Formalizing MLTL in Isabelle/HOL

Zili Wang and Katherine Kosaian

January 28, 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* ⇒ *nat* ⇒ *nat* ⇒ *WEST-bit*
  where *WEST-get-bit regex timestep var* = (
  if timestep ≥ *length regex then S*
  else let regex-index = *regex* ! *timestep in*
  if var ≥ *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 ⇒ nat ⇒ nat ⇒ state-regex
  where WEST-get-state regex time num-vars = (
  if time ≥ length regex then (map (λ 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 ⇒ state-regex ⇒ bool
  where match-timestep state regex-state = (∀ x::nat. x < length regex-state ⟶
(
  ((regex-state ! x = One) ⟶ x ∈ state) ∧
  ((regex-state ! x = Zero) ⟶ x ∉ state)))

fun trim-reversed-regex:: trace-regex ⇒ trace-regex
  where trim-reversed-regex [] = []
  | trim-reversed-regex (h#t) = (if (∀ i<length h. (h!i) = S)
  then (trim-reversed-regex t) else (h#t))

fun trim-regex:: trace-regex ⇒ trace-regex
  where trim-regex regex = rev (trim-reversed-regex (rev regex))

definition match-regex:: nat set list ⇒ trace-regex ⇒ bool
  where match-regex trace regex = ((∀ time<length regex.
  (match-timestep (trace ! time) (regex ! time)))
  ∧(length trace ≥ length regex))

definition match:: nat set list ⇒ WEST-regex ⇒ bool
  where match trace regex-list = (∃ i. i < length regex-list ∧
  (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 ⇒ WEST-regex ⇒ bool
  where regex-equiv rl1 rl2 = (
  ∀ π::nat set list. (match π rl1) ⟷ (match π 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* (λ*y*. (*x*, *y*)) *xs* @ *enumerate-pairs xs*

fun *enum-pairs*:: *'a list* ⇒ (*nat* ∗ *nat*) *list*
where *enum-pairs L* = *enumerate-pairs* [*0* ..< *length L*]

fun *remove-element-at-index*:: *nat* ⇒ *'a list* ⇒ *'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* ⇒ (*nat* × *nat*) ⇒ *nat* ⇒ *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*) ≤ (*length L*)^*2*
⟨*proof*⟩

lemma *length-enum-pairs*:
shows *length* (*enum-pairs L*) ≤ (*length L*)^*2*
⟨*proof*⟩

lemma *enumerate-pairs-fact*:
assumes ∀ *i j*. (*i* < *j* ∧ *i* < *length L* ∧ *j* < *length L*) ⟶ (*L*!*i*) < (*L*!*j*)
shows ∀ *pair* ∈ *set* (*enumerate-pairs L*). (*fst pair*) < (*snd pair*)
⟨*proof*⟩

lemma *enum-pairs-fact*:
shows ∀ *pair* ∈ *set* (*enum-pairs L*). (*fst pair*) < (*snd pair*)
⟨*proof*⟩

lemma *enum-pairs-bound-snd*:
assumes *pair* ∈ *set* (*enumerate-pairs L*)
shows (*snd pair*) ∈ *set L*
⟨*proof*⟩

lemma *enum-pairs-bound*:
shows ∀ *pair* ∈ *set* (*enum-pairs L*). (*snd pair*) < *length L*
⟨*proof*⟩

lemma *WEST-simp-termination1-bound*:
fixes *a*::*nat*
shows *a*^*3*+*a*^*2* < (*a*+*1*)^*3*
⟨*proof*⟩

lemma *WEST-simp-termination1*:
fixes *L*::*WEST-regex*

assumes ¬ (*idx-pairs* ≠ *enum-pairs L* ∨ *length idx-pairs* ≤ *i*)
assumes *check-simp* (*L* ! *fst* (*idx-pairs* ! *i*)) (*L* ! *snd* (*idx-pairs* ! *i*))
assumes *x* = *update-L L* (*idx-pairs* ! *i*) *num-vars*
shows ((*x*, *enum-pairs x*, *0*, *num-vars*), *L*, *idx-pairs*, *i*, *num-vars*)
∈ *measure* (λ(*L*, *idx-list*, *i*, *num-vars*). *length L* ^ *3* + *length idx-list* − *i*)
⟨*proof*⟩

function *WEST-simp-helper*:: *WEST-regex* ⇒ (*nat* × *nat*) *list* ⇒ *nat* ⇒ *nat* ⇒ *WEST-regex*
where *WEST-simp-helper L idx-pairs i num-vars* =
(*if* (*idx-pairs* ≠ *enum-pairs L* ∨ *i* ≥ *length idx-pairs*) *then L else*
(*if* (*check-simp* (*L*!(*fst* (*idx-pairs*!*i*))) (*L*!(*snd* (*idx-pairs*!*i*)))) *then*
(*let newL* = *update-L L* (*idx-pairs*!*i*) *num-vars in*
WEST-simp-helper newL (*enum-pairs newL*) *0 num-vars*)
else WEST-simp-helper L idx-pairs (*i*+*1*) *num-vars*))
⟨*proof*⟩
termination
⟨*proof*⟩

declare *WEST-simp-helper.simps*[*simp del*]

fun *WEST-simp*:: *WEST-regex* ⇒ *nat* ⇒ *WEST-regex*
where *WEST-simp L num-vars* =
WEST-simp-helper L (*enum-pairs L*) *0 num-vars*

value *WEST-simp* [[[*S*, *S*, *One*]],[[*S*, *One*, *S*]], [[*S*, *S*, *Zero*]]] *3*
value *WEST-simp* [[[*S*, *One*]],[[*One*, *S*]], [[*Zero*, *Zero*]]] *2*
value *WEST-simp* [[[*One*, *One*]],[[*Zero*, *Zero*]], [[*One*, *Zero*]], [[*Zero*, *One*]]] *2*

1.3.3 AND and OR operations with WEST-simp

fun *WEST-and-simp*:: *WEST-regex* ⇒ *WEST-regex* ⇒ *nat* ⇒ *WEST-regex*
where *WEST-and-simp L1 L2 num-vars* = *WEST-simp* (*WEST-and L1 L2*) *num-vars*

fun *WEST-or-simp*:: *WEST-regex* ⇒ *WEST-regex* ⇒ *nat* ⇒ *WEST-regex*
where *WEST-or-simp L1 L2 num-vars* = *WEST-simp* (*L1*@*L2*) *num-vars*

