

Formalizing MLTL in Isabelle/HOL

Zili Wang and Katherine Kosaian

October 13, 2025

Abstract

Building on the formalization of Mission-time Linear Temporal Logic (MLTL) in Isabelle/HOL, we formalize the correctness of the algorithms for the WEST tool [1, 2], which converts MLTL formulas to regular expressions. We use Isabelle/HOL's code export to generate Haskell code to validate the existing (unverified) implementation of the WEST tool.

Contents

1	Key algorithms for WEST	2
1.1	Custom Types	2
1.2	Trace Regular Expressions	2
1.3	WEST Operations	4
1.3.1	AND	4
1.3.2	Simp	5
1.3.3	AND and OR operations with WEST-simp	7
1.3.4	Useful Helper Functions	7
1.3.5	WEST Temporal Operations	8
1.3.6	WEST recursive reg Function	9
1.3.7	Adding padding	11
2	Some examples and Code Export	12
3	WEST Proofs	12
3.1	Useful Definitions	12
3.2	Proofs about Traces Matching Regular Expressions	13
3.3	Facts about the WEST and operator	13
3.3.1	Commutative	13
3.3.2	Identity and Zero	16
3.3.3	WEST-and-state	16
3.3.4	WEST-and-trace	18
3.3.5	WEST-and correct	21

3.4	Facts about the WEST or operator	22
3.5	Pad and Match Facts	22
3.6	Facts about WEST num vars	23
3.6.1	Facts about num vars for different WEST operators	23
3.7	Correctness of WEST-simp	27
3.7.1	WEST-count-diff facts	27
3.7.2	Orsimp-trace Facts	29
3.7.3	WEST-orsimp-trace-correct	30
3.7.4	Simp-helper Correct	31
3.7.5	WEST-simp Correct	32
3.8	Correctness of WEST-and-simp/WEST-or-simp	33
3.9	Facts about the WEST future operator	33
3.10	Facts about the WEST global operator	34
3.11	Facts about the WEST until operator	35
3.12	Facts about the WEST release Operator	36
3.13	Top level result: Shows that WEST reg is correct	37
3.14	Top level result for padded version	37
4	Key algorithms for WEST	38
5	Regex Equivalence Correctness	39

1 Key algorithms for WEST

theory *WEST-Algorithms*

imports *Mission-Time-LTL.MLTL-Properties*

begin

1.1 Custom Types

datatype *WEST-bit* = *Zero* | *One* | *S*

type-synonym *state* = *nat set*

type-synonym *trace* = *nat set list*

type-synonym *state-regex* = *WEST-bit list*

type-synonym *trace-regex* = *WEST-bit list list*

type-synonym *WEST-regex* = *WEST-bit list list list*

1.2 Trace Regular Expressions

fun *WEST-get-bit:: trace-regex* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *WEST-bit*

where *WEST-get-bit regex timestep var* = (
if timestep \geq *length regex* *then S*
else let regex-index = *regex ! timestep* *in*
if var \geq *length regex-index* *then S*
else regex-index ! var

)

Returns the state at time i, list of variable states

```
fun WEST-get-state:: trace-regex  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  state-regex
where WEST-get-state regex time num-vars = (
  if time  $\geq$  length regex then (map ( $\lambda$  k. S) [0 ..< num-vars])
  else regex ! time
)
```

Checks if one state of a trace matches one timeslice of a WEST regex

```
definition match-timestep:: nat set  $\Rightarrow$  state-regex  $\Rightarrow$  bool
where match-timestep state regex-state = ( $\forall$  x::nat. x < length regex-state  $\longrightarrow$ 
(
  ((regex-state ! x = One)  $\longrightarrow$  x  $\in$  state)  $\wedge$ 
  ((regex-state ! x = Zero)  $\longrightarrow$  x  $\notin$  state)))
```

```
fun trim-reversed-regex:: trace-regex  $\Rightarrow$  trace-regex
where trim-reversed-regex [] = []
| trim-reversed-regex (h#t) = (if ( $\forall$  i<length h. (h!i) = S)
then (trim-reversed-regex t) else (h#t))
```

```
fun trim-regex:: trace-regex  $\Rightarrow$  trace-regex
where trim-regex regex = rev (trim-reversed-regex (rev regex))
```

```
definition match-regex:: nat set list  $\Rightarrow$  trace-regex  $\Rightarrow$  bool
where match-regex trace regex = (( $\forall$  time<length regex.
  (match-timestep (trace ! time) (regex ! time)))
 $\wedge$ (length trace  $\geq$  length regex))
```

```
definition match:: nat set list  $\Rightarrow$  WEST-regex  $\Rightarrow$  bool
where match trace regex-list = ( $\exists$  i. i < length regex-list  $\wedge$ 
  (match-regex trace (regex-list ! i)))
```

```
lemma match-example:
shows match [{0::nat,1}, {1}, {0}]
[
  [[Zero,Zero]],
  [[S,S], [S,One]]
] = True
<proof>
```

```
definition regex-equiv:: WEST-regex  $\Rightarrow$  WEST-regex  $\Rightarrow$  bool
where regex-equiv rl1 rl2 = (
 $\forall$   $\pi$ ::nat set list. (match  $\pi$  rl1)  $\longleftrightarrow$  (match  $\pi$  rl2))
```

```
lemma (regex-equiv [[[S,S]]
  [[[S,One]]],
```

```

    [[One,S],
     [[Zero,Zero]]]) = True
⟨proof⟩

```

1.3 WEST Operations

1.3.1 AND

```

fun WEST-and-bitwise:: WEST-bit ⇒
    WEST-bit ⇒
    WEST-bit option
where WEST-and-bitwise b One = (if b = Zero then None else Some One)
| WEST-and-bitwise b Zero = (if b = One then None else Some Zero)
| WEST-and-bitwise b S = Some b

```

```

fun WEST-and-state:: state-regex ⇒ state-regex ⇒ state-regex option
where WEST-and-state [] [] = Some []
| WEST-and-state (h1#t1) (h2#t2) =
(case WEST-and-bitwise h1 h2 of
  None ⇒ None
  | Some b ⇒ (case WEST-and-state t1 t2 of
    None ⇒ None
    | Some L ⇒ Some (b#L)))
| WEST-and-state - - = None

```

```

fun WEST-and-trace:: trace-regex ⇒ trace-regex ⇒ trace-regex option
where WEST-and-trace trace [] = Some trace
| WEST-and-trace [] trace = Some trace
| WEST-and-trace (h1#t1) (h2#t2) =
(case WEST-and-state h1 h2 of
  None ⇒ None
  | Some state ⇒ (case WEST-and-trace t1 t2 of
    None ⇒ None
    | Some trace ⇒ Some (state#trace)))

```

```

fun WEST-and-helper:: trace-regex ⇒ WEST-regex ⇒ WEST-regex
where WEST-and-helper trace [] = []
| WEST-and-helper trace (t#traces) =
(case WEST-and-trace trace t of
  None ⇒ WEST-and-helper trace traces
  | Some res ⇒ res#(WEST-and-helper trace traces))

```

```

fun WEST-and:: WEST-regex ⇒ WEST-regex ⇒ WEST-regex
where WEST-and traceList [] = []

```

```

| WEST-and [] traceList = []
| WEST-and (trace#traceList1) traceList2 =
(case WEST-and-helper trace traceList2 of
 [] => WEST-and traceList1 traceList2
 | traceList => traceList@(WEST-and traceList1 traceList2))

```

1.3.2 Simp

Bitwise simplification operation **fun** *WEST-simp-bitwise*:: *WEST-bit* =>
WEST-bit => *WEST-bit*
where *WEST-simp-bitwise* *b* *S* = *S*
| *WEST-simp-bitwise* *b* *Zero* = (if *b* = *Zero* then *Zero* else *S*)
| *WEST-simp-bitwise* *b* *One* = (if *b* = *One* then *One* else *S*)

fun *WEST-simp-state*:: *state-regex* => *state-regex* => *state-regex*
where *WEST-simp-state* *s1* *s2* = (
map (λ *k*. *WEST-simp-bitwise* (*s1* ! *k*) (*s2* ! *k*)) [0 ..< (length *s1*)])

fun *WEST-simp-trace*:: *trace-regex* => *trace-regex* => *nat* => *trace-regex*
where *WEST-simp-trace* *trace1* *trace2* *num-vars* = (
map (λ *k*. (*WEST-simp-state* (*WEST-get-state* *trace1* *k* *num-vars*) (*WEST-get-state*
trace2 *k* *num-vars*)))
[0 ..< (Max {(length *trace1*), (length *trace2*)})])

Helper functions for defining WEST-simp **fun** *count-nonS-trace*:: *state-regex*
=> *nat*
where *count-nonS-trace* [] = 0
| *count-nonS-trace* (*h*#*t*) = (if (*h* ≠ *S*) then (1 + (*count-nonS-trace* *t*)) else
(*count-nonS-trace* *t*))

fun *count-diff-state*:: *state-regex* => *state-regex* => *nat*
where *count-diff-state* [] [] = 0
| *count-diff-state* *trace* [] = *count-nonS-trace* *trace*
| *count-diff-state* [] *trace* = *count-nonS-trace* *trace*
| *count-diff-state* (*h1*#*t1*) (*h2*#*t2*) = (if (*h1* = *h2*) then (*count-diff-state* *t1* *t2*)
else (1 + (*count-diff-state* *t1* *t2*)))

fun *count-diff*:: *trace-regex* => *trace-regex* => *nat*
where *count-diff* [] [] = 0
| *count-diff* [] (*h*#*t*) = (*count-diff-state* [] *h*) + (*count-diff* [] *t*)
| *count-diff* (*h*#*t*) [] = (*count-diff-state* [] *h*) + (*count-diff* [] *t*)
| *count-diff* (*h1*#*t1*) (*h2*#*t2*) = (*count-diff-state* *h1* *h2*) + (*count-diff* *t1* *t2*)

fun *check-simp*:: *trace-regex* => *trace-regex* => *bool*
where *check-simp* *trace1* *trace2* = ((*count-diff* *trace1* *trace2*) ≤ 1 ∧ length *trace1*
= length *trace2*)

fun *enumerate-pairs* :: *nat* *list* => (*nat* * *nat*) *list* **where**

$enumerate_pairs [] = []$ |
 $enumerate_pairs (x\#xs) = map (\lambda y. (x, y)) xs @ enumerate_pairs xs$

fun $enum_pairs:: 'a list \Rightarrow (nat * nat) list$
where $enum_pairs L = enumerate_pairs [0 ..< length L]$

fun $remove_element_at_index:: nat \Rightarrow 'a list \Rightarrow 'a list$
where $remove_element_at_index n L = (take n L)@(drop (n+1) L)$

This assumes $(fst h) < (snd h)$

fun $update_L:: WEST_regex \Rightarrow (nat \times nat) \Rightarrow nat \Rightarrow WEST_regex$
where $update_L L h num_vars =$
 $(remove_element_at_index (fst h) (remove_element_at_index (snd h) L))@[WEST_simp_trace$
 $(L!(fst h)) (L!(snd h)) num_vars]$

