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

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−
  let *?regexList* = [[[*Zero*,*Zero*]],[[*S*,*S*], [*S*,*One*]]]
  let *?trace* = [{*0*::*nat*,*1*}, {*1*}, {*0*}]
  have (*match-regex ?trace* (*?regexList*!*1*))
    unfolding *match-regex-def*
    by (*simp add*: *match-timestep-def nth-Cons′*)
  then show *?thesis*
   by (*metis One-nat-def add.commute le-imp-less-Suc le-numeral-extra*(*4*) *list.size*(*3*)
list.size(*4*) *match-def plus-1-eq-Suc*)
qed

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

lemma (*regex-equiv* [[[*S,S*]]]
  [[[*S,One*]],
   [[*One,S*]],
   [[*Zero,Zero*]]]) $=$ *True*
proof $-$
  have *d1*: *match π* [[[*S, One*]], [[*One, S*]], [[*Zero, Zero*]]] **if** *match*: *match π* [[[*S, S*]]] **for** *π*
  proof $-$
    have *match-ss*: *match-regex π* [[*S, S*]]
     using *match* **unfolding** *match-def*
     by (*metis One-nat-def length-Cons less-one list.size(3) nth-Cons-0*)
    {**assume** $*$: $\neg$ (*match-regex π* [[*S, One*]]) $\wedge$ $\neg$ (*match-regex π* [[*One, S*]])
     have *match-regex π* [[*Zero, Zero*]]
      using *match-ss* **unfolding** *match-regex-def*
      by (*smt (verit)* $*$ *One-nat-def WEST-bit.simps(2) length-Cons less-2-cases less-one list.size(3) match-regex-def match-timestep-def nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
    }
    then show *?thesis*
    unfolding *match-def*
    by (*metis length-Cons less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc*)
  qed
  have *d2*: *match π* [[[*S, S*]]] **if** *match*: *match π* [[[*S, One*]], [[*One, S*]], [[*Zero, Zero*]]] **for** *π*
  proof $-$
    {**assume** $*$: *match-regex π* [[*S, One*]]
    then have *match-regex π* [[*S, S*]]
     unfolding *match-regex-def*
      by (*smt (verit, ccfv-SIG) One-nat-def WEST-bit.simps(4) length-Cons less-2-cases less-one list.size(3) match-timestep-def nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
    then have *match π* [[[*S, S*]]]
     unfolding *match-def* **by** *simp*
    } **moreover** {**assume** $*$: *match-regex π* [[*One, S*]]
    then have *match-regex π* [[*S, S*]]
     unfolding *match-regex-def*
      by (*smt (verit, ccfv-SIG) One-nat-def WEST-bit.simps(4) length-Cons less-2-cases less-one list.size(3) match-timestep-def nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
    then have *match π* [[[*S, S*]]]
     unfolding *match-def* **by** *simp*
    } **moreover** {**assume** $*$: *match-regex π* [[*Zero, Zero*]]

4

then have *match-regex π [[S, S]]*
  unfolding *match-regex-def*
  by (*smt* (*verit*) *One-nat-def WEST-bit.distinct*(5) *length-Cons less-2-cases-iff*
less-one list.size(3) *match-timestep-def nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
then have *match π [[[S, S]]]*
  unfolding *match-def* **by** *simp*
  }
  ultimately show *?thesis* **using** *match* **unfolding** *regex-equiv-def*
  by (*smt* (*verit, del-insts*) *length-Cons less-Suc-eq-0-disj match-def nth-Cons-0*
nth-Cons-Suc)
qed
show *?thesis* **using** *d1 d2*
  unfolding *regex-equiv-def* **by** *metis*
qed

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 (*induction* *L*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons* *a* *L*)
  have *length-L*: (*length* (*a* # *L*))² = (*1* + (*length* *L*))^*2* **by** *auto*
  then have *length-L*: (*length* (*a* # *L*))² = *1* + *2*∗(*length* *L*) + (*length* *L*)^*2* **by**
algebra
  have *length* (*map* (*Pair* *a*) *L*) ≤ *length* *L*
    by *simp*
  then show *?case*
    unfolding *enumerate-pairs.simps* **using** *Cons* *length-L* **by** *simp*
qed

lemma *length-enum-pairs*:
  shows *length* (*enum-pairs* *L*) ≤ (*length* *L*)^*2*
proof−
  show *?thesis* **unfolding** *enum-pairs.simps* **using** *length-enumerate-pairs*
    by (*metis* *length-upt* *minus-nat.diff-0*)
qed

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*)
 using *assms*
proof(*induct length L arbitrary*:*L*)
 case *0*
 then show *?case* **by** *auto*
next
 case (*Suc x*)
 then obtain *h T* **where** *obt-hT*: *L* = *h*#*T*
  by (*metis length-Suc-conv*)
 then have *enum-L*: *enumerate-pairs L* = *map* (*Pair h*) *T* @ *enumerate-pairs T*
  using *enumerate-pairs.simps obt-hT* **by** *blast*
 then have ⋀ *pair*. *pair* ∈ *set* (*enumerate-pairs L*) ⟹ *fst pair* < *snd pair*
 proof−
  fix *pair*
  assume *pair* ∈ *set* (*enumerate-pairs L*)
  then have *pair* ∈ *set* (*map* (*Pair h*) *T* @ *enumerate-pairs T*) **using** *enum-L*
by *auto*
  then have *pair-or*: *pair* ∈ *set* (*map* (*Pair h*) *T*) ∨ *pair* ∈ *set*(*enumerate-pairs*
T)
   by *auto*
  {**assume** *in-base*: *pair* ∈ *set* (*map* (*Pair h*) *T*)
   have ∀ *j*. *0* < *j* ∧ *j* < *length L* ⟶ *h* < *L* ! *j*
    using *Suc.prems obt-hT* **by** *force*
   then have ∀ *j* < *length T*. *h* < *T*!*j*
    using *obt-hT* **by** *force*
   then have ∀ *t* ∈ *set T*. *h* < *t*
    using *obt-hT* **by** (*metis in-set-conv-nth*)
   then have *fst pair* < *snd pair*
    using *in-base* **by** *auto*
  } **moreover** {
   assume *in-rec*: *pair* ∈ *set*(*enumerate-pairs T*)
   have *fst pair* < *snd pair*
    using *Suc.hyps*(*1*)[*of T*] *Suc.prems obt-hT in-rec*
    by (*smt* (*verit, ccfv-SIG*) *Ex-less-Suc Suc.hyps*(*1*) *Suc.hyps*(*2*) *length-Cons*
less-trans-Suc nat.inject nth-Cons-Suc)
  }
  ultimately show *fst pair* < *snd pair* **using** *enum-L obt-hT pair-or* **by** *blast*
 qed
 then show *?case* **by** *blast*
qed

lemma *enum-pairs-fact*:
 shows ∀ *pair* ∈ *set* (*enum-pairs L*). (*fst pair*) < (*snd pair*)
 unfolding *enum-pairs.simps* **using** *enumerate-pairs-fact*[*of* [*0*..<*length L*]]
 by *simp*

lemma *enum-pairs-bound-snd*:

assumes *pair* ∈ *set* (*enumerate-pairs L*)
shows (*snd pair*) ∈ *set L*
using *assms*
proof (*induct length L arbitrary*: *L*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc x*)
  then obtain *h T* **where** *ht*: *L = h#T*
    by (*cases L*) *simp-all*
  then have *eo*: *pair* ∈ *set* (*map* (*Pair h*) *T*) ∨ *pair* ∈ *set* (*enumerate-pairs T*)
    using *Suc* **by** *simp*
  {**assume** ∗: *pair* ∈ *set* (*map* (*Pair h*) *T*)
    then have *?case*
      using *ht*
      using *imageE* **by** *auto*
  } **moreover** {**assume** ∗: *pair* ∈ *set* (*enumerate-pairs T*)
    then have *snd pair* ∈ *set T*
      using *Suc(1)*[*of T*] *ht*
      using *Suc.hyps(2)* **by** *fastforce*
    then have *?case* **using** *ht*
      by *simp*
  }
  ultimately show *?case* **using** *eo* **by** *blast*
qed

lemma *enum-pairs-bound*:
  shows ∀ *pair* ∈ *set* (*enum-pairs L*). (*snd pair*) < *length L*
  unfolding *enum-pairs.simps enumerate-pairs.simps*
proof(*induct length L arbitrary*: *L*)
  case *0*
  then show *?case* **by** *simp*
next
  case (*Suc x*)
  then have *enum-L*: *enumerate-pairs* ([*0..<length L*]) =
    map (*Pair 0*) [*1..<length L*] @ *enumerate-pairs* [*1..<length L*]
    using *enumerate-pairs.simps(2)*[*of 0* [*1 ..< length L*]]
    by (*metis One-nat-def upt-conv-Cons zero-less-Suc*)
  then have *pair*∈*set* (*enumerate-pairs* [*0..<length L*]) ⟹ *snd pair* < *length L*
for *pair*
    using *enum-pairs-bound-snd*[*of pair* [*0..<length L*]]
    by *auto*
  then show *?case* **unfolding** *enum-pairs.simps* **by** *blast*
qed

lemma *WEST-simp-termination1-bound*:
  fixes *a*::*nat*
  shows $a\char`^3 + a\char`^2 < (a+1)\char`^3$

9

proof −
  have *cubed*: $(a+1)\hat{\ }3 = a\hat{\ }3 + 3*a\hat{\ }2 + 3*a + 1$
  proof −
    have $(a+1)\hat{\ }3 = (a+1)*(a+1)*(a+1)$
      by *algebra*
    then show *?thesis*
      by (*simp add: add.commute add-mult-distrib2 mult.commute power2-eq-square power3-eq-cube*)
  qed
  have $0 < 2*a\hat{\ }2 + 2*a + 1$ **by** *simp*
  then have $a\hat{\ }3 + a\hat{\ }2 < a\hat{\ }3 + 3*a\hat{\ }2 + 3*a + 1$ **by** *simp*
  then show *?thesis* **using** *cubed*
    by *simp*
qed

lemma *WEST-simp-termination1*:
  fixes *L*::*WEST-regex*
  assumes $\neg$ (*idx-pairs* $\neq$ *enum-pairs L* $\vee$ *length idx-pairs* $\leq$ *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*)
    $\in$ *measure* (λ(*L*, *idx-list*, *i*, *num-vars*). *length L* $\hat{\ }3 + $ *length idx-list* $- i$)
proof −
  let *?i* = *fst* (*idx-pairs ! i*)
  let *?j* = *snd* (*idx-pairs ! i*)
  have *i-le-j*: *?i* < *?j* **using** *enum-pairs-fact assms*
    by (*metis linorder-le-less-linear nth-mem*)
  have *j-bound*: *?j* < *length L*
    using *assms*(*1*) *enum-pairs-bound*[*of L*]
    by *simp*
  then have *i-bound*: *?i* < (*length L*)−*1*
    using *i-le-j* **by** *auto*
  have *len-orsimp*: *length* [*WEST-simp-trace* (*L ! ?i*) (*L ! ?j*) *num-vars*] = *1*
    by *simp*
  have *length* (*remove-element-at-index ?j L*) = *length L* − *1*
    using *assms*(*3*) *j-bound* **by** *auto*
  then have *length* (*remove-element-at-index ?i* (*remove-element-at-index ?j L*))
= *length L* − *2*
    using *assms*(*3*) *i-bound j-bound* **by** *simp*
  then have *length-x*: *length x* = (*length L*) − *1*
    using *assms*(*3*) *len-orsimp*
    unfolding *update-L.simps*[*of L idx-pairs ! i num-vars*]
    by (*metis* (*no-types, lifting*) *add.commute add-diff-inverse-nat diff-diff-left gr-implies-not0 i-bound length-append less-one nat-1-add-1*)
  have *i-bound*: *i* < *length idx-pairs* **using** *assms* **by** *force*

  { **assume** *short-L*: *length L* = *0*
    then have *?thesis* **using** *assms*
      using *j-bound* **by** *linarith*

10

} moreover {
    assume *long-L: length L $\geq$ 1*
    then have *length L − 1 $\geq$ 0* **by** *blast*
    then have *(length L − 1) $\hat{\ }$ 3 + (length L − 1) $\hat{\ }$ 2 < length L $\hat{\ }$ 3*
      using *WEST-simp-termination1-bound[of length L−1]*
      by *(metis long-L ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add)*
    then have *(length L − 1) $\hat{\ }$ 3 + length (enumerate-pairs [0..<length x]) <*
length L $\hat{\ }$ 3
      using *length-enumerate-pairs[of [0..<length x]] length-x* **by** *auto*
    then have *length x $\hat{\ }$ 3 + length (enumerate-pairs [0..<length x])*
      < length L $\hat{\ }$ 3 + length idx-pairs − i
      using *i-bound length-x* **by** *simp*
    then have *?thesis* **by** *simp*
}
  ultimately show *?thesis* **by** *linarith*
qed

function *WEST-simp-helper*:: *WEST-regex $\Rightarrow$ (nat $\times$ nat) list $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$*
WEST-regex
  where *WEST-simp-helper L idx-pairs i num-vars =*
  (if (idx-pairs $\neq$ enum-pairs L $\vee$ i $\geq$ 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))
  apply *fast* **by** *blast*
termination
  apply *(relation measure (λ(L , idx-list, i, num-vars). (length L$\hat{\ }$3 + length idx-list*
− i)))
    apply *simp* **using** *WEST-simp-termination1* **apply** *blast* **by** *auto*

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

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

fun *WEST-or-simp*:: *WEST-regex $\Rightarrow$ WEST-regex $\Rightarrow$ nat $\Rightarrow$ WEST-regex*

11

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*) ⟹

$(\bigwedge \varphi\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Not\text{-}mltl\ \varphi),\ num\text{-}vars) \implies P) \implies P$

proof −
  fix *x*::*nat mltl* × *nat*
  fix *P*:: *bool*
  assume *t*: $(\bigwedge num\text{-}vars::nat.\ x = (True\text{-}mltl,\ num\text{-}vars) \implies P)$
  assume *fa*: $(\bigwedge num\text{-}vars::nat.\ x = (False\text{-}mltl,\ num\text{-}vars) \implies P)$
  assume *p*: $(\bigwedge p\ num\text{-}vars::nat.\ x = (Prop\text{-}mltl\ p,\ num\text{-}vars) \implies P)$
  assume *n1*: $(\bigwedge p\ num\text{-}vars::nat.\ x = (Not\text{-}mltl\ (Prop\text{-}mltl\ p),\ num\text{-}vars) \implies P)$

  assume *o*: $(\bigwedge \varphi\ \psi\ num\text{-}vars.\ x = (Or\text{-}mltl\ \varphi\ \psi,\ num\text{-}vars) \implies P)$
  assume *a*: $(\bigwedge \varphi\ \psi\ num\text{-}vars.\ x = (And\text{-}mltl\ \varphi\ \psi,\ num\text{-}vars) \implies P)$
  assume *f*: $(\bigwedge \varphi\ a\ b\ num\text{-}vars.\ x = (Future\text{-}mltl\ \varphi\ a\ b,\ num\text{-}vars) \implies P)$
  assume *g*: $(\bigwedge \varphi\ a\ b\ num\text{-}vars.\ x = (Global\text{-}mltl\ \varphi\ a\ b,\ num\text{-}vars) \implies P)$
  assume *u*: $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars.\ x = (Until\text{-}mltl\ \varphi\ \psi\ a\ b,\ num\text{-}vars) \implies P)$
  assume *r*: $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars.\ x = (Release\text{-}mltl\ \varphi\ \psi\ a\ b,\ num\text{-}vars) \implies P)$
  assume *n2*: $(\bigwedge num\text{-}vars.\ x = (Not\text{-}mltl\ True\text{-}mltl,\ num\text{-}vars) \implies P)$
  assume *n3*: $(\bigwedge num\text{-}vars.\ x = (Not\text{-}mltl\ False\text{-}mltl,\ num\text{-}vars) \implies P)$
  assume *n4*: $(\bigwedge \varphi\ \psi\ num\text{-}vars.\ x = (Not\text{-}mltl\ (And\text{-}mltl\ \varphi\ \psi),\ num\text{-}vars) \implies P)$

  assume *n5*: $(\bigwedge \varphi\ \psi\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Or\text{-}mltl\ \varphi\ \psi),\ num\text{-}vars) \implies P)$
  assume *n6*: $(\bigwedge \varphi\ a\ b\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Future\text{-}mltl\ \varphi\ a\ b),\ num\text{-}vars)$
  $\implies P)$
  assume *n7*: $(\bigwedge \varphi\ a\ b\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Global\text{-}mltl\ \varphi\ a\ b),\ num\text{-}vars)$
  $\implies P)$
  assume *n8*: $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Until\text{-}mltl\ \varphi\ \psi\ a\ b),\ num\text{-}vars)$
  $\implies P)$
  assume *n9*: $(\bigwedge \varphi\ \psi\ a\ b\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Release\text{-}mltl\ \varphi\ \psi\ a\ b),\ num\text{-}vars)$
  $\implies P)$
  assume *n10*: $(\bigwedge \varphi\ num\text{-}vars.\ x = (Not\text{-}mltl\ (Not\text{-}mltl\ \varphi),\ num\text{-}vars) \implies P)$
  show *P* **proof** (*cases fst x*)
    case *True-mltl*
    then show *?thesis* **using** *t*
      by (*metis eq-fst-iff*)
  next
    case *False-mltl*
    then show *?thesis* **using** *fa eq-fst-iff* **by** *metis*
  next
    case (*Prop-mltl p*)
    then show *?thesis* **using** *p*
      by (*metis prod.collapse*)
  next
    case (*Not-mltl* φ)
    then have *fst-x*: *fst x = Not-mltl* φ
      by *auto*
    then show *?thesis* **proof** (*cases* φ)
      case *True-mltl*
      then show *?thesis* **using** *n2*
        by (*metis Not-mltl split-pairs*)
    next

14

case *False-mltl*
then show *?thesis* **using** *n3*
  by (*metis Not-mltl prod.collapse*)
next
  case (*Prop-mltl p*)
  then show *?thesis* **using** *n1*
    by (*metis Not-mltl split-pairs*)
next
  case (*Not-mltl φ1*)
  then show *?thesis* **using** *n10 fst-x*
    by (*metis prod.collapse*)
next
  case (*And-mltl φ1 φ2*)
  then show *?thesis*
    by (*metis Not-mltl n4 prod.collapse*)
next
  case (*Or-mltl φ1 φ2*)
  then show *?thesis* **using** *n5 Not-mltl*
    by (*metis prod.collapse*)
next
  case (*Future-mltl a b φ1*)
  then show *?thesis* **using** *n6 Not-mltl*
    by (*metis prod.collapse*)
next
  case (*Global-mltl a b φ1*)
  then show *?thesis* **using** *n7 Not-mltl*
    by (*metis prod.collapse*)
next
  case (*Until-mltl φ1 a b φ2*)
  then show *?thesis* **using** *n8 Not-mltl*
    by (*metis prod.collapse*)
next
  case (*Release-mltl φ1 a b φ2*)
  then show *?thesis* **using** *n9 Not-mltl*
    by (*metis prod.collapse*)
qed
next
  case (*And-mltl φ1 φ2*)
  then show *?thesis* **using** *a*
    by (*metis prod.collapse*)
next
  case (*Or-mltl φ1 φ2*)
  then show *?thesis* **using** *o*
    by (*metis prod.collapse*)
next
  case (*Future-mltl a b φ1*)
  then show *?thesis* **using** *f*
    by (*metis split-pairs*)
next

case (*Global-mltl a b φ1*)
then show *?thesis* **using** *g*
by (*metis prod.collapse*)
next
case (*Until-mltl φ1 a b φ2*)
then show *?thesis* **using** *u*
by (*metis split-pairs*)
next
case (*Release-mltl φ1 a b φ2*)
then show *?thesis* **using** *r*
by (*metis split-pairs*)
qed
qed

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)
  apply (*induction* φ) **by** *simp-all*

function *WEST-reg-aux*:: (*nat*) *mltl* $\Rightarrow$ *nat* $\Rightarrow$ *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*
    using *exhaustive convert-nnf.cases* **using** *exhaustive* **apply** (*smt* (*z3*))
  defer apply *blast* **apply** *simp-all* .
termination
  apply (*relation measure* (λ(*F,num-vars*). (*WEST-termination-measure F*)))
  using *WEST-termination-measure-not* **by** *simp-all*

fun *WEST-num-vars*:: (*nat*) *mltl* $\Rightarrow$ *nat*

17

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

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

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

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

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 −
  have *get-state-empty*: ($WEST\text{-}get\text{-}state$ $[]$ *time num-vars*) = ($map\ (\lambda\ k.\ S)\ [0\ ..<$
num-vars]) **for** *time*
    by *auto*
  have *match-arbitrary-state*: ($match\text{-}timestep\ state\ (map\ (\lambda\ k.\ S)\ [0\ ..< num\text{-}vars]))$
$=$ *True* **if** *state-of-vars*:($\forall\, p \in state.\ p < num\text{-}vars$) **for** *state*
    using *state-of-vars*

  unfolding *match-timestep-def*
  by *simp*
 have *eliminate-forall*: *match-timestep* (π ! *time*) (*WEST-get-state* [] *time num-vars*)
if *time-bounded*:*time* < *length* π **for** *time*
  using *time-bounded* π-*of-num-vars get-state-empty*[*of time*] *match-arbitrary-state*[*of*
π ! *time*] **unfolding** *match-regex-def trace-of-vars-def*
  by (*metis* (*mono-tags*, *lifting*))
 then show *?thesis*
  unfolding *match-regex-def trace-of-vars-def*
  by *auto*
qed

lemma *WEST-and-state-difflengths-is-none*:
 assumes *length s1* $\neq$ *length s2*
 shows *WEST-and-state s1 s2* = *None*
 using *assms*
 proof (*induction s1 arbitrary*: *s2*)
  case *Nil*
  then show *?case*
   apply (*induction s2*) **by** *simp-all*
 next
  case (*Cons a s1*)
  then show *?case*
  proof (*induction s2*)
   case *Nil*
   then show *?case* **by** *auto*
  next
   case (*Cons b s2*)
   have *length s1* $\neq$ *length s2* **using** *Cons.prems*(*2*)
    by *auto*
  then have *and-s1-s2-none*: *WEST-and-state s1 s2* = *None* **using** *Cons.prems*(*1*)[*of*
s2]
    by *simp*
   {assume *ab-none*: *WEST-and-bitwise a b* = *None*
    then have *?case*
     by *simp*
   }
   moreover {assume *ab-not-none*: *WEST-and-bitwise a b* $\neq$ *None*
    then have *?case* **using** *and-s1-s2-none* **using** *WEST-and-state.simps*(*2*)[*of*
a s1 b s2]
     by *auto*
   }
   ultimately show *?case*
    by *blast*
  qed
 qed

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*
  apply (*cases b2*)
    apply (*cases b1*) **apply** *simp-all*
    apply(*cases b1*) **apply** *simp-all*
  apply (*cases b1*) **by** *simp-all*

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−
  show *?thesis* **using** *same-len*
  proof (*induct length s1 arbitrary*: *s1 s2*)
    case *0*
    then show *?case* **using** *WEST-and-state.simps* **by** *simp*
  next
    case (*Suc x*)
    obtain *h1 T1* **where** *s1 = h1 # T1*
      using *Suc.hyps(2)*
      by (*metis length-Suc-conv*)
    obtain *h2 T2* **where** *s2 = h2 # T2*
      using *Suc.prems(1) Suc.hyps(2)*
      by (*metis length-Suc-conv*)
    then show *?case* **using** *WEST-and-bitwise-commutative*[*of h1 h2*] *WEST-and-state.simps(2)*[*of
h1 T1 h2 T2*]
      WEST-and-state.simps(2)[*of h2 T2 h1 T1*]
    by (*metis (no-types, lifting) Suc.hyps(1) Suc.hyps(2) Suc.prems(1) Suc-length-conv
WEST-and-bitwise-commutative ‹s1 = h1 # T1› list.inject option.simps(4) op-
tion.simps(5) remove-option-type-bit.cases*)
  qed
qed

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 reg-*

trace1)

proof −

  have *WEST-and-bitwise-commutative*: *WEST-and-bitwise b1 b2 = WEST-and-bitwise b2 b1* **for** *b1 b2*

    apply (*cases b2*)

     apply (*cases b1*) **apply** *simp-all*

     apply(*cases b1*) **apply** *simp-all*

    apply (*cases b1*) **by** *simp-all*

  then have *WEST-and-state-commutative*: *WEST-and-state s1 s2 = WEST-and-state s2 s1* **if** *same-len*: (*length s1*) = (*length s2*) **for** *s1 s2*

  using *same-len*

  proof (*induct length s1 arbitrary*: *s1 s2*)

    case *0*

    then show *?case* **using** *WEST-and-state.simps* **by** *simp*

  next

    case (*Suc x*)

    obtain *h1 T1* **where** *s1 = h1#T1*

     using *Suc.hyps*(*2*)

     by (*metis length-Suc-conv*)

    obtain *h2 T2* **where** *s2 = h2#T2*

     using *Suc.prems*(*2*) *Suc.hyps*(*2*)

     by (*metis length-Suc-conv*)

    then show *?case* **using** *WEST-and-bitwise-commutative*[*of h1 h2*] *WEST-and-state.simps*(*2*)[*of h1 T1 h2 T2*]

     WEST-and-state.simps(*2*)[*of h2 T2 h1 T1*]

    by (*metis* (*no-types, lifting*) *Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc.prems*(*2*) *Suc-length-conv* *WEST-and-bitwise-commutative* ‹*s1 = h1 # T1*› *list.inject option.simps*(*4*) *option.simps*(*5*) *remove-option-type-bit.cases*)

  qed

  show *?thesis* **using** *regtrace1-of-num-vars regtrace2-of-num-vars*

  proof (*induction regtrace1 arbitrary*: *regtrace2*)

    case *Nil*

    then show *?case* **using** *WEST-and-trace.simps*(*1−2*)

     by (*metis neq-Nil-conv*)

  next

    case (*Cons h1 T1*)

    {assume ∗: *regtrace2* = []

     then have *?case* **using** *WEST-and-trace.simps*

      by *simp*

    } moreover {assume ∗: *regtrace2* ≠ []

     then obtain *h2 T2* **where** *h2T2*: *regtrace2 = h2#T2*

      by (*meson list.exhaust*)

     have *comm-1*: *WEST-and-trace T1 T2 = WEST-and-trace T2 T1*

      using *Cons h2T2*

      unfolding *trace-regex-of-vars-def*

      by (*metis Suc-less-eq length-Cons nth-Cons-Suc*)

     have *comm-2*: *WEST-and-state h1 h2 = WEST-and-state h2 h1*

      using *WEST-and-state-commutative*[*of h1 h2*] *h2T2*

      Cons(*2−3*) **unfolding** *trace-regex-of-vars-def*

       by (*metis WEST-and-state-difflengths-is-none*)
     have *?case* **using** *WEST-and-trace.simps(3)[of h1 T1 h2 T2]*
      h2T2 WEST-and-trace.simps(3)[of h2 T2 h1 T1] comm-1 comm-2
      by *presburger*
   }
   ultimately show *?case* **by** *blast*
  qed
qed

lemma *WEST-and-helper-subset*:
  shows *set* (*WEST-and-helper h L*) ⊆ *set* (*WEST-and-helper h* (*a # L*))
proof −
  {assume ∗: *WEST-and-trace h a = None*
   then have *WEST-and-helper h L = WEST-and-helper h* (*a # L*)
    using *WEST-and-helper.simps(2)[of h a L]* **by** *auto*
   then have *?thesis* **by** *simp*
  }
  moreover {assume ∗: *WEST-and-trace h a ≠ None*
   then obtain *res* **where** *WEST-and-trace h a = Some res*
    by *auto*
   then have *WEST-and-helper h* (*a#L*) = *res # WEST-and-helper h L*
    using *WEST-and-helper.simps(2)[of h a L]* **by** *auto*
   then have *?thesis* **by** *auto*
  }
  ultimately show *?thesis* **by** *blast*
qed

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 −
  show *?thesis* **using** *assumption*
  proof (*induct L*)
   case *Nil*
   then show *?case* **using** *WEST-and-helper.simps(1)*
    by *simp*
  next
   case (*Cons a L*)
   then obtain *loc sometrace* **where** *obt*: *loc < length* (*a#L*) ∧ *WEST-and-trace*
h ((*a#L*) *! loc*) = *Some sometrace* ∧ *regtrace = sometrace*
    by *blast*
   {assume ∗: *loc = 0*
    then have *WEST-and-trace h a = Some sometrace* ∧ *regtrace = sometrace*
     using *obt*
     by *simp*

then have *?case* **using** *WEST-and-helper.simps(2)[of h a L]*
      by *auto*
   } moreover {assume *∗: loc > 0*
   then have *loc: loc−1 < length L ∧*
      WEST-and-trace h (L ! (loc−1)) = Some sometrace ∧ regtrace = sometrace
      using *obt* **by** *auto*
   have *set (WEST-and-helper h L) ⊆ set (WEST-and-helper h (a # L))*
      using *WEST-and-helper-subset* **by** *blast*
   then have *?case* **using** *Cons(1) loc* **by** *blast*
   }
   ultimately show *?case* **by** *auto*
   qed
qed

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))*
using *assms* **proof** *(induction L)*
  case *Nil*
  then show *?case* **by** *simp*
next
  case *(Cons a L)*
  {assume *∗: WEST-and-trace h a = None*
   then have *?case* **using** *WEST-and-helper.simps(2)[of h a L] Cons*
      by *force*
  } moreover {assume *∗: WEST-and-trace h a ≠ None*
   then obtain *res* **where** *res: WEST-and-trace h a = Some res*
      by *auto*
   then have *WEST-and-helper h (a#L) = res # WEST-and-helper h L*
      using *WEST-and-helper.simps(2)[of h a L]* **by** *auto*
   then have *eo: regtrace = res ∨ regtrace ∈ set (WEST-and-helper h L)*
      using *Cons(2)*
      by *auto*
   {assume *∗: regtrace = res*
      then have *?case* **using** *res* **by** *auto*
   } moreover {assume *∗: regtrace ∈ set (WEST-and-helper h L)*
      then obtain *loc* **where** *loc-prop: loc<length L ∧*
         *(∃ sometrace. WEST-and-trace h (L ! loc) = Some sometrace ∧ regtrace =
sometrace)*
         using *Cons.IH* **by** *blast*
      then have *loc+1<length (a#L) ∧*
         *(∃ sometrace. WEST-and-trace h ((a#L) ! (loc+1)) = Some sometrace ∧
regtrace = sometrace)*
         by *auto*
      then have *?case* **by** *blast*
   }

ultimately have *?case* **using** *eo*
  by *blast*
}
ultimately show *?case* **by** *blast*
qed

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*))
 using *WEST-and-helper-set-member-forward WEST-and-helper-set-member-converse*
  by *blast*

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*))
  using *assms* **proof** (*induct L1 arbitrary*: *L2*)
  case *Nil*
  then show *?case* **using** *WEST-and.simps*(*2*) *WEST-and.simps*(*1*)
    by (*metis list.distinct*(*1*) *list.exhaust list.set-cases*)
next
  case (*Cons head tail*)
  {assume *L2-empty*: *L2* = []
   then have *?case*
    using *Cons.prems*(*3*) **by** *auto*
  }
  moreover { assume *L2-not-empty*: *L2* ≠ []
   {assume *regtrace-in-head-L2*: *regtrace* ∈ *set* (*WEST-and-helper head L2*)
    then obtain *loc2* **where** (*loc2*<*length L2* ∧
    (∃ *sometrace*. *WEST-and-trace head* (*L2* ! *loc2*) = *Some sometrace* ∧ *regtrace*
= *sometrace*))
      using *WEST-and-helper-set-member*[*of regtrace head L2*]
      by *blast*
    then have *0* < *length* (*head* # *tail*) ∧
    loc2 < *length L2* ∧
    (∃ *sometrace*.
      WEST-and-trace ((*head* # *tail*) ! *0*) (*L2* ! *loc2*) = *Some sometrace* ∧
      regtrace = *sometrace*)
      using *regtrace-in-head-L2*

        by *simp*
      then have *?case*
        by *blast*
    }
    moreover {assume *regtrace-notin-head-L2*: *regtrace* $\notin$ *set* (*WEST-and-helper head L2*)
      obtain *h2 T2* **where** *h2T2*:*L2 = h2*#*T2* **using** *L2-not-empty*
       by (*meson list.exhaust*)
      {assume $*$: *WEST-and-helper head* (*h2 # T2*) = []
       then have *WEST-and* (*head # tail*) *L2 = WEST-and tail L2*
        using *WEST-and.simps*(*3*)[*of head tail h2 T2*] *h2T2* **by** *simp*
      }
      moreover {assume $*$: *WEST-and-helper head* (*h2 # T2*) $\neq$ []
       then have *WEST-and* (*head # tail*) *L2 =* (*WEST-and-helper head L2*) @
WEST-and tail L2
        using *WEST-and.simps*(*3*)[*of head tail h2 T2*] *h2T2*
        by (*simp add*: *list.case-eq-if*)
      }
      ultimately have *e-o*: *WEST-and* (*head # tail*) *L2 = WEST-and tail L2* $\lor$
WEST-and (*head # tail*) *L2 =* (*WEST-and-helper head L2*) @ *WEST-and tail L2*
       by *blast*
      have *regtrace-in*: *regtrace* $\in$ *set* (*WEST-and tail L2*) **using** *L2-not-empty*
regtrace-notin-head-L2 Cons.prems(*3*) *h2T2 e-o*
       by *fastforce*
     have $\forall$ *k*<*length* (*head # tail*). *trace-regex-of-vars* ((*head # tail*) ! *k*) *num-vars*
       using *Cons.prems*(*1*) **unfolding** *WEST-regex-of-vars-def* **by** *argo*
      then have *regtracelist-tail*: *WEST-regex-of-vars tail num-vars*
       unfolding *WEST-regex-of-vars-def* **by** *auto*
      obtain *loc1 loc2* **where** *loc1* < *length tail* $\land$
      loc2 < *length L2* $\land$ ($\exists$ *sometrace*. *WEST-and-trace* (*tail* ! *loc1*) (*L2* ! *loc2*)
= *Some sometrace* $\land$ *regtrace* = *sometrace*)
       using *Cons.hyps*[*OF regtracelist-tail Cons.prems*(*2*) *regtrace-in*] **by** *blast*
      then have *loc1+1* < *length* (*head # tail*) $\land$
      loc2 < *length L2* $\land$
      ($\exists$ *sometrace*.
       WEST-and-trace ((*head # tail*) ! (*loc1+1*)) (*L2* ! *loc2*) = *Some sometrace*
$\land$
        regtrace = sometrace)
       by *simp*
      then have *?case*
       by *blast*
    }
    ultimately have *?case*
     by *blast*
  }
  ultimately show *?case*
    by *blast*
qed

lemma *WEST-and-subset*:
  shows *set* (*WEST-and T1 L2*) ⊆ *set* (*WEST-and* (*h1*#*T1*) *L2*)
proof −
  {assume ∗: *L2* = []
    then have *?thesis* **by** *auto*
  } moreover {assume ∗: *L2* ≠ []
    then obtain *h2 T2* **where** *L2* = *h2*#*T2*
      using *list.exhaust-sel* **by** *blast*
    then have *?thesis*
      using *WEST-and.simps*(*3*)[*of h1 T1 h2 T2*]
      by (*simp add*: *list.case-eq-if*)
  }
  ultimately show *?thesis* **by** *blast*
qed

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*) **using** *assms*
proof (*induct L1 arbitrary*: *L2*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons h1 T1*)
  then obtain *loc1 loc2* **where** *loc1loc2*: *loc1* < *length* (*h1* # *T1*) ∧
      loc2 < *length L2* ∧
      (∃*sometrace*.
        WEST-and-trace ((*h1* # *T1*) ! *loc1*) (*L2* ! *loc2*) = *Some sometrace* ∧
        regtrace = *sometrace*) **by** *blast*
  {assume ∗: *L2* = []
    then have *?case* **using** *Cons* **by** *auto*
  } moreover {assume ∗: *L2* ≠ []
    then obtain *h2 T2* **where** *h2T2*: *L2* = *h2*#*T2*
      using *list.exhaust-sel* **by** *blast*
    have ∀ *k*<*length* (*h1* # *T1*). *trace-regex-of-vars* ((*h1* # *T1*) ! *k*) *num-vars*
      using *Cons.prems*(*1*) **unfolding** *WEST-regex-of-vars-def* **by** *argo*
    then have *regtraceList-T1*: *WEST-regex-of-vars T1 num-vars*
      unfolding *WEST-regex-of-vars-def* **by** *auto*
    {assume ∗∗: *WEST-and-helper h1 L2* = []
      then have *loc1* > *0*
        using *loc1loc2*
      by (*metis WEST-and-helper.simps*(*1*) *WEST-and-helper-set-member gr-implies-not-zero*
list.size(*3*) *not-gr0 nth-Cons-0*)

then have *exi*: ∃ *loc1 loc2*.
  loc1 < length T1 ∧
  loc2 < length L2 ∧
  (∃ *sometrace*.
      WEST-and-trace (*T1 ! loc1*) (*L2 ! loc2*) = *Some sometrace* ∧
      regtrace = *sometrace*)
    using *loc1loc2*
      by (*metis One-nat-def Suc-pred bot-nat-0.not-eq-extremum length-Cons*
nat-add-left-cancel-less nth-Cons′ plus-1-eq-Suc)
    then have *?case*
      using *Cons.hyps*[*OF regtraceList-T1 Cons*(*3*) *exi*] *WEST-and-subset*
      by *auto*
  } moreover {assume ∗∗: *WEST-and-helper h1 L2* ≠ []
    then have *WEST-and* (*h1 # T1*) (*h2 # T2*) = *WEST-and-helper h1* (*h2*
 # T2) @ *WEST-and T1* (*h2 # T2*)
      by (*simp add: list.case-eq-if*)
    then have *?case*
      using *Cons.hyps*[*OF regtraceList-T1 Cons.prems*(*2*)]
    by (*metis One-nat-def Suc-pred Un-iff WEST-and-helper-set-member-converse*
gr-implies-not-zero h2T2 length-Cons linorder-neqE-nat loc1loc2 nat-add-left-cancel-less
nth-Cons′ plus-1-eq-Suc set-append)
  }
  ultimately have *?case*
    by *auto*
 }
 ultimately show *?case*
   by *auto*
qed

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*))
   using *WEST-and-set-member-dir1 WEST-and-set-member-dir2 assms* **by** *blast*

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 −
  obtain *loc1 loc2* **where** *loc1loc2*: *loc1* < *length L1* ∧
      loc2 < *length L2* ∧
      (∃ *sometrace*.
          WEST-and-trace (*L1* ! *loc1*) (*L2* ! *loc2*) = *Some sometrace* ∧
          regtrace = *sometrace*)
  using *WEST-and-set-member*[*OF L1-of-num-vars L2-of-num-vars*] *regtrace-in*
  by *auto*
  have *loc1*: *trace-regex-of-vars* (*L1* ! *loc1*) *num-vars*
    using *L1-of-num-vars loc1loc2* **unfolding** *WEST-regex-of-vars-def*
    by (*meson less-imp-le-nat*)
  have *loc2*: *trace-regex-of-vars* (*L2* ! *loc2*) *num-vars*
    using *L2-of-num-vars loc1loc2* **unfolding** *WEST-regex-of-vars-def*
    by (*meson less-imp-le-nat*)
    have *loc1* < *length L1* ∧
      loc2 < *length L2* ∧
      (∃ *sometrace*.
          WEST-and-trace (*L2* ! *loc2*) (*L1* ! *loc1*) = *Some sometrace* ∧
          regtrace = *sometrace*)
    using *loc1loc2 WEST-and-trace-commutative*[*OF loc1 loc2*]
    by *simp*
  then show *?thesis* **using** *loc1loc2*
    using *WEST-and-set-member*[*OF L2-of-num-vars L1-of-num-vars*]
    by *blast*
qed

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*)
  using *WEST-and-commutative-sets-member*[*OF L1-of-num-vars L2-of-num-vars*]
    WEST-and-commutative-sets-member[*OF L2-of-num-vars L1-of-num-vars*]
  by *blast*

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 −
  have *set* (*WEST-and L1 L2*) = *set* (*WEST-and L2 L1*)
    using *WEST-and-commutative-sets assms*
    by *blast*
  then have *match* π (*WEST-and L1 L2*) = *match* π (*WEST-and L2 L1*) **for** π

29

    using *match-def match-regex-def*
    by (*metis in-set-conv-nth*)
  then show *?thesis*
    unfolding *regex-equiv-def* **by** *auto*
qed

