ys)$   
*<proof>*

**definition** *wf\_ts* ::  $'a \text{ MFOTL.trace} \Rightarrow \text{nat} \Rightarrow 'a \text{ MFOTL.formula} \Rightarrow 'a \text{ MFOTL.formula} \Rightarrow \text{ts list} \Rightarrow \text{bool}$  **where**  
*wf\_ts*  $\sigma \ j \ \varphi \ \psi \ ts \longleftrightarrow \text{list\_all2} \ (\lambda i \ t. \ t = \tau \ \sigma \ i) \ [\min \ (\text{progress } \sigma \ \varphi \ j) \ (\text{progress } \sigma \ \psi \ j)] \ ..< j] \ ts$

**abbreviation** *Since*  $\text{pos } \varphi \ I \ \psi \equiv \text{MFOTL.Since} \ (\text{if } \text{pos} \text{ then } \varphi \ \text{else } \text{MFOTL.Neg } \varphi) \ I \ \psi$

**definition**  $wf\_since\_aux :: 'a MFOTL.trace \Rightarrow nat \Rightarrow 'a list set \Rightarrow bool \Rightarrow 'a MFOTL.formula \Rightarrow \mathcal{I} \Rightarrow 'a MFOTL.formula \Rightarrow 'a msaux \Rightarrow nat \Rightarrow bool$  **where**  
 $wf\_since\_aux \sigma n R pos \varphi I \psi aux ne \longleftrightarrow sorted\_wrt (\lambda x y. fst x > fst y) aux \wedge$   
 $(\forall t X. (t, X) \in set aux \longrightarrow ne \neq 0 \wedge t \leq \tau \sigma (ne-1) \wedge \tau \sigma (ne-1) - t \leq right I \wedge (\exists i. \tau \sigma i = t) \wedge$   
 $qtable n (MFOTL.fv \psi) (mem\_restr R) (\lambda v. MFOTL.sat \sigma (map the v) (ne-1) (Sincep pos \varphi (point (\tau \sigma (ne-1) - t)) \psi)) X) \wedge$   
 $(\forall t. ne \neq 0 \wedge t \leq \tau \sigma (ne-1) \wedge \tau \sigma (ne-1) - t \leq right I \wedge (\exists i. \tau \sigma i = t) \longrightarrow$   
 $(\exists X. (t, X) \in set aux))$

**lemma**  $qtable\_mem\_restr\_UNIV: qtable n A (mem\_restr UNIV) Q X = wf\_table n A Q X$   
 $\langle proof \rangle$

**lemma**  $wf\_since\_aux\_UNIV\_alt:$

$wf\_since\_aux \sigma n UNIV pos \varphi I \psi aux ne \longleftrightarrow sorted\_wrt (\lambda x y. fst x > fst y) aux \wedge$   
 $(\forall t X. (t, X) \in set aux \longrightarrow ne \neq 0 \wedge t \leq \tau \sigma (ne-1) \wedge \tau \sigma (ne-1) - t \leq right I \wedge (\exists i. \tau \sigma i = t) \wedge$   
 $wf\_table n (MFOTL.fv \psi)$   
 $(\lambda v. MFOTL.sat \sigma (map the v) (ne-1) (Sincep pos \varphi (point (\tau \sigma (ne-1) - t)) \psi)) X) \wedge$   
 $(\forall t. ne \neq 0 \wedge t \leq \tau \sigma (ne-1) \wedge \tau \sigma (ne-1) - t \leq right I \wedge (\exists i. \tau \sigma i = t) \longrightarrow$   
 $(\exists X. (t, X) \in set aux))$   
 $\langle proof \rangle$

**definition**  $wf\_until\_aux :: 'a MFOTL.trace \Rightarrow nat \Rightarrow 'a list set \Rightarrow bool \Rightarrow$

$'a MFOTL.formula \Rightarrow \mathcal{I} \Rightarrow 'a MFOTL.formula \Rightarrow 'a muaux \Rightarrow nat \Rightarrow bool$  **where**  
 $wf\_until\_aux \sigma n R pos \varphi I \psi aux ne \longleftrightarrow list\_all2 (\lambda x i. case x of (t, r1, r2) \Rightarrow t = \tau \sigma i \wedge$   
 $qtable n (MFOTL.fv \varphi) (mem\_restr R) (\lambda v. if pos then (\forall k \in \{i..<ne+length aux\}. MFOTL.sat \sigma$   
 $(map the v) k \varphi)$   
 $else (\exists k \in \{i..<ne+length aux\}. MFOTL.sat \sigma (map the v) k \varphi)) r1 \wedge$   
 $qtable n (MFOTL.fv \psi) (mem\_restr R) (\lambda v. (\exists j. i \leq j \wedge j < ne + length aux \wedge mem (\tau \sigma j - \tau \sigma$   
 $i) I \wedge$   
 $MFOTL.sat \sigma (map the v) j \psi \wedge$   
 $(\forall k \in \{i..<j\}. if pos then MFOTL.sat \sigma (map the v) k \varphi else \neg MFOTL.sat \sigma (map the v) k \varphi)))$   
 $r2)$   
 $aux [ne..<ne+length aux]$

**lemma**  $wf\_until\_aux\_UNIV\_alt:$

$wf\_until\_aux \sigma n UNIV pos \varphi I \psi aux ne \longleftrightarrow list\_all2 (\lambda x i. case x of (t, r1, r2) \Rightarrow t = \tau \sigma i \wedge$   
 $wf\_table n (MFOTL.fv \varphi) (\lambda v. if pos$   
 $then (\forall k \in \{i..<ne+length aux\}. MFOTL.sat \sigma (map the v) k \varphi)$   
 $else (\exists k \in \{i..<ne+length aux\}. MFOTL.sat \sigma (map the v) k \varphi)) r1 \wedge$   
 $wf\_table n (MFOTL.fv \psi) (\lambda v. \exists j. i \leq j \wedge j < ne + length aux \wedge mem (\tau \sigma j - \tau \sigma i) I \wedge$   
 $MFOTL.sat \sigma (map the v) j \psi \wedge$   
 $(\forall k \in \{i..<j\}. if pos then MFOTL.sat \sigma (map the v) k \varphi else \neg MFOTL.sat \sigma (map the v) k \varphi)))$   
 $r2)$   
 $aux [ne..<ne+length aux]$   
 $\langle proof \rangle$

**inductive**  $wf\_mformula :: 'a MFOTL.trace \Rightarrow nat \Rightarrow$

$nat \Rightarrow 'a list set \Rightarrow 'a mformula \Rightarrow 'a MFOTL.formula \Rightarrow bool$

**for**  $\sigma j$  **where**

$Eq: MFOTL.is\_Const t1 \vee MFOTL.is\_Const t2 \Longrightarrow$

$\forall x \in MFOTL.fv\_trm t1. x < n \Longrightarrow \forall x \in MFOTL.fv\_trm t2. x < n \Longrightarrow$

$wf\_mformula \sigma j n R (MRel (eq\_rel n t1 t2)) (MFOTL.Eq t1 t2)$

$| neq\_Const: \varphi = MRel (neq\_rel n (MFOTL.Const x) (MFOTL.Const y)) \Longrightarrow$

$wf\_mformula \sigma j n R \varphi (MFOTL.Neg (MFOTL.Eq (MFOTL.Const x) (MFOTL.Const y)))$