1.3.4 Useful Helper Functions

fun *arbitrary-state*::*nat* ⇒ *state-regex*
where *arbitrary-state num-vars* = *map* (λ *k*. *S*) [*0* ..< *num-vars*]

fun *arbitrary-trace*::*nat* ⇒ *nat* ⇒ *trace-regex*
where *arbitrary-trace num-vars num-pad* = *map* (λ *k*. (*arbitrary-state num-vars*)) [*0* ..< *num-pad*]

fun *shift*:: *WEST-regex* ⇒ *nat* ⇒ *nat* ⇒ *WEST-regex*

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

fun *pad*:: *trace-regex* ⇒ *nat* ⇒ *nat* ⇒ *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* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *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* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *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* ⇒ *WEST-regex* ⇒ *nat* ⇒
nat ⇒ *nat* ⇒ *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* ⇒ *WEST-regex* ⇒
nat ⇒ *nat* ⇒ *nat* ⇒ *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-and-simp* (*WEST-global L-ψ a ub num-vars*)
(*WEST-global L-φ ub ub num-vars*) *num-vars*) *num-vars*
else []))

fun *WEST-release*:: *WEST-regex* ⇒ *WEST-regex* ⇒ *nat*
⇒ *nat* ⇒ *nat* ⇒ *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 ⋀*x*:: *nat mltl* × *nat*. ⋀*P*::*bool*. (⋀*num-vars*::*nat*. *x* = (*True-mltl*, *num-vars*) ⟹ *P*) ⟹
(⋀*num-vars*::*nat*. *x* = (*False-mltl*, *num-vars*) ⟹ *P*) ⟹
(⋀*p num-vars*::*nat*. *x* = (*Prop-mltl p*, *num-vars*) ⟹ *P*) ⟹
(⋀*p num-vars*::*nat*. *x* = (*Not-mltl* (*Prop-mltl p*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ num-vars*. *x* = (*Or-mltl φ ψ*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ num-vars*. *x* = (*And-mltl φ ψ*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ a b num-vars*. *x* = (*Future-mltl φ a b*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ a b num-vars*. *x* = (*Global-mltl φ a b*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ a b num-vars*. *x* = (*Until-mltl φ ψ a b*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ a b num-vars*. *x* = (*Release-mltl φ ψ a b*, *num-vars*) ⟹ *P*) ⟹
(⋀*num-vars*. *x* = (*Not-mltl True-mltl*, *num-vars*) ⟹ *P*) ⟹
(⋀*num-vars*. *x* = (*Not-mltl False-mltl*, *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ num-vars*. *x* = (*Not-mltl* (*And-mltl φ ψ*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ num-vars*. *x* = (*Not-mltl* (*Or-mltl φ ψ*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ a b num-vars*. *x* = (*Not-mltl* (*Future-mltl φ a b*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ a b num-vars*. *x* = (*Not-mltl* (*Global-mltl φ a b*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ a b num-vars*. *x* = (*Not-mltl* (*Until-mltl φ ψ a b*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ ψ a b num-vars*. *x* = (*Not-mltl* (*Release-mltl φ ψ a b*), *num-vars*) ⟹ *P*) ⟹
(⋀*φ num-vars*. *x* = (*Not-mltl* (*Not-mltl φ*), *num-vars*) ⟹ *P*) ⟹ *P*
⟨*proof*⟩

fun *WEST-termination-measure*:: (*nat*) *mltl* ⇒ *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 φ::(*nat*) *mltl*
shows *WEST-termination-measure* (*Not-mltl* φ) = *1* + *3* ∗ (*WEST-termination-measure* φ)
⟨*proof*⟩

function *WEST-reg-aux*:: (*nat*) *mltl* ⇒ *nat* ⇒ *WEST-regex*
where *WEST-reg-aux* $True_m$ *num-vars* = [[(*map* (λ *j*. *S*) [*0* ..< *num-vars*])]]
| *WEST-reg-aux* $False_m$ *num-vars* = []
| *WEST-reg-aux* ($Prop_m$ (*p*)) *num-vars* = [[(*map* (λ *j*. (*if* (*p*=*j*) *then One else S*)) [*0* ..< *num-vars*])]]
| *WEST-reg-aux* (Not_m ($Prop_m$ (*p*))) *num-vars* = [[(*map* (λ *j*. (*if* (*p*=*j*) *then Zero 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* ⇒ *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* ⇒ *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* ⇒ *WEST-regex*
where *pad-WEST-reg* φ = (*let unpadded* = *WEST-reg* φ *in*

(*let complen* = *complen-mltl* φ *in*
(*let num-vars* = *WEST-num-vars* φ *in*
(*map* (λ *L*. (*if* (*length L* < *complen*)*then* (*pad L num-vars* (*complen*−(*length L*))) *else L*))) *unpadded*)))

fun *simp-pad-WEST-reg*:: *nat mltl* ⇒ *WEST-regex*
where *simp-pad-WEST-reg* φ = *WEST-simp* (*pad-WEST-reg* φ) (*WEST-num-vars* φ)

2 Some examples and Code Export

Base cases

value *WEST-reg* $True_m$
value *WEST-reg* $False_m$
value *WEST-reg* ($Prop_m$ (*1*))
value *WEST-reg* (Not_m ($Prop_m$ (*0*)))

Test cases for recursion

value *WEST-reg* ((Not_m ($Prop_m$ (*0*))) And_m ($Prop_m$ (*1*)))
value *WEST-reg* (F_m[*0*,*2*] ($Prop_m$ (*1*)))
value *WEST-reg* ((Not_m ($Prop_m$ (*0*))) Or_m ($Prop_m$ (*0*)))

value *pad-WEST-reg* (($Prop_m$ (*0*)) U_m[*0*,*2*] ($Prop_m$ (*0*)))
value *simp-pad-WEST-reg* ((*Prop-mltl 0*) U_m[*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* ⇒ *nat* ⇒ *bool*
where *trace-of-vars trace num-vars* = (
∀ *k*. (*k* < (*length trace*) ⟶ (∀ *p*∈(*trace*!*k*). *p* < *num-vars*)))

definition *state-regex-of-vars*::*state-regex* ⇒ *nat* ⇒ *bool*
where *state-regex-of-vars state num-vars* = ((*length state*) = *num-vars*)

definition *trace-regex-of-vars*::*trace-regex* ⇒ *nat* ⇒ *bool*
where *trace-regex-of-vars trace num-vars* =
(∀ *i* < (*length trace*). *length* (*trace*!*i*) = *num-vars*)

definition *WEST-regex-of-vars*::*WEST-regex* ⇒ *nat* ⇒ *bool*
where *WEST-regex-of-vars traceList num-vars* =
(∀ *k*<*length traceList*. *trace-regex-of-vars* (*traceList*!*k*) *num-vars*)