3.3.2 Identity and Zero

lemma *WEST-and-helper-identity*:
  shows *WEST-and-helper [] trace = trace*
proof (*induct trace*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons h T*)
  then show *?case*
    using *WEST-and-helper.simps(2)[of [] h T]*
    by (*smt* (*verit*) *WEST-and-trace.elims list.discI option.simps(5)*)
qed

lemma *WEST-and-identity*: *WEST-and [[]] L = L*
proof−
  {assume *∗*: *L = []*
    then have *?thesis*
      by *auto*
  } moreover {assume *∗*: *L ≠ []*
    then obtain *h T* **where** *hT*: *L = h#T*
      using *list.exhaust* **by** *auto*
    then have *?thesis* **using** *WEST-and.simps(3)[of [] [] h T]*
      using *hT*
      by (*metis* (*no-types, lifting*) *WEST-and.simps(2) WEST-and-helper-identity*
append.right-neutral list.simps(5))
  }
  ultimately show *?thesis* **by** *linarith*
qed

lemma *WEST-and-zero*: *WEST-and L [] = []*
  by *simp*

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 *∀ x∈state. x < num-vars*
  shows *∀ x∈(advance-state state). x < (num-vars−1)*

using *assms advance-state.simps* **by** *auto*

lemma *advance-state-elt-member*:
  fixes *state*::*state*
  fixes *x*::*nat*
  assumes *x+1* ∈ *state*
  shows *x* ∈ *advance-state state*
  using *assms advance-state.simps* **by** *force*

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−
  have (∀ *x<length (h # t)*.
          ((*h # t*) ! *x = One* ⟶ *x* ∈ *state*) ∧ ((*h # t*) ! *x = Zero* ⟶ *x* ∉ *state*))
using *assms* **unfolding** *match-timestep-def* **by** *argo*
  then have ∀ *x<length t*.
          ((*h # t*) ! (*x+1*) = *One* ⟶ (*x+1*) ∈ *state*) ∧ ((*h # t*) ! (*x+1*) = *Zero*
⟶ (*x+1*) ∉ *state*) **by** *auto*
  then have ∀ *x<length t*.
          (*t* ! *x = One* ⟶ *x* ∈ (*advance-state state*)) ∧ (*t* ! *x = Zero* ⟶ *x* ∉
(*advance-state state*))
    using *advance-state.simps advance-state-elt-member* **by** *fastforce*
  then show *?thesis* **using** *assms* **unfolding** *match-timestep-def* **by** *metis*
qed

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−
  have *same-length*: *length s1 = length s2*
    using *assms* **unfolding** *state-regex-of-vars-def* **by** *simp*
  have (∀ *x*. *x < num-vars* ⟶ (((*s1* ! *x = One*) ⟶ *x* ∈ *state*) ∧ ((*s1* ! *x =*
Zero) ⟶ *x* ∉ *state*)))
    using *assms* **unfolding** *match-timestep-def state-regex-of-vars-def* **by** *metis*
  then have *match-timestep-s1-unfold*: ∀ *x*∈*state*. *x < num-vars* ⟶ ((*s1* ! *x =*
One) ∨ (*s1* ! *x = S*))
    using *assms* **by** (*meson WEST-bit.exhaust*)
  then have *x-in-state-s1*: ∀ *x*. (*x < num-vars* ∧ *x* ∈ *state*) ⟶ ((*s1* ! *x = One*)
∨ (*s1* ! *x = S*)) **by** *blast*

31

then have *x-notin-state-s1*: $\forall\,x.\ (x < num\text{-}vars \wedge x \notin state) \longrightarrow ((s1\ !\ x = Zero) \vee (s1\ !\ x = S))$

   using *match-timestep-s1-unfold*

   by (*meson WEST-bit.exhaust* ‹$\forall\,x{<}num\text{-}vars.\ (s1\ !\ x = One \longrightarrow x \in state) \wedge (s1\ !\ x = Zero \longrightarrow x \notin state)$›)

have *match-timestep-s2-unfold*: $(\forall\ x.\ x < num\text{-}vars \longrightarrow (((s2\ !\ x = One) \longrightarrow x \in state) \wedge ((s2\ !\ x = Zero) \longrightarrow x \notin state)))$

   using *assms* **unfolding** *match-timestep-def state-regex-of-vars-def* **by** *metis*

then have $\forall\,x{\in}state.\ x < num\text{-}vars \longrightarrow ((s2\ !\ x = One) \vee (s2\ !\ x = S))$

   using *assms* **by** (*meson WEST-bit.exhaust*)

then have *x-in-state-s2*: $\forall\,x.\ (x < num\text{-}vars \wedge x \in state) \longrightarrow ((s2\ !\ x = One) \vee (s2\ !\ x = S))$ **by** *blast*

   then have *x-notin-state-s2*: $\forall\,x.\ (x < num\text{-}vars \wedge x \notin state) \longrightarrow ((s2\ !\ x = Zero) \vee (s2\ !\ x = S))$

   using *match-timestep-s1-unfold*

   by (*meson WEST-bit.exhaust* ‹$\forall\,x{<}num\text{-}vars.\ (s2\ !\ x = One \longrightarrow x \in state) \wedge (s2\ !\ x = Zero \longrightarrow x \notin state)$›)

have *no-contradictory-bits1*: $\forall\,x{\in}state.\ x < num\text{-}vars \longrightarrow WEST\text{-}and\text{-}bitwise\ (s1\ !\ x)\ (s2\ !\ x) \neq None$

   using *x-in-state-s1 x-notin-state-s1 x-in-state-s2 x-notin-state-s2 WEST-and-bitwise.simps*

   by (*metis match-timestep-s2-unfold not-Some-eq*)

then have *no-contradictory-bits2*: $\forall\,x.\ (x \notin state \wedge x < num\text{-}vars) \longrightarrow WEST\text{-}and\text{-}bitwise\ (s1\ !\ x)\ (s2\ !\ x) \neq None$

   using *x-in-state-s1 x-notin-state-s1 x-in-state-s2 x-notin-state-s2 WEST-and-bitwise.simps*

   by *fastforce*

have *no-contradictory-bits*: $\forall\,x.\ x < num\text{-}vars \longrightarrow WEST\text{-}and\text{-}bitwise\ (s1\ !\ x)\ (s2\ !\ x) \neq None$

   using *no-contradictory-bits1 no-contradictory-bits2*

   by *blast*

show *?thesis* **using** *same-length no-contradictory-bits assms*

proof (*induct s1 arbitrary*: *s2 num-vars state*)

   case *Nil*

   then show *?case* **by** *auto*

next

   case (*Cons a s1*)

   then have *num-vars-bound*: $num\text{-}vars = (length\ s1) + 1$

    unfolding *state-regex-of-vars-def* **by** *simp*

   then have *len-s2*: $length\ s2 = num\text{-}vars$ **using** *Cons* **by** *simp*

   then have $length\ s2 \geq 1$ **using** *num-vars-bound* **by** *simp*

   then have *s2-ht-exists*: $\exists\,h\ t.\ s2 = h\#t$

    by (*metis Suc-eq-plus1 Suc-le-length-iff* ‹$length\ s2 = num\text{-}vars$› *not-less-eq-eq num-vars-bound*)

   obtain *h t* **where** *s2-ht*: $s2 = h\#t$ **using** *s2-ht-exists* **by** *blast*

   {**assume** $*$: $WEST\text{-}and\text{-}bitwise\ a\ h = None$

    then have *?case* **using** *WEST-and-state.simps(2)*

     using *Cons.prems(2)* ‹$length\ s2 = num\text{-}vars$› *s2-ht* **by** *force*

   } **moreover** {**assume** $**$: $WEST\text{-}and\text{-}bitwise\ a\ h \neq None$

    have *h1*: $length\ s1 = length\ t$

     using *len-s2 num-vars-bound s2-ht* **by** *simp*

obtain *num-var-minus1* **where** *nvm1-def*: *num-var-minus1 = num-vars −*
1 **by** *simp*
    then have $\forall\, x<(num\text{-}vars-1)$. *WEST-and-bitwise* $((a\#s1)$! $(x+1))$ $((h\#t)$
! $(x+1)) \neq$ *None*
      using *Cons.prems(2)*
      using *num-vars-bound s2-ht* **by** *auto*
     then have *h2*: $\forall\, x<num\text{-}var\text{-}minus1$. *WEST-and-bitwise* (*s1* ! *x*) (*t* ! *x*) $\neq$
None
      using *nvm1-def* **by** *simp*
     obtain *adv-state* **where** *adv-state-def*: *adv-state = advance-state state* **by**
simp
   have *h4*: *state-regex-of-vars s1 num-var-minus1*
    using *Cons.prems* **unfolding** *state-regex-of-vars-def*
    by (*simp add*: *add-implies-diff num-vars-bound nvm1-def*)
   have *h5*: *state-regex-of-vars t num-var-minus1*
    using *h4 h1* **unfolding** *state-regex-of-vars-def* **by** *simp*
   have *h6*: *match-timestep adv-state s1* $\wedge$ *match-timestep adv-state t*
    using *Cons.prems(5) s2-ht adv-state-def*
    using *advance-state-match-timestep* **by** *blast*
   have *ih*: *WEST-and-state s1 t* $\neq$ *None*
    using *Cons.hyps[of t num-var-minus1 adv-state] h1 h2 h4 h5 h6* **by** *auto*
   have *?case* **using** *WEST-and-state.simps(2)[of a s1 h t] ∗∗ ih s2-ht* **by** *auto*
  }
  ultimately show *?case*
   by *blast*
 qed
qed

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*) $\neq$ *None*
 shows $\exists\, r$. *length r = length s1* $\wedge$ *WEST-and-state s1 s2 = Some r*
proof−
 have *s1s2-exists*: $\exists\, r$. *WEST-and-state s1 s2 = Some r*
  using *assms* **by** *simp*
 then obtain *r* **where** *s1s2-obt*: *WEST-and-state s1 s2 = Some r* **by** *auto*
 let *?n = length s1*
 have *s1s2-length-n*: *length r = ?n*
  using *assms s1s2-obt*
 proof (*induct ?n arbitrary*: *s1 s2 r*)
  case *0*
  then show *?case* **using** *WEST-and-state.simps(1)* **by** *simp*
 next
  case (*Suc x*)
  have *length s1* $\geq$ *1* **using** *Suc.hyps(2)* **by** *simp*
  then have $\exists\, h1\ t1$. *s1 = h1 # t1* **by** (*simp add*: *Suc-le-length-iff*)
  then obtain *h1 t1* **where** *h1t1*: *s1 = h1 # t1* **by** *blast*
  have *length s2* $\geq$ *1* **using** *Suc.hyps(2) Suc.prems(1)* **by** *auto*

<div align="center">33</div>

    then have ∃ *h2 t2. s2 = h2 # t2* **by** (*simp add*: *Suc-le-length-iff*)
    then obtain *h2 t2* **where** *h2t2*: *s2 = h2 # t2* **by** *blast*
    have *WEST-and-bitwise h1 h2 ≠ None*
      using *Suc.prems h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      by (*metis option.simps(4)*)
    then obtain *h1h2* **where** *h1h2-and*: *Some h1h2 = WEST-and-bitwise h1 h2*
by *auto*
    have *WEST-and-state t1 t2 ≠ None*
      using *Suc.prems h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      by (*metis* (*no-types, lifting*) *not-None-eq option.simps(4) option.simps(5)*)
     then obtain *t1t2* **where** *t1t2-and*: *Some t1t2 = WEST-and-state t1 t2* **by**
auto
    have *cond1*: *x = length t1* **using** *h1t1 Suc.hyps(2)* **by** *auto*
    have *cond2*: *length t1 = length t2* **using** *h1t1 h2t2 Suc.prems(1)* **by** *auto*
    have *len-t1t2*: *length t1t2 = length t1*
      using *Suc.hyps(1)*[*of t1 t2 t1t2*] **using** *cond1 cond2 t1t2-and*
      using ‹*WEST-and-state t1 t2 ≠ None*› **by** *fastforce*
    have *r-decomp*: *r = h1h2 # t1t2*
      using *Suc.prems(3) h1h2-and t1t2-and WEST-and-state.simps(2)*
      by (*metis h1t1 h2t2 option.inject option.simps(5)*)
    show *?case* **using** *r-decomp len-t1t2 h1t1 h2t2* **by** *auto*
  qed
  then show *?thesis* **using** *assms s1s2-obt s1s2-exists* **by** *simp*
qed

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 −
  have (*a = One ⟶ 0 ∈ state*) **using** *assms* **by** *auto*
  then have *a-one*: (*a#t*)!*0 = One ⟶ 0 ∈ state* **by** *simp*
  have *t-one*: ∀ *x<length t.* (*t!x*) = *One ⟶ x + 1 ∈ state* **using** *assms* **by** *auto*
  have ∀ *x<*(*length t*)+*1.* (*x ≠ 0 ∧* (*a#t*)!*x = One*) *⟶ x ∈ state*
    using *t-one assms(2)*
  by (*metis* (*no-types, lifting*) *Suc-diff-1 Suc-less-eq add-Suc-right cancel-comm-monoid-add-class.diff-cancel*
gr-zeroI less-numeral-extra(1) linordered-semidom-class.add-diff-inverse nth-Cons′
verit-comp-simplify1(1))
  then have *at-one*: ∀ *x<length* (*a#t*). ((*a#t*) ! *x = One ⟶ x ∈ state*)
    using *a-one t-one* **by** (*simp add*: *nth-Cons′*)
  have (*a = Zero ⟶ 0 ∉ state*) **using** *assms* **by** *auto*
  then have *a-zero*: (*a#t*)!*0 = Zero ⟶ 0 ∉ state* **by** *simp*
  have *t-zero*: ∀ *x<length t.* (*t!x*) = *Zero ⟶ x + 1 ∉ state* **using** *assms* **by** *auto*

have $\forall\, x{<}(length\ t){+}1.\ (x \neq 0 \land (a\#t)!x = Zero) \longrightarrow x \notin state$
  using *t-zero assms(2)*
  by (*metis Nat.add-0-right Suc-diff-1 Suc-less-eq add-Suc-right cancel-comm-monoid-add-class.diff-cancel
less-one not-gr-zero nth-Cons$'$*)
  then have *at-zero*: $\forall\, x{<}length\ (a\#t).\ ((a\#t)\ !\ x = Zero \longrightarrow x \notin state)$
    using *a-zero t-zero* **by** (*simp add: nth-Cons$'$*)
  show *?thesis* **using** *at-one at-zero* **by** *blast*
qed

lemma *index-shift-reverse*:
  fixes *a*::*WEST-bit*
  fixes *t*::*state-regex*
  fixes *state*::*state*
  assumes $\forall\, x{<}length\ (a\#t).\ ((a\#t)\ !\ x = One \longrightarrow x \in state) \land ((a\#t)\ !\ x = Zero \longrightarrow x \notin state)$
  shows $\forall\, x{<}length\ t.\ ((t!x) = One \longrightarrow x + 1 \in state) \land ((t!x) = Zero \longrightarrow x + 1 \notin state)$
proof$-$
  have $length\ (a\#t) = (length\ t) + 1$ **by** *simp*
  then have $\forall\, x{<}(length\ t){+}1.\ ((a\#t)\ !\ x = One \longrightarrow x \in state) \land ((a\#t)\ !\ x = Zero \longrightarrow x \notin state)$
    using *assms* **by** *metis*
  then show *?thesis* **by** *simp*
qed

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* $\land$ *match-timestep state s2*
  shows $\exists\, somestate.\ (match\text{-}timestep\ state\ somestate) \land (WEST\text{-}and\text{-}state\ s1\ s2) = Some\ somestate$
proof$-$
  have $WEST\text{-}and\text{-}state\ s1\ s2 \neq None$
    using *WEST-and-state-well-defined assms* **by** *simp*
  then have $\exists\, somestate.\ WEST\text{-}and\text{-}state\ s1\ s2 = Some\ somestate$ **by** *auto*
  then obtain *somestate* **where** *somestate-obt*: $WEST\text{-}and\text{-}state\ s1\ s2 = Some\ somestate$ **by** *auto*
  have *samelength*: $length\ s1 = length\ s2$ **using** *assms(1, 2)* **unfolding** *state-regex-of-vars-def*
by *auto*
  have *len-s1*: $length\ s1 = num\text{-}vars$ **using** *assms* **unfolding** *state-regex-of-vars-def*
by *auto*
  have *len-s2*: $length\ s2 = num\text{-}vars$ **using** *samelength len-s1* **by** *auto*
  have *len-somestate*: $length\ somestate = num\text{-}vars$
    using *samelength somestate-obt WEST-and-state.simps WEST-and-state-length*

    using *len-s1 len-s2*
    by *fastforce*
  have *s1-bits*: $\forall x{<}num\text{-}vars.\ (s1\ !\ x = One \longrightarrow x \in state) \wedge (s1\ !\ x = Zero \longrightarrow$
$x \notin state)$
    using *assms(3) len-s1* **unfolding** *match-timestep-def* **by** *metis*
  have *s2-bits*: $\forall x{<}num\text{-}vars.\ (s2\ !\ x = One \longrightarrow x \in state) \wedge (s2\ !\ x = Zero \longrightarrow$
$x \notin state)$
    using *assms(3) len-s2* **unfolding** *match-timestep-def len-s2* **by** *metis*
  have *somestate-bits*: $\forall x{<}num\text{-}vars.\ (somestate\ !\ x = One \longrightarrow x \in state)$
$$\wedge (somestate\ !\ x = Zero \longrightarrow x \notin state)$$
    using *s1-bits s2-bits somestate-obt len-s1 len-s2 len-somestate assms(1)*
  proof(*induct somestate arbitrary: s1 s2 num-vars state*)
    case *Nil*
    then show *?case*
      by (*metis less-nat-zero-code list.size(3)*)
  next
    case (*Cons a t*)
    have *length s1* $\geq$ *1* **using** *Cons.prems(4, 5, 6)* **by** *auto*
    then have $\exists\,h1\ t1.\ s1 = h1\ \#\ t1$ **by** (*simp add: Suc-le-length-iff*)
    then obtain *h1 t1* **where** *h1t1*: $s1 = h1\ \#\ t1$ **by** *auto*
    have *length s2* $\geq$ *1* **using** *Cons.prems(4, 5, 6)* **by** *auto*
    then have $\exists\,h2\ t2.\ s2 = h2\ \#\ t2$ **by** (*simp add: Suc-le-length-iff*)
    then obtain *h2 t2* **where** *h2t2*: $s2 = h2\ \#\ t2$ **by** *auto*
    have *h1h2-not-none*: *WEST-and-bitwise h1 h2* $\neq$ *None*
      using *Cons.prems(3) h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      by (*metis option.discI option.simps(4)*)
    then have *t1t2-not-none*: *WEST-and-state t1 t2* $\neq$ *None*
      using *Cons.prems(3) h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      by (*metis option.case-eq-if option.distinct(1)*)
    have *h1h2-is-a*: *WEST-and-bitwise h1 h2 = Some a*
      using *Cons.prems(3) h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      using *t1t2-not-none h1h2-not-none* **by** *auto*
    have *t1t2-is-t*: *WEST-and-state t1 t2 = Some t*
      using *Cons.prems(3) h1t1 h2t2 WEST-and-state.simps(2)*[*of h1 t1 h2 t2*]
      using *t1t2-not-none h1h2-not-none* **by** *auto*
    let *?num-vars-m1 = num-vars* $-$ *1*
    have *len-t*: *Suc* (*length t*) = *num-vars*
      using *Cons.prems(1−6)* **by** *simp*
    then have *length-t*: *length t = ?num-vars-m1* **by** *simp*
    then have *length-t1*: *length t1 = ?num-vars-m1* **using** *Cons.prems(1−6) h1t1*
by *simp*
    then have *length-t2*: *length t2 = ?num-vars-m1* **using** *Cons.prems(1−6) h2t2*
by *simp*
    have ($a = One \longrightarrow 0 \in state$) $\wedge$ ($a = Zero \longrightarrow 0 \notin state$)
      using *h1h2-is-a Cons.prems(1, 2) h1t1 h2t2 WEST-and-bitwise.simps*
    by (*smt* (*verit*) *WEST-and-bitwise.elims len-t nth-Cons-0 option.inject zero-less-Suc*)
    then have *a-fact*: (($a\ \#\ t$) *!* $0 = One \longrightarrow 0 \in state$) $\wedge$ (($a\ \#\ t$) *!* $0 = Zero$
$\longrightarrow 0 \notin state$) **by** *auto*
    let *?adv-state = advance-state state*

have $\forall x < num\text{-}vars.((h1\#t1)\ !\ x = One \longrightarrow x \in state) \wedge ((h1\#t1)\ !\ x = Zero \longrightarrow x \notin state)$
  using *Cons.prems(1) h1t1 advance-state.simps[of state]* **by** *blast*
 then have *cond1*: $\forall x < num\text{-}vars-1.(t1\ !\ x = One \longrightarrow (x+1) \in state) \wedge (t1\ !\ x = Zero \longrightarrow (x+1) \notin state)$
  using *index-shift-reverse[of h1 t1]* **by** *simp*
 then have *cond1*: $\forall x < num\text{-}vars-1.(t1\ !\ x = One \longrightarrow x \in ?adv\text{-}state) \wedge (t1\ !\ x = Zero \longrightarrow x \notin ?adv\text{-}state)$
  using *advance-state-elt-member* **by** *fastforce*
 have $\forall x < num\text{-}vars.((h2\#t2)\ !\ x = One \longrightarrow x \in state) \wedge ((h2\#t2)\ !\ x = Zero \longrightarrow x \notin state)$
  using *Cons.prems(2) h2t2 advance-state.simps[of state]* **by** *blast*
 then have $\forall x < num\text{-}vars-1.(t2\ !\ x = One \longrightarrow (x+1) \in state) \wedge (t2\ !\ x = Zero \longrightarrow (x+1) \notin state)$
  using *index-shift-reverse[of h2 t2]* **by** *simp*
 then have *cond2*: $\forall x < num\text{-}vars-1.(t2\ !\ x = One \longrightarrow x \in ?adv\text{-}state) \wedge (t2\ !\ x = Zero \longrightarrow x \notin ?adv\text{-}state)$
  using *advance-state-elt-member* **by** *fastforce*
 have *t-fact*: $\forall x < length\ t.\ (t\ !\ x = One \longrightarrow x \in ?adv\text{-}state) \wedge (t\ !\ x = Zero \longrightarrow x \notin ?adv\text{-}state)$
  using *Cons.hyps[of ?num-vars-m1 t1 ?adv-state t2]*
  using *length-t length-t1 length-t2 t1t2-is-t cond1 cond2*
  by (*metis (mono-tags, opaque-lifting) state-regex-of-vars-def*)
 then have *t-fact*: $\forall x < length\ t.\ (t\ !\ x = One \longrightarrow (x+1) \in state) \wedge (t\ !\ x = Zero \longrightarrow (x+1) \notin state)$
  using *advance-state-elt-member* **by** *auto*
 have *cons-index*: $\forall x < length\ (a\#t).\ (t\ !\ x) = (a\#t)!(x+1)$ **by** *auto*
 have *somestate-fact*: $\forall x < length\ (a\#t).\ ((a\ \#\ t)\ !\ x = One \longrightarrow x \in state) \wedge ((a\ \#\ t)\ !\ x = Zero \longrightarrow x \notin state)$
  using *a-fact t-fact index-shift[of a state] Cons.prems(5,6)*
  using ⟨$(a = One \longrightarrow 0 \in state) \wedge (a = Zero \longrightarrow 0 \notin state)$⟩ **by** *blast*
  show *?case*
  using *somestate-fact len-t* **by** *auto*
 qed
 have *match-somestate*: *match-timestep state somestate*
  using *somestate-obt assms somestate-bits*
  using *len-s2 len-somestate*
  unfolding *match-timestep-def*
  by *metis*
 then show *?thesis* **using** *somestate-obt* **by** *simp*
qed

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)*
 using *assms*

proof(*induct s arbitrary*: *s1 s2 x*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons h t*)
  then obtain *h1 t1* **where** *h1t1*: *s1 = h1 # t1*
      by (*metis WEST-and-state.simps(1) length-greater-0-conv neq-Nil-conv option.inject*)
  obtain *h2 t2* **where** *h2t2*: *s2 = h2 # t2*
    using *Cons*
      by (*metis WEST-and-state.simps(1) length-greater-0-conv neq-Nil-conv option.inject*)
  have *notnone1*: *WEST-and-bitwise h1 h2 ≠ None* **using** *h1t1 h2t2 Cons(2)*
WEST-and-state.simps(2)[of h1 t1 h2 t2]
    by (*metis option.distinct(1) option.simps(4)*)
  have *notnone2*: *WEST-and-state t1 t2 ≠ None* **using** *h1t1 h2t2 Cons(2)*
WEST-and-state.simps(2)[of h1 t1 h2 t2]
    by (*metis option.case-eq-if option.discI*)
  have *someh*: *WEST-and-bitwise h1 h2 = Some h* **using** *h1t1 h2t2 Cons(2)*
WEST-and-state.simps(2)[of h1 t1 h2 t2]
    notnone1 notnone2 **by** *auto*
 have *somet*: *WEST-and-state t1 t2 = Some t* **using** *h1t1 h2t2 Cons(2) WEST-and-state.simps(2)[of h1 t1 h2 t2]*
    notnone1 notnone2 **by** *auto*
  then have *some-t*: *x < length t ⟹ Some (t ! x) = WEST-and-bitwise (t1 ! x) (t2 ! x)* **for** *x*
    using *h1t1 h2t2 Cons(1)[OF somet] Cons(3)*
    by *simp*
  have *some-zero*: *Some ((h # t) ! 0) = WEST-and-bitwise (s1 ! 0) (s2 ! 0)*
    using *someh h1t1 h2t2* **by** *simp*
 {assume *∗*: *x = 0*
   then have *?case*
     using *some-zero* **by** *auto*
  } moreover {assume *∗*: *x > 0*
   then have *xminus-lt*: *x−1 < length t*
     using *Cons(4)* **by** *simp*
   have *Some ((h # t) ! x) = Some (t ! (x−1))*
     using *∗*
     by *auto*
   then have *?case*
     using *some-t[OF xminus-lt] h1t1 h2t2*
     by (*simp add*: *∗*)
 }
  ultimately show *?case*
    by *blast*
qed

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*: $\exists$ *somestate.* (*match-timestep state somestate*) $\wedge$ (*WEST-and-state s1 s2*) = *Some somestate*

shows *match-timestep state s1*

proof $-$

have *s1-bits*: ($\forall x < length\ s1$. (*s1 ! x = One* $\longrightarrow$ *x* $\in$ *state*) $\wedge$ (*s1 ! x = Zero* $\longrightarrow$ *x* $\notin$ *state*))

  using *assms*

 proof(*induct s1 arbitrary: s2 num-vars state*)

  case *Nil*

  then show *?case* **by** *auto*

 next

  case (*Cons h1 t1*)

  obtain *somestate* **where**

  somestate-obt: (*match-timestep state somestate*) $\wedge$ (*WEST-and-state* (*h1#t1*) *s2*) = *Some somestate*

   using *Cons.prems(3)* **by** *auto*

 have *len-s1*: *length* (*h1#t1*) = *num-vars* **using** *Cons.prems* **unfolding** *state-regex-of-vars-def* **by** *simp*

 have *len-s2*: *length s2 = num-vars* **using** *Cons.prems* **unfolding** *state-regex-of-vars-def* **by** *simp*

  then obtain *h2 t2* **where** *h2t2*: *s2=h2#t2*

   by (*metis WEST-and-state.simps(3) neq-Nil-conv not-Some-eq somestate-obt*)

  have *len-somestate*: *length somestate = num-vars*

  using *somestate-obt WEST-and-state-length*[*of - s2*] **unfolding** *state-regex-of-vars-def len-s2*

   using *len-s1* **by** *fastforce*

  then obtain *h t* **where** *ht*: *somestate = h#t* **using** *len-s1*

   by (*metis Ex-list-of-length Zero-not-Suc length-Cons neq-Nil-conv*)

  have *somestate-bits*: ($\forall x < length\ somestate$. (*somestate ! x = One* $\longrightarrow$ *x* $\in$ *state*) $\wedge$ (*somestate ! x = Zero* $\longrightarrow$ *x* $\notin$ *state*))

   using *somestate-obt* **unfolding** *match-timestep-def* **by** *argo*

  then have *somestate-bits-conv*: ($\forall x < length\ somestate$. (*x* $\in$ *state* $\longrightarrow$ (*somestate ! x = One* $\vee$ *somestate ! x = S*)) $\wedge$

                            (*x* $\notin$ *state* $\longrightarrow$ (*somestate ! x = Zero* $\vee$ *somestate ! x = S*)))

   by (*meson WEST-bit.exhaust*)

  have *WEST-and-state* (*h1#t1*) *s2 = Some somestate* **using** *somestate-obt* **by** *blast*

  then have *somestate-and*: *WEST-and-state* (*h1#t1*) (*h2#t2*) = *Some* (*h#t*)

   using *h2t2 ht* **by** *simp*

  have (*somestate ! 0 = One* $\longrightarrow$ *0* $\in$ *state*) $\wedge$ (*somestate ! 0 = Zero* $\longrightarrow$ *0* $\notin$ *state*)

using *somestate-bits len-somestate len-s1* **by** *simp*

then have *somestate-bit0*: $(h = One \longrightarrow 0 \in state) \land (h = Zero \longrightarrow 0 \notin state)$

using *ht* **by** *simp*

have *h1h2-not-none*: *WEST-and-bitwise h1 h2* $\neq$ *None*

using *somestate-and WEST-and-state.simps(2)[of h1 t1 h2 t2] h2t2*

using *option.simps(4)* **by** *fastforce*

have *t1t2-not-none*: *WEST-and-state t1 t2* $\neq$ *None*

using *h1h2-not-none somestate-and WEST-and-state.simps(2)[of h1 t1 h2 t2]*

using *option.simps(4)* **by** *fastforce*

then have *h1h2-is-h*: *WEST-and-bitwise h1 h2* $=$ *Some h*

using *somestate-and WEST-and-state.simps(2)[of h1 t1 h2 t2] h1h2-not-none*

by *auto*

have *h-fact-converse*: $(0 \in state \longrightarrow (h1 = One \lor h1 = S)) \land (0 \notin state \longrightarrow (h1 = Zero \lor h1 = S))$

using *somestate-bit0 h1h2-is-h WEST-and-bitwise.simps[of h1] h1h2-not-none*

by (*metis* (*full-types*) *WEST-and-bitwise.elims option.inject*)

then have *h-fact*: $(h1 = One \longrightarrow 0 \in state) \land (h1 = Zero \longrightarrow 0 \notin state)$ **by** *auto*

have *somestate-bits-t*: $\forall x < length\ t.\ (t!x = One \longrightarrow (x+1) \in state) \land (t!x = Zero \longrightarrow (x+1) \notin state)$

using *index-shift-reverse[of h t] Cons.prems(1) somestate-bits len-somestate len-s1 ht* **by** *blast*

have *t1t2-is-t*: *WEST-and-state t1 t2* $=$ *Some t*

using *somestate-and WEST-and-state.simps(2)[of h1 t1 h2 t2] t1t2-not-none h1h2-not-none* **by** *auto*

then have *t1t2-is-t-indices*: $\forall x < length\ t.\ Some\ (t!x) = WEST\text{-}and\text{-}bitwise\ (t1!x)\ (t2!x)$

using *WEST-and-state-indices[of t1 t2 t] len-s1 len-s2 h2t2* **by** *simp*

have *t-fact-converse1*: $\bigwedge x.\ x < length\ t1 \implies (((x+1) \in state \longrightarrow (t1!x = One \lor t1!x = S)) \land ((x+1) \notin state \longrightarrow (t1!x = Zero \lor t1!x = S)))$

proof $-$

fix *x*

assume *x-lt*: $x < length\ t1$

have $*$: $(t!x = One \longrightarrow (x+1) \in state) \land (t!x = Zero \longrightarrow (x+1) \notin state)$

using *x-lt somestate-bits-t len-s1 len-somestate ht* **by** *auto*

have $**$: $Some\ (t\ !\ x) = WEST\text{-}and\text{-}bitwise\ (t1\ !\ x)\ (t2\ !\ x)$

using *x-lt somestate-bits-t len-s1 len-somestate ht t1t2-is-t-indices* **by** *auto*

{**assume** *case1*: $(x+1) \in state$

then have $t!x = One \lor t1!x = S$

using $*$

by (*smt* (*verit*) $**$ *WEST-and-bitwise.elims WEST-and-bitwise.simps(2) option.distinct(1) option.inject*)

then have $(t1!x = One \lor t1!x = S)$

using *x-lt WEST-and-bitwise.simps[of t1!x]* $* **$

by (*metis* (*full-types*) *WEST-bit.exhaust not-None-eq option.inject*)

} **moreover** {**assume** *case2*: $(x+1) \notin state$

then have *t!x = Zero ∨ t1!x = S*
   using *∗*
   by (*smt* (*verit*) *∗∗ WEST-and-bitwise.elims WEST-and-bitwise.simps*(*2*)
option.distinct(*1*) *option.inject*)
     then have (*t1!x = Zero ∨ t1!x = S*)
      using *x-lt WEST-and-bitwise.simps*[*of t1!x*] *∗ ∗∗*
      by (*metis* (*full-types*) *WEST-bit.exhaust not-None-eq option.inject*)
   }
   ultimately show (((*x+1*) *∈ state* ⟶ (*t1!x = One ∨ t1!x = S*)) *∧* ((*x+1*)
∉ state ⟶ (*t1!x = Zero ∨ t1!x = S*)))
     by *blast*
  qed
  then have *t-fact*: *∀ x<length t1*. (*t1!x = One* ⟶ (*x+1*)*∈state*) *∧* (*t1!x =*
Zero ⟶ (*x+1*)*∉state*)
   by *force*

 show *?case*
   using *h-fact t-fact Cons.prems len-s2 len-somestate index-shift*[*of h1 state*]
   unfolding *state-regex-of-vars-def* **by** *blast*
 qed

 show *?thesis*
   using *s1-bits assms*(*1*) **unfolding** *match-timestep-def*
   using *state-regex-of-vars-def s1-of-num-vars* **by** *presburger*
qed

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−
 have *match-s1*: *match-timestep state s1* **using** *assms WEST-and-state-correct-converse-s1*
by *simp*
 have *match-s2*: *match-timestep state s2*
  using *assms WEST-and-state-correct-converse-s1 WEST-and-state-commutative*
   by (*simp add*: *state-regex-of-vars-def*)
 show *?thesis* **using** *match-s1 match-s2* **by** *simp*
qed

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* $\wedge$ *match-timestep state s2*) $\longleftrightarrow$ ($\exists$ *somestate.*
match-timestep state somestate $\wedge$ (*WEST-and-state s1 s2*) = *Some somestate*)
    using *assms WEST-and-state-correct-converse*
          WEST-and-state-correct-forward **by** *metis*

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* $\wedge$ *match-regex π r2*
  shows *WEST-and-trace r1 r2* $\neq$ *None*
proof$-$
  show *?thesis* **using** *assms*
  proof(*induct r1 arbitrary*: *r2 π num-vars*)
    case *Nil*
    {**assume** *r2-empty*:*r2* = []
      then have *?case* **using** *WEST-and-trace.simps* **by** *blast*
    } **moreover** {**assume** *r2-nonempty*: *r2*$\neq$[]
      then obtain *h2 t2* **where** *r2* = *h2#t2*
        by (*metis trim-reversed-regex.cases*)
      then have*?case* **using** *WEST-and-trace.simps(2)[of h2 t2]* **by** *blast*
      }
      ultimately show *?case* **by** *blast*
  next
    case (*Cons h1 t1*)
    {**assume** *r2-empty*:*r2* = []
      then have *?case* **using** *WEST-and-trace.simps* **by** *blast*
    } **moreover** {**assume** *r2-nonempty*: *r2*$\neq$[]
      then obtain *h2 t2* **where** *h2t2*: *r2* = *h2#t2*
        by (*metis trim-reversed-regex.cases*)

      have *h1t1-nv*: $\forall$*i$<$length* (*h1* # *t1*). *length* ((*h1* # *t1*) ! *i*) = *num-vars*
        using *Cons.prems(1)* **unfolding** *trace-regex-of-vars-def* **by** *argo*
      then have *length* ((*h1* # *t1*) ! *0*) = *num-vars* **by** *blast*
      then have *h1-nv*: *state-regex-of-vars h1 num-vars*
        unfolding *state-regex-of-vars-def* **by** *simp*
      have *h2t2-nv*: $\forall$*i$<$length* (*h2* # *t2*). *length* ((*h2* # *t2*) ! *i*) = *num-vars*
        using *Cons.prems(2) h2t2* **unfolding** *trace-regex-of-vars-def* **by** *metis*
      then have *length* ((*h2* # *t2*) ! *0*) = *num-vars* **by** *blast*
      then have *h2-nv*: *state-regex-of-vars h2 num-vars*
        unfolding *state-regex-of-vars-def* **by** *simp*

      have *match-timestep* (π ! *0*) *h1* $\wedge$ *match-timestep* (π ! *0*) *h2*

using *Cons(4)* **unfolding** *match-regex-def*
　　by (*metis h2t2 length-greater-0-conv list.distinct(1) nth-Cons-0*)
　then have *h1h2-notnone*: *WEST-and-state h1 h2 ≠ None*
　　using *WEST-and-state-well-defined[of h1 num-vars h2 π!0, OF h1-nv h2-nv]*
by *blast*

　have *t1-nv*: *trace-regex-of-vars t1 num-vars*
　　using *h1t1-nv* **unfolding** *trace-regex-of-vars-def* **by** *auto*
　have *t2-nv*: *trace-regex-of-vars t2 num-vars*
　　using *h2t2-nv* **unfolding** *trace-regex-of-vars-def* **by** *auto*

　have *unfold-prem3*: (∀ *time*<*length* (*h1* # *t1*). *match-timestep* (π ! *time*) ((*h1* # *t1*) ! *time*)) ∧
　　length (*h1* # *t1*) ≤ *length* π ∧ (∀ *time*<*length r2*. *match-timestep* (π ! *time*) (*r2* ! *time*)) ∧ *length r2* ≤ *length* π
　　using *Cons.prems(3)* **unfolding** *match-regex-def* **by** *argo*

　have *unfold-prem3-bounds*: *length* (*h1* # *t1*) ≤ *length* π ∧ *length r2* ≤ *length* π
　　using *unfold-prem3* **by** *blast*
　have *π-drop1-len*: *length* (*drop 1* π) = (*length* π)−1 **by** *simp*
　have *len-t1t2*: *length t1* = *length* (*h1*#*t1*)−1 ∧ *length t2* = *length* (*h2*#*t2*)−1
by *simp*
　have *bounds*: *length t1* ≤ *length* (*drop 1* π) ∧ *length t2* ≤ *length* (*drop 1* π)
　　using *unfold-prem3-bounds h2t2 π-drop1-len len-t1t2 h2t2*
　　by (*metis diff-le-mono*)

　have *unfold-prem3-matches*: (∀ *time*<*length* (*h1* # *t1*). *match-timestep* (π ! *time*) ((*h1* # *t1*) ! *time*)) ∧
　　　　　　　　　　(∀ *time*<*length* (*h2* # *t2*). *match-timestep* (π ! *time*) ((*h2* # *t2*) ! *time*))
　　using *unfold-prem3 h2t2* **by** *blast*

　have *h1t1-match*:(∀ *time*<*length* (*h1* # *t1*). *match-timestep* (π ! *time*) ((*h1* # *t1*) ! *time*))
　　using *unfold-prem3-matches* **by** *blast*
　then have (⋀*time*. *time*<*length t1* ⟹ *match-timestep* (*drop 1* π ! *time*) (*t1* ! *time*))
　proof −
　　fix *time*
　　assume *time-bound*: *time* < *length t1*
　　have *time+1* < *length* (*h1*#*t1*) **using** *time-bound* **by** *auto*
　　then have *match-timestep* (π ! (*time+1*)) ((*h1* # *t1*) ! (*time+1*)) **using** *h1t1-match* **by** *auto*
　　then show *match-timestep* (*drop 1* π ! *time*) (*t1* ! *time*)
　　　using *cancel-comm-monoid-add-class.diff-cancel unfold-prem3* **by** *auto*
　qed
　then have *t1-match*: (∀ *time*<*length t1*. *match-timestep* (*drop 1* π ! *time*) (*t1* ! *time*))

43

by *blast*

have *h2t2-match*: $\forall$ *time* < *length* (*h2* # *t2*). *match-timestep* (π ! *time*) ((*h2* # *t2*) ! *time*)
   using *unfold-prem3-matches* **by** *blast*
then have ($\bigwedge$*time*. *time*<*length t2* $\implies$ *match-timestep* (*drop 1* π ! *time*) (*t2* ! *time*))
   proof −
   fix *time*
   assume *time-bound*: *time* < *length t2*
   have *time+1* < *length* (*h2*#*t2*) **using** *time-bound* **by** *auto*
   then have *match-timestep* (π ! (*time+1*)) ((*h2* # *t2*) ! (*time+1*)) **using** *h2t2-match* **by** *auto*
   then show *match-timestep* (*drop 1* π ! *time*) (*t2* ! *time*)
      using *cancel-comm-monoid-add-class.diff-cancel unfold-prem3* **by** *auto*
   qed
then have *t2-match*: ($\forall$ *time*<*length t2*. *match-timestep* (*drop 1* π ! *time*) (*t2* ! *time*))
   by *blast*

then have *matches*: ($\forall$ *time*<*length t1*. *match-timestep* (*drop 1* π ! *time*) (*t1* ! *time*)) $\wedge$
               ($\forall$ *time*<*length t2*. *match-timestep* (*drop 1* π ! *time*) (*t2* ! *time*))
   using *t1-match t2-match* **by** *blast*
have *match-regex* (*drop 1* π) *t1* $\wedge$ *match-regex* (*drop 1* π) *t2*
   using *bounds matches* **unfolding** *match-regex-def h2t2* **by** *auto*
then have *t1t2-notnone*: *WEST-and-trace t1 t2* $\neq$ *None*
   using *Cons.hyps*[*of num-vars t2 drop 1* π, *OF t1-nv t2-nv*] **by** *simp*

have *WEST-and-trace* (*h1* # *t1*) (*h2* # *t2*) $\neq$ *None*
   using *h1h2-notnone t1t2-notnone WEST-and-trace.simps*(*3*) **by** *auto*
then have *?case* **using** *h2t2* **by** *auto*
 }
 ultimately show *?case* **by** *blast*
qed
qed