$| neq\_Var: x < n \Longrightarrow$

$wf\_mformula \sigma j n R (MRel \text{ empty\_table}) (MFOTL.Neg (MFOTL.Eq (MFOTL.Var x) (MFOTL.Var x)))$   
 $| \text{ Pred: } \forall x \in MFOTL.fv (MFOTL.Pred e ts). x < n \implies$   
 $wf\_mformula \sigma j n R (MPred e ts) (MFOTL.Pred e ts)$   
 $| \text{ And: } wf\_mformula \sigma j n R \varphi \varphi' \implies wf\_mformula \sigma j n R \psi \psi' \implies$   
 $\text{ if pos then } \chi = MFOTL.And \varphi' \psi' \wedge \neg (\text{safe\_formula } (MFOTL.Neg \psi') \wedge MFOTL.fv \psi' \subseteq MFOTL.fv \varphi')$   
 $\varphi')$   
 $\text{ else } \chi = MFOTL.And\_Not \varphi' \psi' \wedge \text{safe\_formula } \psi' \wedge MFOTL.fv \psi' \subseteq MFOTL.fv \varphi' \implies$   
 $wf\_mbuf2' \sigma j n R \varphi' \psi' \text{ buf} \implies$   
 $wf\_mformula \sigma j n R (MAnd \varphi \text{ pos } \psi \text{ buf}) \chi$   
 $| \text{ Or: } wf\_mformula \sigma j n R \varphi \varphi' \implies wf\_mformula \sigma j n R \psi \psi' \implies$   
 $MFOTL.fv \varphi' = MFOTL.fv \psi' \implies$   
 $wf\_mbuf2' \sigma j n R \varphi' \psi' \text{ buf} \implies$   
 $wf\_mformula \sigma j n R (MOr \varphi \psi \text{ buf}) (MFOTL.Or \varphi' \psi')$   
 $| \text{ Exists: } wf\_mformula \sigma j (Suc n) (\text{lift\_envs } R) \varphi \varphi' \implies$   
 $wf\_mformula \sigma j n R (MExists \varphi) (MFOTL.Exists \varphi')$   
 $| \text{ Prev: } wf\_mformula \sigma j n R \varphi \varphi' \implies$   
 $\text{first} \longleftrightarrow j = 0 \implies$   
 $\text{list\_all2 } (\lambda i. \text{qtable } n (MFOTL.fv \varphi') (\text{mem\_restr } R) (\lambda v. MFOTL.sat \sigma (\text{map the } v) i \varphi'))$   
 $[\text{min } (\text{progress } \sigma \varphi' j) (j-1)..<\text{progress } \sigma \varphi' j] \text{ buf} \implies$   
 $\text{list\_all2 } (\lambda i t. t = \tau \sigma i) [\text{min } (\text{progress } \sigma \varphi' j) (j-1)..<j] \text{ nts} \implies$   
 $wf\_mformula \sigma j n R (MPrev I \varphi \text{ first buf nts}) (MFOTL.Prev I \varphi')$   
 $| \text{ Next: } wf\_mformula \sigma j n R \varphi \varphi' \implies$   
 $\text{first} \longleftrightarrow \text{progress } \sigma \varphi' j = 0 \implies$   
 $\text{list\_all2 } (\lambda i t. t = \tau \sigma i) [\text{progress } \sigma \varphi' j - 1..<j] \text{ nts} \implies$   
 $wf\_mformula \sigma j n R (MNext I \varphi \text{ first nts}) (MFOTL.Next I \varphi')$   
 $| \text{ Since: } wf\_mformula \sigma j n R \varphi \varphi' \implies wf\_mformula \sigma j n R \psi \psi' \implies$   
 $\text{if pos then } \varphi'' = \varphi' \text{ else } \varphi'' = MFOTL.Neg \varphi' \implies$   
 $\text{safe\_formula } \varphi'' = \text{pos} \implies$   
 $MFOTL.fv \varphi' \subseteq MFOTL.fv \psi' \implies$   
 $wf\_mbuf2' \sigma j n R \varphi' \psi' \text{ buf} \implies$   
 $wf\_ts \sigma j \varphi' \psi' \text{ nts} \implies$   
 $wf\_since\_aux \sigma n R \text{pos } \varphi' I \psi' \text{ aux } (\text{progress } \sigma (MFOTL.Since \varphi'' I \psi') j) \implies$   
 $wf\_mformula \sigma j n R (MSince \text{pos } \varphi I \psi \text{ buf nts aux}) (MFOTL.Since \varphi'' I \psi')$   
 $| \text{ Until: } wf\_mformula \sigma j n R \varphi \varphi' \implies wf\_mformula \sigma j n R \psi \psi' \implies$   
 $\text{if pos then } \varphi'' = \varphi' \text{ else } \varphi'' = MFOTL.Neg \varphi' \implies$   
 $\text{safe\_formula } \varphi'' = \text{pos} \implies$   
 $MFOTL.fv \varphi' \subseteq MFOTL.fv \psi' \implies$   
 $wf\_mbuf2' \sigma j n R \varphi' \psi' \text{ buf} \implies$   
 $wf\_ts \sigma j \varphi' \psi' \text{ nts} \implies$   
 $wf\_until\_aux \sigma n R \text{pos } \varphi' I \psi' \text{ aux } (\text{progress } \sigma (MFOTL.Until \varphi'' I \psi') j) \implies$   
 $\text{progress } \sigma (MFOTL.Until \varphi'' I \psi') j + \text{length aux} = \text{min } (\text{progress } \sigma \varphi' j) (\text{progress } \sigma \psi' j) \implies$   
 $wf\_mformula \sigma j n R (MUntil \text{pos } \varphi I \psi \text{ buf nts aux}) (MFOTL.Until \varphi'' I \psi')$

**definition**  $wf\_mstate :: 'a MFOTL.formula \Rightarrow 'a MFOTL.prefix \Rightarrow 'a \text{ list set} \Rightarrow 'a \text{ mstate} \Rightarrow \text{bool where}$   
 $wf\_mstate \varphi \pi R st \longleftrightarrow \text{mstate\_n } st = MFOTL.nfv \varphi \wedge (\forall \sigma. \text{prefix\_of } \pi \sigma \longrightarrow$   
 $\text{mstate\_i } st = \text{progress } \sigma \varphi (\text{plen } \pi) \wedge$   
 $wf\_mformula \sigma (\text{plen } \pi) (\text{mstate\_n } st) R (\text{mstate\_m } st) \varphi)$

### 6.5.2 Initialisation

**lemma**  $\text{minit0\_And: } \neg (\text{safe\_formula } (MFOTL.Neg \psi) \wedge MFOTL.fv \psi \subseteq MFOTL.fv \varphi) \implies$   
 $\text{minit0 } n (MFOTL.And \varphi \psi) = MAnd (\text{minit0 } n \varphi) \text{ True } (\text{minit0 } n \psi) ([], [])$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{minit0\_And\_Not: } \text{safe\_formula } \psi \wedge MFOTL.fv \psi \subseteq MFOTL.fv \varphi \implies$   
 $\text{minit0 } n (MFOTL.And\_Not \varphi \psi) = (MAnd (\text{minit0 } n \varphi) \text{ False } (\text{minit0 } n \psi) ([], []))$   
 $\langle \text{proof} \rangle$

**lemma** *wf\_mbuf2'\_0*:  $wf\_mbuf2' \sigma 0 n R \varphi \psi (\ [], [] )$   
 ⟨proof⟩

**lemma** *wf\_ts\_0*:  $wf\_ts \sigma 0 \varphi \psi []$   
 ⟨proof⟩

**lemma** *wf\_since\_aux\_Nil*:  $wf\_since\_aux \sigma n R pos \varphi' I \psi' [] 0$   
 ⟨proof⟩

**lemma** *wf\_until\_aux\_Nil*:  $wf\_until\_aux \sigma n R pos \varphi' I \psi' [] 0$   
 ⟨proof⟩

**lemma** *wf\_minit0*:  $safe\_formula \varphi \implies \forall x \in MFOTL.fv \varphi. x < n \implies$   
 $wf\_mformula \sigma 0 n R (minit0 n \varphi) \varphi$   
 ⟨proof⟩

**lemma** *wf\_mstate\_minit*:  $safe\_formula \varphi \implies wf\_mstate \varphi pnil R (minit \varphi)$   
 ⟨proof⟩

### 6.5.3 Evaluation

**lemma** *match\_wf\_tuple*:  $Some f = match\ ts\ xs \implies wf\_tuple\ n (\bigcup_{t \in set\ ts} MFOTL.fv\_trm\ t)$  (tabulate  
 $f\ 0\ n$ )  
 ⟨proof⟩

**lemma** *match\_fvi\_trm\_None*:  $Some f = match\ ts\ xs \implies \forall t \in set\ ts. x \notin MFOTL.fv\_trm\ t \implies f\ x =$   
 $None$   
 ⟨proof⟩