3.2 Proofs about Traces Matching Regular Expressions

value *match-regex* [{*0*::*nat*}, {*0*,*1*}, {}] []
lemma *arbitrary-regtrace-matches-any-trace*:
fixes *num-vars*::*nat*
fixes π::*trace*
assumes π*-of-num-vars*: *trace-of-vars* π *num-vars*
shows *match-regex* π []
⟨*proof*⟩

lemma *WEST-and-state-difflengths-is-none*:
assumes *length s1* ≠ *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* ⇒ *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 (*WEST-and-trace regtrace1 regtrace2*) = (*WEST-and-trace regtrace2 regtrace1*)
⟨*proof*⟩

lemma *WEST-and-helper-subset*:
shows *set* (*WEST-and-helper h L*) ⊆ *set* (*WEST-and-helper h* (*a* # *L*))
⟨*proof*⟩

lemma *WEST-and-helper-set-member-converse*:
fixes *regtrace h*::*trace-regex*
fixes *L*::*WEST-regex*
assumes *assumption*: (∃ *loc*. *loc* < *length L* ∧ (∃ *sometrace*. *WEST-and-trace h* (*L* ! *loc*) = *Some sometrace* ∧ *regtrace* = *sometrace*))
shows *regtrace* ∈ *set* (*WEST-and-helper h L*)
⟨*proof*⟩

lemma *WEST-and-helper-set-member-forward*:
fixes *regtrace h*::*trace-regex*
fixes *L*::*WEST-regex*
assumes *regtrace* ∈ *set* (*WEST-and-helper h L*)
shows (∃ *loc*. *loc* < *length L* ∧ (∃ *sometrace*. *WEST-and-trace h* (*L* ! *loc*) = *Some sometrace* ∧ *regtrace* = *sometrace*))
⟨*proof*⟩

lemma *WEST-and-helper-set-member*:
fixes *regtrace h*::*trace-regex*
fixes *L*::*WEST-regex*
shows *regtrace* ∈ *set* (*WEST-and-helper h L*) ⟷
(∃ *loc*. *loc* < *length L* ∧ (∃ *sometrace*. *WEST-and-trace h* (*L* ! *loc*) = *Some sometrace* ∧ *regtrace* = *sometrace*))
⟨*proof*⟩

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 *regtrace* ∈ *set* (*WEST-and L1 L2*)
shows (∃ *loc1 loc2*. *loc1* < *length L1* ∧ *loc2* < *length L2* ∧
(∃ *sometrace*. *WEST-and-trace* (*L1* ! *loc1*) (*L2* ! *loc2*) = *Some sometrace* ∧ *regtrace* = *sometrace*))
⟨*proof*⟩

lemma *WEST-and-subset*:
shows *set* (*WEST-and T1 L2*) ⊆ *set* (*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*: (∃ *loc1 loc2*. *loc1* < *length L1* ∧ *loc2* < *length L2* ∧
(∃ *sometrace*. *WEST-and-trace* (*L1* ! *loc1*) (*L2* ! *loc2*) = *Some sometrace* ∧ *regtrace* = *sometrace*))
shows *regtrace* ∈ *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* ∈ *set* (*WEST-and L1 L2*) ⟷
(∃ *loc1 loc2*. *loc1* < *length L1* ∧ *loc2* < *length L2* ∧
(∃ *sometrace*. *WEST-and-trace* (*L1* ! *loc1*) (*L2* ! *loc2*) = *Some sometrace* ∧ *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* ∈ *set* (*WEST-and L1 L2*)
shows *regtrace* ∈ *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* = $\{x-1 \mid x.\ (x \in state \land x \neq 0)\}$

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

lemma *advance-state-elt-member*:
fixes *state*::*state*
fixes *x*::*nat*
assumes $x+1 \in state$
shows $x \in advance\text{-}state\ state$
⟨*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-of-num-vars*: *state-regex-of-vars s2 num-vars*
assumes *π-match-r1-r2*: *match-timestep state s1* ∧ *match-timestep state s2*
shows *WEST-and-state s1 s2* ≠ *None*
⟨*proof*⟩

Correct Forward **lemma** *WEST-and-state-length*:
fixes *s1 s2*::*state-regex*
assumes *samelen*: *length s1* = *length s2*
assumes *r-exists*: (*WEST-and-state s1 s2*) ≠ *None*
shows ∃ *r*. *length r* = *length s1* ∧ *WEST-and-state s1 s2* = *Some r*
⟨*proof*⟩

lemma *index-shift*:
fixes *a*::*WEST-bit*
fixes *t*::*state-regex*
fixes *state*::*state*
assumes (*a* = *One* ⟶ *0* ∈ *state*) ∧ (*a* = *Zero* ⟶ *0* ∉ *state*)
assumes ∀ *x*<*length t*. ((*t*!*x*) = *One* ⟶ *x* + *1* ∈ *state*) ∧ ((*t*!*x*) = *Zero* ⟶ *x* + *1* ∉ *state*)
shows ∀ *x*<*length* (*a*#*t*). ((*a*#*t*) ! *x* = *One* ⟶ *x* ∈ *state*) ∧ ((*a*#*t*) ! *x* = *Zero* ⟶ *x* ∉ *state*)
⟨*proof*⟩

lemma *index-shift-reverse*:
fixes *a*::*WEST-bit*
fixes *t*::*state-regex*
fixes *state*::*state*
assumes ∀ *x*<*length* (*a*#*t*). ((*a*#*t*) ! *x* = *One* ⟶ *x* ∈ *state*) ∧ ((*a*#*t*) ! *x* = *Zero* ⟶ *x* ∉ *state*)
shows ∀ *x*<*length t*. ((*t*!*x*) = *One* ⟶ *x* + *1* ∈ *state*) ∧ ((*t*!*x*) = *Zero* ⟶ *x* + *1* ∉ *state*)
⟨*proof*⟩

lemma *WEST-and-state-correct-forward*:
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-of-num-vars*: *state-regex-of-vars s2 num-vars*
assumes *match-both*: *match-timestep state s1* ∧ *match-timestep state s2*
shows ∃ *somestate*. (*match-timestep state somestate*) ∧ (*WEST-and-state s1 s2*) = *Some somestate*
⟨*proof*⟩