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 −
 have *ind-h*: ($\forall$ *time*<*length* (*h*#*t*). *match-timestep* (π ! *time*) ((*h*#*t*) ! *time*)) $\wedge$
length (*h*#*t*) $\leq$ *length* π
   using *assms* **unfolding** *match-regex-def* **by** *metis*
 then have *part1*: *match-timestep* (π ! *0*) *h*
  by *auto*
 have *part2*: *match-timestep* (*drop 1* π ! *time*) (*t* ! *time*) **if** *time-lt*: *time*<*length*
t **for** *time*
  proof −

44

have *match*: *match-timestep* (π ! (*time+1*)) ((*h* # *t*) ! (*time+1*))
  using *ind-h time-lt* **by** *auto*
have (π ! (*time + 1*)) = (*drop 1 π ! time*)
   using *add.commute add-gr-0 impossible-Cons ind-h less-add-same-cancel2*
less-eq-iff-succ-less **by** *auto*
then show *?thesis* **using** *match* **by** *auto*
qed
have *part3*: *length t $\leq$ length* (*drop 1 π*)
  using *ind-h* **by** *auto*
show *?thesis* **using** *part1 part2 part3* **unfolding** *match-regex-def* **by** *simp*
qed

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$-$
  have *all-time-t*: $\forall$ *time<length t. match-timestep* (*txi ! time*) (*t ! time*)
   using *assms(3)* **unfolding** *match-regex-def* **by** *argo*
  have *len-t-leq*: *length t $\leq$ length txi*
   using *assms(3)* **unfolding** *match-regex-def* **by** *argo*
 have *match-ht*: *match-timestep* (π ! *time*) ((*h* # *t*) ! *time*) **if** *time-ht*: *time<length*
(*h* # *t*)
   for *time*
  proof $-$
   {assume $*$: *time = 0*
    then have *?thesis*
     using *assms(2) assms(1)*
     by *auto*
   } moreover {assume $*$: *time > 0*
    then have *?thesis*
    using *time-ht all-time-t assms(1)*
    by *auto*
   }
   ultimately show *?thesis*
    by *blast*
  qed
  have *len-condition*: *length* (*h* # *t*) $\leq$ *length π*
   using *assms(1) len-t-leq* **by** *auto*
  then show *?thesis*
   using *match-ht len-condition* **unfolding** *match-regex-def* **by** *simp*
qed

lemma *WEST-and-trace-correct-forward-empty-trace*:
  fixes *num-vars::nat*
  fixes π::*trace*
  fixes *r1 r2:: trace-regex*

45

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 ∃ *sometrace. match-regex* [] *sometrace* ∧ (*WEST-and-trace r1 r2*) = *Some sometrace*
proof −
  have *r1-empty*: *length r1* ≤ *length* []
    using *match1* **unfolding** *match-regex-def*
    by (*metis list.size(3)*)
  have *r2-empty*: *length r2* ≤ *length* []
    using *match2* **unfolding** *match-regex-def*
  by (*metis list.size(3)*)
  have *r1r2*: *r1* = [] ∧ *r2* = []
    using *r1-empty r2-empty* **by** *simp*
  have *match-regex* [] [] ∧ (*WEST-and-trace* [] []) = *Some* []
    unfolding *WEST-and-trace.simps match-regex-def* **by** *simp*
  then show *?thesis* **using** *r1r2*
    by *blast*
qed

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* ∧ *match-regex π r2*
  assumes *length π* > *0*
  shows ∃ *sometrace. match-regex π sometrace* ∧ (*WEST-and-trace r1 r2*) = *Some sometrace*
proof −
  have *WEST-and-trace r1 r2* ≠ *None*
    using *WEST-and-trace-well-defined*[*of r1 num-vars r2 π*] *assms* **by** *blast*
  then obtain *sometrace* **where** *sometrace-obt*: *WEST-and-trace r1 r2* = *Some sometrace* **by** *blast*

  have *match-regex π sometrace*
    using *assms sometrace-obt*
  proof(*induct sometrace arbitrary*: *r1 r2 π*)
    case *Nil*
    then show *?case* **unfolding** *match-regex-def* **by** *auto*
  next
    case (*Cons h t*)

    have *match-r1*: (∀ *time*<*length r1. match-timestep* (*π* ! *time*) (*r1* ! *time*))
      using *Cons.prems(3)* **unfolding** *match-regex-def* **by** *argo*

    have *match-r2*: (∀ *time*<*length r2. match-timestep* (*π* ! *time*) (*r2* ! *time*))

46

using *Cons.prems(3)* **unfolding** *match-regex-def* **by** *argo*

have *match-h-match-t*: *match-timestep* (π!0) h ∧ *match-regex* (*drop 1* π) t
proof−
  {**assume** *r1r2-empty*: *r1* = [] ∧ *r2* = []
    have *WEST-and-trace r1 r2* = *Some* []
      using *WEST-and-trace.simps r1r2-empty* **by** *blast*
    then have *ht-empty*: *h* = [] ∧ *t* = []
      using *Cons.prems* **by** *simp*
    have *match-timestep* (π!0) [] ∧ *match-regex* (*drop 1* π) []
      unfolding *match-regex-def match-timestep-def* **by** *simp*
    then have *match-timestep* (π!0) h ∧ *match-regex* (*drop 1* π) t
      using *ht-empty* **by** *simp*
  } **moreover** {
    assume *r1-empty*: *r1* = [] ∧ *r2* ≠ []
    obtain *h2 t2* **where** *h2t2*: *r2* = *h2#t2*
      by (*meson neq-Nil-conv r1-empty*)
    have *WEST-and-trace r1 r2* = *Some* (*h2#t2*)
      using *r1-empty WEST-and-trace.simps(2)[of h2 t2] h2t2* **by** *blast*
    then have *hh2-tt2*: *h=h2* ∧ *t=t2*
      using *Cons.prems* **by** *simp*
    have *match-timestep* (π!0) *h2* ∧ *match-regex* (*drop 1* π) *t2*
      using *WEST-and-trace-correct-forward-aux[of* π *h2 t2] Cons(4) h2t2* **by**
auto
    then have *match-timestep* (π!0) h ∧ *match-regex* (*drop 1* π) t
      using *hh2-tt2* **by** *simp*
  } **moreover** {
    assume *r2-empty*: *r2* = [] ∧ *r1* ≠ []
    obtain *h1 t1* **where** *h1t1*: *r1* = *h1#t1*
      by (*meson neq-Nil-conv r2-empty*)
    have *WEST-and-trace r1 r2* = *Some* (*h1#t1*)
      using *r2-empty WEST-and-trace.simps(1)[of r1] h1t1*
      by *blast*
    then have *hh1-tt1*: *h=h1* ∧ *t=t1*
      using *Cons.prems* **by** *simp*
    have *match-timestep* (π!0) h ∧ *match-regex* (*drop 1* π) t
        using *WEST-and-trace-correct-forward-aux[of* π *h1 t1] Cons(4) h1t1*
hh1-tt1
    by *blast*
  } **moreover** {
    assume *r1r2-nonempty*: *r1* ≠ [] ∧ *r2* ≠ []
    obtain *h1 t1* **where** *h1t1*: *r1* = *h1#t1*
      by (*meson neq-Nil-conv r1r2-nonempty*)
    obtain *h2 t2* **where** *h2t2*: *r2* = *h2#t2*
      by (*meson neq-Nil-conv r1r2-nonempty*)

    have *ht*: *WEST-and-trace* (*h1#t1*) (*h2#t2*) = *Some* (*h # t*)
      using *Cons(6) h1t1 h2t2* **by** *blast*
    then have *h1h2-notnone*: *WEST-and-state h1 h2* ≠ *None*

      using *WEST-and-trace.simps(3)*[*of h1 t1 h2 t2*]
      using *not-None-eq* **by** *fastforce*
    then have *t1t2-notnone*: *WEST-and-trace t1 t2 $\neq$ None*
      using *WEST-and-trace.simps(3)*[*of h1 t1 h2 t2*]
      using *not-None-eq*
    using ‹*WEST-and-trace (h1 # t1) (h2 # t2) = Some (h # t)*› **by** *fastforce*
    have *h-is*: (*WEST-and-state h1 h2*) = *Some h*
    using *WEST-and-trace.simps(3)*[*of h1 t1 h2 t2*] *h1h2-notnone t1t2-notnone*

ht

      by *auto*
    have *t-is*: (*WEST-and-trace t1 t2*) = *Some t*
    using *WEST-and-trace.simps(3)*[*of h1 t1 h2 t2*] *h1h2-notnone t1t2-notnone*

ht

      by *auto*

    have *h1t1-nv*: $\forall$ *i<length (h1#t1). length ((h1#t1) ! i) = num-vars*
      using *Cons.prems(1) h1t1* **unfolding** *trace-regex-of-vars-def* **by** *meson*
    then have *hyp1*: *trace-regex-of-vars t1 num-vars*
      unfolding *trace-regex-of-vars-def* **by** *auto*
    have *h2t2-nv*: $\forall$ *i<length (h2#t2). length ((h2#t2) ! i) = num-vars*
      using *Cons.prems(2) h2t2* **unfolding** *trace-regex-of-vars-def* **by** *meson*
    then have *hyp2*: *trace-regex-of-vars t2 num-vars*
      unfolding *trace-regex-of-vars-def* **by** *auto*

    have *hyp3a*: *match-regex (drop 1 π) t1*
    using *WEST-and-trace-correct-forward-aux*[*of π h1 t1*] *h1t1 Cons.prems(3)*
by *blast*
    have *hyp3b*: *match-regex (drop 1 π) t2*
    using *WEST-and-trace-correct-forward-aux*[*of π h2 t2*] *h2t2 Cons.prems(3)*
by *blast*
    have *hyp3*: *match-regex (drop 1 π) t1 $\wedge$ match-regex (drop 1 π) t2*
      using *hyp3a hyp3b* **by** *auto*

    have *match-regex (drop 1 π) t* **if** [] = (*drop 1 π*)
      using *WEST-and-trace-correct-forward-empty-trace*[*of t1 num-vars t2*]
      using *hyp3a hyp3b hyp1 hyp2*
      using *t-is that* **by** *auto*

    then have *match-t*: *match-regex (drop 1 π) t*
      using *Cons.hyps*[*of t1 t2 (drop 1 π), OF hyp1 hyp2 hyp3*] *t-is*
      by *fastforce*

    have *h1-nv*: *state-regex-of-vars h1 num-vars*
      using *h1t1-nv* **unfolding** *state-regex-of-vars-def* **by** *auto*
    have *h2-nv*: *state-regex-of-vars h2 num-vars*
      using *h2t2-nv* **unfolding** *state-regex-of-vars-def* **by** *auto*
    have *match-h1*: *match-timestep (π ! 0) h1*
      using *Cons.prems(3) h1t1* **unfolding** *match-regex-def*
      using *Cons.prems(3) WEST-and-trace-correct-forward-aux* **by** *blast*

48

have *match-h2*: *match-timestep* (π ! *0*) *h2*
  using *Cons.prems*(*3*) *h2t2* **unfolding** *match-regex-def*
  using *Cons.prems*(*3*) *WEST-and-trace-correct-forward-aux* **by** *blast*
have *match-h*: *match-timestep* (π!*0*) *h*
  using *WEST-and-state-correct-forward*[*of h1 num-vars h2 π!0, OF h1-nv*
h2-nv] *h-is*
  using *match-h1 match-h2* **by** *simp*

  have *match-timestep* (π!*0*) *h* $\wedge$ *match-regex* (*drop 1 π*) *t*
    using *match-h match-t* **by** *blast*
  }
  ultimately show *match-timestep* (π!*0*) *h* $\wedge$ *match-regex* (*drop 1 π*) *t*
    by *blast*
qed

have *match-h*: *match-timestep* (π!*0*) *h*
  using *match-h-match-t* **by** *auto*
have *match-t*: *match-regex* (*drop 1 π*) *t*
  using *match-h-match-t* **by** *auto*

have *len-π*: *length* (*drop 1 π*) = (*length π*)-1 **by** *auto*
have *len-ht*: *length t* = *length* (*h#t*)-1 **by** *auto*
have *length t* $\leq$ *length* (*drop 1 π*) **using** *match-t* **unfolding** *match-regex-def*
by *argo*
then have (*length* (*h#t*))-1 $\leq$ (*length π*)-1 **using** *len-π len-ht* **by** *argo*
then have *ht-less-π*: *length* (*h#t*) $\leq$ *length π*
  using *Cons.prems*(*4*)
  by *linarith*

have ($\bigwedge$*time. time<length* (*h # t*) $\Longrightarrow$ (*match-timestep* (π ! *time*) ((*h # t*) !
time)) $\wedge$
  length (*h # t*) $\leq$ *length π*)
proof$-$
  fix *time*
  assume *time-bound*: *time<length* (*h # t*)
  {assume $*$:*time=0*
   have (*match-timestep* (π ! *0*) *h*) $\wedge$ *length* (*h # t*) $\leq$ *length π*
    using *match-h ht-less-π* **by** *simp*
   then have (*match-timestep* (π ! *time*) ((*h # t*) ! *time*)) $\wedge$ *length* (*h # t*) $\leq$
length π
    using $*$ **by** *simp*
  } moreover {
   assume $**$: *time > 0*
   have *time-m1*: *time-1 < length t*
    using *time-bound*
    using $**$ *len-ht* **by** *linarith*
   have ($\forall$ *time<length t. match-timestep* (*drop 1 π ! time*) (*t ! time*))
    using *match-t* **unfolding** *match-regex-def* **by** *argo*
   then have *fact0*: *match-timestep* (*drop 1 π ! (time-1)*) (*t ! (time-1)*)

49

      using *time-m1* **by** *blast*
     have *fact1*: $(t \; ! \; (time{-}1)) = ((h \; \# \; t) \; ! \; time)$
      by (*simp add*: $**$)
     have *fact2*: $(drop \; 1 \; \pi \; ! \; (time{-}1)) = (\pi \; ! \; time)$
      using $**$ *time-m1 ht-less-*π **by** *force*

     then have $(match\text{-}timestep \; (\pi \; ! \; time) \; ((h \; \# \; t) \; ! \; time))$
      using *fact1 fact2 fact0* **by** *simp*
     then have $(match\text{-}timestep \; (\pi \; ! \; time) \; ((h \; \# \; t) \; ! \; time)) \wedge length \; (h \; \# \; t) \leq$
length π
      using *ht-less-*π **by** *simp*
    }
   ultimately show $(match\text{-}timestep \; (\pi \; ! \; time) \; ((h \; \# \; t) \; ! \; time)) \wedge length \; (h \; \#$
$t) \leq length \; \pi$
     by (*metis bot-nat-0.not-eq-extremum*)
  qed
  then show *?case* **unfolding** *match-regex-def* **by** *auto*
 qed
 then show *?thesis* **using** *sometrace-obt* **by** *blast*
qed

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
 using *WEST-and-trace-correct-forward-empty-trace WEST-and-trace-correct-forward-nonempty-trace*
 assms **by** *fast*

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* $\wedge$ *WEST-and-trace t1 t2* = *Some t*
proof $-$
 have *h1h2-nn*: (*WEST-and-state h1 h2*) $\neq$ *None*
  using *WEST-and-trace.simps(3)[of h1 t1 h2 t2] assms*
  using *option.simps(4)* **by** *fastforce*
 then have *t1t2-nn*: *WEST-and-trace t1 t2* $\neq$ *None*
  using *assms WEST-and-trace.simps(3)[of h1 t1 h2 t2]*
  by (*metis (no-types, lifting) WEST-and-state-difflengths-is-none WEST-and-state-length*
option.distinct(1) option.simps(4) option.simps(5))

 have *nn*: *WEST-and-trace* $(h1 \; \# \; t1)$ $(h2 \; \# \; t2)$ $\neq$ *None* **using** *assms* **by** *blast*
 then have *h-fact*: *WEST-and-state h1 h2* = *Some h*

using *h1h2-nn t1t2-nn assms WEST-and-trace.simps(3)[of h1 t1 h2 t2]* **by** *auto*
  then have *t-fact: WEST-and-trace t1 t2 = Some t*
    using *t1t2-nn h1h2-nn assms WEST-and-trace.simps(3)[of h1 t1 h2 t2] nn* **by**
auto
  show *?thesis* **using** *h-fact t-fact* **by** *blast*
qed

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*
  using *assms*
proof(*induction r1 arbitrary:r2 sometrace*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons h1 t1*)
  {assume *r2-empty*: *r2* = []
    have *WEST-and-trace* (*h1* # *t1*) *r2* = *Some* (*h1* # *t1*)
      using *Cons WEST-and-trace.simps(1) r2-empty* **by** *blast*
    then have *?case* **using** *Cons* **by** *simp*
  } moreover {
    assume *r2-nonempty*: *r2* ≠ []
    obtain *h2 t2* **where** *h2t2*: *r2* = *h2*#*t2*
      by (*meson neq-Nil-conv r2-nonempty*)
    have *h1t1-and-h2t2*: *WEST-and-trace* (*h1* # *t1*) (*h2* # *t2*) = *Some sometrace*
      using *Cons WEST-and-trace.simps(3) h2t2* **by** *blast*
    then have *h1h2-nn*: (*WEST-and-state h1 h2*) ≠ *None*
      using *WEST-and-trace.simps(3)[of h1 t1 h2 t2]*
      using *option.simps(4)* **by** *fastforce*
    then have *t1t2-nn*: *WEST-and-trace t1 t2* ≠ *None*
      using *h1t1-and-h2t2 WEST-and-trace.simps(3)[of h1 t1 h2 t2]*
    by (*metis* (*no-types, lifting*) *WEST-and-state-difflengths-is-none WEST-and-state-length*
option.distinct(1) option.simps(4) option.simps(5))
    obtain *h* **where** *h-obt*: *WEST-and-state h1 h2* = *Some h* **using** *h1h2-nn* **by**
blast
    obtain *t* **where** *t-obt*: *WEST-and-trace t1 t2* = *Some t* **using** *t1t2-nn* **by** *blast*
    then have ∗: *sometrace* = *h* # *t*
      using *h-obt t-obt h1t1-and-h2t2* **by** *auto*
    then have *sometrace-ht*: *WEST-and-trace* (*h1* # *t1*) (*h2* # *t2*) = *Some* (*h* #
t)
      using *h2t2 h1t1-and-h2t2* **by** *blast*

    have ∀*i*<*length* (*h1* # *t1*). *length* ((*h1* # *t1*) ! *i*) = *n*
      using *Cons.prems* **unfolding** *trace-regex-of-vars-def* **by** *argo*
    then have *hyp1*: *trace-regex-of-vars t1 n*
      unfolding *trace-regex-of-vars-def* **by** *auto*
    have ∀*i*<*length* (*h2* # *t2*). *length* ((*h2* # *t2*) ! *i*) = *n*

51

using *Cons.prems h2t2* **unfolding** *trace-regex-of-vars-def* **by** *meson*
  then have *hyp2*: *trace-regex-of-vars t2 n*
    unfolding *trace-regex-of-vars-def* **by** *auto*

  have *length t ≥ length t1*
    using *Cons(1)[of t2 t, OF hyp1 hyp2 t-obt]* **by** *simp*
  then have *?case* **using** *∗* **by** *simp*
  }
  ultimately show *?case* **by** *blast*
qed

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*
  using *assms WEST-and-trace-lengths-r1 WEST-and-trace-commutative*
proof−
  have *lenr1*: *length r1 ≤ length sometrace*
    using *assms WEST-and-trace-lengths-r1[of r1 n r2 sometrace]* **by** *blast*
  have *WEST-and-trace r1 r2 = WEST-and-trace r2 r1*
    using *WEST-and-trace-commutative assms* **by** *blast*
  then have *lenr2*: *length r2 ≤ length sometrace*
    using *WEST-and-trace-lengths-r1[of r2 n r1 sometrace] assms* **by** *auto*
  show *?thesis* **using** *lenr1 lenr2* **by** *auto*
qed

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*
  using *assms*
proof(*induct r1 arbitrary*: *r2 π*)
    case *Nil*
  then show *?case*
    unfolding *match-regex-def* **by** *auto*
  next
    case (*Cons h1 t1*)
   obtain *sometrace* **where** *sometrace-obt*: *match-regex π sometrace* ∧ (*WEST-and-trace*
(*h1#t1*) *r2*) = *Some sometrace*
     using *Cons.prems* **by** *blast*
    have *match-sometrace-pre*: *match-regex π sometrace* **using** *sometrace-obt* **by**
simp
     have *r1r2-is-sometrace*: (*WEST-and-trace* (*h1#t1*) *r2*) = *Some sometrace*

52

using *sometrace-obt* **by** *simp*

    have *match-sometrace*: $\forall$ *time*<*length sometrace*. *match-timestep* (π ! *time*) (*sometrace* ! *time*)

    using *match-sometrace-pre* **unfolding** *match-regex-def* **by** *argo*

  have *len-r1*: *length* (*h1#t1*) $\leq$ *length* π

  using *Cons.prems sometrace-obt WEST-and-trace-lengths*

  by (*meson le-trans match-regex-def*)

  {assume *empty-trace*: $\pi = []$

  then have *?case* **using** *len-r1* **by** *simp*

  } moreover {

  assume *nonempty-trace*: $\pi \neq []$

  {assume *r2-empty*: *r2* = []

   have *WEST-and-trace* (*h1#t1*) *r2* = *Some* (*h1#t1*)

    using *sometrace-obt WEST-and-trace.simps r2-empty* **by** *simp*

   then have *?case* **using** *sometrace-obt*

    unfolding *match-regex-def* **by** *force*

  } moreover {

  assume *r2-nonempty*: *r2* $\neq []$

   obtain *hxi txi* **where** *hxitxi*: π = *hxi#txi* **using** *nonempty-trace* **by** (*meson list.exhaust*)

    obtain *h2 t2* **where** *h2t2*: *r2* = *h2#t2* **using** *r2-nonempty* **by** (*meson list.exhaust*)

   have *not-none*: *WEST-and-trace* (*h1#t1*) (*h2#t2*) = *Some sometrace*

    using *sometrace-obt h2t2* **by** *blast*

   have *h1h2-nn*: *WEST-and-state h1 h2* $\neq$ *None*

    using *not-none WEST-and-trace.simps(3)[of h1 t1 h2 t2] not-none*

    using *option.simps(4)* **by** *fastforce*

   then have *t1t2-nn*: *WEST-and-trace t1 t2* $\neq$ *None*

    using *not-none WEST-and-trace.simps(3)[of h1 t1 h2 t2] not-none*

    using *option.simps(4)* **by** *fastforce*

   obtain *h t* **where** *sometrace-ht*: *sometrace* = *h#t*

    using *not-none h1h2-nn t1t2-nn* **by** *auto*

   have *h1h2-h*: *WEST-and-state h1 h2* = *Some h*

    using *WEST-and-trace-nonempty-args[of h1 t1 h2 t2 h t] not-none sometrace-ht*

    by *blast*

   have *t1t2-t*: *WEST-and-trace t1 t2* = *Some t*

    using *WEST-and-trace-nonempty-args[of h1 t1 h2 t2 h t] not-none sometrace-ht*

    by *blast*

   have *match-ht*: $\forall$ *time*<*length* (*h#t*). *match-timestep* ((*hxi* # *txi*) ! *time*) (((*h#t*)) ! *time*)

    using *sometrace-ht sometrace-obt hxitxi* **unfolding** *match-regex-def*

    by *meson*

   have *h1-nv*: *state-regex-of-vars h1 num-vars*

   using *Cons.prems* **unfolding** *trace-regex-of-vars-def state-regex-of-vars-def*
   by (*metis Ex-list-of-length append-self-conv2 arbitrary-regtrace-matches-any-trace*
    bot-nat-0.not-eq-extremum le-0-eq less-nat-zero-code list.pred-inject(*2*)
   list-all-length list-ex-length list-ex-Cons-iff match-regex-def nth-append-length
    trace-of-vars-def)
   have *h2-nv*: *state-regex-of-vars h2 num-vars*
  using *Cons.prems* **unfolding** *trace-regex-of-vars-def h2t2 state-regex-of-vars-def*
  by (*metis Ex-list-of-length append-self-conv2 arbitrary-regtrace-matches-any-trace*
   bot-nat-0.not-eq-extremum le-0-eq less-nat-zero-code list.pred-inject(*2*)
  list-all-length list-ex-length list-ex-Cons-iff match-regex-def nth-append-length
   trace-of-vars-def)
  have *match-h*: *match-timestep hxi h*
   using *match-ht* **unfolding** *match-regex-def* **by** *auto*
  have *match-h1*: *match-timestep hxi h1*
    using *WEST-and-state-correct-converse-s1*[*of h1 num-vars h2 hxi, OF*
h1-nv h2-nv]
   using *sometrace-ht h1h2-h match-h* **by** *blast*

  have $\forall$ *i*<*length* (*h1 # t1*). *length* ((*h1 # t1*) ! *i*) = *num-vars*
   using *Cons.prems* **unfolding** *trace-regex-of-vars-def* **by** *argo*
  then have *t1-nv*: *trace-regex-of-vars t1 num-vars*
   unfolding *trace-regex-of-vars-def* **by** *auto*
  have $\forall$ *i*<*length* (*h2 # t2*). *length* ((*h2 # t2*) ! *i*) = *num-vars*
   using *Cons.prems h2t2* **unfolding** *trace-regex-of-vars-def* **by** *metis*
  then have *t2-nv*: *trace-regex-of-vars t2 num-vars*
   unfolding *trace-regex-of-vars-def h2t2* **by** *auto*
  have *match-regex* π (*h # t*)
   using *sometrace-ht sometrace-obt hxitxi* **unfolding** *match-regex-def*
   by *blast*
  then have *match-regex txi t*
   using *hxitxi WEST-and-trace-correct-forward-aux*[*of* π *h t*]
   unfolding *match-regex-def* **by** *fastforce*
  then have *match-t1*: *match-regex txi t1*
   using *Cons.hyps*[*of t2 txi, OF t1-nv t2-nv*] *t1t2-t* **by** *blast*

  have *?case*
   using *match-h1 match-t1 len-r1*
    using *WEST-and-trace-correct-forward-aux-converse*[*OF - match-h1*
match-t1 , of π] *hxitxi*
   by *blast*
  }
  ultimately have *?case* **by** *blast*
 }
 ultimately show *?case* **by** *blast*
 qed

lemma *WEST-and-trace-correct-converse*:
 fixes *num-vars*::*nat*

54

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 ($\exists$ *sometrace. match-regex π sometrace* $\wedge$ (*WEST-and-trace r1 r2*) =
Some sometrace)
  shows *match-regex π r1* $\wedge$ *match-regex π r2*
proof $-$
  show *?thesis* **using** *WEST-and-trace-correct-converse-r1 WEST-and-trace-commutative*
    using *assms(3) r1-of-num-vars r2-of-num-vars* **by** *presburger*
qed

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*)
  using *WEST-and-trace-correct-forward WEST-and-trace-correct-converse assms*
by *blast*

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*))
  using *assms*
proof (*induct L1*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons h1 T1*)
  {**assume** $*$: *h1 = elem*
    then have *?case* **using** *WEST-and.simps(3)[of h1 T1 h2 T2]*
      by (*simp add*: *list.case-eq-if*)
  } **moreover** {**assume** $*$: *h1 $\neq$ elem*
    then have *List.member T1 elem*
      using *Cons*
      by *simp*
    then have *?case* **using** *Cons WEST-and-subset* **by** *blast*
  }
  ultimately show *?case* **by** *blast*
qed

lemma *WEST-and-trace-element-of-WEST-and-helper*:
  assumes *List.member L2 elem2*

assumes (*WEST-and-trace elem1 elem2*) = *Some sometrace*
  shows *sometrace* ∈ *set* (*WEST-and-helper elem1 L2*)
  using *assms*
proof (*induct L2*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons h2 T2*)
  {**assume** ∗: *elem2* = *h2*
    then have *?case*
      using *WEST-and-helper.simps*(*2*)[*of elem1 h2 t2*]
      using *assms*(*2*) **by** *fastforce*
  } **moreover** {**assume** ∗: *elem2* ≠ *h2*
    then have *List.member T2 elem2* **using** *Cons*(*2*)
      by *simp*
    then have *?case* **using** *Cons*(*1*, *3*) *WEST-and-helper-subset*
      by *blast*
}
  ultimately show *?case* **by** *blast*
qed

lemma *index-of-L-in-L*:
  assumes *i* < *length L*
  shows *List.member L* (*L* ! *i*)
  using *assms* **by** *force*

lemma *WEST-and-indices*:
  fixes *L1 L2*::*WEST-regex*
  fixes *sometrace*::*trace-regex*
  assumes ∃*i1 i2*. *i1* < *length L1* ∧ *i2* < *length L2* ∧ *WEST-and-trace* (*L1* ! *i1*)
(*L2* ! *i2*) = *Some sometrace*
  shows ∃*i*<*length* (*WEST-and L1 L2*). *WEST-and L1 L2* ! *i* = *sometrace*
proof−
 obtain *i1 i2* **where** *i1-e2-prop*: *i1* < *length L1* ∧ *i2* < *length L2* ∧ *WEST-and-trace*
(*L1* ! *i1*) (*L2* ! *i2*) = *Some sometrace*
    using *assms* **by** *blast*

  then have *elem*: *List.member L1* (*L1* ! *i1*)
    using *index-of-L-in-L i1-e2-prop* **by** *blast*
  have *elem2*: *List.member L2* (*L2* ! *i2*)
    using *index-of-L-in-L i1-e2-prop* **by** *blast*

  let *?L* = *WEST-and L1 L2*
  have *L1-nonempty*: *L1* ≠ []
    using *i1-e2-prop* **by** *fastforce*
  have *L2-nonempty*: *L2* ≠ []
    using *i1-e2-prop* **by** *fastforce*

obtain *h1 t1* **where** *h1t1*: *L1 = h1#t1* **using** *L1-nonempty* **using** *list.exhaust*
by *blast*
  obtain *h2 t2* **where** *h2t2*: *L2 = h2#t2* **using** *L2-nonempty* **using** *list.exhaust*
by *blast*

  then have *set-subset*: *set* (*WEST-and-helper* (*L1 ! i1*) *L2*) ⊆ *set* (*WEST-and*
L1 L2)
    using *h2t2 WEST-and-helper-subset-of-WEST-and*[*of L1* (*L1 ! i1*) *h2 t2*] *elem*
    by *blast*

  have *sometrace-in*: *sometrace* ∈ *set* (*WEST-and-helper* (*L1 ! i1*) *L2*)
    using *WEST-and-trace-element-of-WEST-and-helper*[*OF elem2*, *of* (*L1 ! i1*)
sometrace]
    i1-e2-prop **by** *blast*

  show *?thesis* **using** *set-subset sometrace-in*
    by (*simp add*: *in-set-conv-nth subset-code*(*1*))
qed

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* ∧ *match π L2*
  shows *match π* (*WEST-and L1 L2*)
proof−
  have *L1-nonempty*: *L1* ≠ []
    using *assms*(*3*) **unfolding** *match-def* **by** *auto*
  have *L2-nonempty*: *L2* ≠ []
    using *assms*(*3*) **unfolding** *match-def* **by** *auto*

  obtain *i1 i2* **where** *∗*:*i1 < length L1* ∧ *i2 < length L2* ∧ *match-regex π* (*L1!i1*)
∧ *match-regex π* (*L2!i2*)
    using *assms*(*3*) **unfolding** *match-def* **by** *metis*

  let *?r1 = L1!i1*
  let *?r2 = L2!i2*
  have *bounds*: *i1 < length L1* ∧ *i2 < length L2* **using** *∗* **by** *auto*
  have *match-r1r2*: *match-regex π ?r1* ∧ *match-regex π ?r2* **using** *∗* **by** *simp*

  have *r1-nv*: *trace-regex-of-vars* (*L1 ! i1*) *n*
    using *bounds assms*(*1*) **unfolding** *WEST-regex-of-vars-def* **by** *metis*
  have *r2-nv*: *trace-regex-of-vars* (*L2 ! i2*) *n*
    using *bounds assms*(*2*) **unfolding** *WEST-regex-of-vars-def* **by** *metis*

  have ∃ *sometrace*. *match-regex π sometrace* ∧ (*WEST-and-trace ?r1 ?r2*) = *Some*
sometrace

    using *WEST-and-trace-correct-forward*[*of ?r1 n ?r2 π, OF r1-nv r2-nv match-r1r2*]
by *blast*
 then obtain *sometrace* **where** *sometrace-obt*: *match-regex π sometrace ∧ (WEST-and-trace ?r1 ?r2) = Some sometrace*
    by *auto*

 have ∃ *i1 i2*.
   i1 < length L1 ∧
   i2 < length L2 ∧ WEST-and-trace (L1 ! i1) (L2 ! i2) = Some sometrace
   using *bounds sometrace-obt* **by** *blast*
 then have ∃ *i < length (WEST-and L1 L2). (WEST-and L1 L2)!i = sometrace*
   using *WEST-and-indices*[*of L1 L2 sometrace*]
   using *sometrace-obt* **by** *force*

 then obtain *i* **where** *sometrace-index*: *i < length (WEST-and L1 L2) ∧ (WEST-and L1 L2)!i = sometrace*
   by *blast*
 have *sometrace-match*: *match-regex π sometrace* **using** *sometrace-obt* **by** *auto*
 have ∃ *i<length (WEST-and L1 L2). match-regex π (WEST-and L1 L2 ! i)*
   using *sometrace-index sometrace-match* **by** *blast*
 then show *?thesis*
   unfolding *match-def* **by** *simp*
qed

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−
 have ∃ *i<length (WEST-and L1 L2). match-regex π ((WEST-and L1 L2) ! i)*
   using *assms* **unfolding** *match-def* **by** *argo*
 then obtain *i* **where** *i-obt*: *i<length (WEST-and L1 L2) ∧*
              match-regex π ((WEST-and L1 L2) ! i) **by** *auto*
  then obtain *i1 i2* **where** *i1i2*: *i1 < length L1 ∧ i2 < length L2 ∧ Some ((WEST-and L1 L2)!i) = WEST-and-trace (L1!i1) (L2!i2)*
   using *WEST-and.simps WEST-and-helper.simps*
   by (*metis L1-of-num-vars L2-of-num-vars WEST-and-set-member nth-mem*)

 have *i1-L1*: *i1 < length L1* **using** *i1i2* **by** *auto*
 have *i2-L2*: *i2 < length L2* **using** *i1i2* **by** *auto*

 let *?r1 = L1!i1*
 let *?r2 = L2!i2*
 let *?r = WEST-and L1 L2 ! i*

have *r1-of-nv*: *trace-regex-of-vars* (*L1* ! *i1*) *n* **using** *assms(1)* *i1-L1*
  unfolding *WEST-regex-of-vars-def* **by** *metis*
have *r2-of-nv*: *trace-regex-of-vars* (*L2* ! *i2*) *n* **using** *assms(2)* *i2-L2*
  unfolding *WEST-regex-of-vars-def* **by** *metis*

have *match-regex* π *?r*
  using *WEST-and-trace-correct-converse*[*of ?r1 n ?r2* π, *OF r1-of-nv r2-of-nv*]
  using *i-obt i1i2* **by** *auto*
then have *match-regex* π (*WEST-and L1 L2* ! *i*) **unfolding** *match-def* **by** *simp*
then have *match-r1r2*: (*match-regex* π (*L1* ! *i1*) $\wedge$ *match-regex* π (*L2* ! *i2*))
  using *WEST-and-trace-correct-converse*[*of ?r1 n ?r2* π, *OF r1-of-nv r2-of-nv*]
  using *i1i2 i-obt* **by** *force*
then have $\exists$ *i<length* [*L1* ! *i1*]. *match-regex* π ([*L1* ! *i1*] ! *i*) **unfolding** *match-def*
by *auto*
then have $\exists$ *i<1*. *match-regex* π ([*L1* ! *i1*] ! *i*) **unfolding** *match-def* **by** *auto*
then have *match-regex* π (*L1* ! *i1*) **by** *simp*
then show *?thesis* **using** *i1-L1*
  unfolding *match-def* **by** *auto*
qed

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* $\wedge$ *match* π *L2*
proof $-$
 show *?thesis* **using** *WEST-and-correct-converse-L1 WEST-and-commutative assms*
   by (*meson regex-equiv-def*)
qed

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* $\wedge$ *match* π *L2* $\longleftrightarrow$ *match* π (*WEST-and L1 L2*)
proof $-$
 show *?thesis* **using** *WEST-and-correct-forward WEST-and-correct-converse assms*
   by *blast*
qed

3.4   Facts about the WEST or operator

lemma *WEST-or-correct*:

59

fixes π::*trace*
fixes *L1 L2*::*WEST-regex*
shows *match π* (*L1@L2*) ⟷ (*match π L1*) ∨ (*match π L2*)
proof−
  have *forward*: *match π* (*L1@L2*) ⟶ (*match π L1*) ∨ (*match π L2*)
    unfolding *match-def*
   by (*metis add-diff-inverse-nat length-append nat-add-left-cancel-less nth-append*)

  have *converse*: (*match π L1*) ∨ (*match π L2*) ⟶ *match π* (*L1@L2*)
    unfolding *match-def* **by** (*metis list-ex-append list-ex-length*)
  show *?thesis*
    using *forward converse* **by** *blast*
qed

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−
  have *drop-a*: (*drop a* ((*arbitrary-trace num-vars a*)@*L*)) = *L*
    using *arbitrary-trace.simps*[*of num-vars a*] **by** *simp*
  let *?padL* = (*arbitrary-trace num-vars a*)@*L*
  have *length* (*arbitrary-trace num-vars a @ L*) = *a* + (*length L*)
    by *auto*
  then have *match-all*: ∀ *time*<*a*+(*length L*). *match-timestep* (π ! *time*) (*?padL* ! *time*)
    using *assms*(*2*) *arbitrary-trace.simps*[*of num-vars a*]
    unfolding *match-regex-def* **by** *metis*

  have *len-xi*: *length π* ≥ *a* + (*length L*)
    using *assms*(*2*) *arbitrary-trace.simps*[*of num-vars a*]
    unfolding *match-regex-def*
    using ‹*length* (*arbitrary-trace num-vars a @ L*) = *a* + *length L*› **by** *argo*

  then have *match-drop-a*: *match-timestep* (*drop a π* ! *time*) (*L* ! *time*)
    if *time-le*: *time* < *length L* **for** *time*
  proof−
    have *time* + *a* < *a* + (*length L*) **using** *time-le* **by** *simp*
    then have *fact1*: *match-timestep* (π ! (*time* + *a*)) (*?padL* ! (*time* + *a*))
      using *match-all* **by** *blast*
    have *fact2*: (π ! (*time* + *a*)) = (*drop a π* ! *time*)
      using *time-le len-xi*
      by (*simp add*: *add.commute*)
    have *fact3*: (*?padL* ! (*time* + *a*)) = (*L* ! *time*)
      using *time-le len-xi*
    by (*metis* ‹*length* (*arbitrary-trace num-vars a @ L*) = *a* + *length L*› *add.commute*
drop-a le-add1 nth-drop)

   show *?thesis*
    using *fact1 fact2 fact3* **by** *argo*
  qed

  have *len-L-drop-a*: *length L ≤ length* (*drop a π*)
   using *assms*(*2*) **unfolding** *match-regex-def*
   by (*metis assms*(*1*) *diff-add drop-a drop-drop drop-eq-Nil length-drop*)
  then have *match-regex* (*drop a π*) *L* **unfolding** *match-regex-def*
   using *match-drop-a* **by** *metis*
  then show *?thesis* **using** *drop-a assms* **by** *argo*
qed

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 −
  have *time-h*: ∀ *time<length* (*L1 @ L2*). *match-timestep* (*π ! time*) ((*L1 @ L2*) !
time)
   using *assms* **unfolding** *match-regex-def* **by** *argo*
  then have *time*: *match-timestep* (*drop a π ! time*) ((*drop a* (*L1 @ L2*)) ! *time*)
if *time-lt*: *time<length* (*drop a* (*L1 @ L2*)) **for** *time*
  proof −
   have *time + a < length* (*L1@L2*)
    using *time-lt assms*(*2*) **by** *auto*
   then have *h0*: *match-timestep* (*π ! (time + a*)) ((*L1 @ L2*) ! (*time + a*))
    using *time-h* **by** *blast*
   have *h1*: *π ! (time + a*) = (*drop a π*) ! *time*
    using *assms*(*1*)
    by (*simp add*: *add.commute*)
   have *h2*: ((*L1 @ L2*) ! (*time + a*)) = (*drop a* (*L1 @ L2*)) ! *time*
    using *assms*(*2*)
    by (*metis add.commute append-eq-conv-conj nth-append-length-plus*)
   then show *?thesis* **using** *assms h0 h1 h2* **by** *simp*
  qed
  have *len-h*: *length* (*L1 @ L2*) *≤ length π*
   using *assms* **unfolding** *match-regex-def* **by** *argo*
  then have *len*: *length* (*drop a* (*L1 @ L2*)) *≤ length* (*drop a π*)
   using *assms*(*1−2*) **by** *auto*
  show *?thesis*
   using *len time* **unfolding** *match-regex-def*
   by *argo*
qed