**lemma** *match\_fvi\_trm\_Some*:  $Some f = match\ ts\ xs \implies t \in set\ ts \implies x \in MFOTL.fv\_trm\ t \implies f\ x =$   
 $\neq None$   
 ⟨proof⟩

**lemma** *match\_eval\_trm*:  $\forall t \in set\ ts. \forall i \in MFOTL.fv\_trm\ t. i < n \implies Some f = match\ ts\ xs \implies$   
 $map\ (MFOTL.eval\_trm\ (tabulate\ (\lambda i. the\ (f\ i))\ 0\ n))\ ts = xs$   
 ⟨proof⟩

**lemma** *wf\_tuple\_tabulate\_Some*:  $wf\_tuple\ n\ A\ (tabulate\ f\ 0\ n) \implies x \in A \implies x < n \implies \exists y. f\ x =$   
 $Some\ y$   
 ⟨proof⟩

**lemma** *ex\_match*:  $wf\_tuple\ n (\bigcup_{t \in set\ ts} MFOTL.fv\_trm\ t)\ v \implies \forall t \in set\ ts. \forall x \in MFOTL.fv\_trm\ t. x$   
 $< n \implies$   
 $\exists f. match\ ts\ (map\ (MFOTL.eval\_trm\ (map\ the\ v))\ ts) = Some\ f \wedge v = tabulate\ f\ 0\ n$   
 ⟨proof⟩

**lemma** *eq\_rel\_eval\_trm*:  $v \in eq\_rel\ n\ t1\ t2 \implies MFOTL.is\_Const\ t1 \vee MFOTL.is\_Const\ t2 \implies$   
 $\forall x \in MFOTL.fv\_trm\ t1 \cup MFOTL.fv\_trm\ t2. x < n \implies$   
 $MFOTL.eval\_trm\ (map\ the\ v)\ t1 = MFOTL.eval\_trm\ (map\ the\ v)\ t2$   
 ⟨proof⟩

**lemma** *in\_eq\_rel*:  $wf\_tuple\ n\ (MFOTL.fv\_trm\ t1 \cup MFOTL.fv\_trm\ t2)\ v \implies$   
 $MFOTL.is\_Const\ t1 \vee MFOTL.is\_Const\ t2 \implies$   
 $MFOTL.eval\_trm\ (map\ the\ v)\ t1 = MFOTL.eval\_trm\ (map\ the\ v)\ t2 \implies$   
 $v \in eq\_rel\ n\ t1\ t2$   
 ⟨proof⟩

**lemma** *table\_eq\_rel*:  $MFOTL.is\_Const\ t1 \vee MFOTL.is\_Const\ t2 \implies$   
 $table\ n\ (MFOTL.fv\_trm\ t1 \cup MFOTL.fv\_trm\ t2)\ (eq\_rel\ n\ t1\ t2)$

*<proof>*

**lemma** *wf\_tuple\_Suc\_fviD*:  $wf\_tuple (Suc\ n) (MFOTL.fvi\ b\ \varphi)\ v \implies wf\_tuple\ n (MFOTL.fvi (Suc\ b)\ \varphi) (tl\ v)$

*<proof>*

**lemma** *table\_fvi\_tl*:  $table (Suc\ n) (MFOTL.fvi\ b\ \varphi)\ X \implies table\ n (MFOTL.fvi (Suc\ b)\ \varphi) (tl\ 'X)$

*<proof>*

**lemma** *wf\_tuple\_Suc\_fvi\_SomeI*:  $0 \in MFOTL.fvi\ b\ \varphi \implies wf\_tuple\ n (MFOTL.fvi (Suc\ b)\ \varphi)\ v \implies wf\_tuple (Suc\ n) (MFOTL.fvi\ b\ \varphi) (Some\ x\ \# v)$

*<proof>*

**lemma** *wf\_tuple\_Suc\_fvi\_NoneI*:  $0 \notin MFOTL.fvi\ b\ \varphi \implies wf\_tuple\ n (MFOTL.fvi (Suc\ b)\ \varphi)\ v \implies wf\_tuple (Suc\ n) (MFOTL.fvi\ b\ \varphi) (None\ \# v)$

*<proof>*

**lemma** *qtable\_project\_fv*:  $qtable (Suc\ n) (fv\ \varphi) (mem\_restr (lift\_envs\ R))\ P\ X \implies$

$qtable\ n (MFOTL.fvi (Suc\ 0)\ \varphi) (mem\_restr\ R)$

$(\lambda v. \exists x. P ((if\ 0 \in fv\ \varphi\ then\ Some\ x\ else\ None)\ \# v)) (tl\ 'X)$

*<proof>*

**lemma** *mprev*:  $mprev\_next\ I\ xs\ ts = (ys,\ xs',\ ts') \implies$

$list\_all2\ P\ [i..<j]\ xs \implies list\_all2 (\lambda i\ t. t = \tau\ \sigma\ i)\ [i..<j]\ ts \implies i \leq j' \implies i < j \implies$

$list\_all2 (\lambda i\ X. if\ mem (\tau\ \sigma (Suc\ i) - \tau\ \sigma\ i)\ I\ then\ P\ i\ X\ else\ X = empty\_table)$

$[i..<min\ j'\ (j-1)]\ ys \wedge$

$list\_all2\ P\ [min\ j'\ (j-1)..<j']\ xs' \wedge$

$list\_all2 (\lambda i\ t. t = \tau\ \sigma\ i)\ [min\ j'\ (j-1)..<j]\ ts'$

*<proof>*

**lemma** *mnext*:  $mprev\_next\ I\ xs\ ts = (ys,\ xs',\ ts') \implies$

$list\_all2\ P\ [Suc\ i..<j']\ xs \implies list\_all2 (\lambda i\ t. t = \tau\ \sigma\ i)\ [i..<j]\ ts \implies Suc\ i \leq j' \implies i < j \implies$

$list\_all2 (\lambda i\ X. if\ mem (\tau\ \sigma (Suc\ i) - \tau\ \sigma\ i)\ I\ then\ P (Suc\ i)\ X\ else\ X = empty\_table)$

$[i..<min (j'-1) (j-1)]\ ys \wedge$

$list\_all2\ P [Suc (min (j'-1) (j-1))..<j']\ xs' \wedge$

$list\_all2 (\lambda i\ t. t = \tau\ \sigma\ i)\ [min (j'-1) (j-1)..<j]\ ts'$

*<proof>*

**lemma** *in\_foldr\_UnI*:  $x \in A \implies A \in set\ xs \implies x \in foldr (\cup)\ xs\ \{\}$

*<proof>*

**lemma** *in\_foldr\_UnE*:  $x \in foldr (\cup)\ xs\ \{\} \implies (\bigwedge A. A \in set\ xs \implies x \in A \implies P) \implies P$

*<proof>*

**lemma** *sat\_the\_restrict*:  $fv\ \varphi \subseteq A \implies MFOTL.sat\ \sigma (map\ the (restrict\ A\ v))\ i\ \varphi = MFOTL.sat\ \sigma (map\ the\ v)\ i\ \varphi$

*<proof>*

**lemma** *update\_since*:

**assumes** *pre*:  $wf\_since\_aux\ \sigma\ n\ R\ pos\ \varphi\ I\ \psi\ aux\ ne$

**and** *qtable1*:  $qtable\ n (MFOTL.fv\ \varphi) (mem\_restr\ R) (\lambda v. MFOTL.sat\ \sigma (map\ the\ v)\ ne\ \varphi)\ rel1$

**and** *qtable2*:  $qtable\ n (MFOTL.fv\ \psi) (mem\_restr\ R) (\lambda v. MFOTL.sat\ \sigma (map\ the\ v)\ ne\ \psi)\ rel2$

**and** *result\_eq*:  $(rel,\ aux') = update\_since\ I\ pos\ rel1\ rel2 (\tau\ \sigma\ ne)\ aux$

**and** *fvi\_subset*:  $MFOTL.fv\ \varphi \subseteq MFOTL.fv\ \psi$

**shows**  $wf\_since\_aux\ \sigma\ n\ R\ pos\ \varphi\ I\ \psi\ aux' (Suc\ ne)$

**and**  $qtable\ n (MFOTL.fv\ \psi) (mem\_restr\ R) (\lambda v. MFOTL.sat\ \sigma (map\ the\ v)\ ne (Sincep\ pos\ \varphi\ I\ \psi))$