Correct Converse **lemma** *WEST-and-state-indices*:
fixes *s s1 s2*::*state-regex*

assumes *WEST-and-state s1 s2* = *Some s*
assumes *length s1* = *length s2*
assumes *x*<*length s*
shows *Some* (*s*!*x*) = *WEST-and-bitwise* (*s1*!*x*) (*s2*!*x*)
⟨*proof*⟩

lemma *WEST-and-state-correct-converse-s1*:
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-of-num-vars*: *state-regex-of-vars s2 num-vars*
assumes *match-and*: ∃ *somestate*. (*match-timestep state somestate*) ∧ (*WEST-and-state s1 s2*) = *Some somestate*
shows *match-timestep state s1*
⟨*proof*⟩

lemma *WEST-and-state-correct-converse*:
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-of-num-vars*: *state-regex-of-vars s2 num-vars*
assumes *match-and*: ∃ *somestate*. (*match-timestep state somestate*) ∧ (*WEST-and-state s1 s2*) = *Some somestate*
shows *match-timestep state s1* ∧ *match-timestep state s2*
⟨*proof*⟩

lemma *WEST-and-state-correct*:
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-of-num-vars*: *state-regex-of-vars s2 num-vars*
shows (*match-timestep state s1* ∧ *match-timestep state s2*) ⟷ (∃ *somestate*. *match-timestep state somestate* ∧ (*WEST-and-state s1 s2*) = *Some somestate*)
⟨*proof*⟩

3.3.4 WEST-and-trace

Well Defined **lemma** *WEST-and-trace-well-defined*:
fixes *num-vars*::*nat*
fixes *π*::*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 *π-match-r1-r2*: *match-regex π r1* ∧ *match-regex π r2*
shows *WEST-and-trace r1 r2* ≠ *None*

⟨*proof*⟩

Correct Forward **lemma** *WEST-and-trace-correct-forward-aux*:
assumes *match-regex* π (*h#t*)
shows *match-timestep* (π!*0*) *h* $\wedge$ *match-regex* (*drop 1* π) *t*
⟨*proof*⟩

lemma *WEST-and-trace-correct-forward-aux-converse*:
assumes π = *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 π::*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 *match1*: *match-regex* [] *r1*
assumes *match2*: *match-regex* [] *r2*
shows $\exists$ *sometrace*. *match-regex* [] *sometrace* $\wedge$ (*WEST-and-trace r1 r2*) = *Some sometrace*
⟨*proof*⟩

lemma *WEST-and-trace-correct-forward-nonempty-trace*:
fixes *num-vars*::*nat*
fixes π::*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 *match-regex* π *r1* $\wedge$ *match-regex* π *r2*
assumes *length* $\pi > 0$
shows $\exists$ *sometrace*. *match-regex* π *sometrace* $\wedge$ (*WEST-and-trace r1 r2*) = *Some sometrace*
⟨*proof*⟩

lemma *WEST-and-trace-correct-forward*:
fixes *num-vars*::*nat*
fixes π::*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 *match-regex* π *r1* $\wedge$ *match-regex* π *r2*
shows $\exists$ *sometrace*. *match-regex* π *sometrace* $\wedge$ (*WEST-and-trace r1 r2*) = *Some sometrace*
⟨*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*) = *Some* (*h* # *t*)
shows *WEST-and-state h1 h2* = *Some h* ∧ *WEST-and-trace t1 t2* = *Some t*
⟨*proof*⟩

lemma *WEST-and-trace-lengths-r1*:
assumes *trace-regex-of-vars r1 n*
assumes *trace-regex-of-vars r2 n*
assumes (*WEST-and-trace r1 r2*) = *Some sometrace*
shows *length sometrace* ≥ *length r1*
⟨*proof*⟩

lemma *WEST-and-trace-lengths*:
assumes *trace-regex-of-vars r1 n*
assumes *trace-regex-of-vars r2 n*
assumes (*WEST-and-trace r1 r2*) = *Some sometrace*
shows *length sometrace* ≥ *length r1* ∧ *length sometrace* ≥ *length r2*
⟨*proof*⟩

lemma *WEST-and-trace-correct-converse-r1*:
fixes *num-vars*::*nat*
fixes π::*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 (∃ *sometrace*. *match-regex* π *sometrace* ∧ (*WEST-and-trace r1 r2*) = *Some sometrace*)
shows *match-regex* π *r1*
⟨*proof*⟩

lemma *WEST-and-trace-correct-converse*:
fixes *num-vars*::*nat*
fixes π::*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 (∃ *sometrace*. *match-regex* π *sometrace* ∧ (*WEST-and-trace r1 r2*) = *Some sometrace*)
shows *match-regex* π *r1* ∧ *match-regex* π *r2*
⟨*proof*⟩

lemma *WEST-and-trace-correct*:
fixes *num-vars*::*nat*
fixes π::*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*
shows *match-regex* π *r1* $\wedge$ *match-regex* π *r2* $\longleftrightarrow$ ($\exists$ *sometrace*. *match-regex* π *sometrace* $\wedge$ (*WEST-and-trace r1 r2*) = *Some sometrace*)
⟨*proof*⟩

3.3.5 WEST-and correct

Correct Forward **lemma** *WEST-and-helper-subset-of-WEST-and*:
assumes *List.member L1 elem*
shows *set* (*WEST-and-helper elem* (*h2#T2*)) $\subseteq$ *set* (*WEST-and L1* (*h2#T2*))
⟨*proof*⟩

lemma *WEST-and-trace-element-of-WEST-and-helper*:
assumes *List.member L2 elem2*
assumes (*WEST-and-trace elem1 elem2*) = *Some sometrace*
shows *sometrace* $\in$ *set* (*WEST-and-helper elem1 L2*)
⟨*proof*⟩

lemma *index-of-L-in-L*:
assumes $i <$ *length L*
shows *List.member L* (*L* ! *i*)
⟨*proof*⟩

lemma *WEST-and-indices*:
fixes *L1 L2*::*WEST-regex*
fixes *sometrace*::*trace-regex*
assumes $\exists$ *i1 i2*. *i1* < *length L1* $\wedge$ *i2* < *length L2* $\wedge$ *WEST-and-trace* (*L1* ! *i1*) (*L2* ! *i2*) = *Some sometrace*
shows $\exists$ *i*<*length* (*WEST-and L1 L2*). *WEST-and L1 L2* ! *i* = *sometrace*
⟨*proof*⟩

lemma *WEST-and-correct-forward*:
fixes *n*::*nat*
fixes π::*trace*
fixes *L1 L2*:: *WEST-regex*
assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
assumes *match* π *L1* $\wedge$ *match* π *L2*
shows *match* π (*WEST-and L1 L2*)
⟨*proof*⟩

Correct Converse **lemma** *WEST-and-correct-converse-L1*:
fixes *n*::*nat*
fixes π::*trace*
fixes *L1 L2*:: *WEST-regex*
assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
assumes *match* π (*WEST-and L1 L2*)
shows *match* π *L1*

⟨*proof*⟩

lemma *WEST-and-correct-converse*:
  fixes *n*::*nat*
  fixes *π*::*trace*
  fixes *L1 L2*:: *WEST-regex*
  assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
  assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
  assumes *match π* (*WEST-and L1 L2*)
  shows *match π L1* ∧ *match π L2*
⟨*proof*⟩

lemma *WEST-and-correct*:
  fixes *π*::*trace*
  fixes *L1 L2*:: *WEST-regex*
  assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
  assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
  shows *match π L1* ∧ *match π L2* ⟷ *match π* (*WEST-and L1 L2*)
⟨*proof*⟩