Defining and Proving Termination of WEST-simp **lemma** $length_enumerate_pairs:$

shows $length (enumerate_pairs L) \leq (length L)^2$
 $\langle proof \rangle$

lemma $length_enum_pairs:$

shows $length (enum_pairs L) \leq (length L)^2$
 $\langle proof \rangle$

lemma $enumerate_pairs_fact:$

assumes $\forall i j. (i < j \wedge i < length L \wedge j < length L) \longrightarrow (L!i) < (L!j)$
shows $\forall pair \in set (enumerate_pairs L). (fst pair) < (snd pair)$
 $\langle proof \rangle$

lemma $enum_pairs_fact:$

shows $\forall pair \in set (enum_pairs L). (fst pair) < (snd pair)$
 $\langle proof \rangle$

lemma $enum_pairs_bound_snd:$

assumes $pair \in set (enumerate_pairs L)$
shows $(snd pair) \in set L$
 $\langle proof \rangle$

lemma $enum_pairs_bound:$

shows $\forall pair \in set (enum_pairs L). (snd pair) < length L$
 $\langle proof \rangle$

lemma $WEST_simp_termination1_bound:$

fixes $a::nat$
shows $a^3 + a^2 < (a+1)^3$
 $\langle proof \rangle$

lemma $WEST_simp_termination1:$

fixes $L::WEST_regex$

assumes $\neg (idx\text{-pairs} \neq enum\text{-pairs } L \vee length\ idx\text{-pairs} \leq i)$
assumes $check\text{-simp } (L ! fst (idx\text{-pairs} ! i)) (L ! snd (idx\text{-pairs} ! i))$
assumes $x = update\text{-L } L (idx\text{-pairs} ! i) num\text{-vars}$
shows $((x, enum\text{-pairs } x, 0, num\text{-vars}), L, idx\text{-pairs}, i, num\text{-vars})$
 $\in measure (\lambda(L, idx\text{-list}, i, num\text{-vars}). length\ L \wedge 3 + length\ idx\text{-list} - i)$
 <proof>

function $WEST\text{-simp-helper}:: WEST\text{-regex} \Rightarrow (nat \times nat) list \Rightarrow nat \Rightarrow nat \Rightarrow WEST\text{-regex}$
where $WEST\text{-simp-helper } L idx\text{-pairs } i num\text{-vars} =$
 $(if (idx\text{-pairs} \neq enum\text{-pairs } L \vee i \geq length\ idx\text{-pairs}) then L else$
 $(if (check\text{-simp } (L!(fst (idx\text{-pairs}!i))) (L!(snd (idx\text{-pairs}!i)))) then$
 $(let newL = update\text{-L } L (idx\text{-pairs}!i) num\text{-vars} in$
 $WEST\text{-simp-helper } newL (enum\text{-pairs } newL) 0 num\text{-vars})$
 $else WEST\text{-simp-helper } L idx\text{-pairs } (i+1) num\text{-vars}))$
 <proof>
termination
 <proof>

declare $WEST\text{-simp-helper.simps}[simp\ del]$

fun $WEST\text{-simp}:: WEST\text{-regex} \Rightarrow nat \Rightarrow WEST\text{-regex}$
where $WEST\text{-simp } L num\text{-vars} =$
 $WEST\text{-simp-helper } L (enum\text{-pairs } L) 0 num\text{-vars}$

value $WEST\text{-simp } [[[S, S, One]], [[S, One, S]], [[S, S, Zero]]] 3$
value $WEST\text{-simp } [[[S, One]], [[One, S]], [[Zero, Zero]]] 2$
value $WEST\text{-simp } [[[One, One]], [[Zero, Zero]], [[One, Zero]], [[Zero, One]]] 2$

1.3.3 AND and OR operations with WEST-simp

fun $WEST\text{-and-simp}:: WEST\text{-regex} \Rightarrow WEST\text{-regex} \Rightarrow nat \Rightarrow WEST\text{-regex}$
where $WEST\text{-and-simp } L1 L2 num\text{-vars} = WEST\text{-simp } (WEST\text{-and } L1 L2) num\text{-vars}$

fun $WEST\text{-or-simp}:: WEST\text{-regex} \Rightarrow WEST\text{-regex} \Rightarrow nat \Rightarrow WEST\text{-regex}$
where $WEST\text{-or-simp } L1 L2 num\text{-vars} = WEST\text{-simp } (L1 @ L2) num\text{-vars}$

1.3.4 Useful Helper Functions

fun $arbitrary\text{-state}:: nat \Rightarrow state\text{-regex}$
where $arbitrary\text{-state } num\text{-vars} = map (\lambda k. S) [0 ..< num\text{-vars}]$

fun $arbitrary\text{-trace}:: nat \Rightarrow nat \Rightarrow trace\text{-regex}$
where $arbitrary\text{-trace } num\text{-vars } num\text{-pad} = map (\lambda k. (arbitrary\text{-state } num\text{-vars})) [0 ..< num\text{-pad}]$

fun $shift:: WEST\text{-regex} \Rightarrow nat \Rightarrow nat \Rightarrow WEST\text{-regex}$

where *shift traceList num-vars num-pad* = *map* (λ *trace*. (*arbitrary-trace num-vars num-pad*)@*trace*) *traceList*

fun *pad*:: *trace-regex* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *trace-regex*
where *pad trace num-vars num-pad* = *trace*@(*arbitrary-trace num-vars num-pad*)

1.3.5 WEST Temporal Operations

fun *WEST-global*:: *WEST-regex* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *WEST-regex*

where *WEST-global L a b num-vars* = (
if (*a = b*) *then* (*shift L num-vars a*)
else (*if* (*a < b*) *then* (*WEST-and-simp* (*shift L num-vars b*)
(*WEST-global L a (b-1) num-vars*) *num-vars*)
else []))

fun *WEST-future*:: *WEST-regex* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *WEST-regex*

where *WEST-future L a b num-vars* = (
if (*a = b*)
then (*shift L num-vars a*)
else (
if (*a < b*)
then *WEST-or-simp* (*shift L num-vars b*) (*WEST-future L a (b-1) num-vars*)
num-vars
else []))

fun *WEST-until*:: *WEST-regex* \Rightarrow *WEST-regex* \Rightarrow *nat* \Rightarrow
nat \Rightarrow *nat* \Rightarrow *WEST-regex*

where *WEST-until L- φ L- ψ a b num-vars* = (
if (*a=b*)
then (*WEST-global L- ψ a a num-vars*)
else (
if (*a < b*)
then *WEST-or-simp* (*WEST-until L- φ L- ψ a (b-1) num-vars*)
(*WEST-and-simp* (*WEST-global L- φ a (b-1) num-vars*)
(*WEST-global L- ψ b b num-vars*) *num-vars*) *num-vars*)
else []))

fun *WEST-release-helper*:: *WEST-regex* \Rightarrow *WEST-regex* \Rightarrow
nat \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *WEST-regex*

where *WEST-release-helper L- φ L- ψ a ub num-vars* = (
if (*a=ub*)
then (*WEST-and-simp* (*WEST-global L- φ a a num-vars*) (*WEST-global L- ψ a a num-vars*) *num-vars*)
else (
if (*a < ub*)
then *WEST-or-simp* (*WEST-release-helper L- φ L- ψ a (ub-1) num-vars*)

$(WEST\text{-}and\text{-}simp (WEST\text{-}global L\text{-}\psi a ub num\text{-}vars)$
 $(WEST\text{-}global L\text{-}\varphi ub ub num\text{-}vars) num\text{-}vars) num\text{-}vars$
else [])

fun *WEST-release*:: *WEST-regex* \Rightarrow *WEST-regex* \Rightarrow *nat*
 \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *WEST-regex*
where *WEST-release* *L-φ L-ψ a b num-vars* = (
if (*b > a*)
then (*WEST-or-simp* (*WEST-global L-ψ a b num-vars*) (*WEST-release-helper*
L-φ L-ψ a (b-1) num-vars) *num-vars*)
else (*WEST-global L-ψ a b num-vars*)

1.3.6 WEST recursive reg Function

lemma *exhaustive*:

shows $\bigwedge x:: nat\ mltl \times nat. \bigwedge P:: bool. (\bigwedge num\text{-}vars:: nat. x = (True\text{-}mtl, num\text{-}vars)$
 $\Rightarrow P) \Rightarrow$
 $(\bigwedge num\text{-}vars:: nat. x = (False\text{-}mtl, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge p\ num\text{-}vars:: nat. x = (Prop\text{-}mtl\ p, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge p\ num\text{-}vars:: nat. x = (Not\text{-}mtl (Prop\text{-}mtl\ p), num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ num\text{-}vars. x = (Or\text{-}mtl\ \varphi\ \psi, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ num\text{-}vars. x = (And\text{-}mtl\ \varphi\ \psi, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ a\ b\ num\text{-}vars. x = (Future\text{-}mtl\ \varphi\ a\ b, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ a\ b\ num\text{-}vars. x = (Global\text{-}mtl\ \varphi\ a\ b, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars. x = (Until\text{-}mtl\ \varphi\ \psi\ a\ b, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars. x = (Release\text{-}mtl\ \varphi\ \psi\ a\ b, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge num\text{-}vars. x = (Not\text{-}mtl\ True\text{-}mtl, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge num\text{-}vars. x = (Not\text{-}mtl\ False\text{-}mtl, num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ num\text{-}vars. x = (Not\text{-}mtl (And\text{-}mtl\ \varphi\ \psi), num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ num\text{-}vars. x = (Not\text{-}mtl (Or\text{-}mtl\ \varphi\ \psi), num\text{-}vars) \Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ a\ b\ num\text{-}vars. x = (Not\text{-}mtl (Future\text{-}mtl\ \varphi\ a\ b), num\text{-}vars) \Rightarrow P)$
 \Rightarrow
 $(\bigwedge \varphi\ a\ b\ num\text{-}vars. x = (Not\text{-}mtl (Global\text{-}mtl\ \varphi\ a\ b), num\text{-}vars) \Rightarrow P)$
 \Rightarrow
 $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars. x = (Not\text{-}mtl (Until\text{-}mtl\ \varphi\ \psi\ a\ b), num\text{-}vars) \Rightarrow$
 $P) \Rightarrow$
 $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars. x = (Not\text{-}mtl (Release\text{-}mtl\ \varphi\ \psi\ a\ b), num\text{-}vars)$
 $\Rightarrow P) \Rightarrow$
 $(\bigwedge \varphi\ num\text{-}vars. x = (Not\text{-}mtl (Not\text{-}mtl\ \varphi), num\text{-}vars) \Rightarrow P) \Rightarrow P$
<proof>

fun *WEST-termination-measure*:: (*nat*) *mtl* \Rightarrow *nat*
where *WEST-termination-measure* *True_m* = 1
| *WEST-termination-measure* (*Not_m True_m*) = 4
| *WEST-termination-measure* *False_m* = 1
| *WEST-termination-measure* (*Not_m False_m*) = 4
| *WEST-termination-measure* (*Prop_m (p)*) = 1
| *WEST-termination-measure* (*Not_m (Prop_m (p))*) = 4

| *WEST-termination-measure* (φ *Or_m* ψ) = 1 + (*WEST-termination-measure* φ) + (*WEST-termination-measure* ψ)
 | *WEST-termination-measure* (φ *And_m* ψ) = 1 + (*WEST-termination-measure* φ) + (*WEST-termination-measure* ψ)
 | *WEST-termination-measure* (*F_m* [a,b] φ) = 1 + (*WEST-termination-measure* φ)
 | *WEST-termination-measure* (*G_m* [a,b] φ) = 1 + (*WEST-termination-measure* φ)
 | *WEST-termination-measure* (φ *U_m*[a,b] ψ) = 1 + (*WEST-termination-measure* φ) + (*WEST-termination-measure* ψ)
 | *WEST-termination-measure* (φ *R_m*[a,b] ψ) = 1 + (*WEST-termination-measure* φ) + (*WEST-termination-measure* ψ)
 | *WEST-termination-measure* (*Not_m* (φ *Or_m* ψ)) = 1 + 3 * (*WEST-termination-measure* (φ *Or_m* ψ))
 | *WEST-termination-measure* (*Not_m* (φ *And_m* ψ)) = 1 + 3 * (*WEST-termination-measure* (φ *And_m* ψ))
 | *WEST-termination-measure* (*Not_m* (*F_m*[a,b] φ)) = 1 + 3 * (*WEST-termination-measure* (*F_m*[a,b] φ))
 | *WEST-termination-measure* (*Not_m* (*G_m*[a,b] φ)) = 1 + 3 * (*WEST-termination-measure* (*G_m*[a,b] φ))
 | *WEST-termination-measure* (*Not_m* (φ *U_m*[a,b] ψ)) = 1 + 3 * (*WEST-termination-measure* (φ *U_m*[a,b] ψ))
 | *WEST-termination-measure* (*Not_m* (φ *R_m*[a,b] ψ)) = 1 + 3 * (*WEST-termination-measure* (φ *R_m*[a,b] ψ))
 | *WEST-termination-measure* (*Not_m* (*Not_m* φ)) = 1 + 3 * (*WEST-termination-measure* (*Not_m* φ))

lemma *WEST-termination-measure-not*:

fixes $\varphi::(\text{nat}) \text{ mltl}$

shows *WEST-termination-measure* (*Not-mltl* φ) = 1 + 3 * (*WEST-termination-measure* φ)