*rel*

*<proof>*

**lemma** *length\_update\_until*:  $\text{length } (\text{update\_until } \text{pos } I \text{ rel1 } \text{rel2 } \text{nt } \text{aux}) = \text{Suc } (\text{length } \text{aux})$   
 ⟨proof⟩

**lemma** *wf\_update\_until*:

**assumes** *pre*:  $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ \text{aux } \text{ne}$   
**and** *qtable1*:  $\text{qtable } n \ (\text{MFOTL.fv } \varphi) \ (\text{mem\_restr } R) \ (\lambda v. \text{MFOTL.sat } \sigma \ (\text{map the } v) \ (\text{ne} + \text{length } \text{aux}) \ \varphi) \ \text{rel1}$   
**and** *qtable2*:  $\text{qtable } n \ (\text{MFOTL.fv } \psi) \ (\text{mem\_restr } R) \ (\lambda v. \text{MFOTL.sat } \sigma \ (\text{map the } v) \ (\text{ne} + \text{length } \text{aux}) \ \psi) \ \text{rel2}$   
**and** *fv\_subset*:  $\text{MFOTL.fv } \varphi \subseteq \text{MFOTL.fv } \psi$   
**shows**  $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ (\text{update\_until } I \ \text{pos } \text{rel1 } \text{rel2 } (\tau \ \sigma \ (\text{ne} + \text{length } \text{aux})) \ \text{aux}) \ \text{ne}$   
 ⟨proof⟩

**lemma** *wf\_until\_aux\_Cons*:  $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ (a \ \# \ \text{aux}) \ \text{ne} \implies$   
 $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ \text{aux} \ (\text{Suc } \text{ne})$   
 ⟨proof⟩

**lemma** *wf\_until\_aux\_Cons1*:  $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ ((t, a1, a2) \ \# \ \text{aux}) \ \text{ne} \implies t = \tau \ \sigma \ \text{ne}$   
 ⟨proof⟩

**lemma** *wf\_until\_aux\_Cons3*:  $\text{wf\_until\_aux } \sigma \ n \ R \ \text{pos } \varphi \ I \ \psi \ ((t, a1, a2) \ \# \ \text{aux}) \ \text{ne} \implies$   
 $\text{qtable } n \ (\text{MFOTL.fv } \psi) \ (\text{mem\_restr } R) \ (\lambda v. (\exists j. \text{ne} \leq j \wedge j < \text{Suc } (\text{ne} + \text{length } \text{aux}) \wedge \text{mem } (\tau \ \sigma \ j - \tau \ \sigma \ \text{ne}) \ I \wedge$   
 $\text{MFOTL.sat } \sigma \ (\text{map the } v) \ j \ \psi \wedge (\forall k \in \{\text{ne}..<j\}. \text{if } \text{pos } \text{ then } \text{MFOTL.sat } \sigma \ (\text{map the } v) \ k \ \varphi \ \text{else } \neg \text{MFOTL.sat } \sigma \ (\text{map the } v) \ k \ \varphi))) \ a2$   
 ⟨proof⟩

**lemma** *upt\_append*:  $a \leq b \implies b \leq c \implies [a..<b] \ @ \ [b..<c] = [a..<c]$   
 ⟨proof⟩

**lemma** *wf\_mbuf2\_add*:

**assumes**  $\text{wf\_mbuf2 } i \ \text{ja } \text{jb } P \ Q \ \text{buf}$   
**and**  $\text{list\_all2 } P \ [ja..<ja'] \ \text{xs}$   
**and**  $\text{list\_all2 } Q \ [jb..<jb'] \ \text{ys}$   
**and**  $\text{ja} \leq \text{ja}' \ \text{jb} \leq \text{jb}'$   
**shows**  $\text{wf\_mbuf2 } i \ \text{ja}' \ \text{jb}' \ P \ Q \ (\text{mbuf2\_add } \text{xs } \text{ys } \text{buf})$   
 ⟨proof⟩

**lemma** *mbuf2\_take\_eqD*:

**assumes**  $\text{mbuf2\_take } f \ \text{buf} = (\text{xs}, \text{buf}')$   
**and**  $\text{wf\_mbuf2 } i \ \text{ja } \text{jb } P \ Q \ \text{buf}$   
**shows**  $\text{wf\_mbuf2 } (\text{min } \text{ja } \text{jb}) \ \text{ja } \text{jb } P \ Q \ \text{buf}'$   
**and**  $\text{list\_all2 } (\lambda i \ z. \exists x \ y. P \ i \ x \wedge Q \ i \ y \wedge z = f \ x \ y) \ [i..<\text{min } \text{ja } \text{jb}] \ \text{xs}$   
 ⟨proof⟩

**lemma** *mbuf2t\_take\_eqD*:

**assumes**  $\text{mbuf2t\_take } f \ z \ \text{buf } \text{nts} = (z', \text{buf}', \text{nts}')$   
**and**  $\text{wf\_mbuf2 } i \ \text{ja } \text{jb } P \ Q \ \text{buf}$   
**and**  $\text{list\_all2 } R \ [i..<j] \ \text{nts}$   
**and**  $\text{ja} \leq j \ \text{jb} \leq j$   
**shows**  $\text{wf\_mbuf2 } (\text{min } \text{ja } \text{jb}) \ \text{ja } \text{jb } P \ Q \ \text{buf}'$   
**and**  $\text{list\_all2 } R \ [\text{min } \text{ja } \text{jb}..<j] \ \text{nts}'$   
 ⟨proof⟩

**lemma** *mbuf2t\_take\_induct[consumes 5, case\_names base step]*:

**assumes**  $\text{mbuf2t\_take } f \ z \ \text{buf } \text{nts} = (z', \text{buf}', \text{nts}')$   
**and**  $\text{wf\_mbuf2 } i \ \text{ja } \text{jb } P \ Q \ \text{buf}$

**and**  $list\_all2\ R\ [i..<j]\ nts$   
**and**  $ja \leq j\ jb \leq j$   
**and**  $U\ i\ z$   
**and**  $\bigwedge k\ x\ y\ t\ z.\ i \leq k \implies Suc\ k \leq ja \implies Suc\ k \leq jb \implies$   
 $P\ k\ x \implies Q\ k\ y \implies R\ k\ t \implies U\ k\ z \implies U\ (Suc\ k)\ (f\ x\ y\ t\ z)$   
**shows**  $U\ (min\ ja\ jb)\ z'$   
 $\langle proof \rangle$

**lemma**  $mbuf2\_take\_add'$ :

**assumes**  $eq: mbuf2\_take\ f\ (mbuf2\_add\ xs\ ys\ buf) = (zs,\ buf')$   
**and**  $pre: wf\_mbuf2'\ \sigma\ j\ n\ R\ \varphi\ \psi\ buf$   
**and**  $xs: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \varphi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \varphi))$   
 $[progress\ \sigma\ \varphi\ j..<progress\ \sigma\ \varphi\ j']\ xs$   
**and**  $ys: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \psi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \psi))$   
 $[progress\ \sigma\ \psi\ j..<progress\ \sigma\ \psi\ j']\ ys$   
**and**  $j \leq j'$   
**shows**  $wf\_mbuf2'\ \sigma\ j'\ n\ R\ \varphi\ \psi\ buf'$   
**and**  $list\_all2\ (\lambda i.\ Z.\ \exists X\ Y.$   
 $qtable\ n\ (MFOTL.fv\ \varphi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \varphi)\ X \wedge$   
 $qtable\ n\ (MFOTL.fv\ \psi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \psi)\ Y \wedge$   
 $Z = f\ X\ Y)$   
 $[min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)..<min\ (progress\ \sigma\ \varphi\ j')\ (progress\ \sigma\ \psi\ j')]\ zs$   
 $\langle proof \rangle$