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−
  have *length* (*drop a* (*L1 @ L2*)) = *length L2*
    using *arbitrary-trace.simps*[*of num-vars a*] *assms* **by** *simp*
  then have *match-L2*: ⋀*time. time<length L2* ⟹ *match-timestep* ((*drop a π*) !
time) (*L2 ! time*)
  proof−
    fix *time*
    assume *time-lt*: *time<length L2*
    then have *time-lt-dropa-L1L2*: *time < length* (*drop a* (*L1 @ L2*))
      using *assms*(*2*) *arbitrary-trace.simps*[*of num-vars a*] **by** *auto*
    have ∀ *time<length* (*drop a* (*L1 @ L2*)). *match-timestep* (*drop a π ! time*) (*drop
a* (*L1 @ L2*) ! *time*)
      using *assms* **unfolding** *match-regex-def* **by** *metis*
    then have *match-timestep* (*drop a π ! time*) (*drop a* (*L1 @ L2*) ! *time*)
      using *time-lt-dropa-L1L2* **by** *blast*
    then show *match-timestep* (*drop a π ! time*) (*L2 ! time*)
      using *assms*(*2*) *arbitrary-trace.simps*[*of num-vars a*] **by** *simp*
  qed
  have *match-L1L2*: *match-timestep* (*π ! time*) ((*L1 @ L2*) ! *time*) **if** *time-le-L1L2*:
time<length (*L1 @ L2*) **for** *time*
  proof−
    {assume *time-le-L1*: *time < length L1*
      {assume *L1-empty*: *L1 =* []
      have *match-timestep* (*π ! time*) (*L2 ! time*)
        using *assms* **unfolding** *match-regex-def arbitrary-trace.simps*
        using *L1-empty time-le-L1* **by** *auto*
      then have *?thesis* **using** *L1-empty* **by** *simp*
      } moreover {
      assume *L1-nonempty*: *L1 ≠* []
      have *L1-arb*: (*L1!time*) = *arbitrary-state num-vars*
        using *assms* **unfolding** *arbitrary-trace.simps time-le-L1*
        using *time-le-L1* **by** *auto*

      have *match-timestep* (*π ! time*) (*arbitrary-state num-vars*)
        unfolding *arbitrary-state.simps match-timestep-def* **by** *auto*
      then have *match-L1*: *match-timestep* (*π ! time*) (*L1!time*)
        using *L1-arb* **by** *auto*

      have (*L1 @ L2*) ! *time = L1!time*
        using *time-le-L1L2 time-le-L1 L1-nonempty* **by** (*meson nth-append*)
      then have *?thesis* **using** *match-L1* **by** *auto*
      }
      ultimately have *?thesis* **by** *blast*
    } moreover {
    assume *time-geq-L1*: *time ≥ length L1*
    then have *time-minus-a-le-L2*: *time − a < length L2*
      using *assms*(*2*) *time-le-L1L2* **unfolding** *arbitrary-trace.simps* **by** *simp*

then have *match-time-minus-a*: *match-timestep* $((drop\ a\ \pi)\ !\ (time - a))$
$(L2\ !\ (time - a))$
      using *match-L2* **by** *blast*

   have *length* $(drop\ a\ (L1\ @\ L2)) \leq length\ (drop\ a\ \pi)$
     using *assms* **unfolding** *match-regex-def* **by** *metis*
   then have *L2-le-dropa-xi*: *length* $L2 \leq length\ (drop\ a\ \pi)$
     using *assms* **unfolding** *arbitrary-trace.simps* **by** *simp*
   then have *fact1-h1*: *length* $L2 \leq length\ \pi - a$ **by** *auto*
   have *fact1-h2*: *length* $L1 \leq time$ **using** *time-geq-L1* **by** *blast*
   have *fact1-h3*: $time - a < length\ L2$ **using** *time-minus-a-le-L2* **by** *auto*
   have *fact1-h4*: $time < length\ L1 + length\ L2$ **using** *time-le-L1L2* **by** *simp*
   have *length* $L2 \leq length\ \pi - a \Longrightarrow$
        length $L1 \leq time \Longrightarrow$
        $time - a < length\ L2 \Longrightarrow$
        $time < length\ L1 + length\ L2 \Longrightarrow \pi\ !\ (a + (time - a)) = \pi\ !\ time$
     using *fact1-h1 fact1-h2 fact1-h3 fact1-h4 time-geq-L1 assms*
     unfolding *arbitrary-trace.simps* **by** *simp*
   then have *fact1*: *drop* $a\ \pi\ !\ (time - a) = \pi\ !\ time$
     using *time-geq-L1 time-minus-a-le-L2 time-le-L1L2 L2-le-dropa-xi* **by** *simp*

   have *L1-a*: *length* $L1 = a$ **using** *assms* **unfolding** *arbitrary-trace.simps* **by**
auto
   then have *fact2*: $L2\ !\ (time - a) = (L1\ @\ L2)\ !\ time$
     using *fact1-h2 fact1-h3 fact1-h4 time-geq-L1*
     by $(metis\ le\text{-}add\text{-}diff\text{-}inverse\ nth\text{-}append\text{-}length\text{-}plus)$

   have *?thesis* **using** *fact1 fact2 match-time-minus-a* **by** *auto*
   }
  ultimately show *?thesis* **by** *force*
 qed
 have *length* $(drop\ a\ (L1\ @\ L2)) \leq length\ (drop\ a\ \pi)$
   using $assms(2)\ arbitrary\text{-}trace.simps[of\ num\text{-}vars\ num\text{-}pad]$
   by $(metis\ assms(3)\ match\text{-}regex\text{-}def)$
 then have *length* $(L1\ @\ L2) \leq length\ \pi$
   using *assms* **unfolding** *match-regex-def* **by** *simp*
 then show *?thesis* **using** *match-L1L2* **unfolding** *match-regex-def* **by** *simp*
qed

lemma *shift-match*:
 assumes *length* $\pi \geq a$
 assumes *match* $\pi\ (shift\ L\ num\text{-}vars\ a)$
 shows *match* $(drop\ a\ \pi)\ L$
proof −
 obtain *i* **where** *i-obt*: $i < length\ (shift\ L\ num\text{-}vars\ a) \wedge match\text{-}regex\ \pi\ (shift\ L$
$num\text{-}vars\ a\ !\ i)$
   using *assms* **unfolding** *match-def* **by** *force*
 have $(shift\ L\ num\text{-}vars\ a\ !\ i) = (arbitrary\text{-}trace\ num\text{-}vars\ a) @ (L!i)$

63

    using *shift.simps*
    using ‹*i < length* (*shift L num-vars a*) ∧ *match-regex* π (*shift L num-vars a !*
i)› **by** *auto*

  then have *match*: *match-regex* π ((*arbitrary-trace num-vars a*)@(*L*!*i*))
    using *i-obt* **by** *argo*

  have *len-at*: *length* (*arbitrary-trace num-vars a*) = *a*
    unfolding *arbitrary-trace.simps* **by** *simp*

  have *drop-a*: (*drop a* (*arbitrary-trace num-vars a*)@(*L*!*i*)) = *L*!*i*
    using *arbitrary-trace.simps*[*of num-vars a*] **by** *simp*

  then have *match-regex* (*drop a* π) (*drop a* (*arbitrary-trace num-vars a*)@(*L*!*i*))
    using *match* **using** *match-regex*[*OF assms(1) len-at*] **by** *simp*
  then have *match-regex* (*drop a* π) (*L* ! *i*)
    using *drop-a* **by** *argo*
  then show *?thesis* **using** *assms i-obt* **unfolding** *match-def* **by** *auto*
qed

lemma *shift-match-converse*:
  assumes *length* π ≥ *a*
  assumes *match* (*drop a* π) *L*
  shows *match* π (*shift L num-vars a*)
proof −
  obtain *i* **where** *i-obt*: *match-regex* (*drop a* π) (*L*!*i*) ∧ *i < length L*
    using *assms* **unfolding** *match-def* **by** *metis*
  then have *match-padLi*: *match-regex* π ((*arbitrary-trace num-vars a*)@(*L*!*i*))
    using *match-regex-converse assms* **by** *auto*
  have *i-bound*: *i*<*length* (*shift L num-vars a*)
    using *shift.simps i-obt* **by** *auto*
  have (*shift L num-vars a ! i*) = (*arbitrary-trace num-vars a*)@(*L*!*i*)
    unfolding *shift.simps*
    by (*simp add: i-obt*)
  then have ∃ *i*<*length* (*shift L num-vars a*). *match-regex* π (*shift L num-vars a !*
i)
    using *assms match-padLi i-bound* **by** *metis*
  then show *?thesis* **unfolding** *match-def* **by** *argo*
qed

lemma *pad-zero*:
  shows *shift L2 num-vars 0* = *L2*
  unfolding *shift.simps arbitrary-trace.simps*
proof −
  have ∃ *wsss*. *L2* = *wsss* ∧ (@) ([]::*trace-regex*) = (λ*wss*. *wss*) ∧ *L2* = *wsss*
    by *blast*
  then show *map* ((@) (*map* (λ*n*. *arbitrary-state num-vars*) [*0*..<*0*])) *L2* = *L2*
    by *simp*

qed

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*
  using *assms* **unfolding** *trace-regex-of-vars-def*
  by (*simp add*: *nth-append*)

lemma *WEST-num-vars-subformulas*:
  assumes $G \in subformulas\ F$
  shows (*WEST-num-vars F*) $\geq$ *WEST-num-vars G*
  using *assms*
proof (*induct F*)
  case *True-mltl*
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case *False-mltl*
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Prop-mltl x*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Not-mltl F*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*And-mltl F1 F2*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Or-mltl F1 F2*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Future-mltl F x2 x3a*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Global-mltl F x2 x3a*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Until-mltl F1 F2 x3a x4a*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
next
  case (*Release-mltl F1 F2 x3a x4a*)
  then show *?case* **unfolding** *subformulas.simps* **by** *auto*
qed

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

proof (*induction depth-mltl φ arbitrary*: φ *rule*: *less-induct*)
  case *less*
  then show *?case* **proof** (*cases φ*)
    case *True-mltl*
    then show *?thesis* **by** *auto*
  next
    case *False-mltl*
    then show *?thesis* **by** *auto*
  next
    case (*Prop-mltl x3*)
    then show *?thesis* **by** *auto*
  next
    case (*Not-mltl p*)
    then show *?thesis* **proof** (*induct p*)
      case *True-mltl*
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case *False-mltl*
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case (*Prop-mltl x*)
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case (*Not-mltl p*)
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case (*And-mltl $\varphi 1$ $\varphi 2$*)
      then have *phi-is*: $\varphi = $ *Not-mltl* (*And-mltl $\varphi 1$ $\varphi 2$*)
        using *Not-mltl* **by** *auto*
      have *ind1*: *WEST-num-vars $\varphi 1$ = WEST-num-vars* (*convert-nnf* (*Not-mltl $\varphi 1$*))
        using *less*[*of Not-mltl $\varphi 1$*] *phi-is* **by** *auto*
      have *ind2*: *WEST-num-vars $\varphi 2$ = WEST-num-vars* (*convert-nnf* (*Not-mltl $\varphi 2$*))
        using *less*[*of Not-mltl $\varphi 2$*] *phi-is* **by** *auto*
      then show *?case* **using** *ind1 ind2 phi-is*
        by *auto*
    next
      case (*Or-mltl $\varphi 1$ $\varphi 2$*)
      then have *phi-is*: $\varphi = $ *Not-mltl* (*Or-mltl $\varphi 1$ $\varphi 2$*)
        using *Not-mltl* **by** *auto*
      have *ind1*: *WEST-num-vars $\varphi 1$ = WEST-num-vars* (*convert-nnf* (*Not-mltl $\varphi 1$*))
        using *less*[*of Not-mltl $\varphi 1$*] *phi-is* **by** *auto*
      have *ind2*: *WEST-num-vars $\varphi 2$ = WEST-num-vars* (*convert-nnf* (*Not-mltl $\varphi 2$*))
        using *less*[*of Not-mltl $\varphi 2$*] *phi-is* **by** *auto*
      then show *?case* **using** *ind1 ind2 phi-is*
        by *auto*

next

　case (*Future-mltl a b φ1*)

　then have *phi-is*: $\varphi = Not\text{-}mltl$ (*Future-mltl a b φ1*)

　　using *Not-mltl*

　　by *auto*

　　have *ind1*: *WEST-num-vars* $\varphi = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ1*))

　　　using *less*[*of Not-mltl φ1*] *phi-is* **by** *auto*

　　then show *?case* **using** *ind1 phi-is*

　　　by *auto*

　next

　case (*Global-mltl a b φ1*)

　then have *phi-is*: $\varphi = Not\text{-}mltl$ (*Global-mltl a b φ1*)

　　using *Not-mltl*

　　by *auto*

　　have *ind1*: *WEST-num-vars* $\varphi = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ1*))

　　　using *less*[*of Not-mltl φ1*] *phi-is* **by** *auto*

　　then show *?case* **using** *ind1 phi-is*

　　　by *auto*

　next

　case (*Until-mltl φ1 a b φ2*)

　then have *phi-is*: $\varphi = Not\text{-}mltl$ (*Until-mltl φ1 a b φ2*)

　　using *Not-mltl* **by** *auto*

　　have *ind1*: *WEST-num-vars* $\varphi1 = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ1*))

　　　using *less*[*of Not-mltl φ1*] *phi-is* **by** *auto*

　　have *ind2*: *WEST-num-vars* $\varphi2 = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ2*))

　　　using *less*[*of Not-mltl φ2*] *phi-is* **by** *auto*

　　then show *?case* **using** *ind1 ind2 phi-is*

　　　by *auto*

　next

　case (*Release-mltl φ1 a b φ2*)

　then have *phi-is*: $\varphi = Not\text{-}mltl$ (*Release-mltl φ1 a b φ2*)

　　using *Not-mltl* **by** *auto*

　　have *ind1*: *WEST-num-vars* $\varphi1 = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ1*))

　　　using *less*[*of Not-mltl φ1*] *phi-is* **by** *auto*

　　have *ind2*: *WEST-num-vars* $\varphi2 = WEST\text{-}num\text{-}vars$ (*convert-nnf* (*Not-mltl φ2*))

　　　using *less*[*of Not-mltl φ2*] *phi-is* **by** *auto*

　　then show *?case* **using** *ind1 ind2 phi-is*

　　　by *auto*

　qed

next

　case (*And-mltl φ1 φ2*)

　then show *?thesis* **using** *less* **by** *auto*

next

case (*Or-mltl φ1 φ2*)
then show *?thesis* **using** *less* **by** *auto*
next
  case (*Future-mltl a b φ*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Global-mltl a b φ*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Until-mltl φ1 a b φ2*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Release-mltl φ1 a b φ2*)
  then show *?thesis* **using** *less* **by** *auto*
qed
qed

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*
  using *assms*
proof (*induct length state1 arbitrary*: *state1 state2 k state rule*: *less-induct*)
  case *less*
  {assume ∗: *k = 0*
    then have *?case* **using** *less(2−3) less(4) WEST-and-state.simps(1)*
      by *auto*
  } moreover {assume ∗: *k > 0*
    obtain *h1 t1* **where** *h1t1*: *state1 = h1#t1*
      using ∗ *less(2)*
      using *list.exhaust* **by** *auto*
    obtain *h2 t2* **where** *h2t2*: *state2 = h2#t2*
      using ∗ *less(3)*
      using *list.exhaust* **by** *auto*
    have *WEST-and-bitwise h1 h2 ≠ None*
        by (*metis WEST-and-state.simps(2) h1t1 h2t2 less.prems(3) option.discI*
option.simps(4))
    then obtain *h* **where** *someh*: *WEST-and-bitwise h1 h2 = Some h*
      by *blast*
    have *WEST-and-state t1 t2 ≠ None*
      by (*metis* (*no-types, lifting*) *WEST-and-state.simps(2) h1t1 h2t2 less.prems(3)*
option.case-eq-if option.discI)
    then obtain *t* **where** *somet*: *WEST-and-state t1 t2 = Some t*
      by *blast*
    then have *length t = k−1*
      using *less(1)[of t1 k−1 t2] h1t1 h2t2*

by (*metis WEST-and-state-difflengths-is-none diff-Suc-1 length-Cons less.prems*(*1*)
lessI option.distinct(*1*))
   then have *?case* **using** *less WEST-and-state.simps*(*2*)[*of h1 t1 h2 t2*]
    using *someh somet*
    by (*metis WEST-and-state-length option.discI option.inject*)
  }
  ultimately show *?case*
   by *auto*
qed

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*
  using *assms*
proof(*induct r1 arbitrary*: *r2 sometrace*)
  case *Nil*
  then have *sometrace = r2*
   using *WEST-and-trace.simps*(*2*)
  by (*metis WEST-and-trace.simps*(*1*) *WEST-and-trace-commutative option.inject*)
  then show *?case* **using** *Nil* **unfolding** *trace-regex-of-vars-def* **by** *blast*
next
  case (*Cons h1 t1*)
  {assume *r2-empty*: *r2 = []*
   then have *sometrace = (h1#t1)*
    using *WEST-and-trace.simps WEST-and-trace-commutative*(*1*) *Cons.prems*
by *auto*
   then have *?case* **using** *Cons*
    unfolding *trace-regex-of-vars-def* **by** *blast*
  } moreover {
   assume *r2-nonempty*: *r2 ≠ []*
   then obtain *h2 t2* **where** *h2t2*: *r2 = h2#t2*
    by (*meson trim-reversed-regex.cases*)
   {assume *sometrace-empty*: *sometrace = []*
    then have *?case* **unfolding** *trace-regex-of-vars-def* **by** *simp*
   } moreover {
   assume *sometrace-nonempty*: *sometrace ≠ []*
   then obtain *h t* **where** *ht-obt*: *WEST-and-state h1 h2 = Some h ∧ WEST-and-trace*
t1 t2 = Some t
    using *WEST-and-trace-nonempty-args*[*of h1 t1 h2 t2*] *Cons.prems*(*3*)
    by (*metis ‹r2 = h2 # t2› trim-reversed-regex.cases*)
   then have *sometrace-ht*: *sometrace = h#t*
    using *Cons.prems*(*3*) **unfolding** *h2t2* **by** *auto*

   have *h1t1-nv*: *∀ i<length (h1 # t1). length ((h1 # t1) ! i) = k*
    using *Cons.prems* **unfolding** *trace-regex-of-vars-def* **by** *argo*
   have *h1-nv*: *length h1 = k*
    using *h1t1-nv* **by** *auto*

have *t1-nv*: *trace-regex-of-vars t1 k*
  using *h1t1-nv* **unfolding** *trace-regex-of-vars-def* **by** *auto*
have *h2t2-nv*: $\forall i$<*length (h2 # t2). length ((h2 # t2) ! i) = k*
  using *Cons.prems h2t2* **unfolding** *trace-regex-of-vars-def* **by** *metis*
have *h2-nv*: *length h2 = k*
  using *h2t2-nv* **by** *auto*
have *t2-nv*: *trace-regex-of-vars t2 k*
  using *h2t2-nv* **unfolding** *trace-regex-of-vars-def* **by** *auto*

have *h1h2-h*: *WEST-and-state h1 h2 = Some h*
  using *ht-obt* **by** *simp*
then have *h-nv*: *length h = k* **using** *h1-nv h2-nv*
  using *length-WEST-and* **by** *blast*

have *t1t2-t*: *WEST-and-trace t1 t2 = Some t*
  using *ht-obt* **by** *simp*
then have *t-nv*: *trace-regex-of-vars t k*
  using *Cons.hyps*[*of t2 t, OF t1-nv t2-nv*] **by** *blast*

have *t-nv-unfold*: $\forall i$<*length t. length (t ! i) = k*
using *h-nv t-nv sometrace-ht* **unfolding** *trace-regex-of-vars-def* **by** *presburger*

then have *length (sometrace ! i) = k* **if** *i-lt*: *i*<*length sometrace* **for** *i*
  using *i-lt sometrace-ht h-nv*
proof−
  {**assume** ∗: *i = 0*
    then have *?thesis*
      using *sometrace-ht h-nv* **by** *auto*
  } **moreover** {**assume** ∗: *i > 0*
    then have *sometrace ! i = t ! (i−1)*
      using *i-lt sometrace-ht* **by** *simp*

    then have *?thesis*
      using *t-nv-unfold i-lt sometrace-ht*
      by (*metis* ∗ *One-nat-def Suc-less-eq Suc-pred length-Cons*)
  }
  ultimately show *?thesis* **by** *auto*
qed
then have *?case* **unfolding** *trace-regex-of-vars-def* **by** *auto*
}
ultimately have *?case* **by** *blast*
}
ultimately show *?case* **by** *blast*
qed

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−
  {**assume** *L1L2-empty*: (*WEST-and L1 L2*) = []
    then have *?thesis* **unfolding** *WEST-regex-of-vars-def* **by** *simp*
  } **moreover** {
    assume *L1L2-nonempty*: *WEST-and L1 L2* $\neq$ []

      have *trace-regex-of-vars* (*WEST-and L1 L2* ! *i*) *k* **if** *i-index*: *i* < *length*
(*WEST-and L1 L2*) **for** *i*
      proof−
        obtain *sometrace* **where** *sometrace-obt*: (*WEST-and L1 L2*)!*i* = *sometrace*
          using *L1L2-nonempty* **by** *simp*
        then obtain *i1 i2* **where** *i1i2-obt*: *i1* < *length L1* $\wedge$ *i2* < *length L2* $\wedge$ *Some*
sometrace = *WEST-and-trace* (*L1!i1*) (*L2!i2*)
          using *WEST-and.simps WEST-and-helper.simps*
          by (*metis WEST-and-set-member-dir1 assms*(*1*) *assms*(*2*) *i-index nth-mem*)

        let *?r1* = *L1!i1*
        let *?r2* = *L2!i2*
        have *r1r2-sometrace*: *Some sometrace* = *WEST-and-trace* (*L1!i1*) (*L2!i2*)
          using *i1i2-obt* **by** *blast*
        have *r1-nv*: *trace-regex-of-vars ?r1 k*
          using *assms i1i2-obt* **unfolding** *WEST-regex-of-vars-def* **by** *metis*
        have *r2-nv*: *trace-regex-of-vars ?r2 k*
          using *assms i1i2-obt* **unfolding** *WEST-regex-of-vars-def* **by** *metis*
        have *trace-regex-of-vars sometrace k*
          using *r1-nv r2-nv r1r2-sometrace WEST-and-trace-num-vars*[*of ?r1 k ?r2*]
by *metis*
        then show *?thesis*
          using *sometrace-obt* **by** *blast*
      qed
      then have *?thesis* **unfolding** *WEST-regex-of-vars-def* **by** *simp*
  }
  ultimately show *?thesis* **by** *blast*
qed

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−
  let *?L* = *L1@L2*
  have *trace-regex-of-vars* (*?L!i*) *k* **if** *i-lt*: *i* < *length ?L* **for** *i*
  proof−
    {**assume** *in-L1*: *i* < *length L1*
      then have *L1-i-nv*: *trace-regex-of-vars* (*L1!i*) *k*
        using *L1-nv* **unfolding** *WEST-regex-of-vars-def* **by** *metis*
      have *?L!i* = *L1!i*

using *in-L1*
           by (*simp add*: *nth-append*)
         then have *?thesis* **using** *L1-i-nv* **by** *simp*
       } moreover {
         assume *in-L2*: $i \geq length\ L1$
         then have $i - length\ L1 < length\ L2$
           using *i-lt* **by** *auto*
         then have *L2-i-nv*: *trace-regex-of-vars* $(L2!(i - length\ L1))\ k$
           using *L2-nv* **unfolding** *WEST-regex-of-vars-def* **by** *metis*
         have $(?L\ !\ i) = L2!(i - length\ L1)$
           using *in-L2*
           by (*simp add*: *nth-append*)
         then have *?thesis* **using** *L2-i-nv* **by** *simp*
       }
       ultimately show *?thesis* **by** *fastforce*
     qed

     then show *?thesis* **unfolding** *WEST-regex-of-vars-def* **by** *simp*
  qed

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 −
  let *?H* = $[h]$
  have *WEST-regex-of-vars ?H num-vars*
    using *assms* **unfolding** *WEST-regex-of-vars-def* **by** *auto*
  then have *WEST-regex-of-vars* $(?H@T)$ *num-vars*
    using *WEST-or-num-vars*[*of ?H num-vars T*] *assms* **by** *simp*
  then show *?thesis* **by** *simp*
qed

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*
  using *assms WEST-simp-state.simps* **by** *auto*

lemma *WEST-get-state-length*:
  assumes *trace-regex-of-vars r num-vars*
  shows *length* (*WEST-get-state r k num-vars*) = *num-vars*
  using *assms* **unfolding** *trace-regex-of-vars-def*
  using *WEST-get-state.simps*[*of r k num-vars*]
  by (*metis leI length-map length-upt minus-nat.diff-0*)

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*
  using *WEST-simp-state-num-vars assms*
  unfolding *WEST-simp-trace.simps trace-regex-of-vars-def*
  using *WEST-get-state-length assms*(*1*) **by** *auto*

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−
  have *length* (*take i L @ drop* (*i + 1*) *L*) = *length L*−*1*
    using *assms* **by** *simp*
  have *take-nv*: *WEST-regex-of-vars* (*take i L*) *num-vars*
    using *assms* **unfolding** *WEST-regex-of-vars-def*
    by (*metis in-set-conv-nth in-set-takeD*)
  have *drop-nv*: *WEST-regex-of-vars* (*drop* (*i + 1*) *L*) *num-vars*
    using *assms* **unfolding** *WEST-regex-of-vars-def*
    by (*metis add.commute length-drop less-diff-conv less-iff-succ-less-eq nth-drop*)
  then show *?thesis*
    using *take-nv drop-nv WEST-or-num-vars* **by** *simp*
qed

lemma *update-L-length*:
  assumes *h* ∈ *set* (*enum-pairs L*)
  shows *length* (*update-L L h num-var*) = *length L* − *1*
proof−
  have *length L* ≤ *1* ⟶ *enum-pairs L* = [ ]
    unfolding *enum-pairs.simps* **using** *enumerate-pairs.simps*
    by (*simp add: upt-rec*)
  then have *len-L*: *length L* ≥ *2*
    using *assms* **by** *auto*
  let *?i* = *fst h*
  let *?j* = *snd h*
  have *i-le-j*: *?i < ?j* **using** *enum-pairs-fact assms*(*1*)
    by *metis*
  have *j-bound*: *?j < length L*
    using *assms*(*1*) *enum-pairs-bound*[*of L*]
    by *metis*
  then have *i-bound*: *?i <* (*length L*)−*1*
    using *i-le-j* **by** *auto*

  have *len-orsimp*: *length* [*WEST-simp-trace* (*L ! fst h*) (*L ! snd h*) *num-var*] = *1*
    by *simp*
  have *length* (*remove-element-at-index* (*snd h*) *L*) = *length L* − *1*
    using *assms j-bound* **by** *auto*

then have *length* (*remove-element-at-index* (*fst h*) (*remove-element-at-index* (*snd h*) *L*)) = *length L* − *2*
  using *assms i-bound j-bound* **by** *simp*
 then show *?thesis*
   using *len-orsimp len-L*
  using *length-append*[*of* (*remove-element-at-index* (*fst h*) (*remove-element-at-index* (*snd h*) *L*)) [*WEST-simp-trace* (*L* ! *fst h*) (*L* ! *snd h*) *num-var*]]
   unfolding *update-L.simps* **by** *linarith*
qed

lemma *update-L-preserves-num-vars*:
 assumes *WEST-regex-of-vars L num-var*
 assumes *h* ∈ *set* (*enum-pairs L*)
 assumes *K* = *update-L L h num-var*
 shows *WEST-regex-of-vars K num-var*
proof−
  have *simp-nv*: *trace-regex-of-vars* (*WEST-simp-trace* (*L* ! *fst h*) (*L* ! *snd h*) *num-var*) *num-var*
   using *WEST-simp-trace-num-vars assms* **unfolding** *WEST-regex-of-vars-def*
   by (*metis enum-pairs-bound enum-pairs-fact order.strict-trans*)
  then have *simp-nv*: *WEST-regex-of-vars* [*WEST-simp-trace* (*L* ! *fst h*) (*L* ! *snd h*) *num-var*] *num-var*
   unfolding *WEST-regex-of-vars-def* **by** *auto*
 have *∗*:*WEST-regex-of-vars* (*remove-element-at-index* (*snd h*) *L*) *num-var*
   using *assms remove-element-at-index-preserves-nv*
   using *enum-pairs-fact*[*of L*] *enum-pairs-bound*[*of L*]
   using *remove-element-at-index-preserves-nv* **by** *blast*
  let *?La* = (*remove-element-at-index* (*snd h*) *L*)
  have *fst h* < *length* (*remove-element-at-index* (*snd h*) *L*)
   using *enum-pairs-fact*[*of L*] *enum-pairs-bound*[*of L*] *assms*(*2*)
   by *auto*
 then have *WEST-regex-of-vars* (*remove-element-at-index* (*fst h*) (*remove-element-at-index* (*snd h*) *L*)) *num-var*
   using *remove-element-at-index-preserves-nv*[*of fst h ?La num-var*] *∗*
   by *blast*
 then show *?thesis*
   using *simp-nv assms*(*3*) **unfolding** *update-L.simps* **using** *WEST-or-num-vars*
   using *WEST-regex-of-vars-def* **by** *blast*
qed

lemma *WEST-simp-helper-can-simp*:
 assumes *simp-L* = *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
 assumes ∃*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
                check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
                            (*L* ! *snd* (*enum-pairs L* ! *j*))
 assumes *min-j* = *Min* {*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
                check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
                            (*L* ! *snd* (*enum-pairs L* ! *j*))}
 assumes *newL* = *update-L L* (*enum-pairs L* ! *min-j*) *num-vars*

74

assumes *i* < *length* (*enum-pairs L*)
shows *simp-L* = *WEST-simp-helper newL* (*enum-pairs newL*) *0 num-vars*
proof−
  let *?j-set* = {*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
          check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
             (*L* ! *snd* (*enum-pairs L* ! *j*))}
  have *cond1*: *finite ?j-set*
    by *fast*
  have *cond2*: *?j-set* ≠ {}
    using *assms*(*2*) **by** *blast*
  have *min-j* ∈ *?j-set*
    using *Min-in*[*OF cond1 cond2*] *assms*(*3*) **by** *blast*
  then have *min-j-props*: *min-j* < *length* (*enum-pairs L*) ∧ *min-j* ≥ *i*
             ∧ *check-simp* (*L* ! *fst* (*enum-pairs L* ! *min-j*))
                 (*L* ! *snd* (*enum-pairs L* ! *min-j*))
    by *blast*
  have *minimality*: ¬ (*check-simp* (*L* ! *fst* (*enum-pairs L* ! *k*))
                (*L* ! *snd* (*enum-pairs L* ! *k*)))
    if *k-prop*: (*k* < *min-j* ∧ *k* < *length* (*enum-pairs L*) ∧ *k* ≥ *i*)
    for *k*
  proof−
    have *k* ∉ *?j-set*
      using *assms*(*3*) *Min-gr-iff*[*of ?j-set k*] *k-prop*
     by (*metis* (*no-types*, *lifting*) *empty-iff finite-nat-set-iff-bounded mem-Collect-eq*
order-less-imp-not-eq2)
    then show *?thesis* **using** *k-prop* **by** *blast*
  qed
  then have *minimality*: ∀ *k*. (*k* < *min-j* ∧ *k* < *length* (*enum-pairs L*) ∧ *k* ≥ *i*)
⟶
             ¬ (*check-simp* (*L* ! *fst* (*enum-pairs L* ! *k*))
                (*L* ! *snd* (*enum-pairs L* ! *k*)))
    by *blast*
  show *?thesis*
    using *assms*(*1*, *4*, *5*) *minimality min-j-props*
  proof(*induction min-j* − *i arbitrary*: *min-j i L simp-L newL*)
    case *0*
    then have *check-simp* (*L* ! *fst* (*enum-pairs L* ! *i*))
    (*L* ! *snd* (*enum-pairs L* ! *i*))
      by *force*
    then show *?case*
      using *0 WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
      by (*metis diff-diff-cancel diff-zero linorder-not-less*)
  next
    case (*Suc x*)
    have *min-j-eq*: *min-j* − *i* = *x+1*
      using *Suc.hyps*(*2*) **by** *auto*
    then have *min-j* > *i*
      by *auto*
    then have *cant-match-i*: ¬ (*check-simp* (*L* ! *fst* (*enum-pairs L* ! *i*))

75

$$(L \; ! \; snd \; (enum\text{-}pairs \; L \; ! \; i)))$$
  using *Suc* **by** *fast*
 let *?simp-L* = *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
 let *?simp-Lnext* = *WEST-simp-helper L* (*enum-pairs L*) (*i+1*) *num-vars*
 let *?newL* = *update-L L* (*enum-pairs L* ! *min-j*) *num-vars*
 have *simp-L-eq*: *?simp-L* = *?simp-Lnext*
  using *cant-match-i WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
Suc.prems(3)
  by *auto*
 have *cond1*: $x = min\text{-}j - (i+1)$
  using *min-j-eq* **by** *auto*
 have *cond2*: *?simp-Lnext* = *WEST-simp-helper L* (*enum-pairs L*) (*i+1*) *num-vars*
  by *simp*
 have *cond3*: *?newL* = *update-L L* (*enum-pairs L* ! *min-j*) *num-vars*
  by *simp*
 have *cond4*: $i + 1 < length$ (*enum-pairs L*)
  using *Suc* **by** *linarith*
 have *cond5*: $\forall k. \; k < min\text{-}j \wedge k < length$ (*enum-pairs L*) $\wedge \; i + 1 \leq k \longrightarrow$
 $\neg$ *check-simp* (*L* ! *fst* (*enum-pairs L* ! *k*))
  (*L* ! *snd* (*enum-pairs L* ! *k*))
  using *Suc*
  using *add-leD1* **by** *blast*
 have *cond6*: $min\text{-}j < length$ (*enum-pairs L*) $\wedge \; i + 1 \leq min\text{-}j \wedge$
    check-simp (*L* ! *fst* (*enum-pairs L* ! *min-j*))
      (*L* ! *snd* (*enum-pairs L* ! *min-j*))
  using *Suc* **by** *linarith*

 have *?simp-Lnext* = *WEST-simp-helper newL* (*enum-pairs newL*) *0 num-vars*
  using *Suc.hyps(1)*[*OF cond1 cond2 cond3 cond4 cond5 cond6*]
  using *Suc.prems* **by** *blast*
 then show *?case*
  using *simp-L-eq Suc.prems(1)* **by** *argo*
 qed
qed

lemma *WEST-simp-helper-cant-simp*:
 assumes *simp-L* = *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
 assumes $\neg(\exists j. \; j < length$ (*enum-pairs L*) $\wedge \; j \geq i \wedge$
    check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
      (*L* ! *snd* (*enum-pairs L* ! *j*)))
 shows *simp-L* = *L*
 using *assms*
proof(*induct length* (*enum-pairs L*) $- \; i$ *arbitrary*: *simp-L L i*)
 case *0*
 then have $i \geq length$ (*enum-pairs L*)
  by *simp*
 then show *?case*
  using *0(2) WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
  by *auto*

next
  case (*Suc x*)
  then have *i-eq*: $i = length$ (*enum-pairs L*) $- (x+1)$
    by *simp*
  let *?simp-L = WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
  let *?simp-nextL = WEST-simp-helper L* (*enum-pairs L*) (*i+1*) *num-vars*
  have *simp-L-eq*: *?simp-L = ?simp-nextL*
    using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
    using *i-eq Suc*
    by (*metis diff-is-0-eq le-refl nat.distinct*(*1*) *zero-less-Suc zero-less-diff*)
  have *cond1*: $x = length$ (*enum-pairs L*) $- (i+1)$
    using *Suc.hyps*(*2*) **by** *auto*
  have *cond2*: *?simp-nextL = WEST-simp-helper L* (*enum-pairs L*) (*i + 1*) *num-vars*
    by *blast*
  have *cond3*: ¬ (∃*j*<*length* (*enum-pairs L*).
      i + 1 ≤ *j* ∧
      check-simp (*L ! fst* (*enum-pairs L ! j*))
      (*L ! snd* (*enum-pairs L ! j*)))
    using *Suc* **by** *auto*
  have *?simp-nextL = L*
    using *Suc.hyps*(*1*)[*OF cond1 cond2 cond3*] **by** *auto*
  then show *?case*
    using *Suc.prems*(*1*) *simp-L-eq* **by** *argo*
qed

lemma *WEST-simp-helper-length*:
  shows *length* (*WEST-simp-helper L* (*enum-pairs L*) *i num-vars*) ≤ *length L*
proof(*induct length L arbitrary*: *L i rule*: *less-induct*)
case *less*
  {**assume** *i-geq*: *length* (*enum-pairs L*) ≤ *i*
    then have *WEST-simp-helper L* (*enum-pairs L*) *i num-vars = L*
      using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
      by *simp*
    then have *?case*
      by *auto*
  } **moreover** {
    assume *i-le*: *length* (*enum-pairs L*) > *i*
    then have *WEST-simp-helper-eq*: *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
=
      (*if check-simp* (*L ! fst* (*enum-pairs L ! i*))
        (*L ! snd* (*enum-pairs L ! i*))
      then let newL = update-L L (*enum-pairs L ! i*) *num-vars*
        in WEST-simp-helper newL (*enum-pairs newL*) *0 num-vars*
      else WEST-simp-helper L (*enum-pairs L*) (*i + 1*) *num-vars*)
      using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
      by *simp*
    let *?simp-L = WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
    {**assume** *can-simp*: ∃*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
          check-simp (*L ! fst* (*enum-pairs L ! j*))

77

$$(L \mathbin{!} snd \; (enum\text{-}pairs \; L \mathbin{!} j))$$
then obtain *min-j* **where** *obt-min-j*: *min-j = Min {j. j < length (enum-pairs L) ∧ j ≥ i ∧*

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

 by *blast*
 let *?newL = update-L L (enum-pairs L ! min-j) num-vars*
 have *?simp-L = WEST-simp-helper ?newL (enum-pairs ?newL) 0 num-vars*
  using *WEST-simp-helper-can-simp[of ?simp-L L i num-vars min-j ?newL]*
  using *obt-min-j can-simp i-le* **by** *blast*
 have *min-j-bounds*: *min-j < length (enum-pairs L) ∧ min-j ≥ i*
  using *can-simp obt-min-j Min-in[of {j. j < length (enum-pairs L) ∧ j ≥ i ∧*

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

 by *fastforce*
 have *length ?newL < length L*
  using *update-L-length[of enum-pairs L ! min-j L num-vars]*
  using *min-j-bounds*
  by (*metis diff-less enum-pairs-bound less-nat-zero-code not-gr-zero nth-mem zero-less-one*)
 then have *?case*
  using *less(1)[of ?newL] less.prems min-j-bounds update-L-preserves-num-vars*
  by (*metis (no-types, lifting) ‹WEST-simp-helper L (enum-pairs L) i num-vars = 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› leD le-trans nat-le-linear*)
 } moreover {
 assume *cant-simp*: ¬(∃j. j < length (enum-pairs L) ∧ j ≥ i ∧

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

 then have *?simp-L = L*
  using *WEST-simp-helper-cant-simp i-le* **by** *blast*
 then have *?case* **by** *simp*
 }
 ultimately have *?case* **using** *WEST-simp-helper-eq* **by** *blast*
 }
 ultimately show *?case*
  using *WEST-simp-helper.simps[of L (enum-pairs L) i num-vars]*
  by *fastforce*
qed

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*
 using *assms*
proof(*induct length L arbitrary*: *L i rule*: *less-induct*)
 case *less*
 {assume *i-geq*: *length (enum-pairs L) ≤ i*

then have *WEST-simp-helper L* (*enum-pairs L*) *i num-vars* = *L*
  using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
  by *simp*
then have *?case*
  using *less* **by** *argo*
} **moreover** {
  assume *i-le*: *length* (*enum-pairs L*) > *i*
then have *WEST-simp-helper-eq*: *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
=