<proof>

function *WEST-reg-aux*:: (*nat*) *mltl* \Rightarrow *nat* \Rightarrow *WEST-regex*

where *WEST-reg-aux* *True_m* *num-vars* = [[(*map* ($\lambda j. S$) [0 ..< *num-vars*])]]

| *WEST-reg-aux* *False_m* *num-vars* = []

| *WEST-reg-aux* (*Prop_m* (p)) *num-vars* = [[(*map* ($\lambda j. (\text{if } (p=j) \text{ then } \text{One} \text{ else } S)$) [0 ..< *num-vars*])]]

| *WEST-reg-aux* (*Not_m* (*Prop_m* (p))) *num-vars* = [[(*map* ($\lambda j. (\text{if } (p=j) \text{ then } \text{Zero} \text{ else } S)$) [0 ..< *num-vars*])]]

| *WEST-reg-aux* (φ *Or_m* ψ) *num-vars* = *WEST-or-simp* (*WEST-reg-aux* φ *num-vars*) (*WEST-reg-aux* ψ *num-vars*) *num-vars*

| *WEST-reg-aux* (φ *And_m* ψ) *num-vars* = (*WEST-and-simp* (*WEST-reg-aux* φ *num-vars*) (*WEST-reg-aux* ψ *num-vars*) *num-vars*)

| *WEST-reg-aux* (*F_m*[a,b] φ) *num-vars* = (*WEST-future* (*WEST-reg-aux* φ *num-vars*) *a b num-vars*)

| *WEST-reg-aux* (*G_m*[a,b] φ) *num-vars* = (*WEST-global* (*WEST-reg-aux* φ *num-vars*) *a b num-vars*)

| *WEST-reg-aux* (φ $U_m[a,b]$ ψ) *num-vars* = (*WEST-until* (*WEST-reg-aux* φ *num-vars*) (*WEST-reg-aux* ψ *num-vars*) a b *num-vars*)
 | *WEST-reg-aux* (φ $R_m[a,b]$ ψ) *num-vars* = *WEST-release* (*WEST-reg-aux* φ *num-vars*) (*WEST-reg-aux* ψ *num-vars*) a b *num-vars*
 | *WEST-reg-aux* (*Not_m* *True_m*) *num-vars* = *WEST-reg-aux* *False_m* *num-vars*
 | *WEST-reg-aux* (*Not_m* *False_m*) *num-vars* = *WEST-reg-aux* *True_m* *num-vars*
 | *WEST-reg-aux* (*Not_m* (φ *And_m* ψ)) *num-vars* = *WEST-reg-aux* ((*Not_m* φ) *Or_m* (*Not_m* ψ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* (φ *Or_m* ψ)) *num-vars* = *WEST-reg-aux* ((*Not_m* φ) *And_m* (*Not_m* ψ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* ($F_m[a,b]$ φ)) *num-vars* = *WEST-reg-aux* ($G_m[a,b]$ (*Not_m* φ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* ($G_m[a,b]$ φ)) *num-vars* = *WEST-reg-aux* ($F_m[a,b]$ (*Not_m* φ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* (φ $U_m[a,b]$ ψ)) *num-vars* = *WEST-reg-aux* ((*Not_m* φ) $R_m[a,b]$ (*Not_m* ψ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* (φ $R_m[a,b]$ ψ)) *num-vars* = *WEST-reg-aux* ((*Not_m* φ) $U_m[a,b]$ (*Not_m* ψ)) *num-vars*
 | *WEST-reg-aux* (*Not_m* (*Not_m* φ)) *num-vars* = *WEST-reg-aux* φ *num-vars*
 <proof>
termination
 <proof>

fun *WEST-num-vars*:: (*nat*) *mltl* \Rightarrow *nat*
where *WEST-num-vars* *True_m* = 1
 | *WEST-num-vars* *False_m* = 1
 | *WEST-num-vars* (*Prop_m* (p)) = $p+1$
 | *WEST-num-vars* (*Not_m* φ) = *WEST-num-vars* φ
 | *WEST-num-vars* (φ *And_m* ψ) = *Max* {(*WEST-num-vars* φ), (*WEST-num-vars* ψ)}
 | *WEST-num-vars* (φ *Or_m* ψ) = *Max* {(*WEST-num-vars* φ), (*WEST-num-vars* ψ)}
 | *WEST-num-vars* ($F_m[a,b]$ φ) = *WEST-num-vars* φ
 | *WEST-num-vars* ($G_m[a,b]$ φ) = *WEST-num-vars* φ
 | *WEST-num-vars* (φ $U_m[a,b]$ ψ) = *Max* {(*WEST-num-vars* φ), (*WEST-num-vars* ψ)}
 | *WEST-num-vars* (φ $R_m[a,b]$ ψ) = *Max* {(*WEST-num-vars* φ), (*WEST-num-vars* ψ)}

fun *WEST-reg*:: (*nat*) *mltl* \Rightarrow *WEST-regex*
where *WEST-reg* F = (*let* *nnf-F* = *convert-nnf* F *in* *WEST-reg-aux* *nnf-F* (*WEST-num-vars* F))

1.3.7 Adding padding

fun *pad-WEST-reg*:: *nat* *mltl* \Rightarrow *WEST-regex*
where *pad-WEST-reg* φ = (*let* *unpadded* = *WEST-reg* φ *in*

```

      (let complen = complen-mltl  $\varphi$  in
        (let num-vars = WEST-num-vars  $\varphi$  in
          (map ( $\lambda$  L. (if (length L < complen) then (pad L num-vars
            (complen-(length L))) else L))) unpadded)))

```

```

fun simp-pad-WEST-reg:: nat mltl  $\Rightarrow$  WEST-regex
  where simp-pad-WEST-reg  $\varphi$  = WEST-simp (pad-WEST-reg  $\varphi$ ) (WEST-num-vars
 $\varphi$ )

```

2 Some examples and Code Export

Base cases

```

value WEST-reg Truem
value WEST-reg Falsem
value WEST-reg (Propm (1))
value WEST-reg (Notm (Propm (0)))

```

Test cases for recursion

```

value WEST-reg ((Notm (Propm (0))) Andm (Propm (1)))
value WEST-reg (Fm[0,2] (Propm (1)))
value WEST-reg ((Notm (Propm (0))) Orm (Propm (0)))

value pad-WEST-reg ((Propm (0)) Um[0,2] (Propm (0)))
value simp-pad-WEST-reg ((Prop-mltl 0) Um[0,2] (Prop-mltl 0))

```

```

export-code WEST-reg in Haskell module-name WEST
export-code simp-pad-WEST-reg in Haskell module-name WEST-simp-pad

```

end

3 WEST Proofs

theory WEST-Proofs

imports WEST-Algorithms

begin

3.1 Useful Definitions

```

definition trace-of-vars::trace  $\Rightarrow$  nat  $\Rightarrow$  bool
  where trace-of-vars trace num-vars = (
     $\forall k. (k < (\text{length trace}) \longrightarrow (\forall p \in (\text{trace!}k). p < \text{num-vars}))$ )

```

```

definition state-regex-of-vars::state-regex  $\Rightarrow$  nat  $\Rightarrow$  bool
  where state-regex-of-vars state num-vars = ((length state) = num-vars)

```

definition *trace-regex-of-vars*::*trace-regex* \Rightarrow *nat* \Rightarrow *bool*
where *trace-regex-of-vars* *trace num-vars* =
 $(\forall i < (\text{length } \text{trace}). \text{length } (\text{trace}!i) = \text{num-vars})$

definition *WEST-regex-of-vars*::*WEST-regex* \Rightarrow *nat* \Rightarrow *bool*
where *WEST-regex-of-vars* *traceList num-vars* =
 $(\forall k < \text{length } \text{traceList}. \text{trace-regex-of-vars } (\text{traceList}!k) \text{ num-vars})$

3.2 Proofs about Traces Matching Regular Expressions

value *match-regex* $[\{0::\text{nat}\}, \{0,1\}, \{\}] \square$

lemma *arbitrary-regex-matches-any-trace*:

fixes *num-vars*::*nat*

fixes π ::*trace*

assumes π -of-num-vars: *trace-of-vars* π *num-vars*

shows *match-regex* $\pi \square$

<proof>

lemma *WEST-and-state-difflengths-is-none*:

assumes *length* *s1* \neq *length* *s2*

shows *WEST-and-state* *s1 s2* = *None*

<proof>

3.3 Facts about the WEST and operator

3.3.1 Commutative

lemma *WEST-and-bitwise-commutative*:

fixes *b1 b2*::*WEST-bit*

shows *WEST-and-bitwise* *b1 b2* = *WEST-and-bitwise* *b2 b1*

<proof>

fun *remove-option-type-bit*::*WEST-bit option* \Rightarrow *WEST-bit*

where *remove-option-type-bit* (*Some i*) = *i*

| *remove-option-type-bit* - = *S*

lemma *WEST-and-state-commutative*:

fixes *s1 s2*::*state-regex*

assumes *same-len*: *length* *s1* = *length* *s2*

shows *WEST-and-state* *s1 s2* = *WEST-and-state* *s2 s1*

<proof>

lemma *WEST-and-trace-commutative*:

fixes *num-vars*::*nat*

fixes *regtrace1*::*trace-regex*

fixes *regtrace2*::*trace-regex*

assumes *regtrace1-of-num-vars*: *trace-regex-of-vars* *regtrace1 num-vars*

assumes *regtrace2-of-num-vars: trace-regex-of-vars regtrace2 num-vars*
shows $(\text{WEST-and-trace } \text{regtrace1 } \text{regtrace2}) = (\text{WEST-and-trace } \text{regtrace2 } \text{regtrace1})$
 $\langle \text{proof} \rangle$

lemma *WEST-and-helper-subset:*
shows $\text{set } (\text{WEST-and-helper } h \ L) \subseteq \text{set } (\text{WEST-and-helper } h \ (a \ \# \ L))$
 $\langle \text{proof} \rangle$

lemma *WEST-and-helper-set-member-converse:*
fixes *regtrace h::trace-regex*
fixes *L::WEST-regex*
assumes *assumption: $(\exists \text{ loc. } \text{loc} < \text{length } L \wedge (\exists \text{ sometrace. } \text{WEST-and-trace } h \ (L \ ! \ \text{loc}) = \text{Some } \text{sometrace} \wedge \text{regtrace} = \text{sometrace}))$*
shows $\text{regtrace} \in \text{set } (\text{WEST-and-helper } h \ L)$
 $\langle \text{proof} \rangle$

lemma *WEST-and-helper-set-member-forward:*
fixes *regtrace h::trace-regex*
fixes *L::WEST-regex*
assumes $\text{regtrace} \in \text{set } (\text{WEST-and-helper } h \ L)$
shows $(\exists \text{ loc. } \text{loc} < \text{length } L \wedge (\exists \text{ sometrace. } \text{WEST-and-trace } h \ (L \ ! \ \text{loc}) = \text{Some } \text{sometrace} \wedge \text{regtrace} = \text{sometrace}))$
 $\langle \text{proof} \rangle$

lemma *WEST-and-helper-set-member:*
fixes *regtrace h::trace-regex*
fixes *L::WEST-regex*
shows $\text{regtrace} \in \text{set } (\text{WEST-and-helper } h \ L) \iff (\exists \text{ loc. } \text{loc} < \text{length } L \wedge (\exists \text{ sometrace. } \text{WEST-and-trace } h \ (L \ ! \ \text{loc}) = \text{Some } \text{sometrace} \wedge \text{regtrace} = \text{sometrace}))$
 $\langle \text{proof} \rangle$

lemma *WEST-and-set-member-dir1:*
fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*
assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
assumes $\text{regtrace} \in \text{set } (\text{WEST-and } L1 \ L2)$
shows $(\exists \text{ loc1 } \text{loc2. } \text{loc1} < \text{length } L1 \wedge \text{loc2} < \text{length } L2 \wedge (\exists \text{ sometrace. } \text{WEST-and-trace } (L1 \ ! \ \text{loc1}) \ (L2 \ ! \ \text{loc2}) = \text{Some } \text{sometrace} \wedge \text{regtrace} = \text{sometrace}))$
 $\langle \text{proof} \rangle$

lemma *WEST-and-subset:*
shows $\text{set } (\text{WEST-and } T1 \ L2) \subseteq \text{set } (\text{WEST-and } (h1 \ \# \ T1) \ L2)$

<proof>

lemma *WEST-and-set-member-dir2:*

fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*
assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
assumes *exists-locs: (\exists loc1 loc2. loc1 < length L1 \wedge loc2 < length L2 \wedge
(\exists sometrace. WEST-and-trace (L1 ! loc1) (L2 ! loc2) = Some sometrace \wedge
regtrace = sometrace))*
shows *regtrace \in set (WEST-and L1 L2) <proof>*

lemma *WEST-and-set-member:*

fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*
assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
shows *regtrace \in set (WEST-and L1 L2) \longleftrightarrow
(\exists loc1 loc2. loc1 < length L1 \wedge loc2 < length L2 \wedge
(\exists sometrace. WEST-and-trace (L1 ! loc1) (L2 ! loc2) = Some sometrace \wedge
regtrace = sometrace))*
<proof>

lemma *WEST-and-commutative-sets-member:*

fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*
assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
assumes *regtrace-in: regtrace \in set (WEST-and L1 L2)*
shows *regtrace \in set (WEST-and L2 L1)*
<proof>

lemma *WEST-and-commutative-sets:*

fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*
assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
shows *set (WEST-and L1 L2) = set (WEST-and L2 L1)*
<proof>

lemma *WEST-and-commutative:*

fixes *num-vars::nat*
fixes *L1::WEST-regex*
fixes *L2::WEST-regex*
assumes *L1-of-num-vars: WEST-regex-of-vars L1 num-vars*

assumes *L2-of-num-vars: WEST-regex-of-vars L2 num-vars*
shows *regex-equiv (WEST-and L1 L2) (WEST-and L2 L1)*
 ⟨*proof*⟩