3.4 Facts about the WEST or operator

lemma *WEST-or-correct*:
  fixes *π*::*trace*
  fixes *L1 L2*::*WEST-regex*
  shows *match π* (*L1*@*L2*) ⟷ (*match π L1*) ∨ (*match π L2*)
⟨*proof*⟩

3.5 Pad and Match Facts

lemma *shift-match-regex*:
  assumes *length π* ≥ *a*
  assumes *match-regex π* ((*arbitrary-trace num-vars a*)@*L*)
  shows *match-regex* (*drop a π*) (*drop a* ((*arbitrary-trace num-vars a*)@*L*))
⟨*proof*⟩

lemma *match-regex*:
  assumes *length π* ≥ *a*
  assumes *length L1* = *a*
  assumes *match-regex π* (*L1*@*L2*)
  shows *match-regex* (*drop a π*) (*drop a* (*L1*@*L2*))
⟨*proof*⟩

lemma *match-regex-converse*:
  assumes *length π* ≥ *a*
  assumes *L1* = (*arbitrary-trace num-vars a*)
  assumes *match-regex* (*drop a π*) (*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 *regtrace-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-nv*: *WEST-regex-of-vars L1 k*
assumes *L2-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-vars*
assumes *WEST-regex-of-vars T num-vars*
shows *WEST-regex-of-vars* (*h#T*) *num-vars*
⟨*proof*⟩

lemma *WEST-simp-state-num-vars*:
assumes *length s1* = *num-vars*
assumes *length s2* = *num-vars*
shows *length* (*WEST-simp-state s1 s2*) = *num-vars*
⟨*proof*⟩

lemma *WEST-get-state-length*:
assumes *trace-regex-of-vars r num-vars*
shows *length* (*WEST-get-state r k num-vars*) = *num-vars*
⟨*proof*⟩

lemma *WEST-simp-trace-num-vars*:
assumes *trace-regex-of-vars r1 num-vars*
assumes *trace-regex-of-vars r2 num-vars*
shows *trace-regex-of-vars* (*WEST-simp-trace r1 r2 num-vars*) *num-vars*
⟨*proof*⟩

lemma *remove-element-at-index-preserves-nv*:
assumes *i* < *length L*
assumes *WEST-regex-of-vars L num-vars*
shows *WEST-regex-of-vars* (*remove-element-at-index i L*) *num-vars*
⟨*proof*⟩

lemma *update-L-length*:
  assumes $h \in set\ (enum\text{-}pairs\ L)$
  shows $length\ (update\text{-}L\ L\ h\ num\text{-}var) = length\ L - 1$
⟨*proof*⟩

lemma *update-L-preserves-num-vars*:
  assumes *WEST-regex-of-vars L num-var*
  assumes $h \in set\ (enum\text{-}pairs\ L)$
  assumes $K = update\text{-}L\ L\ h\ num\text{-}var$
  shows *WEST-regex-of-vars K num-var*
⟨*proof*⟩

lemma *WEST-simp-helper-can-simp*:
  assumes $simp\text{-}L = WEST\text{-}simp\text{-}helper\ L\ (enum\text{-}pairs\ L)\ i\ num\text{-}vars$
  assumes $\exists j.\ j < length\ (enum\text{-}pairs\ L) \wedge j \geq i \wedge$
      $check\text{-}simp\ (L\ !\ fst\ (enum\text{-}pairs\ L\ !\ j))$
        $(L\ !\ snd\ (enum\text{-}pairs\ L\ !\ j))$
  assumes $min\text{-}j = Min\ \{j.\ j < length\ (enum\text{-}pairs\ L) \wedge j \geq i \wedge$
      $check\text{-}simp\ (L\ !\ fst\ (enum\text{-}pairs\ L\ !\ j))$
        $(L\ !\ snd\ (enum\text{-}pairs\ L\ !\ j))\}$
  assumes $newL = update\text{-}L\ L\ (enum\text{-}pairs\ L\ !\ min\text{-}j)\ num\text{-}vars$
  assumes $i < length\ (enum\text{-}pairs\ L)$
  shows $simp\text{-}L = WEST\text{-}simp\text{-}helper\ newL\ (enum\text{-}pairs\ newL)\ 0\ num\text{-}vars$
⟨*proof*⟩

lemma *WEST-simp-helper-cant-simp*:
  assumes $simp\text{-}L = WEST\text{-}simp\text{-}helper\ L\ (enum\text{-}pairs\ L)\ i\ num\text{-}vars$
  assumes $\neg(\exists j.\ j < length\ (enum\text{-}pairs\ L) \wedge j \geq i \wedge$
      $check\text{-}simp\ (L\ !\ fst\ (enum\text{-}pairs\ L\ !\ j))$
        $(L\ !\ snd\ (enum\text{-}pairs\ L\ !\ j)))$
  shows $simp\text{-}L = L$
  ⟨*proof*⟩

lemma *WEST-simp-helper-length*:
  shows $length\ (WEST\text{-}simp\text{-}helper\ L\ (enum\text{-}pairs\ L)\ i\ num\text{-}vars) \leq length\ L$
⟨*proof*⟩

lemma *WEST-simp-helper-num-vars*:
  assumes *WEST-regex-of-vars L num-vars*
  shows *WEST-regex-of-vars (WEST-simp-helper L (enum-pairs L) i num-vars) num-vars*
  ⟨*proof*⟩

lemma *WEST-simp-num-vars*:
  assumes *WEST-regex-of-vars L num-vars*
  shows *WEST-regex-of-vars (WEST-simp L num-vars) 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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*: $\exists\ \psi.\ F1 = (convert\text{-}nnf\ \psi)$
assumes $k \geq$ *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 $k < length\ r1 \wedge k < length\ r2$
shows *count-diff r1 r2* $\geq$ *count-diff-state* (*r1* ! *k*) (*r2* ! *k*)
⟨*proof*⟩