    (*if check-simp* (*L* ! *fst* (*enum-pairs L* ! *i*))
      (*L* ! *snd* (*enum-pairs L* ! *i*))
    then let newL = *update-L L* (*enum-pairs L* ! *i*) *num-vars*
      in WEST-simp-helper newL (*enum-pairs newL*) *0 num-vars*
    else WEST-simp-helper L (*enum-pairs L*) (*i* + *1*) *num-vars*)
  using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
  by *simp*
let *?simp-L* = *WEST-simp-helper L* (*enum-pairs L*) *i num-vars*
{**assume** *can-simp*: ∃*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
      check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
        (*L* ! *snd* (*enum-pairs L* ! *j*))
  then obtain *min-j* **where** *obt-min-j*: *min-j* = *Min* {*j*. *j* < *length* (*enum-pairs*
L) ∧ *j* ≥ *i* ∧
      check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
        (*L* ! *snd* (*enum-pairs L* ! *j*))}
    by *blast*
  let *?newL* = *update-L L* (*enum-pairs L* ! *min-j*) *num-vars*
  have *?simp-L* = *WEST-simp-helper ?newL* (*enum-pairs ?newL*) *0 num-vars*
    using *WEST-simp-helper-can-simp*[*of ?simp-L L i num-vars min-j ?newL*]
    using *obt-min-j can-simp i-le* **by** *blast*
  have *min-j-bounds*: *min-j* < *length* (*enum-pairs L*) ∧ *min-j* ≥ *i*
    using *can-simp obt-min-j Min-in*[*of* {*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i*
∧
      check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
        (*L* ! *snd* (*enum-pairs L* ! *j*))}]
    by *fastforce*
  have *length ?newL* < *length L*
    using *update-L-length*[*of enum-pairs L* ! *min-j L num-vars*]
    using *min-j-bounds*
    by (*metis diff-less enum-pairs-bound less-nat-zero-code not-gr-zero nth-mem*
zero-less-one)
  then have *?case*
    using *less*(*1*)[*of ?newL*] *less.prems min-j-bounds update-L-preserves-num-vars*
    by (*metis ‹WEST-simp-helper L* (*enum-pairs L*) *i num-vars* = *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› nth-mem*)
} **moreover** {
  assume *cant-simp*: ¬(∃*j*. *j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧
      check-simp (*L* ! *fst* (*enum-pairs L* ! *j*))
        (*L* ! *snd* (*enum-pairs L* ! *j*)))

then have *?simp-L = L*
using *WEST-simp-helper-cant-simp i-le* **by** *blast*
then have *?case* **using** *less* **by** *simp*
   }
   ultimately have *?case* **using** *WEST-simp-helper-eq* **by** *blast*
 }
 ultimately show *?case*
   using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
   by *fastforce*
qed

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*
 unfolding *WEST-simp.simps*
 using *WEST-simp-helper-num-vars assms* **by** *blast*

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*
 unfolding *WEST-and-simp.simps*
 using *WEST-simp-num-vars WEST-and-num-vars assms* **by** *blast*

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*
 unfolding *WEST-or-simp.simps*
 using *WEST-simp-num-vars WEST-or-num-vars assms* **by** *blast*

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*
 using *assms*
proof(*induct L*)
 case *Nil*
 then show *?case*
   unfolding *WEST-regex-of-vars-def* **by** *auto*
next
 case (*Cons h t*)
 let *?padding = arbitrary-trace k a*
 let *?padh = ?padding @ h*
 let *?padt = shift t k a*
 have *padding-nv*: $\forall i < length$ (*arbitrary-trace k a*). *length* (*arbitrary-trace k a ! i*)

80

$= k$
    unfolding *trace-regex-of-vars-def* **by** *auto*
  have *h-nv*: *trace-regex-of-vars h k*
    using *Cons.prems* **unfolding** *WEST-regex-of-vars-def*
    by (*metis length-greater-0-conv list.distinct(1) nth-Cons-0*)
  then have *h-nv*: $\forall i{<}length\ h.\ length\ (h\ !\ i) = k$
    unfolding *trace-regex-of-vars-def* **by** *metis*
  have *length* $((\textit{?padding}\ @\ h)\ !\ i) = k$ **if** *i-lt*: $i < length\ (\textit{?padding}\ @\ h)$ **for** *i*
  proof$-$
    {assume *in-padding*: $i < length\ \textit{?padding}$
      then have *?thesis*
        using *padding-nv*
        by (*metis nth-append*)
    } moreover {
      assume *in-h*: $i \geq length\ \textit{?padding}$
      let *?index* $= i - (length\ \textit{?padding})$
      have $i - (length\ \textit{?padding}) < length\ h$
        using *i-lt in-h* **by** *auto*
      then have $h!\textit{?index} = (\textit{?padding}@h)!i$
        using *i-lt in-h* **by** (*simp add*: *nth-append*)
      then have *?thesis* **using** *h-nv*
        by (*metis ‹i − length (arbitrary-trace k a) < length h›*)
    }
    ultimately show *?thesis* **by** *fastforce*
  qed
  then have *padh-nv*: *trace-regex-of-vars ?padh k*
    unfolding *trace-regex-of-vars-def* **by** *simp*
  have $\forall ka{<}length\ (h\ \#\ t).\ \textit{trace-regex-of-vars}\ ((h\ \#\ t)\ !\ ka)\ k$
    using *Cons.prems* **unfolding** *WEST-regex-of-vars-def* **by** *metis*
  then have *WEST-regex-of-vars t k*
    unfolding *WEST-regex-of-vars-def* **by** *auto*
  then have *padt-nv*: *WEST-regex-of-vars ?padt k*
    using *Cons.hyps* **by** *simp*
  then show *?case* **using** *padh-nv padt-nv*
    using *regtraceList-cons-num-vars*[*of ?padh k ?padt*] **by** *simp*
qed

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*
  using *assms*
proof(*induct b−a arbitrary*: *L a b*)
  case *0*
  then have $a = b$ **by** *simp*
  then have *WEST-future-base*: (*WEST-future L a b k*) $=$ *shift L k a*
    using *WEST-future.simps*[*of L a b k*] **by** *auto*
  have *WEST-regex-of-vars* (*shift L k a*) *k*

    using *shift-num-vars 0* **by** *blast*
  then show *?case* **using** *WEST-future-base* **by** *simp*
next
  case (*Suc x*)
  then have *b = a + (Suc x)* **by** *auto*
  then have *west-future*: *WEST-future L a b k = WEST-or-simp (shift L k b)*
(*WEST-future L a (b − 1) k) k*
    using *WEST-future.simps*[*of L a b k*]
    by (*metis Suc.hyps(2) Zero-not-Suc cancel-comm-monoid-add-class.diff-cancel
diff-is-0-eq′ linorder-le-less-linear*)
  have *fact*: *WEST-regex-of-vars (shift L k b) k*
    using *shift-num-vars Suc* **by** *blast*
  have *indh*: *WEST-regex-of-vars (WEST-future L a (b − 1) k) k*
    using *Suc.hyps Suc.prems* **by** *simp*
  show *?case*
    using *west-future WEST-or-simp-num-vars fact indh* **by** *metis*
qed

lemma *WEST-global-num-vars*:
  assumes *WEST-regex-of-vars L k*
  assumes *a ≤ b*
  shows *WEST-regex-of-vars (WEST-global L a b k) k*
  using *assms*
proof(*induct b−a arbitrary*: *L a b*)
  case *0*
  then have *a = b* **by** *simp*
  then have *WEST-global-base*: (*WEST-global L a b k) = shift L k a*
    using *WEST-global.simps*[*of L a b k*] **by** *auto*
  have *WEST-regex-of-vars (shift L k a) k*
    using *shift-num-vars 0* **by** *blast*
  then show *?case* **using** *WEST-global-base* **by** *simp*
next
  case (*Suc x*)
  then have *b = a + (Suc x)* **by** *auto*
  then have *west-global*: *WEST-global L a b k = WEST-and-simp (shift L k b)*
(*WEST-global L a (b − 1) k) k*
    using *WEST-global.simps*[*of L a b k*]
    by (*metis Suc.hyps(2) Suc.prems(2) add-leE cancel-comm-monoid-add-class.diff-cancel
le-numeral-extra(3) nat-less-le not-one-le-zero plus-1-eq-Suc*)
  have *fact*: *WEST-regex-of-vars (shift L k b) k*
    using *shift-num-vars Suc* **by** *blast*
  have *indh*: *WEST-regex-of-vars (WEST-global L a (b − 1) k) k*
    using *Suc.hyps Suc.prems* **by** *simp*
  show *?case*
    using *west-global WEST-and-simp-num-vars fact indh*
    by *metis*
qed

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*
  using *assms*
proof(*induct b−a arbitrary*: *L1 L2 a b*)
  case *0*
  then have $a = b$ **by** *auto*
  have *WEST-until L1 L2 a b k* = *WEST-global L2 a a k*
    using *WEST-until.simps*[*of L1 L2 a b k*] *0* **by** *auto*
  then show *?case* **using** *0 WEST-global-num-vars*[*of L2 k a b*] **by** *simp*
next
  case (*Suc x*)
  then have $b = a + (Suc\ x)$ **by** *auto*
  then have *west-until*: *WEST-until L1 L2 a b k* = *WEST-or-simp* (*WEST-until L1 L2 a* (*b − 1*) *k*)

                                        (*WEST-and-simp* (*WEST-global L1 a* (*b − 1*) *k*) (*WEST-global L2 b b k*) *k*) *k*
    using *WEST-until.simps*[*of L1 L2 a b k*]
    by (*metis Suc.prems*(*3*) *Zero-neq-Suc add-eq-self-zero order-neq-le-trans*)

  have *fact1*: *WEST-regex-of-vars* (*WEST-global L1 a* (*b − 1*) *k*) *k*
    using *WEST-global-num-vars Suc* **by** *auto*
  have *fact2*: *WEST-regex-of-vars* (*WEST-global L2 b b k*) *k*
    using *WEST-global-num-vars Suc* **by** *blast*
  have *indh*: *WEST-regex-of-vars* (*WEST-until L1 L2 a* (*b − 1*) *k*) *k*
    using *Suc.hyps Suc.prems* **by** *simp*
  show *?case*
    using *west-until WEST-and-num-vars fact1 fact2 indh*
    using *WEST-and-simp-num-vars WEST-or-simp-num-vars* **by** *metis*
qed

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*
  using *assms*
proof(*induct b−a arbitrary*: *L1 L2 a b*)
  case *0*
  then have $a = b$ **by** *auto*
  then have *WEST-release-helper L1 L2 a b k* = *WEST-and-simp* (*WEST-global L1 a a k*) (*WEST-global L2 a a k*) *k*
    using *WEST-release-helper.simps*[*of L1 L2 a b k*] **by** *argo*
  have *fact1*: *WEST-regex-of-vars* (*WEST-global L1 a a k*) *k*
    using *WEST-global-num-vars*[*of L1 k a a*] *0* **by** *blast*

83

have *fact2*: *WEST-regex-of-vars* (*WEST-global L2 a a k*) *k*
  using *WEST-global-num-vars*[*of L2 k a a*] *0* **by** *blast*
then show *?case* **using** *WEST-release-helper.simps*[*of L1 L2 a b k*] *0*
  using *fact1 fact2 WEST-and-simp-num-vars* **by** *auto*
next
  case (*Suc x*)
  then have *b = a + (Suc x)* **by** *auto*
 then have *west-release-helper*: *WEST-release-helper L1 L2 a b k = WEST-or-simp*
(*WEST-release-helper L1 L2 a (b − 1) k*)
      (*WEST-and-simp* (*WEST-global L2 a b k*) (*WEST-global L1 b b k*) *k*) *k*
  using *WEST-release-helper.simps*[*of L1 L2 a b k*]
 by (*metis Suc.hyps(2) Suc.prems(3) add-eq-0-iff-both-eq-0 cancel-comm-monoid-add-class.diff-cancel
le-neq-implies-less plus-1-eq-Suc zero-neq-one*)

  have *fact1*: *WEST-regex-of-vars* ((*WEST-global L2 a b k*)) *k*
    using *WEST-global-num-vars Suc* **by** *auto*
  have *fact2*: *WEST-regex-of-vars* (*WEST-global L1 b b k*) *k*
    using *WEST-global-num-vars Suc* **by** *blast*
  have *indh*: *WEST-regex-of-vars* (*WEST-release-helper L1 L2 a (b − 1) k*) *k*
    using *Suc.hyps Suc.prems* **by** *simp*
  show *?case* **using** *WEST-release-helper.simps*[*of L1 L2 a b k*]
   using *fact1 fact2 indh WEST-and-simp-num-vars WEST-or-simp-num-vars Suc*
    by *presburger*
qed

lemma *WEST-release-num-vars*:
  assumes *WEST-regex-of-vars L1 k*
  assumes *WEST-regex-of-vars L2 k*
  assumes *a ≤ b*
  shows *WEST-regex-of-vars* (*WEST-release L1 L2 a b k*) *k*
  using *assms*
proof−
  {assume *a-eq-b*: *a = b*
    then have *WEST-release L1 L2 a b k = WEST-global L2 a b k*
     using *WEST-release.simps*[*of L1 L2 a b k*] **by** *auto*
    then have *?thesis* **using** *WEST-global-num-vars assms* **by** *auto*
  } moreover {
    assume *a-neq-b*: *a ≠ b*
    then have *b-pos*: *b > 0* **using** *assms* **by** *simp*
    have *a-leq-bm1*: *a ≤ b−1* **using** *a-neq-b assms* **by** *auto*
    then have *a-le-b*: *a < b* **using** *b-pos* **by** *auto*
    have *WEST-release L1 L2 a b k = WEST-or-simp* (*WEST-global L2 a b k*)
(*WEST-release-helper L1 L2 a (b − 1) k*) *k*
     using *WEST-release.simps*[*of L1 L2 a b k*] *a-le-b* **by** *argo*
    then have *?thesis*
     using *WEST-global-num-vars*[*of L2 a b k*]
     using *WEST-release-helper-num-vars*[*of L1 k L2 a b*]
    using *WEST-or-simp-num-vars*[*of WEST-global L2 a b k k WEST-release-helper*

L1 L2 a (b − 1) k]
    using *WEST-global-num-vars WEST-release-helper-num-vars a-leq-bm1 assms(1)*
assms(2) assms(3) **by** *presburger*
  }
  ultimately show *?thesis* **by** *blast*
qed

lemma *WEST-reg-aux-num-vars*:
  assumes *is-nnf*: ∃ *ψ. F1 = (convert-nnf ψ)*
  assumes *k ≥ WEST-num-vars F1*
  assumes *intervals-welldef F1*
  shows *WEST-regex-of-vars (WEST-reg-aux F1 k) k*
  using *assms*
proof (*induct F1 rule*: *nnf-induct*)
  case *nnf*
  then show *?case* **using** *is-nnf* **by** *simp*
next
  case *True*
  then show *?case* **using** *WEST-reg-aux.simps(1)[of k]*
    unfolding *WEST-regex-of-vars-def trace-regex-of-vars-def* **by** *auto*
next
  case *False*
  show *?case* **using** *WEST-reg-aux.simps(2)*
    unfolding *WEST-regex-of-vars-def trace-regex-of-vars-def* **by** *auto*
next
  case (*Prop p*)
  then show *?case* **using** *WEST-reg-aux.simps(3)[of p k]*
    unfolding *WEST-regex-of-vars-def trace-regex-of-vars-def* **by** *auto*
next
  case (*NotProp F p*)
  then show *?case* **using** *WEST-reg-aux.simps(3)[of p k]*
    unfolding *WEST-regex-of-vars-def trace-regex-of-vars-def* **by** *auto*
next
  case (*And F F1 F2*)
  have *nnf-F1*: ∃ψ. *F1 = convert-nnf ψ* **using** *And(1, 4)*
    by (*metis convert-nnf.simps(4) convert-nnf-convert-nnf mltl.inject(3)*)
  then have *F1-k*: *WEST-regex-of-vars (WEST-reg-aux F1 k) k*
    using *And* **by** *auto*
  have *nnf-F2*: ∃ψ. *F2 = convert-nnf ψ*
    by (*metis And.hyps(1) And.prems(1) convert-nnf.simps(4) convert-nnf-convert-nnf*
mltl.inject(3))
  then have *F2-k*: *WEST-regex-of-vars (WEST-reg-aux F2 k) k*
    using *And* **by** *auto*
  have *nv-F1*: *WEST-num-vars F1 ≤ k*
    using *WEST-num-vars-subformulas[of F1 And-mltl F1 F2] And(1,5)* **unfold-**
ing *subformulas.simps*
    by *simp*
  have *nv-F2*: *WEST-num-vars F2 ≤ k*

    using *WEST-num-vars-subformulas*[*of F2 And-mltl F1 F2*] *And*(*1,5*) **unfolding** *subformulas.simps*
      by *simp*
   show *?case*
     using *WEST-reg-aux.simps*(*6*)[*of F1 F2 k*] *And And*(*2*)[*OF nnf-F1 nv-F1*] *And*(*3*)[*OF nnf-F2 nv-F2*]
     using *WEST-and-simp-num-vars*[*of* (*WEST-reg-aux F1 k*) *k* (*WEST-reg-aux F2 k*)]
    by *auto*
next
  case (*Or F F1 F2*)
  have *nnf-F1*: ∃ψ. *F1 = convert-nnf ψ* **using** *Or*
    by (*metis convert-nnf.simps*(*5*) *convert-nnf-convert-nnf mltl.inject*(*4*))
  then have *F1-k*: *WEST-regex-of-vars* (*WEST-reg-aux F1 k*) *k*
    using *Or* **by** *auto*
  have *nnf-F2*: ∃ψ. *F2 = convert-nnf ψ*
   by (*metis Or.hyps*(*1*) *Or.prems*(*1*) *convert-nnf.simps*(*5*) *convert-nnf-convert-nnf mltl.inject*(*4*))
  then have *F2-k*: *WEST-regex-of-vars* (*WEST-reg-aux F2 k*) *k*
    using *Or* **by** *auto*
  let *?L1* = (*WEST-reg-aux F1 k*)
  let *?L2* = (*WEST-reg-aux F2 k*)
  have *WEST-regex-of-vars ?L1 k*
    using *Or nnf-F1* **by** *simp*
  then have *L1-nv*: ∀ *i*<*length* (*WEST-reg-aux F1 k*). *trace-regex-of-vars* (*WEST-reg-aux F1 k ! i*) *k*
    unfolding *WEST-regex-of-vars-def* **by** *metis*
  have *WEST-regex-of-vars ?L2 k*
    using *Or nnf-F2* **by** *simp*
  then have *L2-nv*: ∀ *j*<*length* (*WEST-reg-aux F2 k*). *trace-regex-of-vars* (*WEST-reg-aux F2 k ! j*) *k*
    unfolding *WEST-regex-of-vars-def* **by** *metis*

  have *L1L2-L*: *WEST-reg-aux F k = WEST-or-simp ?L1 ?L2 k*
    using *WEST-reg-aux.simps*(*5*)[*of F1 F2 k*] *Or* **by** *blast*
  let *?L = ?L1@?L2*
  show *?case*
    using *WEST-or-simp-num-vars*[*of ?L1 k ?L2, OF*] *L1-nv L2-nv L1L2-L*
    unfolding *WEST-regex-of-vars-def* **by** *auto*
next
  case (*Final F F1 a b*)
  let *?L1 = WEST-reg-aux F1 k*
  have *F1-nnf*: ∃ψ. *F1 = convert-nnf ψ* **using** *Final*
    by (*metis convert-nnf.simps*(*6*) *convert-nnf-convert-nnf mltl.inject*(*5*))
  then have *L1-nv*: *WEST-regex-of-vars ?L1 k*
    using *Final* **by** *simp*
  have *WEST-reg-future*: *WEST-reg-aux* (*Future-mltl a b F1*) *k = WEST-future ?L1 a b k*
    using *WEST-reg-aux.simps*(*7*)[*of a b F1 k*] **by** *blast*

let *?L = WEST-future ?L1 a b k*
have *WEST-regex-of-vars ?L k*
  using *L1-nv WEST-future-num-vars[of ?L1 k a b] Final* **by** *auto*
then show *?case* **using** *WEST-reg-future Final* **by** *simp*
next
  case (*Global F F1 a b*)
  let *?L1 = WEST-reg-aux F1 k*
  have *F1-nnf*: ∃ψ. *F1 = convert-nnf ψ* **using** *Global*
    by (*metis convert-nnf.simps(7) convert-nnf-convert-nnf mltl.inject(6)*)
  then have *L1-nv*: *WEST-regex-of-vars ?L1 k*
    using *Global* **by** *simp*
  have *WEST-regex-of-vars* (*WEST-global ?L1 a b k*) *k*
    using *L1-nv WEST-global-num-vars[of ?L1 k a b] Global* **by** *simp*
  then show *?case* **using** *WEST-reg-aux.simps(8)[of a b F1 k] Global(1)* **by** *simp*
next
  case (*Until F F1 F2 a b*)
  have *nnf-F1*: ∃ψ. *F1 = convert-nnf ψ* **using** *Until*
    by (*metis convert-nnf.simps(8) convert-nnf-convert-nnf mltl.inject(7)*)
  then have *F1-k*: *WEST-regex-of-vars* (*WEST-reg-aux F1 k*) *k*
    using *Until* **by** *auto*
  have *nnf-F2*: ∃ψ. *F2 = convert-nnf ψ* **using** *Until*
    by (*metis convert-nnf.simps(8) convert-nnf-convert-nnf mltl.inject(7)*)
  then have *F2-k*: *WEST-regex-of-vars* (*WEST-reg-aux F2 k*) *k*
    using *Until* **by** *auto*
  let *?L1 = (WEST-reg-aux F1 k)*
  let *?L2 = (WEST-reg-aux F2 k)*
  have *L1-nv*:*WEST-regex-of-vars ?L1 k*
    using *Until nnf-F1* **by** *simp*
  have *L2-nv*:*WEST-regex-of-vars ?L2 k*
    using *Until nnf-F2* **by** *simp*

  have *WEST-regex-of-vars* (*WEST-until* (*WEST-reg-aux F1 k*) (*WEST-reg-aux F2 k*) *a b k*) *k*
    using *WEST-until-num-vars[of ?L1 k ?L2 a b, OF L1-nv L2-nv] Until* **by** *auto*
  then show *?case* **using** *Until(1) WEST-reg-aux.simps(9)[of F1 a b F2 k]* **by** *auto*
next
  case (*Release F F1 F2 a b*)
  have *nnf-F1*: ∃ψ. *F1 = convert-nnf ψ* **using** *Release*
    by (*metis convert-nnf.simps(9) convert-nnf-convert-nnf mltl.inject(8)*)

  then have *F1-k*: *WEST-regex-of-vars* (*WEST-reg-aux F1 k*) *k*
    using *Release* **by** *auto*
  have *nnf-F2*: ∃ψ. *F2 = convert-nnf ψ* **using** *Release*
    by (*metis convert-nnf.simps(9) convert-nnf-convert-nnf mltl.inject(8)*)
  then have *F2-k*: *WEST-regex-of-vars* (*WEST-reg-aux F2 k*) *k*
    using *Release* **by** *auto*
  let *?L1 = (WEST-reg-aux F1 k)*
  let *?L2 = (WEST-reg-aux F2 k)*

have *L1-nv*:*WEST-regex-of-vars ?L1 k*
  using *Release nnf-F1* **by** *simp*
have *L2-nv*:*WEST-regex-of-vars ?L2 k*
  using *Release nnf-F2* **by** *simp*

 have *WEST-regex-of-vars* (*WEST-release* (*WEST-reg-aux F1 k*) (*WEST-reg-aux*
F2 k) *a b k*) *k*
  using *WEST-release-num-vars*[*of ?L1 k ?L2 a b, OF L1-nv L2-nv*] *Release* **by**
auto
 then show *?case* **using** *WEST-reg-aux.simps*(*10*)[*of F1 a b F2 k*] *Release* **by**
argo
qed

lemma *nnf-intervals-welldef*:
 assumes *intervals-welldef F1*
 shows *intervals-welldef* (*convert-nnf F1*)
 using *assms*
proof (*induct depth-mltl F1 arbitrary*: *F1 rule*: *less-induct*)
 case *less*
 have *iwd*: *intervals-welldef F2* $\implies$
    F1 = Not-mltl F2 $\implies$
    intervals-welldef (*convert-nnf* (*Not-mltl F2*))
  for *F2* **apply** (*cases F2*) **using** *less* **by** *simp-all*
 then show *?case* **using** *less*
  apply (*cases F1*) **by** *simp-all*
qed

lemma *WEST-reg-num-vars*:
 assumes *intervals-welldef F1*
 shows *WEST-regex-of-vars* (*WEST-reg F1*) (*WEST-num-vars F1*)
proof −
 have *WEST-num-vars* (*convert-nnf F1*) = *WEST-num-vars F1*
  using *WEST-num-vars-nnf* **by** *presburger*
 then have *wnv*: *WEST-num-vars* (*convert-nnf F1*) ≤ (*WEST-num-vars F1*)
  by *simp*
 have *iwd*: *intervals-welldef* (*convert-nnf F1*)
  using *assms nnf-intervals-welldef*
  by *auto*
 show *?thesis*
  using *assms WEST-reg-aux-num-vars*[*OF - wnv iwd*]
  unfolding *WEST-reg.simps*
  by *auto*
qed

3.7 Correctness of WEST-simp

3.7.1 WEST-count-diff facts

lemma *count-diff-property-aux*:
 assumes *k < length r1* ∧ *k < length r2*

shows *count-diff r1 r2 ≥ count-diff-state (r1 ! k) (r2 ! k)*
  using *assms*
proof (*induct length r1 arbitrary: r1 r2 k*)
  case *0*
  then show *?case* **by** *simp*
next
  case (*Suc x*)
  obtain *h1 t1 h2 t2* **where** *r1r2: r1 = h1#t1 r2 = h2#t2*
    using *Suc*
    by (*metis length-0-conv not-less-zero trim-reversed-regex.cases*)
  have *cd: count-diff r1 r2 = count-diff-state h1 h2 + count-diff t1 t2*
      using *r1r2 count-diff.simps(4)[of h1 t1 h2 t2]* **by** *simp*
  {**assume** *∗: k = 0*
    have *count-diff r1 r2 ≥ count-diff-state h1 h2*
      using *cd*
      by *auto*
    then have *?case* **using** *∗ r1r2*
      by *auto*
  } **moreover** {**assume** *∗: k > 0*
    have *t1t2: t1 ! (k−1) = r1 ! k ∧ t2 ! (k−1) = r2 ! k*
      using *Suc(3) ∗ r1r2*
      by *simp*
     have *count-diff-state (t1 ! (k − 1)) (t2 ! (k − 1))*
    ≤ count-diff t1 t2
      using *∗ Suc(1)[of t1 k−1 t2]*
      Suc(2−3) r1r2
      by (*metis One-nat-def Suc-less-eq Suc-pred diff-Suc-1' length-Cons*)
    then have *?case* **using** *cd t1t2*
      by *auto*
  }
  ultimately show *?case* **by** *blast*
qed

lemma *count-diff-state-property:*
   assumes *count-diff-state t1 t2 = 0*
   assumes *ka < length t1 ∧ ka < length t2*
   shows *t1 ! ka = t2 ! ka*
  using *assms*
  proof (*induct length t1 arbitrary: t1 t2 ka*)
    case *0*
    then show *?case* **by** *simp*
  next
    case (*Suc x*)
    obtain *h1 T1 h2 T2* **where** *t1t2: t1 = h1#T1 t2 = h2#T2*
    using *Suc*
    by (*metis count-nonS-trace.cases length-0-conv less-nat-zero-code*)
  have *cd: h1 = h2 ∧ count-diff-state t1 t2 = count-diff-state T1 T2*
    using *t1t2 count-diff-state.simps(4)[of h1 T1 h2 T2]*
    Suc(3) **by** *presburger*

```
    then have ind0: count-diff-state T1 T2 = 0
      using Suc(3) by auto
    {assume *: ka = 0
      then have ?case using cd t1t2
        by auto
    } moreover {assume *: ka > 0
      have T1T2: T1 ! (ka−1) = t1 ! ka ∧ T2 ! (ka−1) = t2 ! ka
        using Suc(3) * t1t2
        by simp
       have T1 ! (ka−1) = T2 ! (ka−1)
       using * Suc(1)[OF - ind0, of ka]
       Suc(2−3) t1t2
          by (metis Suc.hyps(1) Suc.prems(2) Suc-less-eq Suc-pred diff-Suc-1 ind0
length-Cons)
      then have ?case using T1T2
        by auto
    }
    ultimately show ?case by blast
  qed

lemma count-diff-property:
  assumes count-diff r1 r2 = 0
  assumes k < length r1 ∧ k < length r2
  assumes ka < length (r1 ! k) ∧ ka < length (r2 ! k)
  shows r2 ! k ! ka = r1 ! k ! ka
proof −
  have count-diff r1 r2 ≥ count-diff-state (r1 ! k) (r2 ! k)
    using count-diff-property-aux[OF assms(2)]
    by auto
  then have cdt: count-diff-state (r1 ! k) (r2 ! k) = 0
    using assms by auto
  show ?thesis
    using count-diff-state-property[OF cdt assms(3)]
    by auto
qed

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]
  using assms
proof(induct num-vars arbitrary: h)
  case 0
  then show ?case by simp
next
  case (Suc num-vars)
  then obtain head tail where head-tail: h = head#tail
    by (meson length-Suc-conv)
    have tail = map (λt. S) [0..<num-vars]
```

90

using *Suc(1)[of tail] head-tail Suc.prems*
  by (*metis Zero-not-Suc count-nonS-trace.simps(2) length-Cons nat.inject plus-1-eq-Suc*)
then have *count-nonS-trace tail = 0*
  using *count-nonS-trace.simps Suc.prems(2)*
  by (*metis Suc.prems(2) add-is-0 head-tail*)
then show *?case*
  using *count-nonS-trace.simps(2)[of head tail] head-tail*
proof −
  have *f1: 0 = Suc 0 + 0 ∨ head = S*
    using *One-nat-def Suc.prems(2)* ‹*count-nonS-trace (head # tail) = (if head ≠ S then 1 + count-nonS-trace tail else count-nonS-trace tail)*› ‹*count-nonS-trace tail = 0*› *head-tail* **by** *argo*
  have *map (λn. S) [0..<Suc num-vars] = S # map (λn. S) [0..<num-vars]*
    using *map-upt-Suc* **by** *blast*
  then show *?thesis*
    using *f1* ‹*tail = map (λt. S) [0..<num-vars]*› *head-tail* **by** *presburger*
qed
qed

lemma *trace-tail-num-vars*:
  assumes *trace-regex-of-vars (h # trace) num-vars*
  shows *trace-regex-of-vars trace num-vars*
proof −
  have ⋀*i. i<length trace ⟹ length (trace ! i) = num-vars*
  proof −
    fix *i*
    assume *i-le: i<length trace*
    have *i+1 < length (h#trace)*
      using *Cons*
      by (*meson i-le impossible-Cons leI le-trans less-iff-succ-less-eq*)
    then have *length ((h # trace) ! (i+1)) = num-vars*
      using *assms* **unfolding** *trace-regex-of-vars-def* **by** *meson*
    then show *length (trace ! i) = num-vars*
      by *auto*
  qed
  then show *?thesis*
    unfolding *trace-regex-of-vars-def* **by** *auto*
qed

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]*
  using *assms*
proof(*induct trace arbitrary: k num-vars*)
  case *Nil*

91

then show *?case* **by** *simp*
next
  case (*Cons h trace*)
  {assume *k-zero*: *k = 0*
    have *cond1*: *length h = num-vars*
      using *Cons.prems(3)* **unfolding** *trace-regex-of-vars-def*
      by (*metis Cons.prems(2) k-zero nth-Cons-0*)
    have *cond2*: *count-nonS-trace h = 0*
      using *Cons.prems(1) count-diff.simps*
      by (*metis add-is-0 count-diff-state.simps(3) count-nonS-trace.elims*)
    have *h = map (λt. S) [0..<num-vars]*
      using *count-nonS-trace-0-allS[OF cond1 cond2]* **by** *simp*
    then have *?case*
      by (*simp add: k-zero*)
  } moreover {
    assume *k-ge-zero*: *k > 0*
    have *cond1*: *count-diff trace [] = 0*
     by (*metis Cons.prems(1) count-diff.simps(2) count-diff.simps(3) neq-Nil-conv zero-eq-add-iff-both-eq-0*)
    have *cond2*: *k−1 < length trace*
      using *k-ge-zero Cons.prems(2)* **by** *auto*
    have *cond3*: *trace-regex-of-vars trace num-vars*
      using *trace-tail-num-vars Cons(4)*
      unfolding *trace-regex-of-vars-def*
      by *blast*
    have *trace ! (k−1) = map (λt. S) [0 ..< num-vars]*
      using *Cons.hyps[OF cond1 cond2 cond3] Cons.prems* **by** *blast*
    then have *?case*
      using *k-ge-zero* **by** *simp*
  }
  ultimately show *?case* **by** *blast*
qed

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−
  have *length r1 > length r2*
    using *assms* **by** *simp*
  let *?tail = drop (length r2) r1*
  have *cond1*: *count-diff ?tail [] = 0*
    using *assms(1, 2)*
  proof(*induct r2 arbitrary: r1 k*)
    case *Nil*
    then show *?case* **by** *simp*

92

next
  case (*Cons a r2*)
  then obtain *h T* **where** *obt-hT*: *r1 = h#T*
    by (*metis length-0-conv less-nat-zero-code trim-reversed-regex.cases*)
  have *count-diff-state h a = 0*
    using *count-diff.simps(4)*[*of h T a r2*] *Cons.prems obt-hT* **by** *simp*
  then have *cond1*: *count-diff T r2 = 0*
    using *count-diff.simps(4)*[*of h T a r2*] *Cons.prems obt-hT* **by** *simp*
  have *count-diff (drop (length r2) T) [] = 0*
    using *Cons.hyps*[*OF cond1*] *Cons.prems obt-hT*
    by (*metis count-diff.simps(1) drop-all linorder-le-less-linear order-refl*)
  then show *?case*
    using *obt-hT* **by** *simp*
qed
have *cond2*: (*k − length r2*) < *length (drop (length r2) r1)*
  using *assms* **by** *auto*
have *cond3*: *trace-regex-of-vars (drop (length r2) r1) num-vars*
  using *assms(3, 2)* **unfolding** *trace-regex-of-vars-def*
  by (*metis ‹length r2 < length r1› add.commute leI length-drop less-diff-conv
nth-drop order.asym*)
have *?tail ! (k − length r2) = map (λt. S) [0 ..< num-vars]*
  using *count-diff-property-S-aux*[*OF cond1 cond2 cond3*] *assms* **by** *blast*
then show *?thesis*
  using *assms* **by** *auto*
qed

lemma *count-diff-state-commutative*:
  shows *count-diff-state e1 e2 = count-diff-state e2 e1*
  proof (*induct e1 arbitrary: e2*)
    case *Nil*
    then show *?case* **using** *count-diff-state.simps*
      by (*metis count-nonS-trace.cases*)
  next
    case (*Cons h1 t1*)
    then show *?case*
      by (*cases e2*) *simp-all*
  qed

lemma *count-diff-commutative*:
  shows *count-diff r1 r2 = count-diff r2 r1*
proof (*induct r1 arbitrary: r2*)
  case *Nil*
  then show *?case* **using** *count-diff.simps*
    by (*metis trim-reversed-regex.cases*)
next
  case (*Cons h1 t1*)
  {**assume** ∗: *r2 = []*
    then have *?case*

using *count-diff* .*simps* **by** *auto*
  } moreover {
    assume ∗: *r2* ≠ []
    then obtain *h2 t2* **where** *r2* = *h2*#*t2*
      by (*meson neq-Nil-conv*)
    then have *?case* **using** *count-diff* .*simps(4)*[*of h1 t1 h2 t2*]
      Cons[*of t2*] ∗ *count-diff-state-commutative*
      by *auto*
  }
  ultimately show *?case* **by** *blast*
qed

lemma *count-diff-same-trace*:
  shows *count-diff trace trace* = *0*
proof(*induct trace*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a trace*)
  have *count-diff-state a a* = *0*
  proof(*induct a*)
    case *Nil*
    then show *?case* **by** *simp*
  next
    case (*Cons a1 a2*)
    then show *?case* **by** *simp*
  qed
  then show *?case*
    using *Cons count-diff* .*simps(4)*[*of a trace a trace*] **by** *auto*
qed

lemma *count-diff-state-0*:
  assumes *count-diff-state h1 h2* = *0*
  assumes *length h1* = *length h2*
  shows *h1* = *h2*
  using *assms*
proof(*induct h1 arbitrary*: *h2*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a h1*)
  then show *?case*
    by (*metis count-diff-state-property nth-equalityI*)
qed

lemma *count-diff-state-1*:
  assumes *length h1* = *length h2*

assumes *count-diff-state h1 h2 = 1*
shows $\exists\, ka < length\ h1.\ h1!ka \neq h2!ka$
using *assms*
proof(*induct h1 arbitrary: h2*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a h1*)
  then obtain *head tail* **where** *obt-headtail*: *h2 = head#tail*
    by (*metis length-0-conv neq-Nil-conv*)
  {**assume** *head-equal*: *a = head*
    then have *count-diff-state h1 tail = 1*
      using *count-diff-state.simps(4)[of a h1 head tail]*
      using *Cons.prems(2) obt-headtail* **by** *auto*
    then have $\exists\, ka < length\ h1.\ h1\ !\ ka \neq tail\ !\ ka$
      using *Cons.hyps[of tail] Cons.prems*
      by (*simp add: obt-headtail*)
    then have *?case* **using** *obt-headtail* **by** *auto*
  } **moreover** {
    assume *head-notequal*: $a \neq head$
    then have *?case* **using** *obt-headtail* **by** *auto*
  }
  ultimately show *?case* **by** *blast*
qed

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 $\forall\, i < length\ h1.\ k \neq i \longrightarrow h1!i = h2!i$
  using *assms*
proof(*induct h1 arbitrary: h2 k*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a h1*)
  then obtain *head tail* **where** *headtail*: *h2 = head#tail*
    by (*metis Suc-length-conv*)
  {**assume** *k0*: *k = 0*
    then have *count-diff-state h1 tail = 0*
      using *Cons.prems headtail count-diff-state.simps(4)[of a h1 head tail]* **by** *auto*
    then have *h1 = tail*
      using *count-diff-state-0 Cons.prems headtail* **by** *simp*
    then have *?case* **using** *k0 headtail* **by** *simp*
  } **moreover** {
    assume *k-not0*: $k \neq 0$
    then have *head-eq*: *a = head*
      using *Cons headtail count-diff-state.simps(4)[of a h1 head tail]*

by (*metis One-nat-def Suc-inject count-diff-state-0 length-Cons nth-Cons′ plus-1-eq-Suc*)
    then have *count-diff-state h1 tail = 1*
      using *Cons headtail count-diff-state.simps(4)[of a h1 head tail]* **by** *argo*
    then have *induction*: $\forall i$<*length h1*. $k-1 \neq i \longrightarrow h1 \ ! \ i = tail \ ! \ i$
      using *Cons.hyps[of h2 k−1] Cons.prems headtail*
      by (*smt (verit) Cons.hyps Suc-less-eq add-diff-inverse-nat k-not0 length-Cons less-one nth-Cons′ old.nat.inject plus-1-eq-Suc*)
    have $\bigwedge i.$ $(i$<*length* $(a \ \# \ h1) \land k \neq i) \Longrightarrow (a \ \# \ h1) \ ! \ i = h2 \ ! \ i$
    proof −
      fix *i*
      assume *i-facts*: $(i$<*length* $(a \ \# \ h1) \land k \neq i)$
      {**assume** *i0*: $i = 0$
        then have $(a \ \# \ h1) \ ! \ i = h2 \ ! \ i$
          using *headtail head-eq* **by** *simp*
      } **moreover** {
        assume *i-not0*: $i \neq 0$
        then have $(a \ \# \ h1) \ ! \ i = h2 \ ! \ i$
          using *induction k-not0 i-facts*
          using *headtail length-Cons nth-Cons′ zero-less-diff* **by** *auto*
      }
      ultimately show $(a \ \# \ h1) \ ! \ i = h2 \ ! \ i$ **by** *blast*
    qed
    then have *?case* **by** *blast*
  }
  ultimately show *?case* **by** *blast*
qed

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*
  using *assms*
proof (*induct r1 arbitrary*: *r2*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons h1 r1*)
  then obtain *h T* **where** *obt-hT*: $r2 = h \# T$
    by (*metis length-0-conv list.exhaust*)
  have *cond1*: *trace-regex-of-vars r1 num-vars*
    using *trace-tail-num-vars Cons.prems* **by** *blast*
  have *cond2*: *trace-regex-of-vars T num-vars*
    using *trace-tail-num-vars Cons.prems obt-hT* **by** *blast*
  have *h1-h-samelen*: *length h1 = length h*
    using *Cons.prems obt-hT* **unfolding** *trace-regex-of-vars-def*
    by (*metis length-greater-0-conv nth-Cons-0*)

have *r1-eq-T*: *r1 = T*
  using *Cons.hyps[OF cond1 cond2] Cons.prems*
  by (*simp add*: *obt-hT*)
then have *count-diff r1 T = 0*
  using *count-diff-same-trace* **by** *auto*
then have *count-diff-state h1 h = 0*
  using *Cons.prems(3) obt-hT count-diff.simps(4)[of h1 r1 h T]* **by** *simp*
then have *h = h1* **using** *h1-h-samelen*
proof(*induct h arbitrary*: *h1*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a h*)
  then show *?case* **using** *count-diff-state.simps*
    Suc-inject count-diff-state.elims length-Cons less-iff-Suc-add not-less-eq
    by (*metis* (*no-types, opaque-lifting*) *count-diff-state-0*)
  qed
then show *?case*
  using *r1-eq-T obt-hT* **by** *blast*
qed