3.3.2 Identity and Zero

lemma *WEST-and-helper-identity:*
shows *WEST-and-helper [] trace = trace*
 ⟨*proof*⟩

lemma *WEST-and-identity: WEST-and [[]] L = L*
 ⟨*proof*⟩

lemma *WEST-and-zero: WEST-and L [] = []*
 ⟨*proof*⟩

3.3.3 WEST-and-state

Well Defined **fun** *advance-state:: state ⇒ state*
where *advance-state state = {x-1 | x. (x∈state ∧ x ≠ 0)}*

lemma *advance-state-elt-bound:*
fixes *state::state*
fixes *num-vars::nat*
assumes $\forall x \in \text{state}. x < \text{num-vars}$
shows $\forall x \in (\text{advance-state state}). x < (\text{num-vars} - 1)$
 ⟨*proof*⟩

lemma *advance-state-elt-member:*
fixes *state::state*
fixes *x::nat*
assumes $x + 1 \in \text{state}$
shows $x \in \text{advance-state state}$
 ⟨*proof*⟩

lemma *advance-state-match-timestep:*
fixes *h::WEST-bit*
fixes *t::state-regex*
fixes *state::state*
assumes *match-timestep state (h#t)*
shows *match-timestep (advance-state state) t*
 ⟨*proof*⟩

lemma *WEST-and-state-well-defined:*
fixes *num-vars::nat*
fixes *state::state*
fixes *s1 s2:: state-regex*
assumes *s1-of-num-vars: state-regex-of-vars s1 num-vars*

assumes $s2\text{-of-num-vars}$: $state\text{-regex-of-vars } s2 \text{ num-vars}$
assumes $\pi\text{-match-r1-r2}$: $match\text{-timestep state } s1 \wedge match\text{-timestep state } s2$
shows $WEST\text{-and-state } s1 \ s2 \neq None$
 $\langle proof \rangle$

Correct Forward lemma $WEST\text{-and-state-length}$:

fixes $s1 \ s2::state\text{-regex}$
assumes $same\text{len}$: $length \ s1 = length \ s2$
assumes $r\text{-exists}$: $(WEST\text{-and-state } s1 \ s2) \neq None$
shows $\exists r. length \ r = length \ s1 \wedge WEST\text{-and-state } s1 \ s2 = Some \ r$
 $\langle proof \rangle$

lemma $index\text{-shift}$:

fixes $a::WEST\text{-bit}$
fixes $t::state\text{-regex}$
fixes $state::state$
assumes $(a = One \longrightarrow 0 \in state) \wedge (a = Zero \longrightarrow 0 \notin state)$
assumes $\forall x < length \ t. ((t!x) = One \longrightarrow x + 1 \in state) \wedge ((t!x) = Zero \longrightarrow x + 1 \notin state)$
shows $\forall x < length \ (a\#t). ((a\#t) ! x = One \longrightarrow x \in state) \wedge ((a\#t) ! x = Zero \longrightarrow x \notin state)$
 $\langle proof \rangle$

lemma $index\text{-shift-reverse}$:

fixes $a::WEST\text{-bit}$
fixes $t::state\text{-regex}$
fixes $state::state$
assumes $\forall x < length \ (a\#t). ((a\#t) ! x = One \longrightarrow x \in state) \wedge ((a\#t) ! x = Zero \longrightarrow x \notin state)$
shows $\forall x < length \ t. ((t!x) = One \longrightarrow x + 1 \in state) \wedge ((t!x) = Zero \longrightarrow x + 1 \notin state)$
 $\langle proof \rangle$

lemma $WEST\text{-and-state-correct-forward}$:

fixes $num\text{-vars}::nat$
fixes $state::state$
fixes $s1 \ s2::state\text{-regex}$
assumes $s1\text{-of-num-vars}$: $state\text{-regex-of-vars } s1 \ num\text{-vars}$
assumes $s2\text{-of-num-vars}$: $state\text{-regex-of-vars } s2 \ num\text{-vars}$
assumes $match\text{-both}$: $match\text{-timestep state } s1 \wedge match\text{-timestep state } s2$
shows $\exists somestate. (match\text{-timestep state } somestate) \wedge (WEST\text{-and-state } s1 \ s2) = Some \ somestate$
 $\langle proof \rangle$

Correct Converse lemma $WEST\text{-and-state-indices}$:

fixes $s \ s1 \ s2::state\text{-regex}$

assumes *WEST-and-state* $s1\ s2 = \text{Some } s$
assumes $\text{length } s1 = \text{length } s2$
assumes $x < \text{length } s$
shows $\text{Some } (s!x) = \text{WEST-and-bitwise } (s1!x) (s2!x)$
<proof>

lemma *WEST-and-state-correct-converse-s1*:
fixes $\text{num-vars}::\text{nat}$
fixes $\text{state}::\text{state}$
fixes $s1\ s2::\text{state-regex}$
assumes $s1\text{-of-num-vars}: \text{state-regex-of-vars } s1\ \text{num-vars}$
assumes $s2\text{-of-num-vars}: \text{state-regex-of-vars } s2\ \text{num-vars}$
assumes $\text{match-and}: \exists \text{somestate}. (\text{match-timestep } \text{state } \text{somestate}) \wedge (\text{WEST-and-state } s1\ s2) = \text{Some } \text{somestate}$
shows $\text{match-timestep } \text{state } s1$
<proof>

lemma *WEST-and-state-correct-converse*:
fixes $\text{num-vars}::\text{nat}$
fixes $\text{state}::\text{state}$
fixes $s1\ s2::\text{state-regex}$
assumes $s1\text{-of-num-vars}: \text{state-regex-of-vars } s1\ \text{num-vars}$
assumes $s2\text{-of-num-vars}: \text{state-regex-of-vars } s2\ \text{num-vars}$
assumes $\text{match-and}: \exists \text{somestate}. (\text{match-timestep } \text{state } \text{somestate}) \wedge (\text{WEST-and-state } s1\ s2) = \text{Some } \text{somestate}$
shows $\text{match-timestep } \text{state } s1 \wedge \text{match-timestep } \text{state } s2$
<proof>

lemma *WEST-and-state-correct*:
fixes $\text{num-vars}::\text{nat}$
fixes $\text{state}::\text{state}$
fixes $s1\ s2::\text{state-regex}$
assumes $s1\text{-of-num-vars}: \text{state-regex-of-vars } s1\ \text{num-vars}$
assumes $s2\text{-of-num-vars}: \text{state-regex-of-vars } s2\ \text{num-vars}$
shows $(\text{match-timestep } \text{state } s1 \wedge \text{match-timestep } \text{state } s2) \longleftrightarrow (\exists \text{somestate}. \text{match-timestep } \text{state } \text{somestate} \wedge (\text{WEST-and-state } s1\ s2) = \text{Some } \text{somestate})$
<proof>

3.3.4 WEST-and-trace

Well Defined lemma *WEST-and-trace-well-defined*:
fixes $\text{num-vars}::\text{nat}$
fixes $\pi::\text{trace}$
fixes $r1\ r2::\text{trace-regex}$
assumes $r1\text{-of-num-vars}: \text{trace-regex-of-vars } r1\ \text{num-vars}$
assumes $r2\text{-of-num-vars}: \text{trace-regex-of-vars } r2\ \text{num-vars}$
assumes $\pi\text{-match-r1-r2}: \text{match-regex } \pi\ r1 \wedge \text{match-regex } \pi\ r2$
shows $\text{WEST-and-trace } r1\ r2 \neq \text{None}$

<proof>

Correct Forward lemma *WEST-and-trace-correct-forward-aux:*

assumes *match-regex* π ($h\#t$)

shows *match-timestep* ($\pi!0$) $h \wedge$ *match-regex* (*drop* 1 π) t

<proof>

lemma *WEST-and-trace-correct-forward-aux-converse:*

assumes $\pi = hxi\#txi$

assumes *match-timestep* (hxi) h

assumes *match-regex* txi t

shows *match-regex* π ($h\#t$)

<proof>

lemma *WEST-and-trace-correct-forward-empty-trace:*

fixes *num-vars::nat*

fixes $\pi::trace$

fixes $r1\ r2::trace\text{-regex}$

assumes *r1-of-num-vars: trace-regex-of-vars* $r1$ *num-vars*

assumes *r2-of-num-vars: trace-regex-of-vars* $r2$ *num-vars*

assumes *match1: match-regex* \square $r1$

assumes *match2: match-regex* \square $r2$

shows \exists *sometraces. match-regex* \square *sometraces* \wedge (*WEST-and-trace* $r1\ r2$) = *Some* *sometraces*

<proof>

lemma *WEST-and-trace-correct-forward-nonempty-trace:*

fixes *num-vars::nat*

fixes $\pi::trace$

fixes $r1\ r2::trace\text{-regex}$

assumes *r1-of-num-vars: trace-regex-of-vars* $r1$ *num-vars*

assumes *r2-of-num-vars: trace-regex-of-vars* $r2$ *num-vars*

assumes *match-regex* π $r1 \wedge$ *match-regex* π $r2$

assumes *length* $\pi > 0$

shows \exists *sometraces. match-regex* π *sometraces* \wedge (*WEST-and-trace* $r1\ r2$) = *Some* *sometraces*

<proof>

lemma *WEST-and-trace-correct-forward:*

fixes *num-vars::nat*

fixes $\pi::trace$

fixes $r1\ r2::trace\text{-regex}$

assumes *r1-of-num-vars: trace-regex-of-vars* $r1$ *num-vars*

assumes *r2-of-num-vars: trace-regex-of-vars* $r2$ *num-vars*

assumes *match-regex* π $r1 \wedge$ *match-regex* π $r2$

shows \exists *sometraces. match-regex* π *sometraces* \wedge (*WEST-and-trace* $r1\ r2$) = *Some* *sometraces*

<proof>

Correct Converse lemma *WEST-and-trace-nonempty-args:*
fixes $h1\ h2::state\ regex$
fixes $t\ t1\ t2::trace\ regex$
assumes *WEST-and-trace* $(h1\ \# \ t1)\ (h2\ \# \ t2) = \text{Some}\ (h\ \# \ t)$
shows *WEST-and-state* $h1\ h2 = \text{Some}\ h \wedge \text{WEST-and-trace}\ t1\ t2 = \text{Some}\ t$
 $\langle proof \rangle$

lemma *WEST-and-trace-lengths-r1:*
assumes *trace-regex-of-vars* $r1\ n$
assumes *trace-regex-of-vars* $r2\ n$
assumes $(\text{WEST-and-trace}\ r1\ r2) = \text{Some}\ \text{sometrace}$
shows $\text{length}\ \text{sometrace} \geq \text{length}\ r1$
 $\langle proof \rangle$

lemma *WEST-and-trace-lengths:*
assumes *trace-regex-of-vars* $r1\ n$
assumes *trace-regex-of-vars* $r2\ n$
assumes $(\text{WEST-and-trace}\ r1\ r2) = \text{Some}\ \text{sometrace}$
shows $\text{length}\ \text{sometrace} \geq \text{length}\ r1 \wedge \text{length}\ \text{sometrace} \geq \text{length}\ r2$
 $\langle proof \rangle$

lemma *WEST-and-trace-correct-converse-r1:*
fixes $num\ vars::nat$
fixes $\pi::trace$
fixes $r1\ r2::trace\ regex$
assumes *r1-of-num-vars: trace-regex-of-vars* $r1\ num\ vars$
assumes *r2-of-num-vars: trace-regex-of-vars* $r2\ num\ vars$
assumes $(\exists\ \text{sometrace}.\ \text{match-regex}\ \pi\ \text{sometrace} \wedge (\text{WEST-and-trace}\ r1\ r2) = \text{Some}\ \text{sometrace})$
shows $\text{match-regex}\ \pi\ r1$
 $\langle proof \rangle$

lemma *WEST-and-trace-correct-converse:*
fixes $num\ vars::nat$
fixes $\pi::trace$
fixes $r1\ r2::trace\ regex$
assumes *r1-of-num-vars: trace-regex-of-vars* $r1\ num\ vars$
assumes *r2-of-num-vars: trace-regex-of-vars* $r2\ num\ vars$
assumes $(\exists\ \text{sometrace}.\ \text{match-regex}\ \pi\ \text{sometrace} \wedge (\text{WEST-and-trace}\ r1\ r2) = \text{Some}\ \text{sometrace})$
shows $\text{match-regex}\ \pi\ r1 \wedge \text{match-regex}\ \pi\ r2$
 $\langle proof \rangle$

lemma *WEST-and-trace-correct:*
fixes $num\ vars::nat$
fixes $\pi::trace$
fixes $r1\ r2::trace\ regex$
assumes *r1-of-num-vars: trace-regex-of-vars* $r1\ num\ vars$

assumes $r2\text{-of-num-vars: trace-regex-of-vars } r2 \text{ num-vars}$
shows $\text{match-regex } \pi \ r1 \wedge \text{match-regex } \pi \ r2 \longleftrightarrow (\exists \text{ sometrace. match-regex } \pi \text{ sometrace} \wedge (\text{WEST-and-trace } r1 \ r2) = \text{Some sometrace})$
 <proof>