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

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

lemma *count-nonS-trace-0-allS*:
assumes *length h = num-vars*
assumes *count-nonS-trace h = 0*
shows *h = map* (*λt. S*) [*0..<num-vars*]
⟨*proof*⟩

lemma *trace-tail-num-vars*:
assumes *trace-regex-of-vars* (*h # trace*) *num-vars*
shows *trace-regex-of-vars trace num-vars*
⟨*proof*⟩

lemma *count-diff-property-S-aux*:
assumes *count-diff trace* [] = *0*
assumes *k < length trace*
assumes *trace-regex-of-vars trace num-vars*
assumes *1 ≤ num-vars*
shows *trace* ! *k = map* (*λt. S*) [*0 ..< num-vars*]
⟨*proof*⟩

lemma *count-diff-property-S*:
assumes *count-diff r1 r2 = 0*
assumes *k < length r1 ∧ length r2 ≤ k*
assumes *trace-regex-of-vars r1 num-vars*
assumes *num-vars ≥ 1*
assumes *ka < num-vars*
shows *r1* ! *k = map* (*λt. S*) [*0..<num-vars*]
⟨*proof*⟩

lemma *count-diff-state-commutative*:
shows *count-diff-state e1 e2 = count-diff-state e2 e1*
⟨*proof*⟩

lemma *count-diff-commutative*:
shows *count-diff r1 r2 = count-diff r2 r1*
⟨*proof*⟩

lemma *count-diff-same-trace*:
shows *count-diff trace trace = 0*
⟨*proof*⟩

lemma *count-diff-state-0*:
assumes *count-diff-state h1 h2 = 0*
assumes *length h1 = length h2*
shows *h1 = h2*
⟨*proof*⟩

lemma *count-diff-state-1*:
assumes *length h1 = length h2*
assumes *count-diff-state h1 h2 = 1*
shows $\exists$ *ka<length h1. h1!ka* $\neq$ *h2!ka*
⟨*proof*⟩

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

lemma *count-diff-same-len*:
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 = length r2*
shows *r1 = r2*
⟨*proof*⟩

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

lemma *count-diff-1-other-states*:
assumes *count-diff r1 r2 = 1*
assumes *length r1 = length r2*
assumes *trace-regex-of-vars r1 num-vars*
assumes *trace-regex-of-vars r2 num-vars*
assumes *count-diff-state (r1!k) (r2!k) = 1*
shows $\forall$ *i<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 *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* ≥ *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* π *r1* ∨ *match-regex* π *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* π *r1* ∨ *match-regex* π *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* < *length* (*enum-pairs L*) $\wedge$ *j* $\geq$ *i* $\wedge$
check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
(*L* ! *snd* (*enum-pairs L* ! *j*))
assumes *i* < *length* (*enum-pairs L*)
shows *length simp-L* < *length L*
⟨proof⟩

lemma *WEST-simp-helper-same-length*:
assumes *WEST-regex-of-vars L num-vars*
assumes *K* = *WEST-simp-helper L* (*enum-pairs L*) *0 num-vars*
assumes *length K* = *length L*
shows *L* = *K*
⟨proof⟩

lemma *WEST-simp-helper-less-length*:
assumes *WEST-regex-of-vars L num-vars*
assumes *length K* < *length L*
assumes *K* = *WEST-simp-helper L* (*enum-pairs L*) *0 num-vars*
shows $\exists$ *min-j*.
(*min-j* < *length* (*enum-pairs L*) $\wedge$
K =
WEST-simp-helper (*update-L L* (*enum-pairs L* ! *min-j*) *num-vars*)
(*enum-pairs*
(*update-L L* (*enum-pairs L* ! *min-j*) *num-vars*))
0 num-vars
$\wedge$ *check-simp* (*L* ! *fst* (*enum-pairs L* ! *min-j*)) (*L* ! *snd* (*enum-pairs L* ! *min-j*)))
⟨proof⟩

lemma *remove-element-at-index-subset*:
fixes *i*::*nat*
assumes *i* < *length L*
shows *set* (*remove-element-at-index i L*) $\subseteq$ *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* ≠ *j1*
  assumes *i* ≠ *j2*
  shows *L* ! *i*
    ∈ *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* ∈ *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 *match* π (*WEST-simp L num-vars*) $\longleftrightarrow$ *match* π *L*
⟨*proof*⟩

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

lemma *WEST-and-simp-correct*:
fixes π::*trace*
fixes *L1 L2*:: *WEST-regex*
assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
shows *match* π *L1* $\wedge$ *match* π *L2* $\longleftrightarrow$ *match* π (*WEST-and-simp L1 L2 n*)
⟨*proof*⟩

lemma *WEST-or-simp-correct*:
fixes π::*trace*
fixes *L1 L2*:: *WEST-regex*
assumes *L1-of-num-vars*: *WEST-regex-of-vars L1 n*
assumes *L2-of-num-vars*: *WEST-regex-of-vars L2 n*
shows *match* π *L1* $\vee$ *match* π *L2* $\longleftrightarrow$ *match* π (*WEST-or-simp L1 L2 n*)
⟨*proof*⟩

3.9 Facts about the WEST future operator

lemma *WEST-future-correct-forward*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*
assumes *match* π (*WEST-future L a b num-vars*)
shows $\pi \models_m (F_m\ [a,b]\ F)$
⟨*proof*⟩

lemma *WEST-future-correct-converse*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*
assumes $\pi \models_m$ (*Future-mltl a b F*)

shows *match* π (*WEST-future L a b num-vars*)
⟨*proof*⟩

lemma *WEST-future-correct*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*
shows *match* π (*WEST-future L a b num-vars*) $\longleftrightarrow$
semantics-mltl π (*Future-mltl a b F*)
⟨*proof*⟩

3.10 Facts about the WEST global operator

lemma *WEST-global-correct-forward*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*
assumes *match* π (*WEST-global L a b num-vars*)
shows *semantics-mltl* π (*Global-mltl a b F*)
⟨*proof*⟩

lemma *WEST-global-correct-converse*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*
assumes *semantics-mltl* π (*Global-mltl a b F*)
shows *match* π (*WEST-global L a b num-vars*)
⟨*proof*⟩

lemma *WEST-global-correct*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F* $\longrightarrow$ (*match* π *L* $\longleftrightarrow$ *semantics-mltl* π *F*))
assumes *WEST-regex-of-vars L num-vars*
assumes *WEST-num-vars F* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl F*) + *b*

shows *match* π (*WEST-global L a b num-vars*) $\longleftrightarrow$
semantics-mltl π (*Global-mltl a b F*)
⟨*proof*⟩

3.11 Facts about the WEST until operator

lemma *WEST-until-correct-forward*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ *complen-mltl* (*Until-mltl F1 a b F2*)
assumes *match* π (*WEST-until L1 L2 a b num-vars*)
shows *semantics-mltl* π (*Until-mltl F1 a b F2*)
⟨*proof*⟩

lemma *WEST-until-correct-converse*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ (*complen-mltl* (*Until-mltl F1 a b F2*))
assumes *semantics-mltl* π (*Until-mltl F1 a b F2*)
shows *match* π (*WEST-until L1 L2 a b num-vars*)
⟨*proof*⟩

lemma *WEST-until-correct*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ *complen-mltl* (*Until-mltl F1 a b F2*)