**lemma**  $mbuf2t\_take\_add'$ :

**assumes**  $eq: mbuf2t\_take\ f\ z\ (mbuf2\_add\ xs\ ys\ buf)\ nts = (z',\ buf',\ nts')$   
**and**  $pre\_buf: wf\_mbuf2'\ \sigma\ j\ n\ R\ \varphi\ \psi\ buf$   
**and**  $pre\_nts: list\_all2\ (\lambda i\ t.\ t = \tau\ \sigma\ i)\ [min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)..<j']\ nts$   
**and**  $xs: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \varphi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \varphi))$   
 $[progress\ \sigma\ \varphi\ j..<progress\ \sigma\ \varphi\ j']\ xs$   
**and**  $ys: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \psi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \psi))$   
 $[progress\ \sigma\ \psi\ j..<progress\ \sigma\ \psi\ j']\ ys$   
**and**  $j \leq j'$   
**shows**  $wf\_mbuf2'\ \sigma\ j'\ n\ R\ \varphi\ \psi\ buf'$   
**and**  $wf\_ts\ \sigma\ j'\ \varphi\ \psi\ nts'$   
 $\langle proof \rangle$

**lemma**  $mbuf2t\_take\_add\_induct'$ [consumes 6, case\_names base step]:

**assumes**  $eq: mbuf2t\_take\ f\ z\ (mbuf2\_add\ xs\ ys\ buf)\ nts = (z',\ buf',\ nts')$   
**and**  $pre\_buf: wf\_mbuf2'\ \sigma\ j\ n\ R\ \varphi\ \psi\ buf$   
**and**  $pre\_nts: list\_all2\ (\lambda i\ t.\ t = \tau\ \sigma\ i)\ [min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)..<j']\ nts$   
**and**  $xs: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \varphi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \varphi))$   
 $[progress\ \sigma\ \varphi\ j..<progress\ \sigma\ \varphi\ j']\ xs$   
**and**  $ys: list\_all2\ (\lambda i.\ qtable\ n\ (MFOTL.fv\ \psi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ i\ \psi))$   
 $[progress\ \sigma\ \psi\ j..<progress\ \sigma\ \psi\ j']\ ys$   
**and**  $j \leq j'$   
**and**  $base: U\ (min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j))\ z$   
**and**  $step: \bigwedge k\ X\ Y\ z.\ min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j) \leq k \implies$   
 $Suc\ k \leq progress\ \sigma\ \varphi\ j' \implies Suc\ k \leq progress\ \sigma\ \psi\ j' \implies$   
 $qtable\ n\ (MFOTL.fv\ \varphi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ k\ \varphi)\ X \implies$   
 $qtable\ n\ (MFOTL.fv\ \psi)\ (mem\_restr\ R)\ (\lambda v.\ MFOTL.sat\ \sigma\ (map\ the\ v)\ k\ \psi)\ Y \implies$   
 $U\ k\ z \implies U\ (Suc\ k)\ (f\ X\ Y\ (\tau\ \sigma\ k)\ z)$   
**shows**  $U\ (min\ (progress\ \sigma\ \varphi\ j')\ (progress\ \sigma\ \psi\ j'))\ z'$   
 $\langle proof \rangle$

**lemma**  $progress\_Until\_le: progress\ \sigma\ (formula.Until\ \varphi\ I\ \psi)\ j \leq min\ (progress\ \sigma\ \varphi\ j)\ (progress\ \sigma\ \psi\ j)$   
 $\langle proof \rangle$

**lemma** *list\_all2\_upt\_Cons*:  $P a x \implies \text{list\_all2 } P [\text{Suc } a..<b] xs \implies \text{Suc } a \leq b \implies \text{list\_all2 } P [a..<b] (x \# xs)$   
 ⟨proof⟩

**lemma** *list\_all2\_upt\_append*:  $\text{list\_all2 } P [a..<b] xs \implies \text{list\_all2 } P [b..<c] ys \implies a \leq b \implies b \leq c \implies \text{list\_all2 } P [a..<c] (xs @ ys)$   
 ⟨proof⟩

**lemma** *meval*:

**assumes** *wf\_mformula*  $\sigma j n R \varphi \varphi'$   
**shows** *case meval*  $n (\tau \sigma j) (\Gamma \sigma j) \varphi$  of  $(xs, \varphi_n) \Rightarrow \text{wf\_mformula } \sigma (\text{Suc } j) n R \varphi_n \varphi' \wedge \text{list\_all2 } (\lambda i. \text{qtable } n (\text{MFOTL.fv } \varphi') (\text{mem\_restr } R) (\lambda v. \text{MFOTL.sat } \sigma (\text{map the } v) i \varphi')) [\text{progress } \sigma \varphi' j..<\text{progress } \sigma \varphi' (\text{Suc } j)] xs$   
 ⟨proof⟩

#### 6.5.4 Monitor step

**lemma** *wf\_mstate\_mstep*:  $\text{wf\_mstate } \varphi \pi R st \implies \text{last\_ts } \pi \leq \text{snd tdb} \implies \text{wf\_mstate } \varphi (\text{psnoc } \pi \text{ tdb}) R (\text{snd } (\text{mstep tdb } st))$   
 ⟨proof⟩

**lemma** *mstep\_output\_iff*:

**assumes** *wf\_mstate*  $\varphi \pi R st \text{last\_ts } \pi \leq \text{snd tdb}$  *prefix\_of*  $(\text{psnoc } \pi \text{ tdb}) \sigma \text{mem\_restr } R v$   
**shows**  $(i, v) \in \text{fst } (\text{mstep tdb } st) \iff \text{progress } \sigma \varphi (\text{plen } \pi) \leq i \wedge i < \text{progress } \sigma \varphi (\text{Suc } (\text{plen } \pi)) \wedge \text{wf\_tuple } (\text{MFOTL.nfv } \varphi) (\text{MFOTL.fv } \varphi) v \wedge \text{MFOTL.sat } \sigma (\text{map the } v) i \varphi$   
 ⟨proof⟩

#### 6.5.5 Monitor function

**definition** *minit\_safe* **where**

*minit\_safe*  $\varphi = (\text{if } \text{mmonitorable\_exec } \varphi \text{ then } \text{minit } \varphi \text{ else undefined})$

**lemma** *minit\_safe\_minit*:  $\text{mmonitorable } \varphi \implies \text{minit\_safe } \varphi = \text{minit } \varphi$   
 ⟨proof⟩

**lemma** (in *monitorable\_mfotl*) *mstep\_mverdicts*:

**assumes** *wf*: *wf\_mstate*  $\varphi \pi R st$   
**and** *le[simp]*:  $\text{last\_ts } \pi \leq \text{snd tdb}$   
**and** *restrict*: *mem\_restr*  $R v$   
**shows**  $(i, v) \in \text{fst } (\text{mstep tdb } st) \iff (i, v) \in M (\text{psnoc } \pi \text{ tdb}) - M \pi$   
 ⟨proof⟩

**primrec** *msteps0* **where**

*msteps0*  $\square st = (\{\}, st)$   
 | *msteps0*  $(\text{tdb } \# \pi) st = (\text{let } (V', st') = \text{mstep tdb } st; (V'', st'') = \text{msteps0 } \pi st' \text{ in } (V' \cup V'', st''))$

**primrec** *msteps0\_stateless* **where**

*msteps0\_stateless*  $\square st = \{\}$   
 | *msteps0\_stateless*  $(\text{tdb } \# \pi) st = (\text{let } (V', st') = \text{mstep tdb } st \text{ in } V' \cup \text{msteps0\_stateless } \pi st')$

**lemma** *msteps0\_msteps0\_stateless*:  $\text{fst } (\text{msteps0 } w st) = \text{msteps0\_stateless } w st$   
 ⟨proof⟩

**lift\_definition** *msteps* ::  $'a \text{MFOTL.prefix} \Rightarrow 'a \text{mstate} \Rightarrow (\text{nat} \times 'a \text{option list}) \text{set} \times 'a \text{mstate}$   
**is** *msteps0* ⟨proof⟩

**lift\_definition** *msteps\_stateless* ::  $'a \text{MFOTL.prefix} \Rightarrow 'a \text{mstate} \Rightarrow (\text{nat} \times 'a \text{option list}) \text{set}$