3.3.5 WEST-and correct

Correct Forward lemma *WEST-and-helper-subset-of-WEST-and:*

assumes $\text{List.member } L1 \ \text{elem}$
shows $\text{set } (\text{WEST-and-helper } \text{elem } (h2\#T2)) \subseteq \text{set } (\text{WEST-and } L1 \ (h2\#T2))$
 <proof>

lemma *WEST-and-trace-element-of-WEST-and-helper:*

assumes $\text{List.member } L2 \ \text{elem2}$
assumes $(\text{WEST-and-trace } \text{elem1 } \text{elem2}) = \text{Some sometrace}$
shows $\text{sometrace} \in \text{set } (\text{WEST-and-helper } \text{elem1 } L2)$
 <proof>

lemma *index-of-L-in-L:*

assumes $i < \text{length } L$
shows $\text{List.member } L \ (L ! i)$
 <proof>

lemma *WEST-and-indices:*

fixes $L1 \ L2::\text{WEST-regex}$
fixes $\text{sometrace}::\text{trace-regex}$
assumes $\exists i1 \ i2. i1 < \text{length } L1 \wedge i2 < \text{length } L2 \wedge \text{WEST-and-trace } (L1 ! i1) \ (L2 ! i2) = \text{Some sometrace}$
shows $\exists i < \text{length } (\text{WEST-and } L1 \ L2). \text{WEST-and } L1 \ L2 ! i = \text{sometrace}$
 <proof>

lemma *WEST-and-correct-forward:*

fixes $n::\text{nat}$
fixes $\pi::\text{trace}$
fixes $L1 \ L2::\text{WEST-regex}$
assumes $L1\text{-of-num-vars: WEST-regex-of-vars } L1 \ n$
assumes $L2\text{-of-num-vars: WEST-regex-of-vars } L2 \ n$
assumes $\text{match } \pi \ L1 \wedge \text{match } \pi \ L2$
shows $\text{match } \pi \ (\text{WEST-and } L1 \ L2)$
 <proof>

Correct Converse lemma *WEST-and-correct-converse-L1:*

fixes $n::\text{nat}$
fixes $\pi::\text{trace}$
fixes $L1 \ L2::\text{WEST-regex}$
assumes $L1\text{-of-num-vars: WEST-regex-of-vars } L1 \ n$
assumes $L2\text{-of-num-vars: WEST-regex-of-vars } L2 \ n$
assumes $\text{match } \pi \ (\text{WEST-and } L1 \ L2)$
shows $\text{match } \pi \ L1$

<proof>

lemma *WEST-and-correct-converse:*

fixes $n::nat$
fixes $\pi::trace$
fixes $L1\ L2::WEST\text{-}regex$
assumes $L1\text{-of-num-vars: } WEST\text{-}regex\text{-of-vars } L1\ n$
assumes $L2\text{-of-num-vars: } WEST\text{-}regex\text{-of-vars } L2\ n$
assumes $match\ \pi\ (WEST\text{-}and\ L1\ L2)$
shows $match\ \pi\ L1 \wedge match\ \pi\ L2$

<proof>

lemma *WEST-and-correct:*

fixes $\pi::trace$
fixes $L1\ L2::WEST\text{-}regex$
assumes $L1\text{-of-num-vars: } WEST\text{-}regex\text{-of-vars } L1\ n$
assumes $L2\text{-of-num-vars: } WEST\text{-}regex\text{-of-vars } L2\ n$
shows $match\ \pi\ L1 \wedge match\ \pi\ L2 \longleftrightarrow match\ \pi\ (WEST\text{-}and\ L1\ L2)$

<proof>

3.4 Facts about the WEST or operator

lemma *WEST-or-correct:*

fixes $\pi::trace$
fixes $L1\ L2::WEST\text{-}regex$
shows $match\ \pi\ (L1@L2) \longleftrightarrow (match\ \pi\ L1) \vee (match\ \pi\ L2)$

<proof>

3.5 Pad and Match Facts

lemma *shift-match-regex:*

assumes $length\ \pi \geq a$
assumes $match\text{-}regex\ \pi\ ((arbitrary\text{-}trace\ num\text{-}vars\ a)@L)$
shows $match\text{-}regex\ (drop\ a\ \pi)\ (drop\ a\ ((arbitrary\text{-}trace\ num\text{-}vars\ a)@L))$

<proof>

lemma *match-regex:*

assumes $length\ \pi \geq a$
assumes $length\ L1 = a$
assumes $match\text{-}regex\ \pi\ (L1@L2)$
shows $match\text{-}regex\ (drop\ a\ \pi)\ (drop\ a\ (L1@L2))$

<proof>

lemma *match-regex-converse:*

assumes $length\ \pi \geq a$
assumes $L1 = (arbitrary\text{-}trace\ num\text{-}vars\ a)$
assumes $match\text{-}regex\ (drop\ a\ \pi)\ (drop\ a\ (L1@L2))$

shows *match-regex* π ($L1@L2$)
(*proof*)

lemma *shift-match*:
assumes *length* $\pi \geq a$
assumes *match* π (*shift* L *num-vars* a)
shows *match* (*drop* a π) L
(*proof*)

lemma *shift-match-converse*:
assumes *length* $\pi \geq a$
assumes *match* (*drop* a π) L
shows *match* π (*shift* L *num-vars* a)
(*proof*)

lemma *pad-zero*:
shows *shift* $L2$ *num-vars* $0 = L2$
(*proof*)

3.6 Facts about WEST num vars

lemma *retrace-append*:
assumes *trace-regex-of-vars* $L1$ k
assumes *trace-regex-of-vars* $L2$ k
shows *trace-regex-of-vars* ($L1@L2$) k
(*proof*)

lemma *WEST-num-vars-subformulas*:
assumes $G \in$ *subformulas* F
shows (*WEST-num-vars* F) \geq *WEST-num-vars* G
(*proof*)

lemma *WEST-num-vars-nnf*:
shows (*WEST-num-vars* φ) = *WEST-num-vars* (*convert-nnf* φ)
(*proof*)

3.6.1 Facts about num vars for different WEST operators

lemma *length-WEST-and*:
assumes *length* $state1 = k$
assumes *length* $state2 = k$
assumes *WEST-and-state* $state1$ $state2 =$ *Some state*
shows *length* $state = k$
(*proof*)

lemma *WEST-and-trace-num-vars*:
assumes *trace-regex-of-vars* $r1$ k
assumes *trace-regex-of-vars* $r2$ k

assumes (*WEST-and-trace* $r1\ r2$) = *Some sometrace*
shows *trace-regex-of-vars sometrace* k
(*proof*)

lemma *WEST-and-num-vars*:
assumes *WEST-regex-of-vars* $L1\ k$
assumes *WEST-regex-of-vars* $L2\ k$
shows *WEST-regex-of-vars* (*WEST-and* $L1\ L2$) k
(*proof*)

lemma *WEST-or-num-vars*:
assumes $L1\text{-nv}$: *WEST-regex-of-vars* $L1\ k$
assumes $L2\text{-nv}$: *WEST-regex-of-vars* $L2\ k$
shows *WEST-regex-of-vars* ($L1@L2$) k
(*proof*)

lemma *regtraceList-cons-num-vars*:
assumes *trace-regex-of-vars* $h\ num\text{-vars}$
assumes *WEST-regex-of-vars* $T\ num\text{-vars}$
shows *WEST-regex-of-vars* ($h\#T$) $num\text{-vars}$
(*proof*)

lemma *WEST-simp-state-num-vars*:
assumes $length\ s1 = num\text{-vars}$
assumes $length\ s2 = num\text{-vars}$
shows $length\ (WEST\text{-simp}\text{-state}\ s1\ s2) = num\text{-vars}$
(*proof*)

lemma *WEST-get-state-length*:
assumes *trace-regex-of-vars* $r\ num\text{-vars}$
shows $length\ (WEST\text{-get}\text{-state}\ r\ k\ num\text{-vars}) = num\text{-vars}$
(*proof*)

lemma *WEST-simp-trace-num-vars*:
assumes *trace-regex-of-vars* $r1\ num\text{-vars}$
assumes *trace-regex-of-vars* $r2\ num\text{-vars}$
shows *trace-regex-of-vars* (*WEST-simp-trace* $r1\ r2\ num\text{-vars}$) $num\text{-vars}$
(*proof*)

lemma *remove-element-at-index-preserves-nv*:
assumes $i < length\ L$
assumes *WEST-regex-of-vars* $L\ num\text{-vars}$
shows *WEST-regex-of-vars* (*remove-element-at-index* $i\ L$) $num\text{-vars}$
(*proof*)

lemma *update-L-length*:

assumes $h \in \text{set } (\text{enum-pairs } L)$

shows $\text{length } (\text{update-L } L \ h \ \text{num-var}) = \text{length } L - 1$

<proof>

lemma *update-L-preserves-num-vars*:

assumes *WEST-regex-of-vars* $L \ \text{num-var}$

assumes $h \in \text{set } (\text{enum-pairs } L)$

assumes $K = \text{update-L } L \ h \ \text{num-var}$

shows *WEST-regex-of-vars* $K \ \text{num-var}$

<proof>

lemma *WEST-simp-helper-can-simp*:

assumes $\text{simp-L} = \text{WEST-simp-helper } L \ (\text{enum-pairs } L) \ i \ \text{num-vars}$

assumes $\exists j. j < \text{length } (\text{enum-pairs } L) \wedge j \geq i \wedge$

$\text{check-simp } (L \ ! \ \text{fst } (\text{enum-pairs } L \ ! \ j))$

$(L \ ! \ \text{snd } (\text{enum-pairs } L \ ! \ j))$

assumes $\text{min-j} = \text{Min } \{j. j < \text{length } (\text{enum-pairs } L) \wedge j \geq i \wedge$

$\text{check-simp } (L \ ! \ \text{fst } (\text{enum-pairs } L \ ! \ j))$

$(L \ ! \ \text{snd } (\text{enum-pairs } L \ ! \ j))\}$

assumes $\text{newL} = \text{update-L } L \ (\text{enum-pairs } L \ ! \ \text{min-j}) \ \text{num-vars}$

assumes $i < \text{length } (\text{enum-pairs } L)$

shows $\text{simp-L} = \text{WEST-simp-helper } \text{newL} \ (\text{enum-pairs } \text{newL}) \ 0 \ \text{num-vars}$

<proof>

lemma *WEST-simp-helper-cant-simp*:

assumes $\text{simp-L} = \text{WEST-simp-helper } L \ (\text{enum-pairs } L) \ i \ \text{num-vars}$

assumes $\neg(\exists j. j < \text{length } (\text{enum-pairs } L) \wedge j \geq i \wedge$

$\text{check-simp } (L \ ! \ \text{fst } (\text{enum-pairs } L \ ! \ j))$

$(L \ ! \ \text{snd } (\text{enum-pairs } L \ ! \ j)))$

shows $\text{simp-L} = L$

<proof>

lemma *WEST-simp-helper-length*:

shows $\text{length } (\text{WEST-simp-helper } L \ (\text{enum-pairs } L) \ i \ \text{num-vars}) \leq \text{length } L$

<proof>

lemma *WEST-simp-helper-num-vars*:

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

shows *WEST-regex-of-vars* $(\text{WEST-simp-helper } L \ (\text{enum-pairs } L) \ i \ \text{num-vars})$

num-vars

<proof>

lemma *WEST-simp-num-vars*:

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

shows *WEST-regex-of-vars* $(\text{WEST-simp } L \ \text{num-vars}) \ \text{num-vars}$