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

3.12 Facts about the WEST release Operator

lemma *WEST-release-correct-forward*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes *a-leq-b*: $a \leq b$
assumes *len*: *length* $\pi \geq$ *complen-mltl* (*Release-mltl F1 a b F2*)
assumes *match* π (*WEST-release L1 L2 a b num-vars*)
shows *semantics-mltl* π (*Release-mltl F1 a b F2*)
⟨*proof*⟩

lemma *WEST-release-correct-converse*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ *complen-mltl* (*Release-mltl F1 a b F2*)
assumes *semantics-mltl* π (*Release-mltl F1 a b F2*)
shows *match* π (*WEST-release L1 L2 a b num-vars*)
⟨*proof*⟩

lemma *WEST-release-correct*:
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F1* $\longrightarrow$ (*match* π *L1* $\longleftrightarrow$ *semantics-mltl* π *F1*))
assumes $\bigwedge\pi$. (*length* $\pi \geq$ *complen-mltl F2* $\longrightarrow$ (*match* π *L2* $\longleftrightarrow$ *semantics-mltl* π *F2*))
assumes *WEST-regex-of-vars L1 num-vars*
assumes *WEST-regex-of-vars L2 num-vars*
assumes *WEST-num-vars F1* $\leq$ *num-vars*
assumes *WEST-num-vars F2* $\leq$ *num-vars*
assumes $a \leq b$
assumes *length* $\pi \geq$ *complen-mltl* (*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*: *length* $\pi \geq$ *complen-mltl F*
assumes *is-nnf*: $\exists\ \psi.\ F = ($*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 φ::(*nat*) *mltl*
fixes π::*trace*
assumes *int-welldef*: *intervals-welldef* φ
assumes π*-long-enough*: *length* $\pi \geq$ *complen-mltl* (*convert-nnf* φ)
shows *match* π (*WEST-reg* φ) $\longleftrightarrow$ *semantics-mltl* π φ
⟨*proof*⟩

lemma *WEST-correct-v2*:
fixes φ::(*nat*) *mltl*
fixes π::*trace*
assumes *intervals-welldef* φ
assumes π*-long-enough*: *length* $\pi \geq$ *complen-mltl* φ
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 φ::(*nat*) *mltl*
fixes π::*trace*
assumes *intervals-welldef* φ
assumes π*-long-enough*: *length* $\pi \geq$ *complen-mltl* φ

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

lemma *WEST-correct-pad*:
fixes φ::(*nat*) *mltl*
fixes π::*trace*
assumes *intervals-welldef* φ
assumes π-*long-enough*: *length* π ≥ *complen-mltl* φ
shows *match* π (*simp-pad-WEST-reg* φ) ⟷ *semantics-mltl* π φ
⟨*proof*⟩

end

4 Key algorithms for WEST

theory *Regex-Equivalence*

imports *WEST-Algorithms WEST-Proofs*

begin

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

function *enumerate-list*:: *state-regex* ⇒ *trace-regex*
where *enumerate-list* [] = [[]]
| *enumerate-list* (*One*#*T*) = (*map* (λ*x*. *One*#*x*) (*enumerate-list* *T*))
| *enumerate-list* (*Zero*#*T*) = (*map* (λ*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* ⇒ *'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 (λ*t*. (*map* (λ*h*. *h*#*t*) *enumerate-H*)) *enumerate-T*)

fun *enumerate-trace*:: *trace-regex* ⇒ *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 (λ*t*. (*map* (λ*h*. *h*#*t*) *enumerate-H*)) *enumerate-T*)

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

fun *enumerate-sets*:: *WEST-regex* ⇒ *trace-regex set*
  where *enumerate-sets* [] = {}
  | *enumerate-sets* (*h*#*T*) = (*set* (*enumerate-trace h*)) ∪ (*enumerate-sets T*)

fun *naive-equivalence*:: *WEST-regex* ⇒ *WEST-regex* ⇒ *bool*
  where *naive-equivalence A B* = (*A* = *B* ∨ (*enumerate-sets A*) = (*enumerate-sets B*))

5 Regex Equivalence Correctness

lemma *enumerate-list-len-alt*:
  shows ∀ *state* ∈ *set* (*enumerate-list state-regex*).
        length state = *length state-regex*
⟨*proof*⟩

lemma *enumerate-list-len*:
  assumes *state* ∈ *set* (*enumerate-list state-regex*)
  shows *length state* = *length state-regex*
  ⟨*proof*⟩

lemma *enumerate-list-prop*:
  assumes (⋀*k*. *List.member j k* ⟹ *k* ≠ *S*)
  shows *enumerate-list j* = [*j*]
  ⟨*proof*⟩

lemma *enumerate-fixed-trace*:
  fixes *h1*:: *trace-regex*
  assumes ⋀*j*. *List.member h1 j* ⟹ (⋀*k*. *List.member j k* ⟹ *k* ≠ *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 $\bigwedge j$. *List.member h1 j* $\Longrightarrow$ ($\bigwedge k$. *List.member j k* $\Longrightarrow$ $k \neq S$)
assumes $\bigwedge j$. *List.member h2 j* $\Longrightarrow$ ($\bigwedge k$. *List.member j k* $\Longrightarrow$ $k \neq S$)
assumes (*set h1*) $\neq$ (*set h2*)
shows *set* (*enumerate-trace h1*) $\neq$ *set* (*enumerate-trace h2*)
⟨*proof*⟩

lemma *enumerate-list-tail-in*:
assumes *head-t#tail-t* $\in$ *set* (*enumerate-list* (*h#trace*))
shows *tail-t* $\in$ *set* (*enumerate-list trace*)
⟨*proof*⟩

lemma *enumerate-list-fixed*:
assumes $t \in$ *set* (*enumerate-list trace*)
shows ($\forall k$. *List.member t k* $\longrightarrow$ $k \neq S$)
⟨*proof*⟩

lemma *map-enum-list-nonempty*:
fixes *t*:: *WEST-bit list list*
fixes *head*:: *WEST-bit list*
shows *map* (λh. *h # t*) (*enumerate-list head*) $\neq$ []
⟨*proof*⟩

lemma *length-of-flatten-list*:
assumes *flat* =
foldr (@)
(*map* (λt. *map* (λh. *h # t*) *H*) *T*) []
shows *length flat* = *length T* $*$ *length H*
⟨*proof*⟩

lemma *flatten-list-idx*:
assumes *flat* = *flatten-list* (*map* (λt. *map* (λh. *h # t*) *head*) *tail*)
assumes $i <$ *length tail*
assumes $j <$ *length head*
shows (*head*!*j*)#(*tail*!*i*) = *flat*!(*i*$*$(*length head*) + *j*) $\wedge$ *i*$*$(*length head*) + *j* < *length flat*
⟨*proof*⟩