**is** *msteps0\_stateless*  $\langle \text{proof} \rangle$

**lemma** *msteps\_msteps\_stateless*:  $\text{fst} (\text{msteps } w \text{ st}) = \text{msteps\_stateless } w \text{ st}$   
 $\langle \text{proof} \rangle$

**lemma** *msteps0\_snoc*:  $\text{msteps0} (\pi @ [\text{tdb}]) \text{ st} =$   
 $(\text{let } (V', \text{st}') = \text{msteps0 } \pi \text{ st}; (V'', \text{st}'') = \text{mstep tdb st' in } (V' \cup V'', \text{st}''))$   
 $\langle \text{proof} \rangle$

**lemma** *msteps\_psnoc*:  $\text{last\_ts } \pi \leq \text{snd tdb} \implies \text{msteps} (\text{psnoc } \pi \text{ tdb}) \text{ st} =$   
 $(\text{let } (V', \text{st}') = \text{msteps } \pi \text{ st}; (V'', \text{st}'') = \text{mstep tdb st' in } (V' \cup V'', \text{st}''))$   
 $\langle \text{proof} \rangle$

**definition** *monitor where*

*monitor*  $\varphi \pi = \text{msteps\_stateless } \pi (\text{minit\_safe } \varphi)$

**lemma** *Suc\_length\_conv\_snoc*:  $(\text{Suc } n = \text{length } xs) = (\exists y \text{ ys. } xs = \text{ys} @ [y] \wedge \text{length } \text{ys} = n)$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *monitorable\_mfotl*) *wf\_mstate\_msteps*:  $\text{wf\_mstate } \varphi \pi R \text{ st} \implies \text{mem\_restr } R v \implies \pi \leq$   
 $\pi' \implies$   
 $X = \text{msteps} (\text{pdrop} (\text{plen } \pi) \pi') \text{ st} \implies \text{wf\_mstate } \varphi \pi' R (\text{snd } X) \wedge$   
 $((i, v) \in \text{fst } X) = ((i, v) \in M \pi' - M \pi)$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *monitorable\_mfotl*) *wf\_mstate\_msteps\_stateless*:  
**assumes**  $\text{wf\_mstate } \varphi \pi R \text{ st mem\_restr } R v \pi \leq \pi'$   
**shows**  $(i, v) \in \text{msteps\_stateless} (\text{pdrop} (\text{plen } \pi) \pi') \text{ st} \longleftrightarrow (i, v) \in M \pi' - M \pi$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *monitorable\_mfotl*) *wf\_mstate\_msteps\_stateless\_UNIV*:  $\text{wf\_mstate } \varphi \pi \text{ UNIV } \text{st} \implies \pi \leq$   
 $\pi' \implies$   
 $\text{msteps\_stateless} (\text{pdrop} (\text{plen } \pi) \pi') \text{ st} = M \pi' - M \pi$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *monitorable\_mfotl*) *mverdicts\_Nil*:  $M \text{ pnil} = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *wf\_mstate\_minit\_safe*:  $\text{mmonitorable } \varphi \implies \text{wf\_mstate } \varphi \text{ pnil } R (\text{minit\_safe } \varphi)$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *monitorable\_mfotl*) *monitor\_mverdicts*:  $\text{monitor } \varphi \pi = M \pi$   
 $\langle \text{proof} \rangle$

## 6.6 Collected correctness results

**context** *monitorable\_mfotl*

**begin**

We summarize the main results proved above.

1. The term  $M$  describes semantically the monitor's expected behaviour:

- *mono\_monitor*:  $\pi \leq \pi' \implies M \pi \subseteq M \pi'$
- *sound\_monitor*:  $\llbracket (i, v) \in M \pi; \text{prefix\_of } \pi \sigma \rrbracket \implies \text{MFOTL.sat } \sigma (\text{map the } v) i \varphi$
- *complete\_monitor*:  $\llbracket \text{prefix\_of } \pi \sigma; \text{wf\_tuple} (\text{MFOTL.nfv } \varphi) (\text{fv } \varphi) v; \bigwedge \sigma. \text{prefix\_of } \pi \sigma \rrbracket \implies \text{MFOTL.sat } \sigma (\text{map the } v) i \varphi \implies \exists \pi'. \text{prefix\_of } \pi' \sigma \wedge (i, v) \in M \pi'$

- $\text{sliceable\_}M: \text{mem\_restr } S \ v \implies ((i, v) \in M (\text{pmap\_}\Gamma (\lambda D. D \cap \text{relevant\_events } \varphi S) \ \pi)) = ((i, v) \in M \ \pi)$
2. The executable monitor's online interface  $\text{minit\_safe}$  and  $\text{mstep}$  preserves the invariant  $\text{wf\_mstate}$  and produces the the verdicts according to  $M$ :
- $\text{wf\_mstate\_minit\_safe}: \text{mmonitorable } \varphi' \implies \text{wf\_mstate } \varphi' \ \text{pnil } R (\text{minit\_safe } \varphi')$
  - $\text{wf\_mstate\_mstep}: \llbracket \text{wf\_mstate } \varphi' \ \pi \ R \ \text{st}; \text{last\_ts } \pi \leq \text{snd } \text{tdb} \rrbracket \implies \text{wf\_mstate } \varphi' (\text{psnoc } \pi \ \text{tdb}) \ R (\text{snd } (\text{mstep } \text{tdb } \text{st}))$
  - $\text{mstep\_mverdicts}: \llbracket \text{wf\_mstate } \varphi \ \pi \ R \ \text{st}; \text{last\_ts } \pi \leq \text{snd } \text{tdb}; \text{mem\_restr } R \ v \rrbracket \implies ((i, v) \in \text{fst } (\text{mstep } \text{tdb } \text{st})) = ((i, v) \in M (\text{psnoc } \pi \ \text{tdb}) - M \ \pi)$
3. The executable monitor's offline interface  $\text{Monitor.monitor}$  implements  $M$ :
- $\text{monitor\_mverdicts}: \text{Monitor.monitor } \varphi \ \pi = M \ \pi$

end

## 7 Slicing framework

This section formalizes the abstract slicing framework and the joint data slicer presented in the article [3, Sections 4.2 and 4.3].

### 7.1 Abstract slicing

#### 7.1.1 Definition 1

Corresponds to locale  $\text{monitor}$  defined in theory  $\text{MFOTL\_Monitor.Abstract\_Monitor}$ .

#### 7.1.2 Definition 2

**locale**  $\text{slicer} = \text{monitor} +$   
**fixes**  $\text{submonitor} :: 'k :: \text{finite} \Rightarrow 'a \ \text{prefix} \Rightarrow (\text{nat} \times 'b \ \text{option list}) \ \text{set}$   
**and**  $\text{splitter} :: 'a \ \text{prefix} \Rightarrow 'k \Rightarrow 'a \ \text{prefix}$   
**and**  $\text{joiner} :: ('k \Rightarrow (\text{nat} \times 'b \ \text{option list}) \ \text{set}) \Rightarrow (\text{nat} \times 'b \ \text{option list}) \ \text{set}$   
**assumes**  $\text{mono\_splitter}: \pi \leq \pi' \implies \text{splitter } \pi \ k \leq \text{splitter } \pi' \ k$   
**and**  $\text{correct\_slicer}: \text{joiner } (\lambda k. \text{submonitor } k (\text{splitter } \pi \ k)) = M \ \pi$   
**begin**

**lemmas**  $\text{sound\_slicer} = \text{equalityD1}[\text{OF correct\_slicer}]$   
**lemmas**  $\text{complete\_slicer} = \text{equalityD2}[\text{OF correct\_slicer}]$

end

**locale**  $\text{self\_slicer} = \text{slicer} \ \text{nfv} \ \text{fv} \ \text{sat} \ M \ \lambda \_ . M \ \text{splitter} \ \text{joiner} \ \text{for} \ \text{nfv} \ \text{fv} \ \text{sat} \ M \ \text{splitter} \ \text{joiner}$

#### 7.1.3 Definition 3

**locale**  $\text{event\_separable\_splitter} =$   
**fixes**  $\text{event\_splitter} :: 'a \Rightarrow 'k :: \text{finite set}$   
**begin**