<proof>

lemma *WEST-and-simp-num-vars*:
assumes *WEST-regex-of-vars L1 k*
assumes *WEST-regex-of-vars L2 k*
shows *WEST-regex-of-vars (WEST-and-simp L1 L2 k) k*
 \langle *proof* \rangle

lemma *WEST-or-simp-num-vars*:
assumes *WEST-regex-of-vars L1 k*
assumes *WEST-regex-of-vars L2 k*
shows *WEST-regex-of-vars (WEST-or-simp L1 L2 k) k*
 \langle *proof* \rangle

lemma *shift-num-vars*:
fixes *L::WEST-regex*
fixes *a k::nat*
assumes *WEST-regex-of-vars L k*
shows *WEST-regex-of-vars (shift L k a) k*
 \langle *proof* \rangle

lemma *WEST-future-num-vars*:
assumes *WEST-regex-of-vars L k*
assumes $a \leq b$
shows *WEST-regex-of-vars (WEST-future L a b k) k*
 \langle *proof* \rangle

lemma *WEST-global-num-vars*:
assumes *WEST-regex-of-vars L k*
assumes $a \leq b$
shows *WEST-regex-of-vars (WEST-global L a b k) k*
 \langle *proof* \rangle

lemma *WEST-until-num-vars*:
assumes *WEST-regex-of-vars L1 k*
assumes *WEST-regex-of-vars L2 k*
assumes $a \leq b$
shows *WEST-regex-of-vars (WEST-until L1 L2 a b k) k*
 \langle *proof* \rangle

lemma *WEST-release-helper-num-vars*:
assumes *WEST-regex-of-vars L1 k*
assumes *WEST-regex-of-vars L2 k*
assumes $a \leq b$

shows *WEST-regex-of-vars* (*WEST-release-helper* *L1 L2 a b k*) *k*
 ⟨*proof*⟩

lemma *WEST-release-num-vars*:
assumes *WEST-regex-of-vars* *L1 k*
assumes *WEST-regex-of-vars* *L2 k*
assumes $a \leq b$
shows *WEST-regex-of-vars* (*WEST-release* *L1 L2 a b k*) *k*
 ⟨*proof*⟩

lemma *WEST-reg-aux-num-vars*:
assumes *is-nnf*: $\exists \psi. F1 = (\text{convert-nnf } \psi)$
assumes $k \geq \text{WEST-num-vars } F1$
assumes *intervals-welldef* *F1*
shows *WEST-regex-of-vars* (*WEST-reg-aux* *F1 k*) *k*
 ⟨*proof*⟩

lemma *nnf-intervals-welldef*:
assumes *intervals-welldef* *F1*
shows *intervals-welldef* (*convert-nnf* *F1*)
 ⟨*proof*⟩

lemma *WEST-reg-num-vars*:
assumes *intervals-welldef* *F1*
shows *WEST-regex-of-vars* (*WEST-reg* *F1*) (*WEST-num-vars* *F1*)
 ⟨*proof*⟩

3.7 Correctness of WEST-simp

3.7.1 WEST-count-diff facts

lemma *count-diff-property-aux*:
assumes $k < \text{length } r1 \wedge k < \text{length } r2$
shows $\text{count-diff } r1 \ r2 \geq \text{count-diff-state } (r1 ! k) (r2 ! k)$
 ⟨*proof*⟩

lemma *count-diff-state-property*:
assumes $\text{count-diff-state } t1 \ t2 = 0$
assumes $ka < \text{length } t1 \wedge ka < \text{length } t2$
shows $t1 ! ka = t2 ! ka$
 ⟨*proof*⟩

lemma *count-diff-property*:
assumes $\text{count-diff } r1 \ r2 = 0$
assumes $k < \text{length } r1 \wedge k < \text{length } r2$
assumes $ka < \text{length } (r1 ! k) \wedge ka < \text{length } (r2 ! k)$
shows $r2 ! k ! ka = r1 ! k ! ka$
 ⟨*proof*⟩

lemma *count-nonS-trace-0-allS*:
assumes $\text{length } h = \text{num-vars}$
assumes $\text{count-nonS-trace } h = 0$
shows $h = \text{map } (\lambda t. S) [0..<\text{num-vars}]$
 $\langle \text{proof} \rangle$

lemma *trace-tail-num-vars*:
assumes $\text{trace-regex-of-vars } (h \# \text{trace}) \text{ num-vars}$
shows $\text{trace-regex-of-vars } \text{trace } \text{num-vars}$
 $\langle \text{proof} \rangle$

lemma *count-diff-property-S-aux*:
assumes $\text{count-diff } \text{trace } [] = 0$
assumes $k < \text{length } \text{trace}$
assumes $\text{trace-regex-of-vars } \text{trace } \text{num-vars}$
assumes $1 \leq \text{num-vars}$
shows $\text{trace } ! k = \text{map } (\lambda t. S) [0 .. < \text{num-vars}]$
 $\langle \text{proof} \rangle$

lemma *count-diff-property-S*:
assumes $\text{count-diff } r1 \ r2 = 0$
assumes $k < \text{length } r1 \wedge \text{length } r2 \leq k$
assumes $\text{trace-regex-of-vars } r1 \ \text{num-vars}$
assumes $\text{num-vars} \geq 1$
assumes $ka < \text{num-vars}$
shows $r1 \ ! k = \text{map } (\lambda t. S) [0..<\text{num-vars}]$
 $\langle \text{proof} \rangle$

lemma *count-diff-state-commutative*:
shows $\text{count-diff-state } e1 \ e2 = \text{count-diff-state } e2 \ e1$
 $\langle \text{proof} \rangle$

lemma *count-diff-commutative*:
shows $\text{count-diff } r1 \ r2 = \text{count-diff } r2 \ r1$
 $\langle \text{proof} \rangle$

lemma *count-diff-same-trace*:
shows $\text{count-diff } \text{trace } \text{trace} = 0$
 $\langle \text{proof} \rangle$

lemma *count-diff-state-0*:
assumes $\text{count-diff-state } h1 \ h2 = 0$
assumes $\text{length } h1 = \text{length } h2$
shows $h1 = h2$
 $\langle \text{proof} \rangle$

lemma *count-diff-state-1*:
assumes $\text{length } h1 = \text{length } h2$
assumes $\text{count-diff-state } h1 \ h2 = 1$
shows $\exists ka < \text{length } h1. h1!ka \neq h2!ka$
<proof>

lemma *count-diff-state-other-states*:
assumes $\text{count-diff-state } h1 \ h2 = 1$
assumes $\text{length } h1 = \text{length } h2$
assumes $h1!k \neq h2!k$
assumes $k < \text{length } h1$
shows $\forall i < \text{length } h1. k \neq i \longrightarrow h1!i = h2!i$
<proof>

lemma *count-diff-same-len*:
assumes $\text{trace-regex-of-vars } r1 \ \text{num-vars}$
assumes $\text{trace-regex-of-vars } r2 \ \text{num-vars}$
assumes $\text{count-diff } r1 \ r2 = 0$
assumes $\text{length } r1 = \text{length } r2$
shows $r1 = r2$
<proof>

lemma *count-diff-1*:
assumes $\text{count-diff } r1 \ r2 = 1$
assumes $\text{length } r1 = \text{length } r2$
assumes $\text{trace-regex-of-vars } r1 \ \text{num-vars}$
assumes $\text{trace-regex-of-vars } r2 \ \text{num-vars}$
shows $\exists k < \text{length } r1. \text{count-diff-state } (r1!k) \ (r2!k) = 1$
<proof>

lemma *count-diff-1-other-states*:
assumes $\text{count-diff } r1 \ r2 = 1$
assumes $\text{length } r1 = \text{length } r2$
assumes $\text{trace-regex-of-vars } r1 \ \text{num-vars}$
assumes $\text{trace-regex-of-vars } r2 \ \text{num-vars}$
assumes $\text{count-diff-state } (r1!k) \ (r2!k) = 1$
shows $\forall i < \text{length } r1. k \neq i \longrightarrow r1!i = r2!i$
<proof>

3.7.2 Orsimp-trace Facts

lemma *WEST-simp-bitwise-identity*:
assumes $b1 = b2$
shows $\text{WEST-simp-bitwise } b1 \ b2 = b1$
<proof>

lemma *WEST-simp-bitwise-commutative*:

shows *WEST-simp-bitwise* $b1\ b2 = WEST-simp-bitwise\ b2\ b1$
(*proof*)

lemma *WEST-simp-state-commutative*:
assumes *length* $s1 = num-vars$
assumes *length* $s2 = num-vars$
shows *WEST-simp-state* $s1\ s2 = WEST-simp-state\ s2\ s1$
(*proof*)

lemma *WEST-simp-trace-commutative*:
assumes *trace-regex-of-vars* $r1\ num-vars$
assumes *trace-regex-of-vars* $r2\ num-vars$
shows *WEST-simp-trace* $r1\ r2\ num-vars = WEST-simp-trace\ r2\ r1\ num-vars$
(*proof*)

lemma *WEST-simp-trace-identity*:
assumes *trace-regex-of-vars* $r1\ num-vars$
assumes *trace-regex-of-vars* $r2\ num-vars$
assumes *count-diff* $r1\ r2 = 0$
assumes *length* $r1 \geq length\ r2$
shows *WEST-simp-trace* $r1\ r2\ num-vars = r1$
(*proof*)

lemma *WEST-simp-trace-length*:
assumes *trace-regex-of-vars* $r1\ num-vars$
assumes *trace-regex-of-vars* $r2\ num-vars$
assumes *length* $r1 = length\ r2$
shows *length* (*WEST-simp-trace* $r1\ r2\ num-vars$) = *length* $r1$
(*proof*)

3.7.3 WEST-orsimp-trace-correct

lemma *WEST-simp-trace-correct-forward*:
assumes *check-simp* $r1\ r2$
assumes *trace-regex-of-vars* $r1\ num-vars$
assumes *trace-regex-of-vars* $r2\ num-vars$
assumes *match-regex* π (*WEST-simp-trace* $r1\ r2\ num-vars$)
shows *match-regex* $\pi\ r1 \vee match-regex\ \pi\ r2$
(*proof*)

lemma *WEST-simp-trace-correct-converse*:
assumes *check-simp* $r1\ r2$
assumes *trace-regex-of-vars* $r1\ num-vars$
assumes *trace-regex-of-vars* $r2\ num-vars$
assumes *match-regex* $\pi\ r1 \vee match-regex\ \pi\ r2$
shows *match-regex* π (*WEST-simp-trace* $r1\ r2\ num-vars$)

<proof>

lemma *WEST-simp-trace-correct:*

assumes *check-simp r1 r2*

assumes *trace-regex-of-vars r1 num-vars*

assumes *trace-regex-of-vars r2 num-vars*

shows *match-regex π (WEST-simp-trace r1 r2 num-vars) \longleftrightarrow match-regex π r1*
 \vee match-regex π r2

<proof>

3.7.4 Simp-helper Correct

lemma *WEST-simp-helper-can-simp-bound:*

assumes *simp-L = WEST-simp-helper L (enum-pairs L) i num-vars*

assumes $\exists j. j < \text{length } (\text{enum-pairs } L) \wedge j \geq i \wedge$

$\text{check-simp } (L ! \text{fst } (\text{enum-pairs } L ! j))$

$(L ! \text{snd } (\text{enum-pairs } L ! j))$

assumes $i < \text{length } (\text{enum-pairs } L)$

shows $\text{length } \text{simp-L} < \text{length } L$

<proof>

lemma *WEST-simp-helper-same-length:*

assumes *WEST-regex-of-vars L num-vars*

assumes $K = \text{WEST-simp-helper } L (\text{enum-pairs } L) 0 \text{ num-vars}$

assumes $\text{length } K = \text{length } L$

shows $L = K$

<proof>

lemma *WEST-simp-helper-less-length:*

assumes *WEST-regex-of-vars L num-vars*

assumes $\text{length } K < \text{length } L$

assumes $K = \text{WEST-simp-helper } L (\text{enum-pairs } L) 0 \text{ num-vars}$

shows $\exists \text{min-j.}$

$(\text{min-j} < \text{length } (\text{enum-pairs } L) \wedge$

$K =$

$\text{WEST-simp-helper } (\text{update-L } L (\text{enum-pairs } L ! \text{min-j}) \text{ num-vars})$

$(\text{enum-pairs}$

$(\text{update-L } L (\text{enum-pairs } L ! \text{min-j}) \text{ num-vars}))$

0 num-vars

$\wedge \text{check-simp } (L ! \text{fst } (\text{enum-pairs } L ! \text{min-j})) (L ! \text{snd } (\text{enum-pairs } L !$

$\text{min-j}))$

<proof>

lemma *remove-element-at-index-subset:*

fixes $i::\text{nat}$

assumes $i < \text{length } L$

shows $\text{set } (\text{remove-element-at-index } i L) \subseteq \text{set } L$