lemma *flatten-list-shape*:
assumes *List.member flat x1*
assumes *flat* = *flatten-list* (*map* (λt. *map* (λh. *h # t*) *H*) *T*)
shows $\exists$ *x1-head x1-tail*. *x1* = *x1-head#x1-tail* $\wedge$ *List.member H x1-head* $\wedge$ *List.member T x1-tail*
⟨*proof*⟩

lemma *flatten-list-len*:
assumes $\bigwedge t.$ *List.member T t* $\Longrightarrow$ *length t* $=$ *n*
assumes *flat* $=$ *flatten-list* (*map* ($\lambda t.$ *map* ($\lambda h.$ *h* # *t*) *H*) *T*)
shows $\bigwedge x1.$ *List.member flat x1* $\Longrightarrow$ *length x1* $=$ *n+1*
⟨*proof*⟩

lemma *flatten-list-lemma*:
assumes $\bigwedge x1.$ *List.member to-flatten x1* $\Longrightarrow$ ($\bigwedge x2.$ *List.member x1 x2* $\Longrightarrow$ *length x2* $=$ *length trace*)
assumes *a* $\in$ *set* (*flatten-list to-flatten*)
shows *length a* $=$ *length trace*
⟨*proof*⟩

lemma *enumerate-trace-len*:
assumes *a* $\in$ *set* (*enumerate-trace trace*)
shows *length a* $=$ *length trace*
⟨*proof*⟩

definition *regex-zeros-and-ones*:: *trace-regex* $\Rightarrow$ *bool*
where *regex-zeros-and-ones tr* $=$
($\forall$ *j. List.member tr j* $\longrightarrow$ ($\forall$ *k. List.member j k* $\longrightarrow$ *k* $\neq$ *S*))

lemma *match-enumerate-state-aux-first-bit*:
assumes *a-head* $=$ *Zero* $\vee$ *a-head* $=$ *One*
assumes *a-head* # *a-tail* $\in$ *set* (*enumerate-list* (*h-head* # *h*))
shows *h-head* $=$ *a-head* $\vee$ *h-head* $=$ *S*
⟨*proof*⟩

lemma *advance-state-iff*:
assumes *x* $>$ *0*
shows *x* $\in$ *state* $\longleftrightarrow$ (*x*$-$*1*) $\in$ *advance-state state*
⟨*proof*⟩

lemma *match-enumerate-state-aux*:
assumes *a* $\in$ *set* (*enumerate-list h*)
assumes *match-timestep state a*
shows *match-timestep state h*
⟨*proof*⟩

lemma *enumerate-list-index-one*:
assumes *j* $<$ *length* (*enumerate-list a*)
shows *One* # *enumerate-list a* ! *j* $=$ *enumerate-list* (*S* # *a*) ! (*length* (*enumerate-list a*) $+$ *j*) $\wedge$
(*length* (*enumerate-list a*) $+$ *j* $<$ *length* (*enumerate-list* (*S* # *a*)))

⟨*proof*⟩

lemma *list-concat-index*:
  assumes $j < length\ L1$
  shows $(L1@L2)!j = L1!j$
  ⟨*proof*⟩

lemma *enumerate-list-index-zero*:
  assumes $j < length\ (enumerate\text{-}list\ a)$
  shows $Zero\ \#\ enumerate\text{-}list\ a\ !\ j = enumerate\text{-}list\ (S\ \#\ a)\ !\ j \wedge$
    $j < length\ (enumerate\text{-}list\ (S\ \#\ a))$
  ⟨*proof*⟩

lemma *match-enumerate-list*:
  assumes *match-timestep state a*
  shows $\exists j<length\ (enumerate\text{-}list\ a).$
    $match\text{-}timestep\ state\ (enumerate\text{-}list\ a\ !\ j)$
  ⟨*proof*⟩

lemma *enumerate-trace-head-in*:
  assumes $a\text{-}head\ \#\ a\text{-}tail \in set\ (enumerate\text{-}trace\ (h\ \#\ trace))$
  shows $a\text{-}head \in set\ (enumerate\text{-}list\ h)$
⟨*proof*⟩

lemma *enumerate-trace-tail-in*:
  assumes $a\text{-}head\ \#\ a\text{-}tail \in set\ (enumerate\text{-}trace\ (h\ \#\ trace))$
  shows $a\text{-}tail \in set\ (enumerate\text{-}trace\ trace)$
⟨*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 $a \in set\ (enumerate\text{-}trace\ trace)$
  assumes $match\text{-}regex\ \pi\ a$
  shows $match\text{-}regex\ \pi\ trace$
⟨*proof*⟩

lemma *match-enumerate-trace*:
  assumes $a \in set\ (enumerate\text{-}trace\ h) \wedge match\text{-}regex\ \pi\ a$
  shows $match\ \pi\ (h\ \#\ T)$
⟨*proof*⟩

lemma *match-enumerate-sets1*:

assumes $(\exists\, r \in (\textit{enumerate-sets } R).\ \textit{match-regex } \pi\ r)$
shows $(\textit{match } \pi\ R)$
⟨*proof*⟩

lemma *match-cases*:
assumes $\textit{match } \pi\ (a \ \#\ R)$
shows $\textit{match } \pi\ [a] \lor \textit{match } \pi\ R$
⟨*proof*⟩

lemma *enumerate-trace-decompose*:
assumes $\textit{state} \in \textit{set } (\textit{enumerate-list } h)$
assumes $\textit{trace} \in \textit{set } (\textit{enumerate-trace } T)$
shows $\textit{state}\#\textit{trace} \in \textit{set } (\textit{enumerate-trace } (h\#T))$
⟨*proof*⟩

lemma *match-enumerate-trace-aux-converse*:
assumes $\textit{match-regex } \pi\ \textit{trace}$
shows $\textit{match } \pi\ (\textit{enumerate-trace trace})$
⟨*proof*⟩

lemma *match-enumerate-sets2*:
assumes $(\textit{match } \pi\ R)$
shows $(\exists\, r \in \textit{enumerate-sets } R.\ \textit{match-regex } \pi\ r)$
⟨*proof*⟩

lemma *match-enumerate-sets*:
shows $(\exists\, r \in \textit{enumerate-sets } R.\ \textit{match-regex } \pi\ r) \longleftrightarrow (\textit{match } \pi\ R)$
⟨*proof*⟩

lemma *regex-equivalence-correct1*:
assumes $(\textit{naive-equivalence } A\ B)$
shows $\textit{match } \pi\ A = \textit{match } \pi\ B$
⟨*proof*⟩

lemma *regex-equivalence-correct*:
shows $(\textit{naive-equivalence } A\ B) \longrightarrow (\textit{regex-equiv } A\ B)$
⟨*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.