**lift\_definition**  $\text{splitter} :: 'a \ \text{prefix} \Rightarrow 'k \Rightarrow 'a \ \text{prefix}$  **is**  
 $\lambda \pi \ k. \text{map } (\lambda (D, t). (\{e \in D. k \in \text{event\_splitter } e\}, t)) \ \pi$   
 $\langle \text{proof} \rangle$

#### 7.1.4 Lemma 1

**lemma** *mono\_splitter*:  $\pi \leq \pi' \implies \text{splitter } \pi \ k \leq \text{splitter } \pi' \ k$   
*<proof>*

**end**

## 7.2 Joint data slicer

**abbreviation** *(input) ok*  $\varphi \ v \equiv \text{wf\_tuple } (\text{MFOTL.nfv } \varphi) (\text{MFOTL.fv } \varphi) \ v$

**locale** *splitting\_strategy* =  
  **fixes**  $\varphi :: 'a \ \text{MFOTL.formula}$   
  **and** *strategy* ::  $'a \ \text{option list} \Rightarrow 'k :: \text{finite set}$   
  **assumes** *strategy\_nonempty*:  $\text{ok } \varphi \ v \implies \text{strategy } v \neq \{\}$   
**begin**

**abbreviation** *slice\_set* **where**  
  *slice\_set*  $k \equiv \{v. \exists v'. \text{map the } v' = v \wedge \text{ok } \varphi \ v' \wedge k \in \text{strategy } v'\}$

**end**

### 7.2.1 Definition 4

**locale** *MFOTL\_monitor* =  
  *monitor*  $\text{MFOTL.nfv } \varphi \ \text{MFOTL.fv } \varphi \ \lambda \sigma \ v \ i. \ \text{MFOTL.sat } \sigma \ v \ i \ \varphi \ M \ \text{for } \varphi \ M$

**locale** *joint\_data\_slicer* = *MFOTL\_monitor*  $\varphi \ M + \text{splitting\_strategy } \varphi \ \text{strategy}$   
  **for**  $\varphi \ M \ \text{strategy}$   
**begin**

**definition** *event\_splitter* **where**  
  *event\_splitter*  $e = (\bigcup (\text{strategy } ' \{v. \text{ok } \varphi \ v \wedge \text{MFOTL.matches } (\text{map the } v) \ \varphi \ e\}))$

**sublocale** *event\_separable\_splitter* **where** *event\_splitter* = *event\_splitter* *<proof>*

**definition** *joiner* **where**  
  *joiner* =  $(\lambda s. \bigcup k. s \ k \cap (\text{UNIV} :: \text{nat set}) \times \{v. k \in \text{strategy } v\})$

**lemma** *splitter\_pslice*:  $\text{splitter } \pi \ k = \text{MFOTL\_slicer.pslice } \varphi \ (\text{slice\_set } k) \ \pi$   
*<proof>*

### 7.2.2 Lemma 2

Corresponds to the following theorem *sat\_slice\_strong* proved in theory *MFOTL\_Monitor.Abstract\_Monitor*:  
 $\llbracket \text{relevant\_events } \varphi' \ S \subseteq E; v \in S \rrbracket \implies \text{MFOTL.sat } \sigma \ v \ i \ \varphi' = \text{MFOTL.sat } (\text{map\_}\Gamma \ (\lambda D. D \cap E) \ \sigma) \ v \ i \ \varphi'$

### 7.2.3 Theorem 1

**sublocale** *joint\_monitor*: *MFOTL\_monitor*  $\varphi \ \lambda \pi. \ \text{joiner } (\lambda k. M \ (\text{splitter } \pi \ k))$   
*<proof>*

### 7.2.4 Corollary 1

**sublocale** *joint\_slicer*: *slicer*  $\text{MFOTL.nfv } \varphi \ \text{MFOTL.fv } \varphi \ \lambda \sigma \ v \ i. \ \text{MFOTL.sat } \sigma \ v \ i \ \varphi$   
   $\lambda \pi. \ \text{joiner } (\lambda k. M \ (\text{splitter } \pi \ k)) \ \lambda \_ . M \ \text{splitter } \text{joiner}$   
*<proof>*

end

### 7.2.5 Definition 5

Corresponds to locale *sliceable\_monitor* defined in theory *MFOTL\_Monitor.Abstract\_Monitor*.

```
locale sliceable_joint_data_slicer =  
  sliceable_monitor MFOTL.nfv  $\varphi$  MFOTL.fv  $\varphi$  relevant_events  $\varphi$   $\lambda\sigma v i.$  MFOTL.sat  $\sigma v i \varphi M +$   
  joint_data_slicer  $\varphi M$  strategy for  $\varphi M$  strategy  
begin
```

```
lemma monitor_split: ok  $\varphi v \implies k \in$  strategy  $v \implies (i, v) \in M$  (splitter  $\pi k$ )  $\longleftrightarrow (i, v) \in M \pi$   
<proof>
```

### 7.2.6 Theorem 2

```
sublocale self_slicer MFOTL.nfv  $\varphi$  MFOTL.fv  $\varphi$   $\lambda\sigma v i.$  MFOTL.sat  $\sigma v i \varphi M$  splitter joiner  
<proof>
```

end

### 7.2.7 Towards Theorem 3

```
fun names :: 'a MFOTL.formula  $\Rightarrow$  MFOTL.name set where  
  names (MFOTL.Pred  $e \_$ ) = { $e$ }  
| names (MFOTL.Eq  $\_ \_$ ) = {}  
| names (MFOTL.Neg  $\psi$ ) = names  $\psi$   
| names (MFOTL.Or  $\alpha \beta$ ) = names  $\alpha \cup$  names  $\beta$   
| names (MFOTL.Exists  $\psi$ ) = names  $\psi$   
| names (MFOTL.Prev  $I \psi$ ) = names  $\psi$   
| names (MFOTL.Next  $I \psi$ ) = names  $\psi$   
| names (MFOTL.Since  $\alpha I \beta$ ) = names  $\alpha \cup$  names  $\beta$   
| names (MFOTL.Until  $\alpha I \beta$ ) = names  $\alpha \cup$  names  $\beta$ 
```

```
fun gen_unique :: 'a MFOTL.formula  $\Rightarrow$  bool where  
  gen_unique (MFOTL.Pred  $\_ \_$ ) = True  
| gen_unique (MFOTL.Eq (MFOTL.Var  $\_$ ) (MFOTL.Const  $\_$ )) = False  
| gen_unique (MFOTL.Eq (MFOTL.Const  $\_$ ) (MFOTL.Var  $\_$ )) = False  
| gen_unique (MFOTL.Eq  $\_ \_$ ) = True  
| gen_unique (MFOTL.Neg  $\psi$ ) = gen_unique  $\psi$   
| gen_unique (MFOTL.Or  $\alpha \beta$ ) = (gen_unique  $\alpha \wedge$  gen_unique  $\beta \wedge$  names  $\alpha \cap$  names  $\beta = \{\}$ )  
| gen_unique (MFOTL.Exists  $\psi$ ) = gen_unique  $\psi$   
| gen_unique (MFOTL.Prev  $I \psi$ ) = gen_unique  $\psi$   
| gen_unique (MFOTL.Next  $I \psi$ ) = gen_unique  $\psi$   
| gen_unique (MFOTL.Since  $\alpha I \beta$ ) = (gen_unique  $\alpha \wedge$  gen_unique  $\beta \wedge$  names  $\alpha \cap$  names  $\beta = \{\}$ )  
| gen_unique (MFOTL.Until  $\alpha I \beta$ ) = (gen_unique  $\alpha \wedge$  gen_unique  $\beta \wedge$  names  $\alpha \cap$  names  $\beta = \{\}$ )
```

```
lemma sat_inter_names_cong: ( $\bigwedge e. e \in$  names  $\varphi \implies \{xs. (e, xs) \in E\} = \{xs. (e, xs) \in F\}$ )  $\implies$   
  MFOTL.sat (map_ $\Gamma$  ( $\lambda D. D \cap E$ )  $\sigma$ )  $v i \varphi \longleftrightarrow$  MFOTL.sat (map_ $\Gamma$  ( $\lambda D. D \cap F$ )  $\sigma$ )  $v i \varphi$   
<proof>
```

```
lemma matches_in_names: MFOTL.matches  $v \varphi x \implies$  fst  $x \in$  names  $\varphi$   
<proof>
```

```
lemma unique_names_matches_absorb: fst  $x \in$  names  $\alpha \implies$  names  $\alpha \cap$  names  $\beta = \{\} \implies$   
  MFOTL.matches  $v \alpha x \vee$  MFOTL.matches  $v \beta x \longleftrightarrow$  MFOTL.matches  $v \alpha x$   
fst  $x \in$  names  $\beta \implies$  names  $\alpha \cap$  names  $\beta = \{\} \implies$   
  MFOTL.matches  $v \alpha x \vee$  MFOTL.matches  $v \beta x \longleftrightarrow$  MFOTL.matches  $v \beta x$   
<proof>
```