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 $\exists k < length\ r1.\ count\text{-}diff\text{-}state\ (r1!k)\ (r2!k) = 1$
  using *assms*
proof(*induct length r1 arbitrary*: *r1 r2*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc x*)
  obtain *h1 T1* **where** *obt-h1T1*: *r1 = h1#T1* **using** *Suc*
    by (*metis length-Suc-conv*)
  obtain *h2 T2* **where** *obt-h2T2*: *r2 = h2#T2* **using** *Suc*
    by (*metis length-Suc-conv*)
  {assume *h1h2-same*: *h1 = h2*
    have *count-diff-state h1 h2 = 0*
      using *h1h2-same count-diff-state-0*
      by (*metis Nat.add-0-right count-diff.simps(4) count-diff-same-trace*)
    then have *cond2*: *count-diff T1 T2 = 1*
      using *h1h2-same Suc.prems(1) obt-h1T1 obt-h2T2*
      using *count-diff.simps(4)[of h1 T1 h2 T2]* **by** *simp*
    have $\exists k < length\ T1.\ count\text{-}diff\text{-}state\ (T1\ !\ k)\ (T2\ !\ k) = 1$
      using *Suc obt-h1T1 obt-h2T2 h1h2-same*
      by (*metis cond2 length-Cons nat.inject trace-tail-num-vars*)
    then have *?case* **using** *obt-h1T1 obt-h2T2*
      by *fastforce*

97

} moreover {
  assume *h1h2-notsame*: *h1 ≠ h2*
  have *h1h2-nv*: *length h1 = length h2*
    using *Suc.prems(3, 4)* **unfolding** *trace-regex-of-vars-def*
   by (*metis Suc.hyps(2) Suc.prems(2) nth-Cons-0 obt-h1T1 obt-h2T2 zero-less-Suc*)
  then have *count-diff-state h1 h2 > 0*
    using *count-diff-state-0 h1h2-notsame* **by** *auto*
  then have *count-diff-state h1 h2 = 1*
    using *count-diff.simps(4)[of h1 T1 h2 T2] Suc obt-h1T1 obt-h2T2* **by** *auto*
  then have *?case* **using** *obt-h1T1 obt-h2T2* **by** *auto*
}
  ultimately show *?case* **by** *blast*
qed

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 $\forall i < length\ r1.\ k \neq i \longrightarrow r1!i = r2!i$
  using *assms*
proof(*induct length r1 arbitrary: r1 r2 k*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc x*)
  obtain *h1 T1* **where** *obt-h1T1*: *r1 = h1#T1* **using** *Suc*
    by (*metis length-Suc-conv*)
  obtain *h2 T2* **where** *obt-h2T2*: *r2 = h2#T2* **using** *Suc*
    by (*metis length-Suc-conv*)
  {assume *k0*: *k = 0*
    have *count-diff T1 T2 = 0*
     using *Suc count-diff.simps(4)[of h1 T1 h2 T2] obt-h1T1 obt-h2T2 k0*
     by *auto*
    then have $\forall i < length\ T1.\ T1\ !\ i = T2\ !\ i$
     using *Suc.prems count-diff-same-len trace-tail-num-vars*
     by (*metis Suc-inject length-Cons obt-h1T1 obt-h2T2*)
    then have *?case* **using** *obt-h1T1 obt-h2T2 k0*
     using *length-Cons* **by** *auto*
  } moreover {
    assume *k-not0*: *k ≠ 0*
    then have *T1T2-diffby1*: *count-diff T1 T2 = 1*
     using *Suc.prems obt-h1T1 obt-h2T2 count-diff.simps(4)[of h1 T1 h2 T2]*
     by (*metis One-nat-def add-right-imp-eq count-diff-same-len count-diff-state-1*
list.size(4) not-gr-zero nth-Cons-pos one-is-add trace-tail-num-vars)
    then have *h1h2-same*: *h1 = h2*
     using *k-not0 count-diff.simps(4)[of h1 T1 h2 T2] Suc.prems obt-h1T1 obt-h2T2*

98

    unfolding *trace-regex-of-vars-def*
        by (*metis Suc.hyps(2) add-cancel-right-left count-diff-state-0 nth-Cons-0 zero-less-Suc*)
    have *induction*: $\forall i{<}length\ T1.\ (k{-}1) \neq i \longrightarrow T1\ !\ i = T2\ !\ i$
        using *Suc.hyps(1)*[*of T1 T2 k−1*] *Suc.hyps(2) Suc.prems T1T2-diffby1*
    by (*metis* (*mono-tags, lifting*) *k-not0 length-Cons nth-Cons′ obt-h1T1 obt-h2T2 old.nat.inject trace-tail-num-vars*)
    then have *?case* **using** *obt-h1T1 obt-h2T2 k-not0 h1h2-same*
        by (*simp add*: *nth-Cons′*)
  }
  ultimately show *?case* **by** *blast*
qed

3.7.2 Orsimp-trace Facts

lemma *WEST-simp-bitwise-identity*:
  assumes *b1 = b2*
  shows *WEST-simp-bitwise b1 b2 = b1*
  using *assms WEST-simp-bitwise.simps*
  by (*metis WEST-bit.exhaust*)

lemma *WEST-simp-bitwise-commutative*:
  shows *WEST-simp-bitwise b1 b2 = WEST-simp-bitwise b2 b1*
  using *WEST-simp-bitwise.simps*
  by (*metis* (*full-types*) *WEST-simp-bitwise.elims*)

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*
  using *WEST-simp-state.simps*[*of s1 s2*]
  using *WEST-simp-bitwise-commutative assms* **by** *simp*

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−
  have *r1-vars*: $\forall k.\ length\ (WEST\text{-}get\text{-}state\ r1\ k\ num\text{-}vars) = num\text{-}vars$
    using *assms WEST-get-state-length* **by** *blast*
  have *r2-vars*: $\forall k.\ length\ (WEST\text{-}get\text{-}state\ r2\ k\ num\text{-}vars) = num\text{-}vars$
    using *assms WEST-get-state-length* **by** *blast*
  have ($\lambda k.$ *WEST-simp-state* (*WEST-get-state r1 k num-vars*)
        (*WEST-get-state r2 k num-vars*)) $= (\lambda k.$ *WEST-simp-state* (*WEST-get-state r2 k num-vars*)
            (*WEST-get-state r1 k num-vars*))
    using *WEST-simp-state-commutative r1-vars r2-vars* **by** *fast*
  then show *?thesis*

unfolding *WEST-simp-trace.simps*[*of r1 r2 num-vars*]
  unfolding *WEST-simp-trace.simps*[*of r2 r1 num-vars*]
  by (*simp add: insert-commute*)
qed

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 −
  have *of-vars*: $\forall$ *i<length r1. length (r1 ! i) = num-vars*
    using *assms* **unfolding** *trace-regex-of-vars-def* **by** *argo*
  have *mapmap*: *map* (λ*k. map* (λ*ka. (r1!k)!ka*)
          [0..< num-vars]) *[0..<length r1] = r1*
    using *assms(1)* **unfolding** *trace-regex-of-vars-def*[*of r1 num-vars*]
    by (*smt* (*verit*) *length-map list-eq-iff-nth-eq map-nth nth-map*)

  have *r1-k-ka*: $\bigwedge$*ka. ka < num-vars* $\Longrightarrow$
     WEST-simp-bitwise (*WEST-get-state r1 k num-vars ! ka*)
             (*WEST-get-state r2 k num-vars ! ka*) = *r1!k!ka*
   if *k-lt*: *k<length r1* **for** *k*
  proof −
    fix *ka*
    assume *ka-lt*: *ka < num-vars*
    {assume $*$: *k < length r2*
     have *length (r1 ! k) = num-vars* $\land$ *length (r2 ! k) = num-vars*
      using *assms* **unfolding** *trace-regex-of-vars-def* $*$ *ka-lt*
      using $*$ *that* **by** *presburger*
     then have (*r2 ! k*) *! ka = (r1 ! k) ! ka*
      using $*$ *ka-lt* **using** *assms(3)*
      using *count-diff-property-aux*
      using *count-diff-property that* **by** *presburger*
      then have *WEST-get-state r2 k num-vars ! ka = WEST-get-state r1 k num-vars ! ka*
      unfolding *WEST-get-state.simps* **using** $*$ *ka-lt*
      using *leD that* **by** *auto*
     then have *WEST-simp-bitwise* (*WEST-get-state r1 k num-vars ! ka*)
              (*WEST-get-state r2 k num-vars ! ka*) = *r1!k!ka*
      using *WEST-simp-bitwise-identity that* **by** *force*
    } moreover {assume $*$: *k ≥ length r2*
     then have *WEST-get-state r2 k num-vars = (map* (λ *k. S*) *[0 ..< num-vars]*)
      by *simp*
     then have *r2-k-ka-S*: (*WEST-get-state r2 k num-vars ! ka*) = *S*
      using *ka-lt* **by** *simp*

     have *r1-k-ka*: (*WEST-get-state r1 k num-vars ! ka*) = *r1!k!ka*

using *k-lt* **by** *simp*
　have $(r1!k!ka) = S$
　　using *count-diff-property-S*
　　using $*$ *ka-lt assms(1, 3, 4)*
　　using *that*
　　by *simp*
　then have *WEST-simp-bitwise* (*WEST-get-state r1 k num-vars* ! *ka*)
　　　　　$S = r1!k!ka$
　　using *r1-k-ka* **by** *simp*
　then have *WEST-simp-bitwise* (*WEST-get-state r1 k num-vars* ! *ka*)
　　　　　(*WEST-get-state r2 k num-vars* ! *ka*) $= r1!k!ka$
　　using *r2-k-ka-S* **by** *simp*
　}
　ultimately show *WEST-simp-bitwise* (*WEST-get-state r1 k num-vars* ! *ka*)
　　　　　(*WEST-get-state r2 k num-vars* ! *ka*) $= r1!k!ka$ **by** *auto*
qed
have *len-lhs*: *length* (*map* ($\lambda k.$ (*f k*)
　　　　$[0..< num\text{-}vars]$)
　$[0..<length\ r1]$) $= length\ r1$ **for** $f :: nat \Rightarrow nat\ list \Rightarrow WEST\text{-}bit\ list$
　by *auto*
have *aux-helper*: $\bigwedge i.\ i < length\ r1 \implies$ (*map* ($\lambda k.$ (*f k*)
　　　　$[0..< num\text{-}vars]$)
　$[0..<length\ r1]$)! $i = r1\ !\ i$ **if** *f-prop*: $\forall k<length\ r1.\ (f\ k)$
　　　　$[0..< num\text{-}vars] = r1!k$ **for** f
　proof $-$
　　fix i
　　assume $i < length\ r1$
　　show *map* ($\lambda k.\ f\ k\ [0..<num\text{-}vars]$) $[0..<length\ r1]\ !\ i = r1\ !\ i$
　　　using *f-prop*
　　　by (*simp add*: $\langle i < length\ r1 \rangle$)
　qed
have *map-prop*: *map* ($\lambda k.$ (*f k*)
　　　　$[0..< num\text{-}vars]$)
　　$[0..<length\ r1] = r1$ **if** *f-prop*: $\forall k<length\ r1.\ (f\ k)$
　　　　$[0..< num\text{-}vars] = r1!k$ **for** f
　　using *len-lhs*[*of f*] *aux-helper*[*of f*] *f-prop*
　　by (*metis nth-equalityI*)

let *?f* $= \lambda i.\ map\ (\lambda ka.\ WEST\text{-}simp\text{-}bitwise\ (WEST\text{-}get\text{-}state\ r1\ i\ num\text{-}vars\ !\ ka)$
　　　　　(*WEST-get-state r2 i num-vars* ! *ka*))

have $\forall k<length\ r1.\ map\ (\lambda ka.\ WEST\text{-}simp\text{-}bitwise\ (WEST\text{-}get\text{-}state\ r1\ k\ num\text{-}vars$
! *ka*)
　　　　　(*WEST-get-state r2 k num-vars* ! *ka*))
　　　　$[0..< num\text{-}vars] = r1!k$
　using *r1-k-ka*
　by (*smt* (*z3*) *length-map length-upt minus-nat.diff-0 nth-equalityI nth-map-upt
of-vars plus-nat.add-0*)

then have $\forall\, k$<*length r1*. (*?f k*)
$[0..< num\text{-}vars] = r1!k$
  by *blast*
then have *map* ($\lambda k.$ (*?f k*)
$[0..< num\text{-}vars]$)
$[0..<length\ r1] = r1$
  using *map-prop*[*of ?f*]
  by *blast*
then have *map* ($\lambda k.\ map\ (\lambda ka.\ WEST\text{-}simp\text{-}bitwise$ (*WEST-get-state r1 k num-vars*
! *ka*)
($WEST\text{-}get\text{-}state\ r2\ k\ num\text{-}vars\ !\ ka$))
$[0..< num\text{-}vars]$)
$[0..<length\ r1] = r1$
  using *of-vars*
  by *blast*
then show *?thesis*
  unfolding *WEST-simp-trace.simps WEST-simp-state.simps*
  using *WEST-simp-bitwise-identity assms WEST-get-state-length*
  by *simp*
qed

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*
  using *assms* **by** *simp*

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* $\lor$ *match-regex* π *r2*
proof $-$
  {**assume** *diff0*: *count-diff r1 r2 = 0*
    then have $*$: (*WEST-simp-trace r1 r2 num-vars*) = *r1*
      using *WEST-simp-trace-identity assms diff0* **by** *fastforce*
    have *r1 = r2*
      using *count-diff-same-len assms diff0* **by** *force*
    then have *?thesis* **using** *assms* $*$ **by** *simp*
  } **moreover** {
    assume *diff1*: *count-diff r1 r2 = 1*
    then obtain *k* **where** *obt-k*: *k < length r1* $\land$ *count-diff-state* (*r1!k*) (*r2!k*) =
1
      using *count-diff-1*[*of r1 r2 num-vars*] *assms* **by** *fastforce*
    then have *length* (*r1 ! k*) = *length* (*r2 ! k*)

102

using *assms* **unfolding** *trace-regex-of-vars-def*
  by (*metis check-simp.simps*)
then obtain *ka* **where** *obt-ka*: *ka < length* (*r1!k*) ∧ (*r1!k!ka*) ≠ (*r2!k!ka*)
  using *count-diff-state-1*[*of r1!k r2!k*] *obt-k assms* **by** *blast*

let *?r1r2 = (WEST-simp-trace r1 r2 num-vars)*
have *rest-of-states*: ∀ *i<length r1 . i≠k* ⟶ *r1!i = r2!i*
  using *count-diff-1-other-states assms obt-k*
  by (*metis* (*no-types, opaque-lifting*) *check-simp.elims*(*2*) *diff1*)
have *fact1*: ⋀*i*. (*i<length r1* ∧ *i≠k*) ⟹
          ((*match-timestep* (*π!i*) (*r1!i*)) ∨ (*match-timestep* (*π!i*) (*r2!i*)))
proof −
  fix *i*
  assume *i-assms*: *i<length r1* ∧ *i≠k*
  then have *states-eq*: *r1!i = r2!i* **using** *rest-of-states* **by** *blast*
  have *?r1r2 = map* (*λk. WEST-simp-state* (*WEST-get-state r1 k num-vars*)
          (*WEST-get-state r2 k num-vars*)) [*0..<length r1*]
    using *assms*(*1*) **unfolding** *check-simp.simps WEST-simp-trace.simps*
    by (*metis* (*mono-tags, lifting*) *Max-singleton insert-absorb2*)
  then have *?r1r2!i = WEST-simp-state* (*WEST-get-state r1 i num-vars*)
          (*WEST-get-state r2 i num-vars*)
    using *i-assms* **by** *simp*
  then have *?r1r2!i = WEST-simp-state* (*r1!i*) (*r2!i*)
    using *WEST-get-state.simps i-assms*
    by (*metis assms*(*1*) *check-simp.elims*(*2*) *leD*)
  then have *?r1r2!i = r1!i*
    using *WEST-simp-state.simps states-eq*
    using *WEST-simp-bitwise.simps*
    using *WEST-simp-bitwise-identity map-nth* **by** *fastforce*
  then show ((*match-timestep* (*π!i*) (*r1!i*)) ∨ (*match-timestep* (*π!i*) (*r2!i*)))
    using *assms states-eq* **unfolding** *match-regex-def*
    by (*metis WEST-simp-trace-length check-simp.elims*(*2*) *i-assms*)
qed
have *?r1r2!k = WEST-simp-state* (*WEST-get-state r1 k num-vars*)
        (*WEST-get-state r2 k num-vars*)
    using *WEST-simp-trace.simps*[*of r1 r2 num-vars*] *obt-k* **by** *force*
then have *r1r2-k*: *?r1r2!k = WEST-simp-state* (*r1!k*) (*r2!k*)
  using *obt-k assms* **by** *auto*
then have *other-states*: ∀ *i<length* (*r1!k*). *i≠ka* ⟶ (*r1!k!i*) = (*r2!k!i*)
  using *count-diff-state-other-states*[*of r1!k r2!k ka*]
  using *obt-ka obt-k assms fact1*
  using ‹*length* (*r1 ! k*) = *length* (*r2 ! k*)› **by** *blast*
have *?r1r2!k = WEST-simp-state* (*WEST-get-state r1 k num-vars*)
        (*WEST-get-state r2 k num-vars*)
    using *WEST-simp-trace.simps*[*of r1 r2 num-vars*] *obt-k* **by** *force*
then have *r1r2-k*: *?r1r2!k = WEST-simp-state* (*r1!k*) (*r2!k*)
  using *obt-k assms* **by** *auto*
then have *other-states*: ∀ *i<length* (*r1!k*). *i≠ka* ⟶ (*r1!k!i*) = (*r2!k!i*)
  using *count-diff-state-other-states*[*of r1!k r2!k ka*]

103

using *obt-ka obt-k assms fact1*

using ‹*length (r1 ! k) = length (r2 ! k)*› **by** *blast*

have *state-fact1*: $\bigwedge i.\ (i<length\ (r1!k) \wedge i \neq ka) \implies (?r1r2!k!i) = (r1!k!i)$

proof−

  fix *i*

  assume *i-fact*: $i<length\ (r1!k) \wedge i \neq ka$

  have *length (r1 ! k) = length (r2 ! k)*

    using *assms obt-k* **unfolding** *trace-regex-of-vars-def*

    by (*simp add*: ‹*length (r1 ! k) = length (r2 ! k)*›)

  then show $(?r1r2!k!i) = (r1!k!i)$

    using *WEST-simp-state.simps*[*of r1!k r2!k*] *i-fact r1r2-k*

    by (*simp add*: *WEST-simp-bitwise-identity* ‹*length (r1 ! k) = length (r2 !*

k)› *map-nth other-states*)

qed

have *r1r2-k-ka*: *?r1r2!k!ka = WEST-simp-bitwise (r1 ! k ! ka) (r2 ! k ! ka)*

  using *WEST-simp-state.simps*[*of r1!k r2!k*] *r1r2-k obt-ka* **by** *simp*

then have *state-fact2*: *?r1r2!k!ka = S*

  using *obt-ka WEST-simp-bitwise.elims*

  by (*metis (full-types*))

then have *cases*: $(r1!k!ka = S) \vee (r2!k!ka = S)$

    $\vee(r1!k!ka = One \wedge r2!k!ka = Zero)$

    $\vee(r1!k!ka = Zero \wedge r2!k!ka = One)$

  using *r1r2-k-ka*

  by (*metis (full-types) WEST-bit.exhaust obt-ka*)

have $\bigwedge x.\ x<length\ (?r1r2 ! k) \implies$

    $(((r1 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (r1 ! k ! x = Zero \longrightarrow x \notin \pi ! k))$

    $\vee((r2 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (r2 ! k ! x = Zero \longrightarrow x \notin \pi ! k)))$

  using *state-fact1 state-fact2*

proof−

  fix *x*

  assume *x-fact*: $x < length\ (?r1r2!k)$

  {assume *x-is-ka*: $x = ka$

    then have $((?r1r2 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (?r1r2 ! k ! x = Zero$

$\longrightarrow x \notin \pi ! k))$

      using *state-fact2* **by** *simp*

  } moreover {

    assume *x-not-ka*: $x \neq ka$

    then have $?r1r2!k!x = r1!k!x$

      using *state-fact1*[*of x*] *x-fact x-not-ka*

     using *assms(3) check-simp.simps obt-k trace-regex-of-vars-def* **by** *fastforce*

    then have $(((r1 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (r1 ! k ! x = Zero \longrightarrow x$

$\notin \pi ! k))$

      $\vee((r2 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (r2 ! k ! x = Zero \longrightarrow x \notin \pi ! k)))$

      using *cases assms WEST-simp-trace-length check-simp.elims obt-k x-fact*

      unfolding *match-timestep-def*

      by (*metis (mono-tags, lifting) match-regex-def match-timestep-def*)

  }

  ultimately show $(((r1 ! k ! x = One \longrightarrow x \in \pi ! k) \wedge (r1 ! k ! x = Zero$

$\longrightarrow x \notin \pi ! k))$

$\lor((r2 \ ! \ k \ ! \ x = One \longrightarrow x \in \pi \ ! \ k) \land (r2 \ ! \ k \ ! \ x = Zero \longrightarrow x \notin \pi \ ! \ k)))$
by (*metis obt-ka*)
  qed
    then have *fact2*: $((match\text{-}timestep \ (\pi!k) \ (r1!k)) \lor (match\text{-}timestep \ (\pi!k)$
$(r2!k)))$
      unfolding *match-timestep-def*
       by (*metis WEST-simp-state-num-vars ‹length (r1 ! k) = length (r2 ! k)›*
other-states r1r2-k)

  have $\forall \ time{<}length \ ?r1r2. \ ((match\text{-}timestep \ (\pi!time) \ (r1!time)) \lor (match\text{-}timestep$
$(\pi!time) \ (r2!time)))$
    using *fact1 fact2 assms*
    by (*metis WEST-simp-trace-length check-simp.elims(2)*)
  then have *?thesis*
    using *assms WEST-simp-trace-length* **unfolding** *match-regex-def*
    by (*smt (verit) check-simp.elims(2) rest-of-states*)
 }
 ultimately show *?thesis*
  using *check-simp.simps[of r1 r2] assms(1)* **by** *force*
qed

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 $\lor$ match-regex π r2*
 shows *match-regex π (WEST-simp-trace r1 r2 num-vars)*
proof−
 {assume *diff0*: *count-diff r1 r2 = 0*
  then have $*$: *(WEST-simp-trace r1 r2 num-vars) = r1*
    using *WEST-simp-trace-identity assms diff0* **by** *fastforce*
  have *r1 = r2*
    using *count-diff-same-len assms diff0* **by** *force*
  then have *?thesis* **using** *assms* $*$ **by** *simp*
 } moreover {
  assume *diff1*: *count-diff r1 r2 = 1*
  then obtain *k* **where** *obt-k*: $k < length \ r1 \land count\text{-}diff\text{-}state \ (r1!k) \ (r2!k) =$
1
    using *count-diff-1[of r1 r2 num-vars] assms* **by** *fastforce*
  then have *length (r1 ! k) = length (r2 ! k)*
    using *assms* **unfolding** *trace-regex-of-vars-def*
    by (*metis check-simp.simps*)
  then obtain *ka* **where** *obt-ka*: $ka < length \ (r1!k) \land (r1!k!ka) \neq (r2!k!ka)$
    using *count-diff-state-1[of r1!k r2!k] obt-k assms* **by** *blast*
  let *?r1r2 = (WEST-simp-trace r1 r2 num-vars)*
  have *rest-of-states*: $\forall \ i{<}length \ r1. \ i \neq k \longrightarrow r1!i = r2!i$
    using *count-diff-1-other-states assms obt-k*
    by (*metis (no-types, opaque-lifting) check-simp.elims(2) diff1*)

have *fact1*: $\bigwedge$*i*. (*i<length r1* $\wedge$ *i$\neq$k*) $\Longrightarrow$ *match-timestep* (π!*i*) (*?r1r2!i*)
proof−
  fix *i*
  assume *i-assms*: *i<length r1* $\wedge$ *i$\neq$k*
  then have *states-eq*: *r1!i = r2!i* **using** *rest-of-states* **by** *blast*
  have *?r1r2 = map* (λ*k. WEST-simp-state* (*WEST-get-state r1 k num-vars*)
      (*WEST-get-state r2 k num-vars*)) [*0..<length r1*]
    using *assms*(*1*) **unfolding** *check-simp.simps WEST-simp-trace.simps*
    by (*metis* (*mono-tags, lifting*) *Max-singleton insert-absorb2*)
  then have *?r1r2!i = WEST-simp-state* (*WEST-get-state r1 i num-vars*)
      (*WEST-get-state r2 i num-vars*)
    using *i-assms* **by** *simp*
  then have *?r1r2!i = WEST-simp-state* (*r1!i*) (*r2!i*)
    using *WEST-get-state.simps i-assms*
    by (*metis assms*(*1*) *check-simp.elims*(*2*) *leD*)
  then have *?r1r2!i = r1!i*
    using *WEST-simp-state.simps states-eq*
    using *WEST-simp-bitwise.simps*
    using *WEST-simp-bitwise-identity map-nth* **by** *fastforce*
  then show *match-timestep* (π!*i*) (*?r1r2!i*)
    using *assms*(*4*) *states-eq* **unfolding** *match-regex-def*
    by (*metis assms*(*1*) *check-simp.elims*(*2*) *i-assms*)
qed
have *?r1r2!k = WEST-simp-state* (*WEST-get-state r1 k num-vars*)
    (*WEST-get-state r2 k num-vars*)
  using *WEST-simp-trace.simps*[*of r1 r2 num-vars*] *obt-k* **by** *force*
then have *r1r2-k*: *?r1r2!k = WEST-simp-state* (*r1!k*) (*r2!k*)
  using *obt-k assms* **by** *auto*
then have *other-states*: $\forall$ *i<length* (*r1!k*). *i$\neq$ka* $\longrightarrow$ (*r1!k!i*) = (*r2!k!i*)
  using *count-diff-state-other-states*[*of r1!k r2!k ka*]
  using *obt-ka obt-k assms fact1*
  using ‹*length* (*r1 ! k*) = *length* (*r2 ! k*)› **by** *blast*
have *state-fact1*: $\bigwedge$*i*. (*i<length* (*r1!k*) $\wedge$ *i$\neq$ka*) $\Longrightarrow$ (*?r1r2!k!i*) = (*r1!k!i*)
proof−
  fix *i*
  assume *i-fact*: *i<length* (*r1!k*) $\wedge$ *i$\neq$ka*
  have *length* (*r1 ! k*) = *length* (*r2 ! k*)
    using *assms obt-k* **unfolding** *trace-regex-of-vars-def*
    by (*simp add:* ‹*length* (*r1 ! k*) = *length* (*r2 ! k*)›)
  then show (*?r1r2!k!i*) = (*r1!k!i*)
    using *WEST-simp-state.simps*[*of r1!k r2!k*] *i-fact r1r2-k*
    by (*simp add: WEST-simp-bitwise-identity* ‹*length* (*r1 ! k*) = *length* (*r2 !*
k)› *map-nth other-states*)
qed
have *?r1r2!k!ka = WEST-simp-bitwise* (*r1 ! k ! ka*) (*r2 ! k ! ka*)
  using *WEST-simp-state.simps*[*of r1!k r2!k*] *r1r2-k obt-ka* **by** *simp*
then have *state-fact2*: *?r1r2!k!ka = S*
  using *obt-ka WEST-simp-bitwise.elims*
  by (*metis* (*full-types*))

106

have *match-state*: *match-timestep* $(\pi!k)$ $(r1!k) \lor$ *match-timestep* $(\pi!k)$ $(r2!k)$
  using *assms(4)* *obt-k* **unfolding** *match-regex-def*
  by (*metis assms(1) check-simp.elims(2)*)
have $\bigwedge x.\ x{<}length\ (?r1r2\ !\ k) \Longrightarrow$
    $((?r1r2\ !\ k\ !\ x = One \longrightarrow x \in \pi\ !\ k) \land (?r1r2\ !\ k\ !\ x = Zero \longrightarrow x \notin \pi\ !\ k))$
  using *state-fact1 state-fact2 match-state*
  proof −
  fix x
  assume *x-fact*: $x < length\ (?r1r2!k)$
  {**assume** *x-is-ka*: $x = ka$
   then have $((?r1r2\ !\ k\ !\ x = One \longrightarrow x \in \pi\ !\ k) \land (?r1r2\ !\ k\ !\ x = Zero \longrightarrow x \notin \pi\ !\ k))$
    using *state-fact2* **by** *simp*
  } **moreover** {
   assume *x-not-ka*: $x \neq ka$
   then have $?r1r2!k!x = r1!k!x$
    using *state-fact1*[*of x*] *x-fact x-not-ka*
    using *assms(3) check-simp.simps obt-k trace-regex-of-vars-def* **by** *fastforce*
   then have $((?r1r2\ !\ k\ !\ x = One \longrightarrow x \in \pi\ !\ k) \land (?r1r2\ !\ k\ !\ x = Zero \longrightarrow x \notin \pi\ !\ k))$
    using *match-state* **unfolding** *match-timestep-def*
    by (*smt* (*verit, best*) *WEST-simp-trace-length WEST-simp-trace-num-vars* ‹$\forall i{<}length\ (r1\ !\ k).\ i \neq ka \longrightarrow r1\ !\ k\ !\ i = r2\ !\ k\ !\ i$› *assms(1) assms(2) assms(3) check-simp.simps obt-k trace-regex-of-vars-def x-fact x-not-ka*)
  }
  ultimately show $((?r1r2\ !\ k\ !\ x = One \longrightarrow x \in \pi\ !\ k) \land (?r1r2\ !\ k\ !\ x = Zero \longrightarrow x \notin \pi\ !\ k))$
   by *blast*
  qed
  then have *fact2*: *match-timestep* $(\pi\ !\ k)$ $(?r1r2\ !\ k)$
   unfolding *match-timestep-def* **by** *argo*
  have $\forall time{<}length\ ?r1r2.\ match\text{-}timestep\ (\pi\ !\ time)\ (?r1r2\ !\ time)$
   using *fact1 fact2 assms*
   by (*metis WEST-simp-trace-length check-simp.elims(2)*)
  then have *?thesis*
   using *assms WEST-simp-trace-length* **unfolding** *match-regex-def*
   by (*metis* (*no-types, lifting*) *check-simp.simps*)
 }
 ultimately show *?thesis* **using** *check-simp.simps*[*of r1 r2*] *assms(1)* **by** *force*
qed

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* $\lor$ *match-regex* π *r2*

using *assms WEST-simp-trace-correct-forward WEST-simp-trace-correct-converse*
by *metis*

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 ∃*j. j < length* (*enum-pairs L*) ∧ *j ≥ i* ∧
                    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 −
  obtain *min-j* **where** *obt-min-j*: *min-j = Min* {*j. j < length* (*enum-pairs L*) ∧ *j*
≥ *i* ∧
                    check-simp (*L ! fst* (*enum-pairs L ! j*))
                              (*L ! snd* (*enum-pairs L ! j*))}
    by *blast*
  then have *min-j-props*: *min-j < length* (*enum-pairs L*) ∧ *min-j ≥ i* ∧
                    check-simp (*L ! fst* (*enum-pairs L ! min-j*))
                              (*L ! snd* (*enum-pairs L ! min-j*))
    using *Min-in*[*of* {*j. j < length* (*enum-pairs L*) ∧
            i ≤ j ∧
            check-simp (*L ! fst* (*enum-pairs L ! j*))
            (*L ! snd* (*enum-pairs L ! j*))}]
  by (*smt* (*verit, ccfv-threshold*) *assms*(*2*) *empty-Collect-eq finite-nat-set-iff-bounded mem-Collect-eq*)
  let *?newL = update-L L* (*enum-pairs L ! min-j*) *num-vars*
  have *length-newL*: *length ?newL = length L − 1*
    using *update-L-length assms min-j-props* **by** *auto*
  have *simp-L = WEST-simp-helper ?newL* (*enum-pairs ?newL*) *0 num-vars*
    using *WEST-simp-helper-can-simp*[*OF assms*(*1*) *assms*(*2*) *obt-min-j, of ?newL*] *assms*
    by *blast*
  then show *?thesis*
    using *assms WEST-simp-helper-length length-newL*
  by (*metis* (*no-types, opaque-lifting*) *diff-diff-cancel enum-pairs-bound gr0I linorder-not-le nth-mem*
      order-less-imp-le zero-less-diff zero-neq-one)
qed

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*
  using *WEST-simp-helper-can-simp*[*of K L 0 num-vars*] *assms WEST-simp-helper-cant-simp*
  by (*metis* (*no-types, lifting*) *WEST-simp-helper-can-simp-bound gr-zeroI less-irrefl-nat less-nat-zero-code*)

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 ∃ *min-j*.
      (*min-j < length* (*enum-pairs L*) ∧
      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
        ∧ *check-simp* (*L ! fst* (*enum-pairs L ! min-j*)) (*L ! snd* (*enum-pairs L !*
min-j)))
  using *assms*
proof −
  have ∃ *j<length* (*enum-pairs L*).
      0 ≤ j ∧
      check-simp (*L ! fst* (*enum-pairs L ! j*))
       (*L ! snd* (*enum-pairs L ! j*))
    using *assms WEST-simp-helper-can-simp*[*of K L 0 num-vars*]
    by (*metis* (*no-types, lifting*) *WEST-simp-helper-cant-simp less-irrefl-nat*)
  then obtain *min-j* **where** *obt-min-j*: *min-j = Min{j. j<length* (*enum-pairs L*)
∧
      0 ≤ j ∧ check-simp (*L ! fst* (*enum-pairs L ! j*))
                    (*L ! snd* (*enum-pairs L ! j*))}
    by *blast*
  then have *min-j-props*: *min-j<length* (*enum-pairs L*) ∧
      0 ≤ min-j ∧ check-simp (*L ! fst* (*enum-pairs L ! min-j*))
                    (*L ! snd* (*enum-pairs L ! min-j*))
    using *Min-in*
      by (*smt* (*verit*) ‹∃ *j<length* (*enum-pairs L*). *0 ≤ j ∧ check-simp* (*L ! fst*
(*enum-pairs L ! j*)) (*L ! snd* (*enum-pairs L ! j*))› *empty-def finite-nat-set-iff-bounded*
mem-Collect-eq)
  let *?newL = update-L L* (*enum-pairs L ! min-j*) *num-vars*
  have *K = WEST-simp-helper ?newL* (*enum-pairs ?newL*) *0 num-vars*
    using *obt-min-j assms*
   using *WEST-simp-helper-can-simp* ‹∃ *j<length* (*enum-pairs L*). *0 ≤ j ∧ check-simp*
(*L ! fst* (*enum-pairs L ! j*)) (*L ! snd* (*enum-pairs L ! j*))› *dual-order.strict-trans2*
by *blast*
  then show *?thesis*
    using *assms min-j-props* **by** *blast*
qed

lemma *remove-element-at-index-subset*:
  fixes *i*::*nat*
  assumes *i < length L*
  shows *set* (*remove-element-at-index i L*) ⊆ *set L*

proof −
  have *fact1*: *set* (*take i L*) ⊆ *set L*
    using *assms* **unfolding** *remove-element-at-index.simps*
    by (*meson set-take-subset*)
  have *fact2*: *set* (*drop* (*i + 1*) *L*) ⊆ *set L*
    using *assms* **unfolding** *remove-element-at-index.simps*
    by (*simp add*: *set-drop-subset*)
  have *set* (*take i L* @ *drop* (*i + 1*) *L*) = *set* (*take i L*) ∪ *set* (*drop* (*i + 1*) *L*)
    by *simp*
  then show *?thesis*
    using *fact1 fact2* **unfolding** *remove-element-at-index.simps*
    by *blast*
qed

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*
  using *assms*
proof (*induct length L* − *length K arbitrary*: *K L num-vars rule*: *less-induct*)
  case *less*
  {assume *same-len*: *length K = length L*
    then have *K = L*
      using *WEST-simp-helper-same-length*[*OF less.prems*(*1*) *less.prems*(*3*)] **by**
blast
    then have *?case* **using** *less* **by** *blast*
  } moreover {
    assume *diff-len*: *length K* ≠ *length L*
    then have *K-le-L*: *length L > length K*
      using *less*(*4*) *WEST-simp-helper-length*[*of L 0 num-vars*] **by** *simp*

    then obtain *min-j* **where** *obt-min-j*: *min-j < length* (*enum-pairs L*) ∧
    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
    ∧ *check-simp* (*L ! fst* (*enum-pairs L ! min-j*)) (*L ! snd* (*enum-pairs L ! min-j*))
      using *WEST-simp-helper-less-length less.prems* **by** *blast*
    let *?nextL* = (*update-L L* ((*enum-pairs L*)!*min-j*) *num-vars*)
    let *?simp-nextL* = *WEST-simp-helper ?nextL* (*enum-pairs ?nextL*) *0 num-vars*
    have *length ?nextL = length L* − *1*
      using *update-L-length obt-min-j* **by** *force*
    then have *cond1*: *length ?nextL* − *length K < length L* − *length K*
      using *obt-min-j*
      by (*metis K-le-L Suc-diff-Suc diff-Suc-eq-diff-pred lessI*)
    have *cond2*: *WEST-regex-of-vars* (*update-L L* (*enum-pairs L ! min-j*) *num-vars*)
num-vars
        using *update-L-preserves-num-vars*[*of L num-vars* (*enum-pairs L*)!*min-j*

?nextL]
    using *less*
    using *nth-mem obt-min-j* **by** *blast*
  let *?h = (enum-pairs L ! min-j)*
  let *?updateL = (update-L L ?h num-vars)*
  have *match π ?updateL*
    using *less.hyps*[*OF cond1 cond2 less.prems*(*2*)] *obt-min-j* **by** *blast*
  have *updateL-eq*: *?updateL = remove-element-at-index* (*fst ?h*)
        (*remove-element-at-index* (*snd ?h*) *L*) @
        [*WEST-simp-trace* (*L ! fst ?h*) (*L ! snd ?h*) *num-vars*]
    using *update-L.simps*[*of L ?h num-vars*] **by** *blast*
  have *fst-le-snd*: *fst ?h < snd ?h*
    using *enum-pairs-fact nth-mem obt-min-j* **by** *blast*
  have *h-bound*: *snd ?h < length L*
    using *enum-pairs-bound*[*of L*] *obt-min-j*
    using *nth-mem* **by** *blast*
  {assume *match-simped-part*: *match π* [*WEST-simp-trace* (*L ! fst ?h*) (*L ! snd ?h*) *num-vars*]
    have *cond1*: *check-simp* (*L ! fst* (*enum-pairs L ! min-j*))
  (*L ! snd* (*enum-pairs L ! min-j*))
      using *obt-min-j* **by** *blast*
    have *cond2*: *trace-regex-of-vars* (*L ! fst* (*enum-pairs L ! min-j*)) *num-vars*
      using *less.prems*(*1*) *fst-le-snd h-bound* **unfolding** *WEST-regex-of-vars-def*
      by (*meson order-less-trans*)
    have *cond3*: *trace-regex-of-vars* (*L ! snd* (*enum-pairs L ! min-j*)) *num-vars*
      using *less.prems*(*1*) *fst-le-snd h-bound* **unfolding** *WEST-regex-of-vars-def*
      by (*meson order-less-trans*)
    have *match-either*: *match-regex π* (*L ! fst ?h*) ∨ *match-regex π* (*L ! snd ?h*)
      using *WEST-simp-trace-correct-forward*[*OF cond1 cond2 cond3*]
      using *match-simped-part* **unfolding** *match-def* **by** *force*
    then have *?case* **unfolding** *match-def*
      using *fst-le-snd h-bound*
      by (*meson Suc-lessD less-trans-Suc*)
  } moreover {
    let *?other-part = (remove-element-at-index* (*fst ?h*)
        (*remove-element-at-index* (*snd ?h*) *L*))
    assume *match-other-part*: *match π ?other-part*
    have *set* (*remove-element-at-index* (*fst* (*enum-pairs L ! min-j*))
      (*remove-element-at-index* (*snd* (*enum-pairs L ! min-j*)) *L*))
      ⊆ *set* (*remove-element-at-index* (*snd* (*enum-pairs L ! min-j*)) *L*)
      using *fst-le-snd h-bound remove-element-at-index-subset*
        [*of fst* (*enum-pairs L ! min-j*) (*remove-element-at-index* (*snd* (*enum-pairs L ! min-j*)) *L*)]
      by *simp*
    then have *other-part-subset*: *set ?other-part* ⊆ *set L*
      using *fst-le-snd h-bound remove-element-at-index-subset*
        [*of snd* (*enum-pairs L ! min-j*) *L*] **by** *blast*
    then obtain *idx* **where** *obt-idx*: *match-regex π* (*?other-part!idx*) ∧ *idx < length ?other-part*

using *match-other-part* **unfolding** *match-def* **by** *metis*
then have (*?other-part!idx*) ∈ *set L*
using *updateL-eq fst-le-snd h-bound other-part-subset*
by (*meson in-mono nth-mem*)
then have *?case*
using *obt-idx* **unfolding** *match-def*
by (*metis in-set-conv-nth*)
}
ultimately have *?case* **using** *updateL-eq WEST-or-correct*
by (*metis ‹match π (update-L L (enum-pairs L ! min-j) num-vars)›*)
}
ultimately show *?case* **by** *blast*
qed