<proof>

lemma *WEST-simp-helper-correct-forward:*
 assumes *WEST-regex-of-vars L num-vars*
 assumes *match π K*
 assumes *K = WEST-simp-helper L (enum-pairs L) 0 num-vars*
 shows *match π L*
 <proof>

lemma *remove-element-at-index-fact:*
 assumes *j1 < j2*
 assumes *j2 < length L*
 assumes *i < length L*
 assumes *i \neq j1*
 assumes *i \neq j2*
 shows *L ! i*
 \in set (remove-element-at-index j1 (remove-element-at-index j2 L))
 <proof>

lemma *update-L-match:*
 assumes *WEST-regex-of-vars L num-var*
 assumes *match π L*
 assumes *h \in set (enum-pairs L)*
 assumes *check-simp (L!(fst h)) (L!(snd h))*
 shows *match π (update-L L h num-var)*
 <proof>

lemma *WEST-simp-helper-correct-converse:*
 assumes *WEST-regex-of-vars L num-vars*
 assumes *match π L*
 assumes *K = WEST-simp-helper L (enum-pairs L) i num-vars*
 shows *match π K*
 <proof>

3.7.5 WEST-simp Correct

lemma *simp-correct-forward:*
 assumes *WEST-regex-of-vars L num-vars*
 assumes *match π (WEST-simp L num-vars)*
 shows *match π L*
 <proof>

lemma *simp-correct-converse:*
 assumes *WEST-regex-of-vars L num-vars*
 assumes *match π L*
 shows *match π (WEST-simp L num-vars)*

<proof>

lemma *simp-correct*:

assumes *WEST-regex-of-vars L num-vars*
shows $\text{match } \pi \text{ (WEST-simp L num-vars)} \longleftrightarrow \text{match } \pi \text{ L}$
<proof>

3.8 Correctness of WEST-and-simp/WEST-or-simp

lemma *WEST-and-simp-correct*:

fixes $\pi::\text{trace}$
fixes $L1 \ L2:: \text{WEST-regex}$
assumes $L1\text{-of-num-vars: WEST-regex-of-vars L1 } n$
assumes $L2\text{-of-num-vars: WEST-regex-of-vars L2 } n$
shows $\text{match } \pi \ L1 \wedge \text{match } \pi \ L2 \longleftrightarrow \text{match } \pi \text{ (WEST-and-simp L1 L2 } n)$
<proof>

lemma *WEST-or-simp-correct*:

fixes $\pi::\text{trace}$
fixes $L1 \ L2:: \text{WEST-regex}$
assumes $L1\text{-of-num-vars: WEST-regex-of-vars L1 } n$
assumes $L2\text{-of-num-vars: WEST-regex-of-vars L2 } n$
shows $\text{match } \pi \ L1 \vee \text{match } \pi \ L2 \longleftrightarrow \text{match } \pi \text{ (WEST-or-simp L1 L2 } n)$
<proof>

3.9 Facts about the WEST future operator

lemma *WEST-future-correct-forward*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F ≤ num-vars*
assumes $a \leq b$
assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$
assumes $\text{match } \pi \text{ (WEST-future L a b num-vars)}$
shows $\pi \models_m (F_m [a, b] F)$
<proof>

lemma *WEST-future-correct-converse*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F ≤ num-vars*
assumes $a \leq b$
assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$
assumes $\pi \models_m (\text{Future-mltl a b } F)$

shows $\text{match } \pi \text{ (WEST-future } L \ a \ b \ \text{num-vars)}$
 $\langle \text{proof} \rangle$

lemma *WEST-future-correct*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

assumes *WEST-num-vars* $F \leq \text{num-vars}$

assumes $a \leq b$

assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$

shows $\text{match } \pi \text{ (WEST-future } L \ a \ b \ \text{num-vars)} \longleftrightarrow$
 $\text{semantics-mltl } \pi \text{ (Future-mltl } a \ b \ F)$

$\langle \text{proof} \rangle$

3.10 Facts about the WEST global operator

lemma *WEST-global-correct-forward*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

assumes *WEST-num-vars* $F \leq \text{num-vars}$

assumes $a \leq b$

assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$

assumes $\text{match } \pi \text{ (WEST-global } L \ a \ b \ \text{num-vars)}$

shows $\text{semantics-mltl } \pi \text{ (Global-mltl } a \ b \ F)$

$\langle \text{proof} \rangle$

lemma *WEST-global-correct-converse*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

assumes *WEST-num-vars* $F \leq \text{num-vars}$

assumes $a \leq b$

assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$

assumes $\text{semantics-mltl } \pi \text{ (Global-mltl } a \ b \ F)$

shows $\text{match } \pi \text{ (WEST-global } L \ a \ b \ \text{num-vars)}$

$\langle \text{proof} \rangle$

lemma *WEST-global-correct*:

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F \longrightarrow (\text{match } \pi \ L \longleftrightarrow \text{semantics-mltl } \pi \ F))$

assumes *WEST-regex-of-vars* $L \ \text{num-vars}$

assumes *WEST-num-vars* $F \leq \text{num-vars}$

assumes $a \leq b$

assumes $\text{length } \pi \geq (\text{complen-mltl } F) + b$

shows $\text{match } \pi \text{ (WEST-global } L \text{ a b num-vars)} \longleftrightarrow$
 $\text{semantics-mltl } \pi \text{ (Global-mltl a b F)}$
 $\langle \text{proof} \rangle$

3.11 Facts about the WEST until operator

lemma *WEST-until-correct-forward:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \text{ L1} \longleftrightarrow \text{semantics-mltl } \pi \text{ F1}))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \text{ L2} \longleftrightarrow \text{semantics-mltl } \pi \text{ F2}))$
assumes *WEST-regex-of-vars* $L1 \text{ num-vars}$
assumes *WEST-regex-of-vars* $L2 \text{ num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes $a \leq b$
assumes $\text{length } \pi \geq \text{complen-mltl (Until-mltl } F1 \text{ a b } F2)$
assumes $\text{match } \pi \text{ (WEST-until } L1 \text{ L2 a b num-vars)}$
shows $\text{semantics-mltl } \pi \text{ (Until-mltl } F1 \text{ a b } F2)$
 $\langle \text{proof} \rangle$

lemma *WEST-until-correct-converse:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \text{ L1} \longleftrightarrow \text{semantics-mltl } \pi \text{ F1}))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \text{ L2} \longleftrightarrow \text{semantics-mltl } \pi \text{ F2}))$
assumes *WEST-regex-of-vars* $L1 \text{ num-vars}$
assumes *WEST-regex-of-vars* $L2 \text{ num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes $a \leq b$
assumes $\text{length } \pi \geq (\text{complen-mltl (Until-mltl } F1 \text{ a b } F2))$
assumes $\text{semantics-mltl } \pi \text{ (Until-mltl } F1 \text{ a b } F2)$
shows $\text{match } \pi \text{ (WEST-until } L1 \text{ L2 a b num-vars)}$
 $\langle \text{proof} \rangle$

lemma *WEST-until-correct:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \text{ L1} \longleftrightarrow \text{semantics-mltl } \pi \text{ F1}))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \text{ L2} \longleftrightarrow \text{semantics-mltl } \pi \text{ F2}))$
assumes *WEST-regex-of-vars* $L1 \text{ num-vars}$
assumes *WEST-regex-of-vars* $L2 \text{ num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes $a \leq b$
assumes $\text{length } \pi \geq \text{complen-mltl (Until-mltl } F1 \text{ a b } F2)$

shows $\text{match } \pi \text{ (WEST-until } L1 \ L2 \ a \ b \ \text{num-vars)} \longleftrightarrow$
 $\text{semantics-mltl } \pi \text{ (Until-mltl } F1 \ a \ b \ F2)$
 $\langle \text{proof} \rangle$

3.12 Facts about the WEST release Operator

lemma *WEST-release-correct-forward:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \ L1 \longleftrightarrow \text{semantics-mltl } \pi \ F1))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \ L2 \longleftrightarrow \text{semantics-mltl } \pi \ F2))$
assumes *WEST-regex-of-vars* $L1 \ \text{num-vars}$
assumes *WEST-regex-of-vars* $L2 \ \text{num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes *a-leq-b*: $a \leq b$
assumes *len*: $\text{length } \pi \geq \text{complen-mltl } (\text{Release-mltl } F1 \ a \ b \ F2)$
assumes $\text{match } \pi \text{ (WEST-release } L1 \ L2 \ a \ b \ \text{num-vars)}$
shows $\text{semantics-mltl } \pi \text{ (Release-mltl } F1 \ a \ b \ F2)$
 $\langle \text{proof} \rangle$

lemma *WEST-release-correct-converse:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \ L1 \longleftrightarrow \text{semantics-mltl } \pi \ F1))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \ L2 \longleftrightarrow \text{semantics-mltl } \pi \ F2))$
assumes *WEST-regex-of-vars* $L1 \ \text{num-vars}$
assumes *WEST-regex-of-vars* $L2 \ \text{num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes $a \leq b$
assumes $\text{length } \pi \geq \text{complen-mltl } (\text{Release-mltl } F1 \ a \ b \ F2)$
assumes $\text{semantics-mltl } \pi \text{ (Release-mltl } F1 \ a \ b \ F2)$
shows $\text{match } \pi \text{ (WEST-release } L1 \ L2 \ a \ b \ \text{num-vars)}$
 $\langle \text{proof} \rangle$

lemma *WEST-release-correct:*

assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F1 \longrightarrow (\text{match } \pi \ L1 \longleftrightarrow \text{semantics-mltl } \pi \ F1))$
assumes $\bigwedge \pi. (\text{length } \pi \geq \text{complen-mltl } F2 \longrightarrow (\text{match } \pi \ L2 \longleftrightarrow \text{semantics-mltl } \pi \ F2))$
assumes *WEST-regex-of-vars* $L1 \ \text{num-vars}$
assumes *WEST-regex-of-vars* $L2 \ \text{num-vars}$
assumes *WEST-num-vars* $F1 \leq \text{num-vars}$
assumes *WEST-num-vars* $F2 \leq \text{num-vars}$
assumes $a \leq b$
assumes $\text{length } \pi \geq \text{complen-mltl } (\text{Release-mltl } F1 \ a \ b \ F2)$

shows *semantics-mltl* π (*Release-mltl* $F1$ a b $F2$) \longleftrightarrow *match* π (*WEST-release* $L1$ $L2$ a b *num-vars*)
 ⟨*proof*⟩

3.13 Top level result: Shows that WEST reg is correct

lemma *WEST-reg-aux-correct*:

assumes π -*long-enough*: $\text{length } \pi \geq \text{complen-mltl } F$
assumes *is-nnf*: $\exists \psi. F = (\text{convert-nnf } \psi)$
assumes φ -*nv*: *WEST-num-vars* $F \leq$ *num-vars*
assumes *intervals-welldef* F
shows *match* π (*WEST-reg-aux* F *num-vars*) \longleftrightarrow *semantics-mltl* π F
 ⟨*proof*⟩

lemma *complen-convert-nnf*:

shows *complen-mltl* (*convert-nnf* φ) = *complen-mltl* φ
 ⟨*proof*⟩

lemma *nnf-int-welldef*:

assumes *intervals-welldef* φ
shows *intervals-welldef* (*convert-nnf* φ)
 ⟨*proof*⟩

lemma *WEST-correct*:

fixes $\varphi::(\text{nat})$ *mltl*
fixes $\pi::\text{trace}$
assumes *int-welldef*: *intervals-welldef* φ
assumes π -*long-enough*: $\text{length } \pi \geq \text{complen-mltl } (\text{convert-nnf } \varphi)$
shows *match* π (*WEST-reg* φ) \longleftrightarrow *semantics-mltl* π φ
 ⟨*proof*⟩

lemma *WEST-correct-v2*:

fixes $\varphi::(\text{nat})$ *mltl*
fixes $\pi::\text{trace}$
assumes *intervals-welldef* φ
assumes π -*long-enough*: $\text{length } \pi \geq \text{complen-mltl } \varphi$
shows *match* π (*WEST-reg* φ) \longleftrightarrow *semantics-mltl* π φ
 ⟨*proof*⟩