**definition** *mergeable\_envs* **where**

*mergeable\_envs*  $n$   $S \longleftrightarrow (\forall v1 \in S. \forall v2 \in S. (\forall A B f. (\forall x \in A. x < n \wedge v1 ! x = f x) \wedge (\forall x \in B. x < n \wedge v2 ! x = f x) \longrightarrow (\exists v \in S. \forall x \in A \cup B. v ! x = f x)))$

**lemma** *mergeable\_envsI*:

**assumes**  $\bigwedge v1 v2 v. v1 \in S \implies v2 \in S \implies \text{length } v = n \implies \forall x < n. v ! x = v1 ! x \vee v ! x = v2 ! x$   
 $\implies v \in S$

**shows** *mergeable\_envs*  $n$   $S$

*<proof>*

**lemma** *in\_listset\_nth*:  $x \in \text{listset } As \implies i < \text{length } As \implies x ! i \in As ! i$

*<proof>*

**lemma** *all\_nth\_in\_listset*:  $\text{length } x = \text{length } As \implies (\bigwedge i. i < \text{length } As \implies x ! i \in As ! i) \implies x \in \text{listset } As$

*<proof>*

**lemma** *mergeable\_envs\_listset*: *mergeable\_envs*  $(\text{length } As)$   $(\text{listset } As)$

*<proof>*

**lemma** *mergeable\_envs\_Exists*: *mergeable\_envs*  $n$   $S \implies \text{MFOTL.nfv } \alpha \leq n \implies \text{MFOTL.nfv } \beta \leq n \implies$

$(\exists v' \in S. \forall x \in \text{fv } \alpha. v' ! x = v ! x) \implies (\exists v' \in S. \forall x \in \text{fv } \beta. v' ! x = v ! x) \implies$

$(\exists v' \in S. \forall x \in \text{fv } \alpha \cup \text{fv } \beta. v' ! x = v ! x)$

*<proof>*

**lemma** *in\_set\_ConsE*:  $xs \in \text{set\_Cons } A \ As \implies (\bigwedge y \ ys. xs = y \# \ ys \implies y \in A \implies ys \in As \implies P)$

$\implies P$

*<proof>*

**lemma** *mergeable\_envs\_set\_Cons*: *mergeable\_envs*  $n$   $S \implies \text{mergeable_envs } (\text{Suc } n)$   $(\text{set\_Cons UNIV } S)$

*<proof>*

**lemma** *slice\_Exists*:  $\text{MFOTL.slicer.slice } (\text{MFOTL.Exists } \varphi) \ S \ \sigma = \text{MFOTL.slicer.slice } \varphi \ (\text{set\_Cons UNIV } S) \ \sigma$

*<proof>*

**lemma** *image\_Suc\_fvi*:  $\text{Suc } \text{' MFOTL.fvi } (\text{Suc } b) \ \varphi = \text{MFOTL.fvi } b \ \varphi - \{0\}$

*<proof>*

**lemma** *nfv\_Exists*:  $\text{MFOTL.nfv } (\text{MFOTL.Exists } \varphi) = \text{MFOTL.nfv } \varphi - 1$

*<proof>*

**lemma** *set\_Cons\_empty\_iff[simp]*:  $\text{set\_Cons } A \ Xs = \{\} \longleftrightarrow A = \{\} \vee Xs = \{\}$

*<proof>*

**lemma** *unique\_sat\_slice\_mem*:  $\text{safe\_formula } \varphi \implies \text{gen\_unique } \varphi \implies S \neq \{\} \implies$

$\text{mergeable\_envs } n \ S \implies \text{MFOTL.nfv } \varphi \leq n \implies$

$\text{MFOTL.sat } (\text{MFOTL.slicer.slice } \varphi \ S \ \sigma) \ v \ i \ \varphi \implies \exists v' \in S. \forall x \in \text{fv } \varphi. v' ! x = v ! x$

*<proof>*

**lemma** *unique\_sat\_slice*:

**assumes** *formula*:  $\text{safe\_formula } \varphi \ \text{gen\_unique } \varphi$

**and** *restr*:  $S \neq \{\} \ \text{mergeable\_envs } (\text{MFOTL.nfv } \varphi) \ S$

**and** *sat\_slice*:  $\text{MFOTL.sat } (\text{MFOTL.slicer.slice } \varphi \ S \ \sigma) \ v \ i \ \varphi$

**shows**  $\text{MFOTL.sat } \sigma \ v \ i \ \varphi$

*<proof>*

### 7.2.8 Lemma 3

**lemma** (in *splitting\_strategy*) *unique\_sat\_strategy*:  
safe\_formula  $\varphi \implies$  gen\_unique  $\varphi \implies$  slice\_set  $k \neq \{\}$   $\implies$   
mergeable\_envs (MFOTL.nfv  $\varphi$ ) (slice\_set  $k$ )  $\implies$   
MFOTL.sat (MFOTL.slicer.slice  $\varphi$  (slice\_set  $k$ )  $\sigma$ ) (map the v)  $i \varphi \implies$   
ok  $\varphi v \implies k \in$  strategy  $v$   
*<proof>*

**locale** skip\_inter = joint\_data\_slicer +  
assumes nonempty: slice\_set  $k \neq \{\}$   
and mergeable: mergeable\_envs (MFOTL.nfv  $\varphi$ ) (slice\_set  $k$ )  
**begin**

### 7.2.9 Definition of J'

**definition** skip\_joiner = ( $\lambda s. \bigcup k. s k$ )

### 7.2.10 Theorem 3

**lemma** skip\_joiner:  
assumes safe\_formula  $\varphi$  gen\_unique  $\varphi$   
shows joiner ( $\lambda k. M$  (splitter  $\pi k$ )) = skip\_joiner ( $\lambda k. M$  (splitter  $\pi k$ ))  
(is ?L = ?R)  
*<proof>*

**sublocale** skip\_joint\_monitor: MFOTL\_monitor  $\varphi$   
 $\lambda \pi. (if$  safe\_formula  $\varphi \wedge$  gen\_unique  $\varphi$  *then* skip\_joiner *else* joiner) ( $\lambda k. M$  (splitter  $\pi k$ ))  
*<proof>*

**end**

## References

- [1] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, 2015.
- [2] D. Basin, F. Klaedtke, and E. Zălinescu. The MonPoly monitoring tool. In G. Reger and K. Havelund, editors, *RV-CuBES 2017*, volume 3 of *Kalpa Publications in Computing*, pages 19–28. EasyChair, 2017.
- [3] J. Schneider, D. Basin, F. Brix, S. Krstić, and D. Traytel. Scalable online first-order monitoring. *Int. J. Softw. Tools Technol. Transf.*, 2020. To appear. Preprint at [http://people.inf.ethz.ch/traytel/papers/sttt20-som\\_long/som\\_long.pdf](http://people.inf.ethz.ch/traytel/papers/sttt20-som_long/som_long.pdf).
- [4] J. Schneider, D. Basin, S. Krstić, and D. Traytel. A formally verified monitor for metric first-order temporal logic. In B. Finkbeiner and L. Mariani, editors, *RV 2019*, volume 11757 of *LNCS*, pages 310–328. Springer, 2019.