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−
{assume *L-small*: *length L ≤ 2*
then have (*remove-element-at-index j1* (*remove-element-at-index j2 L*)) = []
unfolding *remove-element-at-index.simps* **using** *assms* **by** *simp*
then have *?thesis* **using** *assms* **by** *auto*
} moreover {
assume *L-big*: *length L ≥ 3*
then have *length* (*remove-element-at-index j1* (*remove-element-at-index j2 L*))
≥ *1*
unfolding *remove-element-at-index.simps* **using** *assms* **by** *auto*
{assume *in-front*: *i < j1*
then have *i-bound*: *i < length* (*take j2 L*)
using *assms* **by** *simp*
have *L!i = (take j1 L)!i*
using *in-front assms* **by** *auto*
then have *L!i ∈ set* (*take j1 L*)
using *in-front assms*
by (*metis length-take min-less-iff-conj nth-mem*)
then have *Li-in*: *L!i ∈ set* (*take j1* (*take j2 L*))
using *assms* **by** *auto*
have *set* (*take j1* (*take j2 L @ drop* (*j2 + 1*) *L*)) = *set* (*take j1* (*take j2 L*))
using *assms(1) assms(2)* **by** *simp*
then have *L!i ∈ set* (*take j1* (*take j2 L @ drop* (*j2 + 1*) *L*))
using *Li-in* **by** *blast*
then have *?thesis* **unfolding** *remove-element-at-index.simps*
by *auto*

112

```
    } moreover {
      assume in-middle: j1 < i ∧ i < j2
      then have i-len: i < length (take j2 L)
        using assms by auto
      then have Li-eq: L!i = (take j2 L)!i
        by simp
      then have L!i ∈ set (take j2  L)
        using ‹i < length (take j2 L)› in-set-conv-nth by fastforce
      have i−(j1+1) < length (drop (j1 + 1) (take j2 L @ drop (j2 + 1) L))
        using assms i-len in-middle by auto
      then have L!i = (drop (j1 + 1) (take j2 L)) ! (i−(j1+1))
        using assms i-len in-middle Li-eq by auto
      then have L!i ∈ set (drop (j1 + 1) (take j2 L))
          by (metis diff-less-mono i-len in-middle length-drop less-iff-succ-less-eq
nth-mem)
      then have ?thesis
        unfolding remove-element-at-index.simps by auto
    } moreover {
      assume in-back: j2 < i
      then have i−(j2+1) < length (drop (j2 + 1) L)
        using assms by auto
      then have Li-eq: L!i = (drop (j2 + 1) L)!(i−(j2+1))
        using assms in-back by auto
      then have L!i ∈ set (drop (j2 + 1) L)
        by (metis ‹i − (j2 + 1) < length (drop (j2 + 1) L)› nth-mem)
      then have L!i ∈ set(drop (j1 + 1) (take j2 L @ drop (j2 + 1) L))
        using assms by auto
      then have ?thesis unfolding remove-element-at-index.simps
        by auto
    }
    ultimately have ?thesis unfolding remove-element-at-index.simps
      using assms L-big by linarith
  }
  ultimately show ?thesis by linarith
qed

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−
  obtain i where i-obt: i < length L ∧ match-regex π (L!i)
    using assms(2) unfolding match-def by metis
  have fst-le-snd: fst h < snd h
    using assms enum-pairs-fact by auto
  have h-bound: snd h < length L
    using assms enum-pairs-bound
```

113

by *blast*
　{assume *in-simped*: $i = fst\ h \lor i = snd\ h$
　　let *?r1* = $(L!(fst\ h))$
　　let *?r2* = $(L!(snd\ h))$
　　have *match-regex* π (*WEST-simp-trace* (L ! *fst h*) (L ! *snd h*) *num-var*)
　　　using *WEST-simp-trace-correct-converse*[*of ?r1 ?r2 num-var*]
　　　using *assms* **unfolding** *WEST-regex-of-vars-def*
　　by (*metis* (*mono-tags*, *lifting*) *WEST-simp-trace-correct-converse i-obt enum-pairs-bound*
enum-pairs-fact in-simped order.strict-trans)
　　then have *?thesis*
　　　unfolding *update-L.simps match-regex-def*
　　by (*metis* (*no-types*, *lifting*) *WEST-or-correct* ‹*match-regex* π (*WEST-simp-trace*
(L ! *fst h*) (L ! *snd h*) *num-var*)› *append.right-neutral append-eq-append-conv2 im-*
possible-Cons le-eq-less-or-eq match-def nat-le-linear nth-append-length same-append-eq)
　} moreover {
　　assume *in-rest*: $i \neq fst\ h \land i \neq snd\ h$
　　have $L!i \in set\ L$
　　　using *i-obt* **by** *simp*
　　have $L!i \in set$ (*remove-element-at-index* (*fst h*) (*remove-element-at-index* (*snd*
h) *L*))
　　　using *fst-le-snd h-bound i-obt in-rest*
　　　using *remove-element-at-index-fact* **by** *blast*
　　then have *match* π
　　(*remove-element-at-index* (*fst h*) (*remove-element-at-index* (*snd h*) *L*))
　　　unfolding *match-def* **using** *i-obt*
　　　by (*metis in-set-conv-nth*)
　　then have *?thesis* **unfolding** *update-L.simps match-def*
　　　using *WEST-or-correct match-def* **by** *blast*
　}
　ultimately show *?thesis* **by** *blast*
qed

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*
　using *assms*
　proof (*induct length L arbitrary*: *K L i num-vars rule*: *less-induct*)
　　case *less*
　　{assume ∗: *length* (*enum-pairs L*) $\leq i$
　　　then have $K = L$
　　　　using *less*(*4*)
　　　　using *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *i num-vars*]
　　　　by *argo*
　　　then have *?case*
　　　　using *less*(*3*)
　　　　by *blast*

} **moreover {assume** ∗: *length* (*enum-pairs L*) > *i*

   {assume ∗∗: ∃ *j. j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧ *check-simp* (*L* ! *fst*
(*enum-pairs L* ! *j*))
            (*L* ! *snd* (*enum-pairs L* ! *j*))
   then obtain *j-min* **where** *j-min-obt*: *j-min* = *Min* {*j. j* < *length* (*enum-pairs*
L) ∧ *j* ≥ *i* ∧ *check-simp* (*L* ! *fst* (*enum-pairs L* ! *j*))
            (*L* ! *snd* (*enum-pairs L* ! *j*))}
       by *blast*
   have *j-min-props*: *j-min* < *length* (*enum-pairs L*) ∧ *j-min* ≥ *i* ∧ *check-simp*
(*L* ! *fst* (*enum-pairs L* ! *j-min*))
            (*L* ! *snd* (*enum-pairs L* ! *j-min*))
       using *j-min-obt Min-in*
       by (*metis* (*mono-tags, lifting*) ∗∗ *Collect-empty-eq finite-nat-set-iff-bounded*
mem-Collect-eq)
   have *K-eq*: *K* = (*let newL* =
                 update-L L (*enum-pairs L* ! *j-min*)
                  num-vars
              in WEST-simp-helper newL
                 (*enum-pairs newL*) *0 num-vars*)
       using *less*(*4*) ∗ ∗∗ *WEST-simp-helper.simps*[*of L* (*enum-pairs L*) *j-min*
num-vars]
       using *WEST-simp-helper-can-simp*
       by (*metis* (*no-types, lifting*) *j-min-obt*)
   let *?h* = (*enum-pairs L* ! *j-min*)
   have *cond1*: *length* (*update-L L* (*enum-pairs L* ! *j-min*) *num-vars*) < *length*
L
       using *update-L-length*[*of ?h L num-vars*] *j-min-props*
       by (*metis diff-less enum-pairs-bound less-nat-zero-code less-one not-gr-zero*
nth-mem)
       have *cond2*: *WEST-regex-of-vars* (*update-L L* (*enum-pairs L* ! *j-min*)
num-vars) *num-vars*
       using *update-L-preserves-num-vars*[*of L num-vars ?h K*] *less*
       using *j-min-props nth-mem update-L-preserves-num-vars* **by** *blast*
   have *cond3*: *match* π (*update-L L* (*enum-pairs L* ! *j-min*) *num-vars*)
       using *update-L-match*[*OF less*(*2*) *less*(*3*), *of ?h*] *j-min-props*
       by *fastforce*
   have *?case*
       using *less*(*1*)[*OF cond1 cond2, of K*]
       using *K-eq*
       by (*metis cond3*)
    }
  moreover {assume ∗∗: ¬(∃ *j. j* < *length* (*enum-pairs L*) ∧ *j* ≥ *i* ∧ *check-simp*
(*L* ! *fst* (*enum-pairs L* ! *j*))
            (*L* ! *snd* (*enum-pairs L* ! *j*)))
   then have *K-eq*: *K* = *L*
       using *WEST-simp-helper-cant-simp less.prems*(*3*)
       by *presburger*
   then have *?case*
       using *less*(*3*)

      by *blast*
    }
    ultimately have *?case*
      by *blast*
  }
  ultimately show *?case*
    by *linarith*
qed

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*
  unfolding *WEST-simp.simps* **using** *WEST-simp-helper-correct-forward assms*
  by (*metis WEST-simp.elims*)

lemma *simp-correct-converse*:
  assumes *WEST-regex-of-vars L num-vars*
  assumes *match π L*
  shows *match π (WEST-simp L num-vars)*
  unfolding *WEST-simp.simps* **using** *WEST-simp-helper-correct-converse assms*
  by *blast*

lemma *simp-correct*:
  assumes *WEST-regex-of-vars L num-vars*
  shows *match π (WEST-simp L num-vars)* $\longleftrightarrow$ *match π L*
  using *simp-correct-forward simp-correct-converse assms*
  by *blast*

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* $\land$ *match π L2* $\longleftrightarrow$ *match π (WEST-and-simp L1 L2 n)*
proof −
  show *?thesis*
    using *simp-correct*[*of WEST-and L1 L2 n π*] *assms WEST-and-correct*[*of L1 n L2 π*]
    unfolding *WEST-and-simp.simps*
    using *WEST-and-num-vars* **by** *blast*
qed

116

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* $\lor$ *match* π *L2* $\longleftrightarrow$ *match* π (*WEST-or-simp L1 L2 n*)
proof $-$
  show *?thesis*
    using *simp-correct*[*of L1@L2 n* π]
    using *assms WEST-or-correct*[*of* π *L1 L2*]
    unfolding *WEST-or-simp.simps*
    using *WEST-or-num-vars* **by** *blast*
qed

3.9   Facts about the WEST future operator

lemma *WEST-future-correct-forward*:
  assumes $\bigwedge\pi$. (*length* π $\geq$ *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* π $\geq$ (*complen-mltl F*) + *b*
  assumes *match* π (*WEST-future L a b num-vars*)
  shows $\pi \models_m$ (F_m [*a,b*] *F*)
  using *assms*
proof(*induct b* $-$ *a arbitrary:* π *L F a b*)
  case *0*
  then have *a-eq-b*: $a = b$ **by** *simp*
  then have *WEST-future L a b num-vars* = *shift L num-vars a*
    using *WEST-future.simps*[*of L a b num-vars*] **by** *simp*
  then have *match* π (*shift L num-vars a*)
    using *0* **by** *simp*
  then have *match-dropa-L*: *match* (*drop a* π) *L*
    using *shift-match*[*of a* π *L num-vars*] *0 a-eq-b* **by** *auto*

  have *complen-mltl F* $\leq$ *length* (*drop a* π)
    using *0*(*2*)[*of* (*drop a* π)] *0*(*6*) *a-eq-b complen-geq-one*[*of F*] **by** *simp*
  then have *semantics-mltl* (*drop a* π) *F*
    using *0*(*2*)[*of* (*drop a* π)] *match-dropa-L* **by** *blast*
  then have $\exists i.$ ($a \leq i \land i \leq b$) $\land$ *semantics-mltl* (*drop i* π) *F*
    using *a-eq-b* **by** *blast*
  then show *?case* **unfolding** *semantics-mltl.simps*
    using *0*(*1*, *6*) *a-eq-b complen-geq-one*[*of F*] **by** *simp*
next
  case (*Suc x*)
  then have *b-asucx*: $b = a + (Suc\ x)$ **by** *simp*
  then have (*WEST-future L a b num-vars*) = *WEST-or-simp* (*shift L num-vars b*) (*WEST-future L a* (*b* $-$ *1*) *num-vars*) *num-vars*

117

using *WEST-future.simps*[*of L a b num-vars*]
 by (*metis Suc.hyps*(*2*) *Suc.prems*(*4*) *add-eq-0-iff-both-eq-0 cancel-comm-monoid-add-class.diff-cancel nat-less-le plus-1-eq-Suc zero-neq-one*)
  then have (*WEST-future L a b num-vars*) = *WEST-or-simp* (*shift L num-vars b*) (*WEST-future L a* (*a + x*) *num-vars*) *num-vars*
   using *b-asucx*
   by (*metis add-diff-cancel-left′ le-add1 ordered-cancel-comm-monoid-diff-class.diff-add-assoc plus-1-eq-Suc*)
  {**assume** *match-head*: *match π* (*shift L num-vars b*)
   then obtain *i* **where** *match-regex π* (*shift L num-vars b* ! *i*)
    unfolding *match-def* **by** *metis*
   have *match* (*drop b π*) *L*
    using *shift-match*[*of b π L num-vars*] *Suc*(*7*) *match-head* **by** *auto*
   then have *semantics-mltl* (*drop b π*) *F*
    using *Suc* **by** *simp*
   then have ∃ *i*. (*a* ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i π*) *F*
    using *Suc.prems*(*4*) **by** *auto*
  } **moreover** {
   assume *match-tail*: *match π* (*WEST-future L a* (*a + x*) *num-vars*)
   have *hyp1*: *x* = *b − 1 − a* **using** *Suc* **by** *simp*
   have *hyp2*: (⋀*π*. *complen-mltl F* ≤ *length π* ⟶ *match π L* = *semantics-mltl π F*)
    using *Suc.prems* **by** *blast*
   have *hyp3*: *WEST-regex-of-vars L num-vars* **using** *Suc.prems* **by** *simp*
   have *hyp4*: *WEST-num-vars F* ≤ *num-vars* **using** *Suc.prems* **by** *blast*
   have *hyp5*: *a* ≤ *b − 1* **using** *Suc.prems Suc.hyps* **by** *auto*
   have *hyp6*: *complen-mltl F* + (*b − 1*) ≤ *length π* **using** *Suc.prems* **by** *simp*
   have *hyp7*: *match π* (*WEST-future L a* (*b − 1*) *num-vars*)
    using *match-tail Suc.hyps*(*2*)
    using *b-asucx* **by** *fastforce*
   have *semantics-mltl π* (*Future-mltl a* (*a+x*) *F*)
    using *Suc.hyps*(*1*)[*of b−1 a F L π, OF hyp1 hyp2 hyp3 hyp4 hyp5 hyp6 hyp7*]
    using *b-asucx* **by** *simp*
   then have ∃ *i*. (*a* ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i π*) *F*
    unfolding *semantics-mltl.simps b-asucx* **by** *auto*
  }
  ultimately have ∃ *i*. (*a* ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i π*) *F*
   unfolding *match-def*
   by (*metis Nat.add-diff-assoc Suc.prems*(*2*) *Suc.prems*(*6*) *WEST-future-num-vars WEST-or-simp-correct shift-num-vars* ‹*WEST-future L a b num-vars* = *WEST-or-simp* (*shift L num-vars b*) (*WEST-future L a* (*b − 1*) *num-vars*) *num-vars*› ‹*match π* (*WEST-future L a* (*a + x*) *num-vars*) ⟹ ∃ *i*. (*a* ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i π*) *F*› ‹*match π* (*shift L num-vars b*) ⟹ ∃ *i*. (*a* ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i π*) *F*› *b-asucx diff-add-inverse le-add1 plus-1-eq-Suc*)
  then show *?case*
   using *Suc* **unfolding** *semantics-mltl.simps* **by** *auto*
 qed

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)*
  using *assms*
proof(*induct b$-$a arbitrary: π L F a b*)
  case *0*
  then have *a-eq-b*: $a = b$ **by** *simp*
  then have *west-future-aa*: *WEST-future L a b num-vars = shift L num-vars a*
    using *WEST-future.simps[of L a b num-vars]* **by** *simp*
  have *match (drop a π) L*
    using *assms(1)[of drop a π] assms complen-geq-one*
    using *0.prems(1) 0.prems(5) 0.prems(6) a-eq-b le-antisym length-drop semantics-mltl.simps(7)* **by** *auto*
  then have *match π (shift L num-vars a)*
    using *shift-match-converse 0* **by** *auto*
  then show *?case* **using** *west-future-aa* **by** *simp*
next
  case *(Suc x)*
  then have *b-asucx*: $b = a + (Suc\ x)$ **by** *simp*
  then have *(WEST-future L a b num-vars) = WEST-or-simp (shift L num-vars b) (WEST-future L a (b $-$ 1) num-vars) num-vars*
    using *WEST-future.simps[of L a b num-vars]*
    by *(metis Suc.hyps(2) Zero-not-Suc cancel-comm-monoid-add-class.diff-cancel diff-is-0-eq' linorder-le-less-linear)*
  then have *(WEST-future L a b num-vars) = WEST-or-simp (shift L num-vars b) (WEST-future L a (a + x) num-vars) num-vars*
    using *b-asucx*
    by *(metis add-Suc-right diff-Suc-1)*
  {assume *sat-b*: *semantics-mltl (drop b π) F*
    then have *match (drop b π) L* **using** *Suc* **by** *simp*
    then have *match π (shift L num-vars b)*
      using *shift-match Suc*
      by *(metis add.commute add-leD1 shift-match-converse)*
    then have *?case* **using** *WEST-future.simps[of L a b num-vars] Suc*
      by *(metis Nat.add-diff-assoc WEST-future-num-vars WEST-or-simp-correct shift-num-vars ‹WEST-future L a b num-vars = WEST-or-simp (shift L num-vars b) (WEST-future L a (b $-$ 1) num-vars) num-vars› b-asucx le-add1 plus-1-eq-Suc)*
  } moreover {
    assume *sat-before-b*: *semantics-mltl π (Future-mltl a (a+x) F)*
    have *match π (WEST-future L a (a + x) num-vars)*
      using *Suc.hyps(1)[of a+x a F L π] Suc sat-before-b* **by** *simp*
    have *?case*

119

    using *WEST-future.simps*[*of L a b num-vars*] *Suc*

      by (*metis Nat.add-diff-assoc WEST-future-num-vars WEST-or-simp-correct*
shift-num-vars ‹*WEST-future L a b num-vars = WEST-or-simp* (*shift L num-vars*
b) (*WEST-future L a* (*b − 1*) *num-vars*) *num-vars*› ‹*match π* (*WEST-future L a*
(*a + x*) *num-vars*)› *diff-add-inverse le-add1 plus-1-eq-Suc*)

  }

  ultimately show *?case* **using** *b-asucx*

    by (*metis* (*no-types, lifting*) *Suc.prems*(*6*) *add-Suc-right le-SucE le-antisym*
semantics-mltl.simps(*7*))

qed

lemma *WEST-future-correct*:

  assumes $\bigwedge\pi$. (*length π ≥ 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 ≤ num-vars*

  assumes *a ≤ b*

  assumes *length π ≥* (*complen-mltl F*) *+ b*

  shows *match π* (*WEST-future L a b num-vars*) $\longleftrightarrow$

          semantics-mltl π (*Future-mltl a b F*)

  using *assms WEST-future-correct-forward WEST-future-correct-converse* **by** *blast*

3.10   Facts about the WEST global operator

lemma *WEST-global-correct-forward*:

  assumes $\bigwedge\pi$. (*length π ≥ 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 ≤ num-vars*

  assumes *a ≤ b*

  assumes *length π ≥* (*complen-mltl F*) *+ b*

  assumes *match π* (*WEST-global L a b num-vars*)

  shows *semantics-mltl π* (*Global-mltl a b F*)

  using *assms*

proof(*induct b−a arbitrary: π L F a b*)

  case *0*

  then have *a-eq-b*: *a = b* **by** *simp*

  then have *WEST-global L a b num-vars = shift L num-vars a*

    using *assms WEST-global.simps*[*of L a b num-vars*] **by** *auto*

  then have *match π* (*shift L num-vars a*) **using** *0* **by** *simp*

  then have *match* (*drop a π*) *L*

    using *shift-match*[*of a π L num-vars*] *0* **by** *auto*

  then have *semantics-mltl* (*drop a π*) *F*

    using *0*(*2*)[*of* (*drop a π*)] *complen-geq-one*[*of F*] *0 a-eq-b* **by** *auto*

  then show *?case* **using** *0*

    unfolding *semantics-mltl.simps* **by** *auto*

next

  case (*Suc x*)

then have *b-asucx*: $b = a + (Suc\ x)$ **by** *simp*
  then have ($WEST\text{-}global\ L\ a\ b\ num\text{-}vars$) = $WEST\text{-}and\text{-}simp$ ($shift\ L\ num\text{-}vars$
b) ($WEST\text{-}global\ L\ a\ (a + x)\ num\text{-}vars$) $num\text{-}vars$
    using $WEST\text{-}global.simps[of\ L\ a\ b\ num\text{-}vars]$
  by ($metis\ add\text{-}diff\text{-}cancel\text{-}left'\ cancel\text{-}comm\text{-}monoid\text{-}add\text{-}class.diff\text{-}cancel\ diff\text{-}is\text{-}0\text{-}eq$
$less\text{-}eq\text{-}Suc\text{-}le\ not\text{-}less\text{-}eq\text{-}eq\ ordered\text{-}cancel\text{-}comm\text{-}monoid\text{-}diff\text{-}class.diff\text{-}add\text{-}assoc\ plus\text{-}1\text{-}eq\text{-}Suc$
$zero\text{-}eq\text{-}add\text{-}iff\text{-}both\text{-}eq\text{-}0$)

  have *nv1*: $WEST\text{-}regex\text{-}of\text{-}vars$ ($shift\ L\ num\text{-}vars\ b$) $num\text{-}vars$
    using $shift\text{-}num\text{-}vars\ Suc$ **by** *blast*
  have *nv2*: $WEST\text{-}regex\text{-}of\text{-}vars$ ($WEST\text{-}global\ L\ a\ (a + x)\ num\text{-}vars$) $num\text{-}vars$
    using $WEST\text{-}global\text{-}num\text{-}vars\ Suc\ b\text{-}asucx$
    by ($metis\ le\text{-}iff\text{-}add$)
  have *match-h*: $match\ \pi$ ($shift\ L\ num\text{-}vars\ b$)
    using $WEST\text{-}and\text{-}correct\text{-}converse\ nv1\ nv2\ Suc$
  by ($metis\ WEST\text{-}and\text{-}simp\text{-}correct\ ‹WEST\text{-}global\ L\ a\ b\ num\text{-}vars = WEST\text{-}and\text{-}simp$
($shift\ L\ num\text{-}vars\ b$) ($WEST\text{-}global\ L\ a\ (a + x)\ num\text{-}vars$) $num\text{-}vars›$)
  then have $match\ (drop\ b\ \pi)\ L$
    using $shift\text{-}match\ Suc$
    using $add\text{-}leD2$ **by** *blast*
  then have *sat-b*: $semantics\text{-}mltl\ (drop\ b\ \pi)\ F$ **using** Suc **by** *auto*

  have *match-t*: $match\ \pi$ ($WEST\text{-}global\ L\ a\ (a + x)\ num\text{-}vars$)
    using $Suc.hyps(1)[of\ a+x\ a\ F\ L\ \pi]\ Suc\ b\text{-}asucx$
  by ($metis\ WEST\text{-}and\text{-}simp\text{-}correct\ ‹WEST\text{-}global\ L\ a\ b\ num\text{-}vars = WEST\text{-}and\text{-}simp$
($shift\ L\ num\text{-}vars\ b$) ($WEST\text{-}global\ L\ a\ (a + x)\ num\text{-}vars$) $num\text{-}vars›\ nv1\ nv2$)
  then have $semantics\text{-}mltl\ \pi$ ($Global\text{-}mltl\ a\ (a+x)\ F$)
    using Suc **by** *fastforce*
  then have *sat-before-b*: $\forall\,i.\ a \leq i \wedge i \leq a + x \longrightarrow semantics\text{-}mltl\ (drop\ i\ \pi)\ F$
    using Suc **unfolding** $semantics\text{-}mltl.simps$ **by** *auto*
  have $\forall\,i.\ a \leq i \wedge i \leq b \longrightarrow semantics\text{-}mltl\ (drop\ i\ \pi)\ F$
    using $sat\text{-}b\ sat\text{-}before\text{-}b$ **unfolding** $semantics\text{-}mltl.simps$
    by ($metis\ add\text{-}Suc\text{-}right\ b\text{-}asucx\ le\text{-}antisym\ not\text{-}less\text{-}eq\text{-}eq$)
  then show *?case* **using** Suc
    unfolding $semantics\text{-}mltl.simps$ **by** *blast*
qed

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

proof(*induct b−a arbitrary: π L F a b*)
  case *0*
  then have *a-eq-b*: $a = b$ **by** *simp*
  then have *west-global-aa*: *WEST-global L a b num-vars = shift L num-vars a*
    using *WEST-global.simps*[*of L a b num-vars*] **by** *simp*
  have *match* (*drop a π*) *L*
    using *assms*(*1*)[*of drop a π*] *assms complen-geq-one*
   by (*metis* (*mono-tags, lifting*) *0.prems*(*1*) *0.prems*(*5*) *0.prems*(*6*) *a-eq-b add-le-imp-le-diff*
drop-all le-trans length-0-conv length-drop not-one-le-zero semantics-mltl.simps(*8*))
  then have *match π* (*shift L num-vars a*)
    using *shift-match-converse 0* **by** *auto*
  then show *?case* **using** *west-global-aa* **by** *simp*
next
  case (*Suc x*)
  then have *b-asucx*: $b = a + (Suc\ x)$ **by** *simp*
  then have *west-global*: (*WEST-global L a b num-vars*) = *WEST-and-simp* (*shift*
L num-vars b) (*WEST-global L a* (*a + x*) *num-vars*) *num-vars*
    using *WEST-global.simps*[*of L a b num-vars*]
   by (*metis add-diff-cancel-left′ add-eq-0-iff-both-eq-0 cancel-comm-monoid-add-class.diff-cancel*
diff-is-0-eq less-eq-Suc-le not-less-eq-eq ordered-cancel-comm-monoid-diff-class.diff-add-assoc
plus-1-eq-Suc)

  have *sat-b*: *semantics-mltl* (*drop b π*) *F*
    using *Suc* **unfolding** *semantics-mltl.simps* **by** *auto*
  then have *match* (*drop b π*) *L* **using** *Suc* **by** *simp*
  then have *match-head*: *match π* (*shift L num-vars b*)
    using *shift-match Suc*
    by (*metis add.commute add-leD1 shift-match-converse*)

  have *sat-before-b*: *semantics-mltl π* (*Future-mltl a* (*a+x*) *F*)
    using *Suc* **unfolding** *semantics-mltl.simps* **by** *auto*
  have *match-tail*: *match π* (*WEST-global L a* (*a + x*) *num-vars*)
    using *Suc.hyps*(*1*)[*of a+x a F L π*] *Suc sat-before-b*
    by (*simp add*: *b-asucx nle-le not-less-eq-eq*)

  have *nv1*: *WEST-regex-of-vars* (*shift L num-vars b*) *num-vars*
    using *shift-num-vars Suc* **by** *blast*
  have *nv2*: *WEST-regex-of-vars* (*WEST-global L a* (*a + x*) *num-vars*) *num-vars*
    using *WEST-global-num-vars Suc b-asucx*
    by (*metis le-iff-add*)
  show *?case* **using** *b-asucx match-head match-tail*
    using *west-global WEST-and-simp-correct nv1 nv2* **by** *metis*
qed

lemma *WEST-global-correct*:
  assumes $\bigwedge\pi.$ (*length π ≥ 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 ≤ num-vars*
assumes $a \leq b$
assumes *length π ≥ (complen-mltl F) + b*
shows *match π (WEST-global L a b num-vars) ⟷*
          semantics-mltl π (Global-mltl a b F)
using *assms WEST-global-correct-forward WEST-global-correct-converse* **by** *blast*

3.11 Facts about the WEST until operator

lemma *WEST-until-correct-forward*:
  assumes $\bigwedge$*π. (length π ≥ complen-mltl F1 ⟶ (match π L1 ⟷ semantics-mltl π F1))*
  assumes $\bigwedge$*π. (length π ≥ complen-mltl F2 ⟶ (match π L2 ⟷ 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 ≤ num-vars*
  assumes *WEST-num-vars F2 ≤ num-vars*
  assumes $a \leq b$
  assumes *length π ≥ 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)*
  using *assms*
proof(*induct b−a arbitrary: π L1 L2 F1 F2 a b*)
  case *0*
  then have *a-eq-b: b = a* **by** *simp*
  have *len-xi: complen-mltl F2 + a ≤ length π*
    using *0 complen-geq-one* **by** *auto*
  have *until-aa: WEST-until L1 L2 a b num-vars = WEST-global L2 a a num-vars*
    using *WEST-until.simps[of L1 L2 a b num-vars] a-eq-b* **by** *auto*
  then have *WEST-global L2 a a num-vars = shift L2 num-vars a* **by** *auto*
  then have *match π (shift L2 num-vars a)*
    using *until-aa 0* **by** *argo*
  then have *match (drop a π) L2*
    using *shift-match[of a π L2 num-vars] 0* **by** *simp*
  then have *semantics-mltl (drop a π) F2* **using** *0* **by** *auto*
  then have *sem-until: (∃ i. (a ≤ i ∧ i ≤ a) ∧*
      semantics-mltl (drop i π) F2 ∧
      (∀ j. a ≤ j ∧ j < i ⟶ semantics-mltl (drop j π) F1))
    by *auto*
  have *max (complen-mltl F1 − 1) (complen-mltl F2) ≥ 1*
    using *complen-geq-one[of F2]* **by** *auto*
  then have *a < length π*
    using *0(9)* **using** *a-eq-b*
    unfolding *complen-mltl.simps*
    by *linarith*
  then show *?case* **using** *sem-until*
    unfolding *a-eq-b semantics-mltl.simps*

by *blast*
next
  case (*Suc x*)
  then have *b-asucx*: $b = a + (Suc\ x)$ **by** *simp*
  have *WEST-until L1 L2 a b num-vars = WEST-or-simp (WEST-until L1 L2 a*
$(a + x)$ *num-vars)*
        (*WEST-and-simp (WEST-global L1 a $(a + x)$ num-vars) (WEST-global*
L2 b b num-vars) num-vars) num-vars
    using *WEST-until.simps*[*of L1 L2 a b num-vars*] *Suc b-asucx*
    by (*metis add-Suc-right cancel-comm-monoid-add-class.diff-cancel diff-Suc-1*
less-add-Suc1 n-not-Suc-n zero-diff)

  let *?rec = WEST-until L1 L2 a $(a + x)$ num-vars*
  let *?base = WEST-and-simp (WEST-global L1 a $(a + x)$ num-vars) (WEST-global*
L2 b b num-vars) num-vars
  have *sem-until*: ($\exists\,i.\ (a \leq i \wedge i \leq b)\ \wedge$
      semantics-mltl (*drop i π*) *F2* $\wedge$
      ($\forall\,j.\ a \leq j \wedge j < i \longrightarrow$ *semantics-mltl* (*drop j π*) *F1*))
  proof−
    {**assume** *match-base*: *match π ?base*
     have *nv1*: *WEST-regex-of-vars (WEST-global L2 b b num-vars) num-vars*
      using *WEST-global-num-vars*[*of L2 num-vars b b*] *Suc* **by** *simp*
      have *nv2*: *WEST-regex-of-vars (WEST-global L1 a $(a + x)$ num-vars)*
num-vars
      using *WEST-global-num-vars*[*of L1 num-vars a a+x*] *Suc* **by** *auto*
     have *match π (WEST-global L2 b b num-vars)*
      using *match-base WEST-and-simp-correct Suc nv1 nv2* **by** *blast*
     then have *match π (shift L2 num-vars b)*
      using *WEST-global.simps*[*of L2 b b num-vars*] **by** *simp*
     then have *cond1*: *semantics-mltl* (*drop b π*) *F2*
      using *shift-match*[*of b π L2 num-vars*] *Suc* **by** *simp*

     have *match π (WEST-global L1 a $(a + x)$ num-vars)*
      using *match-base WEST-and-simp-correct Suc nv1 nv2* **by** *blast*
     then have *semantics-mltl π (Global-mltl a $(a+x)$ F1)*
      using *WEST-global-correct*[*of F1 L1 num-vars a a+x π*] *Suc* **by** *auto*
     then have $\forall\,i.\ a \leq i \wedge i \leq a + x \longrightarrow$ *semantics-mltl* (*drop i π*) *F1*
      using *Suc* **by** *auto*
     then have *cond2*: $\forall\,j.\ a \leq j \wedge j < b \longrightarrow$ *semantics-mltl* (*drop j π*) *F1*
      using *b-asucx* **by** *auto*

     have *semantics-mltl* (*drop b π*) *F2* $\wedge$
      ($\forall\,j.\ a \leq j \wedge j < b \longrightarrow$ *semantics-mltl* (*drop j π*) *F1*)
      using *cond1 cond2* **by** *auto*
     then have *?thesis* **using** *Suc* **by** *blast*
    } **moreover** {
     assume *match-rec*: *match π ?rec*
     then have *semantics-mltl π (Until-mltl F1 a $(a+x)$ F2)*
      using *Suc.hyps(1)*[*of a+x a F1 L1 F2 L2 π*] *Suc* **by** *auto*

124

then obtain *i* **where** *i-obt*: $(a \leq i \wedge i \leq (a+x)) \wedge$
    semantics-mltl (*drop i π*) *F2* $\wedge$ ($\forall j.\ a \leq j \wedge j < i \longrightarrow$ *semantics-mltl* (*drop j π*) *F1*)
      by *auto*
    have *?thesis* **using** *i-obt b-asucx* **by** *auto*
   }
   ultimately show *?thesis* **using** *WEST-until.simps*[*of L1 L2 a b num-vars*] *Suc*
     using *WEST-or-simp-correct*
     using ‹*WEST-until L1 L2 a b num-vars = WEST-or-simp* (*WEST-until L1 L2 a* (*a + x*) *num-vars*) (*WEST-and-simp* (*WEST-global L1 a* (*a + x*) *num-vars*) (*WEST-global L2 b b num-vars*) *num-vars*) *num-vars*›
     by (*metis* (*no-types*, *lifting*) *WEST-and-simp-num-vars WEST-global-num-vars WEST-until-num-vars le-add1 order-refl*)
   qed
   have *a < length π*
     using *Suc*(*10*) **using** *b-asucx complen-geq-one* **by** *auto*
   then show *?case* **using** *sem-until*
     unfolding *semantics-mltl.simps* **by** *auto*
 qed

lemma *WEST-until-correct-converse*:
  assumes $\bigwedge \pi.$ (*length π* $\geq$ *complen-mltl F1* $\longrightarrow$ (*match π L1* $\longleftrightarrow$ *semantics-mltl π F1*))
  assumes $\bigwedge \pi.$ (*length π* $\geq$ *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 π* $\geq$ (*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*)
  using *assms*
proof(*induct b−a arbitrary*: *π L1 L2 F1 F2 a b*)
  case *0*
  then have *a-eq-b*: *b = a* **using** *0* **by** *simp*
  then have *semantics-mltl* (*drop a π*) *F2*
    using *assms* **unfolding** *semantics-mltl.simps*
    by (*metis 0.prems*(*9*) *le-antisym semantics-mltl.simps*(*9*))
  then have *match* (*drop a π*) *L2*
    using *0* **by** *simp*
  then have *match π* (*WEST-global L2 a a num-vars*)
    using *shift-match-converse*[*of a π L2 num-vars*] *0* **by** *auto*
  then show *?case* **using** *WEST-until.simps*[*of L1 L2 a a num-vars*] *a-eq-b* **by** *simp*
next
  case (*Suc x*)

have *max* (*complen-mltl F1 − 1*) (*complen-mltl F2*) ≥ *1*
  using *complen-geq-one*[*of F2*] **by** *auto*
then have *b-lt*: *b ≤ length π* **using** *Suc.prems(8)* **unfolding** *complen-mltl.simps*
  by *linarith*
have *b-asucx*: *b = a + (Suc x)* **using** *Suc* **by** *simp*
{assume *sat-b*: *semantics-mltl (drop b π) F2* ∧
     (∀ *j. a ≤ j ∧ j < b ⟶ semantics-mltl (drop j π) F1*)

  have *match (drop b π) L2*
    using *sat-b Suc* **by** *auto*
  then have *match π (shift L2 num-vars b)*
    using *shift-match*[*of b π L2*] *shift-match-converse*[*OF b-lt*] **by** *auto*
  then have *match-L2*: *match π (WEST-global L2 b b num-vars)*
    using *WEST-global.simps*[*of L2 b b num-vars*] **by** *simp*

  have *semantics-mltl π (Global-mltl a (b−1) F1)*
    using *sat-b Suc* **unfolding** *semantics-mltl.simps* **by** *auto*
  then have *match-L1*: *match π (WEST-global L1 a (b−1) num-vars)*
    using *WEST-global-correct*[*of F1 L1 num-vars a b−1 π*] *Suc* **by** *auto*

  have *nv1*: *WEST-regex-of-vars (WEST-global L1 a (b − 1) num-vars) num-vars*
    using *WEST-global-num-vars*[*of L1 num-vars a b−1*] *Suc* **by** *auto*
  have *nv2*: *WEST-regex-of-vars ((WEST-global L2 b b num-vars)) num-vars*
    using *WEST-global-num-vars*[*of L2 num-vars b b*] *Suc* **by** *auto*
  have *match π (WEST-and-simp (WEST-global L1 a (b − 1) num-vars) (WEST-global*
L2 b b num-vars) num-vars)
    using *match-L2 match-L1 nv1 nv2 WEST-and-simp-correct* **by** *blast*
  then have *?case*
    using *WEST-until.simps*[*of L1 L2 a b num-vars*]
  by (*metis Suc.prems(3) Suc.prems(4) Suc.prems(7) WEST-and-simp-num-vars*
WEST-or-simp-correct WEST-until-num-vars ‹semantics-mltl π (Global-mltl a (b
− 1) F1)› le-antisym linorder-not-less match-L2 nv1 nv2 semantics-mltl.simps(8))
} moreover {
  assume ¬(*semantics-mltl (drop b π) F2* ∧
     (∀ *j. a ≤ j ∧ j < b ⟶ semantics-mltl (drop j π) F1*))
  then have *sab-before-b*: (∃ *i. (a ≤ i ∧ i ≤ (a+x))* ∧
    semantics-mltl (drop i π) F2 ∧
    (∀ *j. a ≤ j ∧ j < i ⟶ semantics-mltl (drop j π) F1*))
    using *Suc(11) b-asucx* **unfolding** *semantics-mltl.simps*
    by (*metis add-Suc-right le-antisym not-less-eq-eq*)
  then have *semantics-mltl π (Until-mltl F1 a (b − 1) F2)*
    using *Suc b-asucx*
    unfolding *semantics-mltl.simps* **by** *auto*
  then have *match-rec*: *match π (WEST-until L1 L2 a (b − 1) num-vars)*
    using *Suc.hyps(1)*[*of b−1 a F1 L1 F2 L2 π*] *Suc* **by** *auto*
  have *WEST-until L1 L2 a b num-vars = WEST-or-simp (WEST-until L1 L2*
a (b − 1) num-vars)
      (*WEST-and-simp (WEST-global L1 a (b − 1) num-vars)*
       (*WEST-global L2 b b num-vars) num-vars*)

126

num-vars
    using *WEST-until.simps[of L1 L2 a b num-vars] Suc*
    by (*metis add-eq-self-zero b-asucx nat.discI nless-le*)
  then have *?case*
    using *match-rec Suc WEST-or-simp-correct*
  by (*metis WEST-and-simp-num-vars WEST-global-num-vars WEST-until-num-vars*
‹*semantics-mltl* π (*Until-mltl F1 a* (*b − 1*) *F2*)› *eq-imp-le semantics-mltl.simps(9)*)
  }
  ultimately show *?case* **by** *blast*
qed

lemma *WEST-until-correct*:
  assumes $\bigwedge\pi$. (*length* π ≥ *complen-mltl F1* ⟶ (*match* π *L1* ⟷ *semantics-mltl*
π *F1*))
  assumes $\bigwedge\pi$. (*length* π ≥ *complen-mltl F2* ⟶ (*match* π *L2* ⟷ *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* ≤ *num-vars*
  assumes *WEST-num-vars F2* ≤ *num-vars*
  assumes *a* ≤ *b*
  assumes *length* π ≥ *complen-mltl* (*Until-mltl F1 a b F2*)
  shows *match* π (*WEST-until L1 L2 a b num-vars*) ⟷
             semantics-mltl π (*Until-mltl F1 a b F2*)
  using *WEST-until-correct-forward*[*OF assms(1) assms(2) assms(3) assms(4)*
assms(5) assms(6) assms(7) assms(8)]
   WEST-until-correct-converse[*OF assms(1) assms(2) assms(3) assms(4) assms(5)*
assms(6) assms(7) assms(8)]
  by *blast*