3.14 Top level result for padded version

lemma *WEST-correct-pad-aux*:

fixes $\varphi::(\text{nat})$ *mltl*
fixes $\pi::\text{trace}$
assumes *intervals-welldef* φ
assumes π -*long-enough*: $\text{length } \pi \geq \text{complen-mltl } \varphi$

shows *match* π (*pad-WEST-reg* φ) \longleftrightarrow *semantics-mltl* π φ
 ⟨*proof*⟩

lemma *WEST-correct-pad*:

fixes $\varphi::(\text{nat})$ *mltl*
fixes $\pi::\text{trace}$
assumes *intervals-welldef* φ
assumes π -*long-enough*: *length* $\pi \geq$ *complen-mltl* φ
shows *match* π (*simp-pad-WEST-reg* φ) \longleftrightarrow *semantics-mltl* π φ
 ⟨*proof*⟩

end

4 Key algorithms for WEST

theory *Regex-Equivalence*

imports *WEST-Algorithms WEST-Proofs*

begin

fun *depth-datatype-list*:: *state-regex* \Rightarrow *nat*
where *depth-datatype-list* [] = 0
 | *depth-datatype-list* (*One*#*T*) = 1 + *depth-datatype-list* *T*
 | *depth-datatype-list* (*Zero*#*T*) = 1 + *depth-datatype-list* *T*
 | *depth-datatype-list* (*S*#*T*) = 2 + 2*(*depth-datatype-list* *T*)

function *enumerate-list*:: *state-regex* \Rightarrow *trace-regex*
where *enumerate-list* [] = [[]]
 | *enumerate-list* (*One*#*T*) = (*map* ($\lambda x.$ *One*#*x*) (*enumerate-list* *T*))
 | *enumerate-list* (*Zero*#*T*) = (*map* ($\lambda x.$ *Zero*#*x*) (*enumerate-list* *T*))
 | *enumerate-list* (*S*#*T*) = (*enumerate-list* (*Zero*#*T*))@(*enumerate-list* (*One*#*T*))
 ⟨*proof*⟩

termination ⟨*proof*⟩

fun *flatten-list*:: 'a *list list* \Rightarrow 'a *list*
where *flatten-list* *L* = *foldr* (@) *L* []

value *flatten-list* [[12, 13::nat], [15]]

value *flatten-list* (*let* *enumerate-H* = *enumerate-list* [*S*, *One*] *in*
let *enumerate-T* = [[]] *in*
map ($\lambda t.$ (*map* ($\lambda h.$ *h*#*t*) *enumerate-H*)) *enumerate-T*)

```

fun enumerate-trace:: trace-regex  $\Rightarrow$  WEST-regex
  where enumerate-trace [] = [[]]
  | enumerate-trace (H#T) = flatten-list
  (let enumerate-H = enumerate-list H in
   let enumerate-T = enumerate-trace T in
   map ( $\lambda t.$  (map ( $\lambda h.$  h#t) enumerate-H)) enumerate-T)

value enumerate-trace [[S, One], [S], [One]]
value enumerate-trace [[]]

fun enumerate-sets:: WEST-regex  $\Rightarrow$  trace-regex set
  where enumerate-sets [] = {}
  | enumerate-sets (h#T) = (set (enumerate-trace h))  $\cup$  (enumerate-sets T)

fun naive-equivalence:: WEST-regex  $\Rightarrow$  WEST-regex  $\Rightarrow$  bool
  where naive-equivalence A B = (A = B  $\vee$  (enumerate-sets A) = (enumerate-sets B))

```

5 Regex Equivalence Correctness

```

lemma enumerate-list-len-alt:
  shows  $\forall$  state  $\in$  set (enumerate-list state-regex).
    length state = length state-regex
  <proof>

```

```

lemma enumerate-list-len:
  assumes state  $\in$  set (enumerate-list state-regex)
  shows length state = length state-regex
  <proof>

```

```

lemma enumerate-list-prop:
  assumes ( $\bigwedge k.$  List.member j k  $\implies$  k  $\neq$  S)
  shows enumerate-list j = [j]
  <proof>

```

```

lemma enumerate-fixed-trace:
  fixes h1:: trace-regex
  assumes  $\bigwedge j.$  List.member h1 j  $\implies$  ( $\bigwedge k.$  List.member j k  $\implies$  k  $\neq$  S)
  shows (enumerate-trace h1) = [h1]
  <proof>

```

If we have two state regexs that don't contain S's, then enumerate trace on each is different.

```

lemma enum-trace-prop:
  fixes h1 h2:: trace-regex

```

assumes $\bigwedge j. \text{List.member } h1 \ j \implies (\bigwedge k. \text{List.member } j \ k \implies k \neq S)$
assumes $\bigwedge j. \text{List.member } h2 \ j \implies (\bigwedge k. \text{List.member } j \ k \implies k \neq S)$
assumes $(\text{set } h1) \neq (\text{set } h2)$
shows $\text{set } (\text{enumerate-trace } h1) \neq \text{set } (\text{enumerate-trace } h2)$
 <proof>

lemma *enumerate-list-tail-in*:
assumes $\text{head-t}\#\text{tail-t} \in \text{set } (\text{enumerate-list } (h\#\text{trace}))$
shows $\text{tail-t} \in \text{set } (\text{enumerate-list } \text{trace})$
 <proof>

lemma *enumerate-list-fixed*:
assumes $t \in \text{set } (\text{enumerate-list } \text{trace})$
shows $(\forall k. \text{List.member } t \ k \longrightarrow k \neq S)$
 <proof>

lemma *map-enum-list-nonempty*:
fixes $t::\text{WEST-bit list list}$
fixes $\text{head}::\text{WEST-bit list}$
shows $\text{map } (\lambda h. h \# t) (\text{enumerate-list } \text{head}) \neq []$
 <proof>

lemma *length-of-flatten-list*:
assumes $\text{flat} = \text{foldr } (@) (\text{map } (\lambda t. \text{map } (\lambda h. h \# t) H) T) []$
shows $\text{length } \text{flat} = \text{length } T * \text{length } H$
 <proof>

lemma *flatten-list-idx*:
assumes $\text{flat} = \text{flatten-list } (\text{map } (\lambda t. \text{map } (\lambda h. h \# t) \text{head}) \text{tail})$
assumes $i < \text{length } \text{tail}$
assumes $j < \text{length } \text{head}$
shows $(\text{head}!\ j)\#\text{(tail}!\ i) = \text{flat}!(i*(\text{length } \text{head}) + j) \wedge i*(\text{length } \text{head}) + j < \text{length } \text{flat}$
 <proof>

lemma *flatten-list-shape*:
assumes $\text{List.member } \text{flat } x1$
assumes $\text{flat} = \text{flatten-list } (\text{map } (\lambda t. \text{map } (\lambda h. h \# t) H) T)$
shows $\exists x1\text{-head } x1\text{-tail}. x1 = x1\text{-head}\#x1\text{-tail} \wedge \text{List.member } H \ x1\text{-head} \wedge \text{List.member } T \ x1\text{-tail}$
 <proof>

lemma *flatten-list-len*:

assumes $\bigwedge t. \text{List.member } T \ t \implies \text{length } t = n$
assumes $\text{flat} = \text{flatten-list } (\text{map } (\lambda t. \text{map } (\lambda h. h \# t) \ H) \ T)$
shows $\bigwedge x1. \text{List.member flat } x1 \implies \text{length } x1 = n+1$
<proof>

lemma *flatten-list-lemma*:

assumes $\bigwedge x1. \text{List.member to-flatten } x1 \implies (\bigwedge x2. \text{List.member } x1 \ x2 \implies \text{length } x2 = \text{length trace})$
assumes $a \in \text{set } (\text{flatten-list to-flatten})$
shows $\text{length } a = \text{length trace}$
<proof>

lemma *enumerate-trace-len*:

assumes $a \in \text{set } (\text{enumerate-trace trace})$
shows $\text{length } a = \text{length trace}$
<proof>

definition *regex-zeros-and-ones*:: $\text{trace-regex} \Rightarrow \text{bool}$

where $\text{regex-zeros-and-ones } tr =$
 $(\forall j. \text{List.member } tr \ j \longrightarrow (\forall k. \text{List.member } j \ k \longrightarrow k \neq S))$

lemma *match-enumerate-state-aux-first-bit*:

assumes $a\text{-head} = \text{Zero} \vee a\text{-head} = \text{One}$
assumes $a\text{-head} \# a\text{-tail} \in \text{set } (\text{enumerate-list } (h\text{-head} \# h))$
shows $h\text{-head} = a\text{-head} \vee h\text{-head} = S$
<proof>

lemma *advance-state-iff*:

assumes $x > 0$
shows $x \in \text{state} \longleftrightarrow (x-1) \in \text{advance-state state}$
<proof>

lemma *match-enumerate-state-aux*:

assumes $a \in \text{set } (\text{enumerate-list } h)$
assumes $\text{match-timestep state } a$
shows $\text{match-timestep state } h$
<proof>

lemma *enumerate-list-index-one*:

assumes $j < \text{length } (\text{enumerate-list } a)$
shows $\text{One} \# \text{enumerate-list } a \ ! \ j = \text{enumerate-list } (S \# a) \ ! \ (\text{length } (\text{enumerate-list } a) + j) \wedge$
 $(\text{length } (\text{enumerate-list } a) + j < \text{length } (\text{enumerate-list } (S \# a)))$

<proof>

lemma *list-concat-index*:

assumes $j < \text{length } L1$

shows $(L1@L2)!j = L1!j$

<proof>

lemma *enumerate-list-index-zero*:

assumes $j < \text{length } (\text{enumerate-list } a)$

shows $\text{Zero} \# \text{enumerate-list } a ! j = \text{enumerate-list } (S \# a) ! j \wedge$
 $j < \text{length } (\text{enumerate-list } (S \# a))$

<proof>

lemma *match-enumerate-list*:

assumes *match-timestep state a*

shows $\exists j < \text{length } (\text{enumerate-list } a).$

$\text{match-timestep state } (\text{enumerate-list } a ! j)$

<proof>

lemma *enumerate-trace-head-in*:

assumes $a\text{-head} \# a\text{-tail} \in \text{set } (\text{enumerate-trace } (h \# \text{trace}))$

shows $a\text{-head} \in \text{set } (\text{enumerate-list } h)$

<proof>

lemma *enumerate-trace-tail-in*:

assumes $a\text{-head} \# a\text{-tail} \in \text{set } (\text{enumerate-trace } (h \# \text{trace}))$

shows $a\text{-tail} \in \text{set } (\text{enumerate-trace } \text{trace})$

<proof>

Intuitively, this says that the traces in *enumerate trace h* are “more specific” than *h*, which is “more generic”—i.e., *h* matches everything that each element of *enumerate trace h* matches.

lemma *match-enumerate-trace-aux*:

assumes $a \in \text{set } (\text{enumerate-trace } \text{trace})$

assumes *match-regex π a*

shows *match-regex π trace*

<proof>

lemma *match-enumerate-trace*:

assumes $a \in \text{set } (\text{enumerate-trace } h) \wedge \text{match-regex } \pi a$

shows *match π (h # T)*

<proof>

lemma *match-enumerate-sets1*:

assumes $(\exists r \in (\text{enumerate-sets } R). \text{match-regex } \pi r)$
shows $(\text{match } \pi R)$
<proof>

lemma *match-cases*:
assumes $\text{match } \pi (a \# R)$
shows $\text{match } \pi [a] \vee \text{match } \pi R$
<proof>

lemma *enumerate-trace-decompose*:
assumes $\text{state} \in \text{set } (\text{enumerate-list } h)$
assumes $\text{trace} \in \text{set } (\text{enumerate-trace } T)$
shows $\text{state}\#\text{trace} \in \text{set } (\text{enumerate-trace } (h\#T))$
<proof>

lemma *match-enumerate-trace-aux-converse*:
assumes $\text{match-regex } \pi \text{ trace}$
shows $\text{match } \pi (\text{enumerate-trace } \text{trace})$
<proof>

lemma *match-enumerate-sets2*:
assumes $(\text{match } \pi R)$
shows $(\exists r \in \text{enumerate-sets } R. \text{match-regex } \pi r)$
<proof>

lemma *match-enumerate-sets*:
shows $(\exists r \in \text{enumerate-sets } R. \text{match-regex } \pi r) \longleftrightarrow (\text{match } \pi R)$
<proof>

lemma *regex-equivalence-correct1*:
assumes $(\text{naive-equivalence } A B)$
shows $\text{match } \pi A = \text{match } \pi B$
<proof>

lemma *regex-equivalence-correct*:
shows $(\text{naive-equivalence } A B) \longrightarrow (\text{regex-equiv } A B)$
<proof>

export-code *naive-equivalence* **in** *Haskell* **module-name** *regex-equiv*

end

References

- [1] J. Elwing, L. Gamboa-Guzman, J. Sorkin, C. Travesset, Z. Wang, and K. Y. Rozier. Mission-time LTL (MLTL) formula validation via regular expressions. In P. Herber and A. Wijs, editors, *iFM*, volume 14300 of *LNCS*, pages 279–301. Springer, 2023.
- [2] Z. Wang, L. P. Gamboa-Guzman, and K. Y. Rozier. WEST: Interactive Validation of Mission-time Linear Temporal Logic (MLTL). 2024.