3.12   Facts about the WEST release Operator

lemma *WEST-release-correct-forward*:
  assumes $\bigwedge\pi$. (*length* π ≥ *complen-mltl F1* ⟶ (*match* π *L1* ⟷ *semantics-mltl*
π *F1*))
  assumes $\bigwedge\pi$. (*length* π ≥ *complen-mltl F2* ⟶ (*match* π *L2* ⟷ *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* ≤ *num-vars*
  assumes *WEST-num-vars F2* ≤ *num-vars*
  assumes *a-leq-b*: *a* ≤ *b*
  assumes *len*: *length* π ≥ *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 −
  {assume *match-base*: *match* π (*WEST-global L2 a b num-vars*)
    {assume ∗ : *a = 0* ∧ *b = 0*

then have *WEST-global L2 a b num-vars = L2*
  using *WEST-global.simps pad-zero* **by** *simp*
then have *matchL2*: *match π L2*
  using *match-base* **by** *auto*
have *complen-mltl F2 ≤ length π*
  using *assms(8)* **by** *auto*
then have (*semantics-mltl π F2*)
  using *matchL2 assms(2)[of π] ∗*
  by *blast*
then have *?thesis* **using** ∗ **by** *simp*
} **moreover** {**assume** ∗ : *b > 0*
then have *semantics-mltl π* (*Global-mltl a b F2*)
  using *match-base WEST-global-correct[of F2 L2 num-vars a b π] assms* **by**
auto
then have ∀ *i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl* (*drop i π*) *F2*
  unfolding *semantics-mltl.simps* **using** *assms ∗ add-cancel-right-left complen-geq-one le-add2 le-trans max-nat.neutr-eq-iff nle-le not-one-le-zero*
  by (*smt (verit, best) add-diff-cancel-left' complen-mltl.simps(9) diff-is-0-eq'*)
then have *?thesis* **unfolding** *semantics-mltl.simps* **using** *assms* **by** *blast*
} **ultimately have** *?thesis* **using** *a-leq-b* **by** *blast*
} **moreover** {
  assume *no-match-base*: *match π* (*WEST-release-helper L1 L2 a* (*b−1*) *num-vars*)
∧ *a < b*
  have *a-le-b*: *a < b* **using** *no-match-base* **by** *simp*
  have *no-match*: *match π* (*WEST-release-helper L1 L2 a* (*b−1*) *num-vars*) **using**
no-match-base **by** *blast*
  have (∃*j≥a. j ≤ b − 1 ∧*
      semantics-mltl (*drop j π*) *F1 ∧*
      (∀ *k. a ≤ k ∧ k ≤ j ⟶ semantics-mltl* (*drop k π*) *F2*))
    using *assms a-le-b no-match*
  proof(*induct b−a−1 arbitrary: π L1 L2 F1 F2 a b*)
    case *0*
    have *max* (*complen-mltl F1 − 1*) (*complen-mltl F2*) ≥ *0*
      by *force*
    then have *a-leq*: *a ≤ length π*
      using *0(8−9)* **unfolding** *complen-mltl.simps*
      by *auto*
    have *b-aplus1*: *b = a+1* **using** *0* **by** *presburger*
    then have *match-rec*: *match π* (*WEST-release-helper L1 L2 a a num-vars*)
    using *0(10)* **using** *WEST-release.simps[of L1 L2 a b num-vars] WEST-or-correct*
0
      by (*metis diff-add-inverse2*)
      then have *match π* (*WEST-and-simp* (*WEST-global L1 a a num-vars*)
(*WEST-global L2 a a num-vars*) *num-vars*)
      using *0 WEST-release-helper.simps* **by** *metis*
    then have *match π* (*WEST-global L1 a a num-vars*) ∧ *match π* (*WEST-global
L2 a a num-vars*)
      using *WEST-and-simp-correct 0*
      using *WEST-global-num-vars[of L1 num-vars a a] WEST-global-num-vars[of*

L2 num-vars a a]
     by *blast*
   then have *match π* (*shift L1 num-vars a*) ∧ *match π* (*shift L1 num-vars a*)
    by *auto*
   then have *match-L1L2*: *match* (*drop a π*) *L1* ∧ *match* (*drop a π*) *L2*
    using *shift-match 0 a-leq*
    by (*metis WEST-global.simps ‹match π* (*WEST-global L1 a a num-vars*) ∧
match π (*WEST-global L2 a a num-vars*)›)
    have *b − a + max* (*complen-mltl F1 − 1*) (*complen-mltl F2*) ≤ *length* (*drop
a π*)
     using *0*(*9*) **unfolding** *complen-mltl.simps* **using** *0*(*1*, *8*) **by** *auto*
   then have *b − a + complen-mltl F1 − 1* ≤ *length* (*drop a π*)
    unfolding *complen-mltl.simps* **using** *0*(*1*) **by** *auto*
   then have *complen-mltl F1* ≤ *length* (*drop a π*)
    using *0*(*1*) *complen-geq-one*[*of F1*]
    by (*simp add*: *b-aplus1*)
    then have *F1-equiv*: *semantics-mltl* (*drop a π*) *F1 = match π* (*shift L1
num-vars a*)
    using *0*
    using ‹*match π* (*shift L1 num-vars a*) ∧ *match π* (*shift L1 num-vars a*)›
match-L1L2 **by** *blast*
    have *b − a + max* (*complen-mltl F2 − 1*) (*complen-mltl F2*) ≤ *length* (*drop
a π*)
     using *0*(*9*) **unfolding** *complen-mltl.simps* **using** *0*(*1*, *8*) **by** *auto*
   then have *b − a + complen-mltl F2* ≤ *length* (*drop a π*)
    unfolding *complen-mltl.simps* **using** *0*(*1*) **by** *auto*
   then have *complen-mltl F2* ≤ *length* (*drop a π*)
    using *0*(*1*) *complen-geq-one*[*of F1*]
    by (*simp add*: *b-aplus1*)
    then have *F2-equiv*: *semantics-mltl* (*drop a π*) *F2 = match π* (*shift L2
num-vars a*)
    using *0 a-leq match-L1L2 shift-match-converse* **by** *blast*
    have *semantics-mltl* (*drop a π*) *F1* ∧ *semantics-mltl* (*drop a π*) *F2*
    using *F1-equiv F2-equiv match-L1L2*
    using *a-leq shift-match-converse* **by** *blast*
    then show *?case* **using** *b-aplus1* **by** *auto*
  next
   case (*Suc x*)
   then have *b-aplus2*: *b = a+x+2* **by** *linarith*
    then have *match-rec*: *match π* (*WEST-release-helper L1 L2 a* (*a+x+1*)
num-vars)
    using *WEST-release.simps*[*of L1 L2 a a+x+2 num-vars*] *WEST-or-correct
Suc*
    by (*metis Suc-1 Suc-eq-plus1 add-Suc-shift add-diff-cancel-right′*)
    have *west-release-helper*: *WEST-release-helper L1 L2 a* (*a+x+1*) *num-vars
= WEST-or-simp* (*WEST-release-helper L1 L2 a* (*a + x*) *num-vars*)
       (*WEST-and-simp* (*WEST-global L2 a* (*a + x + 1*) *num-vars*)
(*WEST-global L1* (*a + x + 1*) (*a + x + 1*) *num-vars*) *num-vars*) *num-vars*
    using *WEST-release-helper.simps*[*of L1 L2 a a+x+1 num-vars*]

by (*metis add.commute add-diff-cancel-right′ less-add-Suc1 less-add-one not-add-less1 plus-1-eq-Suc*)

  let *?rec = WEST-release-helper L1 L2 a (a + x) num-vars*

  let *?base = WEST-and-simp (WEST-global L2 a (a + x + 1) num-vars) (WEST-global L1 (a + x + 1) (a + x + 1) num-vars) num-vars*

  have *match-rec-or-base*: *match π ?rec ∨ match π ?base*

  using *WEST-or-simp-correct WEST-release-helper-num-vars WEST-and-simp-num-vars WEST-global-num-vars*

  by (*metis (mono-tags, lifting) Suc.prems(3) Suc.prems(4) ab-semigroup-add-class.add-ac(1) eq-imp-le le-add1 match-rec west-release-helper*)

  have ∃*j≥a. j ≤ a+x+1 ∧*
    semantics-mltl (drop j π) F1 ∧ (∀ k. a ≤ k ∧ k ≤ j ⟶ semantics-mltl (drop k π) F2)

  proof−

   {assume *match-rec*: *match π (WEST-release-helper L1 L2 a (a + x) num-vars)*

    have *x-is*: *x = a + x + 1 − a − 1*
     by *auto*

    have *a-leq*: *a ≤ a + x + 1*
     by *simp*

    have *a-lt*: *a < a + x + 1*
     by *auto*

    have *complen*: *complen-mltl (Release-mltl F1 a (a + x + 1) F2) ≤ length π*

     using *Suc(10) Suc(2)* **by** *simp*

    have *sum*: *a + x + 1 = b − 1*
     using *Suc(2)* **by** *auto*

    have *important-match*: *match π (WEST-release-helper L1 L2 a (b−2) num-vars)*

     using *match-rec sum b-aplus2* **by** *simp*

    have *match π (WEST-or-simp (WEST-global L2 a (b − 1) num-vars) (WEST-release-helper L1 L2 a (b − 2) num-vars) num-vars)*

     using *important-match b-aplus2*

     using *WEST-or-simp-correct[of WEST-global L2 a (b − 1) num-vars num-vars WEST-release-helper L1 L2 a (b − 2) num-vars π]*

    by (*metis Suc.prems(3) Suc.prems(4) WEST-global-num-vars WEST-release-helper-num-vars a-leq diff-add-inverse2 le-add1 sum*)

   then have *match1*: *match π (WEST-release L1 L2 a (a + x + 1) num-vars)*

    unfolding *WEST-release.simps*

    using *b-aplus2 sum*

    by (*metis (full-types) Suc-1 a-lt diff-diff-left plus-1-eq-Suc*)

    have *match2*: *match π (WEST-release-helper L1 L2 a (a + x + 1 − 1) num-vars)*

    using *important-match b-aplus2* **by** *auto*

    have ∃*j≥a. j ≤ a + x ∧*
    semantics-mltl (drop j π) F1 ∧ (∀ k. a ≤ k ∧ k ≤ j ⟶ semantics-mltl (drop k π) F2)

     using *Suc.hyps(1)[OF x-is Suc(3) Suc(4) Suc(5) Suc(6) Suc(7) Suc(8) a-leq complen - a-lt]*

130

match1 match2
      by (*metis add-diff-cancel-right'*)
    then have *?case* **using** *b-aplus2* **by** *auto*
  } moreover {
    assume *match-base*: *match π* (*WEST-and-simp* (*WEST-global L2 a* (*a +
x + 1*) *num-vars*)
                                (*WEST-global L1* (*a + x + 1*) (*a + x + 1*) *num-vars*)
num-vars)
      have *match π* (*WEST-global L2 a* (*a + x + 1*) *num-vars*)
        using *match-base WEST-and-simp-correct WEST-global-num-vars*
      by (*metis Suc.prems(3) Suc.prems(4) add.commute eq-imp-le less-add-Suc1
order-less-le plus-1-eq-Suc*)
      then have *semantics-mltl π* (*Global-mltl a* (*a + x + 1*) *F2*)
        using *WEST-global-correct*[*of F2 L2 num-vars a a + x + 1 π*]
        using *Suc.prems*(*2*, *4*, *6*, *8*) *Suc.hyps*(*2*) **by** *simp*
      then have *fact2*: (∀ *k. a ≤ k ∧ k ≤* (*a + x + 1*) ⟶ *semantics-mltl* (*drop
k π*) *F2*)
          unfolding *semantics-mltl.simps* **using** *Suc.prems*(*8*, *10*)
          unfolding *complen-mltl.simps* **by** *simp*
      have *match π* (*WEST-global L1* (*a + x + 1*) (*a + x + 1*) *num-vars*)
        using *match-base WEST-and-simp-correct WEST-global-num-vars*
      by (*metis Suc.prems(3) Suc.prems(4) add.commute eq-imp-le less-add-Suc1
order-less-le plus-1-eq-Suc*)
      then have *match π* (*shift L1 num-vars* (*a + x + 1*))
          using *WEST-global.simps*[*of L1 a + x + 1 a + x + 1 num-vars*] **by**
metis
      then have *match* (*drop* (*a + x + 1*) *π*) *L1*
        using *shift-match*[*of a + x + 1 π L1 num-vars*]
          using *Suc.prems*(*8*) **unfolding** *complen-mltl.simps* **using** *b-aplus2* **by**
simp
      then have *fact1*: *semantics-mltl* (*drop* (*a + x + 1*) *π*) *F1*
        using *Suc.prems*(*1*)[*of drop* (*a + x + 1*) *π*]
          using *Suc.prems*(*8*) **unfolding** *complen-mltl.simps* **using** *b-aplus2* **by**
auto
      have *?case* **using** *b-aplus2 fact1 fact2*
      by (*smt* (*verit*) *Suc.hyps*(*2*) *Suc.prems*(*10*) *Suc-diff-Suc ab-semigroup-add-class.add-ac*(*1*)
add.commute add-diff-cancel-left' antisym-conv1 le-iff-add order-less-imp-le plus-1-eq-Suc)
    }
    ultimately show *?thesis* **using** *match-rec-or-base*
    by (*smt* (*verit*, *best*) *Suc.hyps*(*2*) *Suc-eq-plus1 add.assoc diff-right-commute
le-trans ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
  qed
  then show *?case* **using** *b-aplus2* **by** *simp*
 qed

 then have *?thesis* **unfolding** *semantics-mltl.simps* **by** *auto*
 }
 ultimately show *?thesis* **using** *WEST-release.simps assms*(*9*)
 by (*smt* (*verit*, *ccfv-SIG*) *WEST-global-num-vars WEST-or-simp-correct WEST-release-helper-num-vars*

a-leq-b add-leD2 add-le-cancel-right assms(3) assms(4) diff-add less-iff-succ-less-eq)
qed

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 −
  have *len-xi*: *a < length π*
    using *assms(7, 8)* **unfolding** *complen-mltl.simps*
  by *(metis (no-types, lifting) add-leD1 complen-geq-one diff-add-inverse diff-is-0-eq′*
le-add-diff-inverse le-neq-implies-less le-zero-eq less-numeral-extra(4) less-one max-nat.eq-neutr-iff)

  {assume *case1*: $\forall i.$ *a $\leq$ i $\wedge$ i $\leq$ b $\longrightarrow$ semantics-mltl (drop i π) F2*
    then have *match π (WEST-global L2 a b num-vars)*
      using *WEST-global-correct-converse assms* **by** *fastforce*
    then have *?thesis* **unfolding** *WEST-release.simps*
      using *WEST-or-simp-correct*
    by *(smt (verit) WEST-global-num-vars WEST-release-helper-num-vars add-leE*
add-le-cancel-right assms(3) assms(4) diff-add less-iff-succ-less-eq)
  } moreover {
    assume *case2*: $\exists j{\geq}a.$ *j $\leq$ b − 1 $\wedge$*
        semantics-mltl (drop j π) F1 $\wedge$
        ($\forall k.$ a $\leq$ k $\wedge$ k $\leq$ j $\longrightarrow$ semantics-mltl (drop k π) F2)
    then obtain *j* **where** *obtain-j*: *a $\leq$ j $\wedge$ j $\leq$ b − 1 $\wedge$*
        semantics-mltl (drop j π) F1 $\wedge$
        ($\forall k.$ a $\leq$ k $\wedge$ k $\leq$ j $\longrightarrow$ semantics-mltl (drop k π) F2)
      by *blast*
    {
      assume *a-eq-b*: *a = b*
      then have *?thesis* **using** *case2*
        using *calculation le-antisym* **by** *blast*
    } moreover {
      assume *a-le-b*: *a < b*

      have *semantics-mltl π (Global-mltl j j F1)* **using** *obtain-j*
        by *auto*
      have *(complen-mltl F1 − 1) + b $\leq$ length π*
        using *assms(8) obtain-j* **unfolding** *complen-mltl.simps* **by** *auto*

then have *complen-mltl F1 + j ≤ length π*
  using *obtain-j a-le-b* **by** *auto*
then have *match-global1*: *match π* (*WEST-global L1 j j num-vars*)
  using *WEST-global-correct-converse*[*of F1 L1 num-vars j j π*] *assms*
  using ‹*semantics-mltl π* (*Global-mltl j j F1*)› **by** *blast*

have *len-xi-f2j*: *complen-mltl F2 + j ≤ length π*
  using *assms*(*8*) *obtain-j* **by** *auto*
have *a ≤ j*
  using *a-le-b obtain-j* **by** *blast*
then have *semantics-mltl π* (*Global-mltl a j F2*)
  using *obtain-j a-le-b*
  unfolding *semantics-mltl.simps* **by** *blast*
then have *match-global2*: *match π* (*WEST-global L2 a j num-vars*)
    using *WEST-global-correct-converse*[*of F2 L2 num-vars a j π*] *len-xi-f2j*
assms
  by *simp*
have *j-bounds*: *a ≤ j ∧ j ≤ b − 1* **using** *obtain-j* **by** *blast*
have *match π* (*WEST-release-helper L1 L2 a* (*b − 1*) *num-vars*)
  using *match-global1 match-global2 a-le-b j-bounds assms*(*1−6*)
proof(*induct b−1−a arbitrary*: *a b L1 L2 F1 F2*)
  case *0*
    then have *match π* (*WEST-and-simp* (*WEST-global L1 a a num-vars*)
(*WEST-global L2 a a num-vars*) *num-vars*)
      using *WEST-and-simp-correct*
      by (*metis WEST-global-num-vars diff-is-0-eq′ diffs0-imp-equal le-trans*)
    then show *?case*
      using *WEST-release-helper.simps*[*of L1 L2 a b−1 num-vars*] *0*
      by (*metis diff-diff-cancel diff-zero le-trans*)
  next
    case (*Suc x*)
    have *match-helper*: *match π* (*WEST-or-simp* (*WEST-release-helper L1 L2
a* (*b − 1 − 1*) *num-vars*)
        (*WEST-and-simp* (*WEST-global L2 a* (*b − 1*) *num-vars*)
        (*WEST-global L1* (*b − 1*) (*b − 1*) *num-vars*) *num-vars*) *num-vars*)
      using *Suc*
    proof −
    {assume *j-eq-bm1*: *j = b−1*
        then have *match π* (*WEST-and-simp* (*WEST-global L2 a* (*b − 1*)
num-vars)
        (*WEST-global L1* (*b − 1*) (*b − 1*) *num-vars*) *num-vars*)
        using *Suc WEST-and-simp-correct*
        by (*meson WEST-global-num-vars*)
      then have *?thesis* **using** *WEST-or-simp-correct*
            by (*metis Suc.hyps*(*2*) *Suc.prems*(*4*) *Suc.prems*(*7*) *Suc.prems*(*8*)
*WEST-and-simp-num-vars WEST-global-num-vars WEST-release-helper-num-vars
cancel-comm-monoid-add-class.diff-cancel diff-less-Suc j-eq-bm1 le-SucE le-add1 not-add-less1
ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
    } moreover {

133

assume *j-le-bm1*: $j < b-1$
have *match π (WEST-release-helper L1 L2 a (b − 1 − 1) num-vars)*
using *Suc.hyps(1)[of b−1 a L1 L2 F1 F2] Suc*
by (*smt (verit) Suc-leI diff-Suc-1 diff-le-mono diff-right-commute*
j-le-bm1 le-eq-less-or-eq not-less-eq-eq)
then have *?thesis* **using** *WEST-or-simp-correct*
using *Suc.hyps(2) Suc.prems(4) Suc.prems(7) Suc.prems(8) WEST-and-simp-num-vars*
WEST-global-num-vars WEST-release-helper-num-vars
by (*smt (verit, del-insts) Nat.lessE Suc-leI diff-Suc-1 j-le-bm1 le-Suc-eq*
le-trans)
}
ultimately show *?thesis* **using** *Suc(6)*
by (*meson le-neq-implies-less*)
qed

have $a < b-1$ **using** *Suc(2)* **by** *simp*
then show *?case*
using *WEST-release-helper.simps[of L1 L2 a b−1 num-vars] match-helper*
by *presburger*
qed

then have *match π (WEST-or-simp (WEST-global L2 a b num-vars) (WEST-release-helper*
L1 L2 a (b − 1) num-vars) num-vars)
using *WEST-or-simp-correct assms*
by (*meson WEST-global-num-vars WEST-release-helper-num-vars j-bounds*
le-trans)
then have *?thesis* **using** *a-le-b* **unfolding** *WEST-release.simps*
by *presburger*
}
ultimately have *?thesis* **using** *assms(7)* **by** *fastforce*
}
ultimately show *?thesis* **unfolding** *semantics-mltl.simps* **using** *len-xi assms(9)*
by *fastforce*
qed

lemma *WEST-release-correct*:
  assumes $\bigwedge\pi.$ *(length π ≥ complen-mltl F1 ⟶ (match π L1 ⟷ semantics-mltl*
π F1))
  assumes $\bigwedge\pi.$ *(length π ≥ complen-mltl F2 ⟶ (match π L2 ⟷ 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 ≤ num-vars*
  assumes *WEST-num-vars F2 ≤ num-vars*
  assumes $a ≤ b$
  assumes *length π ≥ complen-mltl (Release-mltl F1 a b F2)*
  shows *semantics-mltl π (Release-mltl F1 a b F2) ⟷ match π (WEST-release*
L1 L2 a b num-vars)

134

using *WEST-release-correct-converse*[*OF assms*(1−8)] *WEST-release-correct-forward*[*OF assms*(1−8)]
  by *blast*

3.13   Top level result: Shows that WEST reg is correct

lemma *WEST-reg-aux-correct*:
  assumes *π-long-enough*: *length π ≥ complen-mltl F*
  assumes *is-nnf*: ∃ *ψ*. *F = (convert-nnf ψ)*
  assumes *φ-nv*: *WEST-num-vars F ≤ num-vars*
  assumes *intervals-welldef F*
  shows *match π (WEST-reg-aux F num-vars) ⟷ semantics-mltl π F*
  using *assms*
  proof (*induction F arbitrary*: *π rule*: *nnf-induct*)
  case *nnf*
  then show *?case* **using** *is-nnf* **by** *auto*
next
  case *True*
  have *semantics-true*: *semantics-mltl π True-mltl = True* **by** *simp*
  have *WEST-reg-aux True-mltl num-vars = [[map (λj. S) [0..<num-vars]]]*
    using *WEST-reg-aux.simps(1)* **by** *blast*
  have *match-state*: *match-timestep (π ! 0) (map (λj. S) [0..<num-vars])*
    unfolding *match-timestep-def* **by** *auto*
  have *length π ≥ 1* **using** *True* **by** *auto*
  then have *match-regex π [(map (λj. S) [0..<num-vars])] = True*
    using *True match-state* **unfolding** *match-regex-def* **by** *simp*
  then have *match π (WEST-reg-aux True-mltl num-vars) = True*
    using *WEST-reg-aux.simps(1)[of num-vars]* **unfolding** *match-def* **by** *simp*
  then show *?case*
    using *semantics-true* **by** *auto*
next
  case *False*
  have *semantics-false*: *semantics-mltl π False-mltl = False* **by** *simp*
  have *match π [] = False*
    unfolding *match-def* **by** *simp*
  then show *?case*
    using *semantics-false* **by** *simp*
next
  case (*Prop p*)
  have *trace-nonempty*: *length π ≥ 1* **using** *Prop* **by** *simp*
  let *?state = π!0*
  {**assume** *p-in*: *p ∈ ?state*
    then have *semantics-prop-true*: *semantics-mltl π (Prop-mltl p) = True*
      using *semantics-mltl.simps(3)[of π] trace-nonempty* **by** *auto*
    have *WEST-prop*: (*WEST-reg-aux (Prop-mltl p) num-vars*) = [[*map (λj. if p = j then One else S) [0..<num-vars]*]]
      using *WEST-reg-aux.simps(3)* **by** *blast*
    have *p < num-vars ⟹ p ∈ π ! 0*
      using *p-in Prop* **by** *blast*

then have *match-timestep ?state (map (λj. if p = j then One else S) [0..<num-vars])*
= True
    unfolding *match-timestep-def p-in* **by** *auto*
  then have *match-regex π (WEST-reg-aux (Prop-mltl p) num-vars ! 0) = True*
    using *trace-nonempty WEST-prop* **unfolding** *match-regex-def* **by** *auto*
  then have *match π (WEST-reg-aux (Prop-mltl p) num-vars) = True*
    unfolding *match-def* **by** *auto*
  then have *?case* **using** *semantics-prop-true* **by** *blast*
 } moreover {
  assume *p-notin*: *p ∉ ?state*
  then have *semantics-prop-false*: *semantics-mltl π (Prop-mltl p) = False*
    using *semantics-mltl.simps(3)[of π] trace-nonempty* **by** *auto*
  have *WEST-prop*: *(WEST-reg-aux (Prop-mltl p) num-vars) = [[map (λj. if p*
= j then One else S) [0..<num-vars]]]
    using *WEST-reg-aux.simps(3)* **by** *blast*
  have *p < num-vars ∧ p ∉ π ! 0*
    using *p-notin Prop* **by** *auto*
 then have *match-timestep ?state (map (λj. if p = j then One else S) [0..<num-vars])*
= False
    unfolding *match-timestep-def p-notin* **by** *auto*
  then have *match-regex π (WEST-reg-aux (Prop-mltl p) num-vars ! 0) = False*
    using *trace-nonempty WEST-prop* **unfolding** *match-regex-def* **by** *auto*
  then have *match π (WEST-reg-aux (Prop-mltl p) num-vars) = False*
    unfolding *match-def* **by** *auto*
  then have *?case* **using** *semantics-prop-false* **by** *blast*
 }
 ultimately show *?case* **by** *blast*
next
 case (*NotProp F p*)
 have *trace-nonempty*: *length π ≥ 1* **using** *NotProp* **by** *simp*
 let *?state = π!0*
 {assume *p-in*: *p ∈ ?state*
   then have *semantics-prop-true*: *semantics-mltl π (Not-mltl (Prop-mltl p)) =*
False
     using *semantics-mltl.simps trace-nonempty* **by** *auto*
  have *WEST-prop*: *(WEST-reg-aux (Not-mltl (Prop-mltl p)) num-vars) = [[map*
(λj. if p = j then Zero else S) [0..<num-vars]]]
     using *WEST-reg-aux.simps* **by** *blast*
   have *p < num-vars ∧ p ∈ π ! 0*
     using *p-in NotProp* **by** *simp*
  then have *match-timestep ?state (map (λj. if p = j then Zero else S) [0..<num-vars])*
= False
     unfolding *match-timestep-def p-in* **by** *auto*
   then have *match-regex π (WEST-reg-aux (Not-mltl (Prop-mltl p)) num-vars !*
0) = False
     using *trace-nonempty WEST-prop* **unfolding** *match-regex-def* **by** *auto*
  then have *match π (WEST-reg-aux (Not-mltl (Prop-mltl p)) num-vars) = False*
     unfolding *match-def* **by** *auto*
   then have *?case* **using** *semantics-prop-true NotProp* **by** *blast*

136

} moreover {
  assume *p-notin*: $p \notin$ *?state*
  then have *semantics-prop-false*: *semantics-mltl* π (*Not-mltl* (*Prop-mltl p*)) =
True
    using *semantics-mltl.simps(3)[of* π*] trace-nonempty* **by** *auto*
  have *WEST-prop*: (*WEST-reg-aux* (*Not-mltl* (*Prop-mltl p*)) *num-vars*) = [[*map*
($\lambda j.$ *if* $p = j$ *then Zero else S*) [*0..<num-vars*]]]
    using *WEST-reg-aux.simps* **by** *blast*
  have $p <$ *num-vars* $\wedge$ $p \notin \pi$! *0*
    using *p-notin NotProp* **by** *auto*
  then have *match-timestep ?state* (*map* ($\lambda j.$ *if* $p = j$ *then Zero else S*) [*0..<num-vars*])
= *True*
    unfolding *match-timestep-def p-notin* **by** *auto*
  then have *match-regex* π (*WEST-reg-aux* (*Not-mltl* (*Prop-mltl p*)) *num-vars* !
0) = *True*
    using *trace-nonempty WEST-prop* **unfolding** *match-regex-def* **by** *auto*
  then have *match* π (*WEST-reg-aux* (*Not-mltl* (*Prop-mltl p*)) *num-vars*) = *True*
    unfolding *match-def* **by** *simp*
  then have *?case* **using** *semantics-prop-false NotProp* **by** *blast*
}
  ultimately show *?case* **by** *blast*
next
  case (*And F F1 F2*)
  have *subformula1*: *WEST-num-vars F1* $\leq$ *num-vars*
    using *WEST-num-vars-subformulas[of F1 F] And(1,6)* **by** *simp*
  have *subformula2*: *WEST-num-vars F2* $\leq$ *num-vars*
    using *WEST-num-vars-subformulas[of F2 F] And(1,6)* **by** *simp*
  have *complen-mltl F1* $\leq$ *complen-mltl F*
    using *And(1) complen-mltl.simps(5)[of F1 F2]* **by** *auto*
  then have *cp-F1*: *complen-mltl F1* $\leq$ *length* π
    using *And.prems* **by** *auto*
  have *h2*: *match* π (*WEST-reg-aux F1 num-vars*) = *semantics-mltl* π *F1*
    using *And(2)[OF cp-F1] subformula1*
    by (*metis And.hyps And.prems(2) And.prems(4) convert-nnf.simps(4) con-*
vert-nnf-convert-nnf intervals-welldef.simps(5) mltl.inject(3))
  have *complen-mltl F2* $\leq$ *complen-mltl F*
    using *And(1) complen-mltl.simps(5)[of F2 F2]* **by** *simp*
  then have *cp-F2*: *complen-mltl F2* $\leq$ *length* π
    using *And.prems* **by** *auto*
  have *h1*: *match* π (*WEST-reg-aux F2 num-vars*) = *semantics-mltl* π *F2*
    using *And.prems(2) And(1) And(3)[OF cp-F2] subformula2*
    by (*metis And.prems(4) convert-nnf.simps(4) convert-nnf-convert-nnf inter-*
vals-welldef.simps(5) mltl.inject(3))
  let *?n* = *num-vars*
  have *F1-nv*: *WEST-regex-of-vars* (*WEST-reg-aux F1 num-vars*) *num-vars*
   using *WEST-reg-aux-num-vars[of F1 num-vars] subformula1 And(1) And.prems(2)*
    using *WEST-num-vars-subformulas*
    by (*metis And.prems(4) convert-nnf.simps(4) convert-nnf-convert-nnf inter-*
vals-welldef.simps(5) mltl.inject(3))

have *F2-nv*: *WEST-regex-of-vars* (*WEST-reg-aux F2 num-vars*) *num-vars*
  using *WEST-reg-aux-num-vars*[*of F2 num-vars*] *subformula1 And*(*1*) *And.prems*(*2*)
   using *WEST-num-vars-subformulas*
    by (*metis And.prems*(*4*) *convert-nnf.simps*(*4*) *convert-nnf-convert-nnf inter-*
vals-welldef.simps(*5*) *mltl.inject*(*3*) *subformula2*)
 have *match*: *match π* (*WEST-and* (*WEST-reg-aux F1 ?n*) (*WEST-reg-aux F2*
?n)) = (*match π* (*WEST-reg-aux F1 ?n*) ∧ *match π* (*WEST-reg-aux F2 ?n*))
   using *WEST-and-correct*[*of WEST-reg-aux F1 ?n ?n WEST-reg-aux F2 ?n π*,
OF F1-nv F2-nv]
   by *blast*
 have *WEST-reg-F*: *WEST-reg-aux F num-vars* = *WEST-and-simp* (*WEST-reg-aux*
F1 num-vars) (*WEST-reg-aux F2 num-vars*) *num-vars*
   using *And*(*1*) *WEST-reg-aux.simps*(*6*)[*of F1 F2 num-vars*] **by** *argo*
 have *semantics-F*: *semantics-mltl π* (*And-mltl F1 F2*) = (*semantics-mltl π F1*
∧ *semantics-mltl π F2*)
   using *semantics-mltl.simps*(*5*)[*of π F1 F2*] **by** *blast*
 have *F1-nnf*: ∃ψ. *F1* = *convert-nnf ψ*
   using *And*(*1*) *And*(*5*) *nnf-subformulas*[*of F - F1*]
   by (*metis convert-nnf.simps*(*4*) *convert-nnf-convert-nnf mltl.inject*(*3*))
 have *F1-correct*: *match π* (*WEST-reg-aux F1 num-vars*) = *semantics-mltl π F1*
   using *And*(*2*)[*OF cp-F1 F1-nnf*] *WEST-num-vars-subformulas And* **by** *auto*
 have *F2-nnf*: ∃ψ. *F2* = *convert-nnf ψ*
   using *And*(*1*) *And*(*5*) *nnf-subformulas*[*of F - F2*]
   by (*metis convert-nnf.simps*(*4*) *convert-nnf-convert-nnf mltl.inject*(*3*))
 have *F2-correct*: *match π* (*WEST-reg-aux F2 num-vars*) = *semantics-mltl π F2*
   using *And*(*3*)[*OF cp-F2 F2-nnf*] *WEST-num-vars-subformulas And* **by** *auto*
 show *?case*
   using *WEST-reg-F F1-correct F2-correct*
   using *semantics-mltl.simps*(*5*)[*of π F1 F2*] *And*(*1*) *match*
   by (*metis F1-nv F2-nv WEST-and-simp-correct*)
next
 case (*Or F F1 F2*)
 have *cp-F1*: *complen-mltl F1* ≤ *length π*
   using *Or complen-mltl.simps*(*6*)[*of F1 F2*] **by** *simp*
 have *cp-F2*: *complen-mltl F2* ≤ *length π*
   using *Or complen-mltl.simps*(*6*)[*of F1 F2*] **by** *simp*
 have *F1-nnf*: ∃ψ. *F1* = *convert-nnf ψ*
   using *Or*(*1*) *nnf-subformulas*[*of F - F1*]
  by (*metis Or.prems*(*2*) *convert-nnf.simps*(*5*) *convert-nnf-convert-nnf mltl.inject*(*4*))
 have *F1-correct*: *match π* (*WEST-reg-aux F1 num-vars*) = *semantics-mltl π F1*
   using *Or*(*2*)[*OF cp-F1 F1-nnf*] *WEST-num-vars-subformulas Or* **by** *simp*
 have *F2-nnf*: ∃ψ. *F2* = *convert-nnf ψ*
   using *Or nnf-subformulas*[*of F - F2*]
   by (*metis convert-nnf.simps*(*5*) *convert-nnf-convert-nnf mltl.inject*(*4*))
 have *F2-correct*: *match π* (*WEST-reg-aux F2 num-vars*) = *semantics-mltl π F2*
   using *Or*(*3*)[*OF cp-F2 F2-nnf*] *WEST-num-vars-subformulas Or* **by** *simp*
 let *?L1* = (*WEST-reg-aux F1 num-vars*)
 let *?L2* = (*WEST-reg-aux F2 num-vars*)
 have *L1-nv*: *WEST-regex-of-vars ?L1 num-vars*

138

using *WEST-reg-aux-num-vars*[*of F1 num-vars, OF F1-nnf*]
  using *Or*(*1*, *6*, *7*) **by** *auto*
 have *L2-nv*: *WEST-regex-of-vars ?L2 num-vars*
  using *WEST-reg-aux-num-vars*[*of F2 num-vars, OF F2-nnf*]
  using *Or*(*1*, *6*, *7*) **by** *auto*
 have (*match* π *?L1* $\vee$ *match* π *?L2*) = *match* π (*WEST-or-simp ?L1 ?L2*
num-vars)
   using *WEST-or-simp-correct*[*of ?L1 num-vars ?L2* π, *OF L1-nv L2-nv*] **by**
blast
 then show *?case*
  using *F1-correct F2-correct*
  using *semantics-mltl.simps*(*6*)[*of* π *F1 F2*]
  unfolding *Or*(*1*) **unfolding** *WEST-reg-aux.simps* **by** *blast*
next
 case (*Final F F1 a b*)
 have *F1-nv*: *WEST-num-vars F1* $\leq$ *num-vars*
  using *Final* **by** *auto*
 have *cp-F1*: *complen-mltl F1* $\leq$ *length* π
  using *Final* **by** *simp*
 then have *len-xi*: *length* π $\geq$ (*complen-mltl F1*) + *b* **using** *Final* **by** *auto*
 have *F1-nnf*: $\exists \psi$. *F1* = *convert-nnf* ψ
  using *Final*
  by (*metis convert-nnf.simps*(*6*) *convert-nnf-convert-nnf mltl.inject*(*5*))
 let *?L1* = (*WEST-reg-aux F1 num-vars*)
 have *match-F1*: *match* π *?L1* = *semantics-mltl* π *F1*
  using *Final*(*2*)[*OF cp-F1 F1-nnf F1-nv*] *Final* **by** *auto*
 have *intervals-welldef-F1*: *intervals-welldef F1*
  using *Final* **by** *auto*
 have *a-le-b*: *a* $\leq$ *b*
  using *Final* **by** *simp*
 show *?case* **using** *WEST-reg-aux.simps*(*7*)[*of a b F1 num-vars*] *Final*
  using *match-F1 WEST-future-correct F1-nv len-xi*
  using *a-le-b intervals-welldef-F1*
  by (*metis F1-nnf WEST-reg-aux-num-vars*)
next
 case (*Global F F1 a b*)
 have *F1-nv*: *WEST-num-vars F1* $\leq$ *num-vars*
  using *Global* **by** *auto*
 have *cp-F1*: *complen-mltl F1* $\leq$ *length* π
  using *Global* **by** *simp*
 then have *len-xi*: *length* π $\geq$ (*complen-mltl F1*) + *b* **using** *Global* **by** *auto*
 have *F1-nnf*: $\exists \psi$. *F1* = *convert-nnf* ψ
  using *Global*
  by (*metis convert-nnf.simps*(*7*) *convert-nnf-convert-nnf mltl.inject*(*6*))
 let *?L1* = (*WEST-reg-aux F1 num-vars*)
 have *match-F1*: *match* π *?L1* = *semantics-mltl* π *F1*
  using *Global*(*2*)[*OF cp-F1 F1-nnf F1-nv*] *Global* **by** *auto*
 then show *?case* **using** *WEST-reg-aux.simps*(*8*)[*of a b F1 num-vars*] *Global*
  using *match-F1 WEST-global-correct F1-nv*

139

by (*metis F1-nnf WEST-reg-aux-num-vars intervals-welldef.simps(8) len-xi*)
next
  case (*Until F F1 F2 a b*)
  have *F1-nv*: *WEST-num-vars F1 ≤ num-vars*
    using *Until* **by** *auto*
  {**assume** ∗: *a = 0 ∧ b = 0*
    have *complen-leq*: *complen-mltl F2 ≤ length π*
      using *Until*(*1*) *Until.prems*(*1*) **by** *simp*
    have *some-nnf*: ∃ ψ. *F2 = convert-nnf ψ*
      using *Until*(*1*) *Until.prems*(*2*)
      by (*metis convert-nnf.simps(8) convert-nnf-convert-nnf mltl.inject(7)*)
    have *F2 ∈ subformulas* (*Until-mltl F1 a b F2*)
      unfolding *subformulas.simps* **by** *blast*
    then have *num-vars*: *WEST-num-vars F2 ≤ num-vars*
      using *Until*(*1*) *Until.prems*(*3*) *WEST-num-vars-subformulas*[*of F2 F*]
      by *auto*
    have *match-F2*: *match π* (*WEST-reg-aux F2 num-vars*) = *semantics-mltl π F2*
      using *Until*(*1*) *Until*(*3*)[*OF complen-leq some-nnf num-vars*] *Until.prems*
      by *simp*
    have *max* (*complen-mltl F1 − 1*) (*complen-mltl F2*) *>= 1*
      using *complen-geq-one*[*of F2*] **by** *auto*
    then have *len-gt*: *length π > 0*
      using *Until.prems*(*1*) *Until*(*1*) **by** *auto*
    have *global*: *WEST-global* (*WEST-reg-aux F2 num-vars*) *0 0 num-vars = shift*
(*WEST-reg-aux F2 num-vars*) *num-vars 0*
      using *WEST-global.simps*[*of - 0 0*] **by** *auto*
    have *map* (*λk. arbitrary-state num-vars*) [*0..<0*] = []
      by *simp*
    then have *padis*: *shift* (*WEST-reg-aux F2 num-vars*) *num-vars 0 = WEST-reg-aux*
F2 num-vars
      unfolding *shift.simps arbitrary-trace.simps* **using** *append.left-neutral list.simps(8)*
map-ident upt-0
      proof −
        have (*@*) (*map* (*λn. arbitrary-state num-vars*) ([]::*nat list*)) = (*λwss. wss*)
          by *blast*
        then show *map* ((*@*) (*map* (*λn. arbitrary-state num-vars*) [*0..<0*])) (*WEST-reg-aux*
F2 num-vars) = *WEST-reg-aux F2 num-vars*
          by *simp*
      qed
    then have *match π* (*WEST-global* (*WEST-reg-aux F2 num-vars*) *0 0 num-vars*)
=
    (*semantics-mltl π F2*)
      using *match-F2 global padis* **by** *simp*
    then have *match π* (*WEST-until* (*WEST-reg-aux F1 num-vars*) (*WEST-reg-aux*
F2 num-vars) *0 0 num-vars*) =
    (*semantics-mltl π F2*)
      using *WEST-until.simps*[*of - - 0 0 num-vars*] **by** *auto*
    then have *match π* (*WEST-until* (*WEST-reg-aux F1 num-vars*) (*WEST-reg-aux*
F2 num-vars) *0 0 num-vars*) =

$(semantics\text{-}mltl\ (drop\ 0\ \pi)\ F2\ \wedge\ (\forall\,j.\ 0 \leq j \wedge j < 0 \longrightarrow semantics\text{-}mltl\ (drop\ j\ \pi)\ F1))$
  by *auto*
  then have *match* π (*WEST-reg-aux* (*Until-mltl F1 0 0 F2*) *num-vars*) $=$
semantics-mltl π (*Until-mltl F1 0 0 F2*)
   using *len-gt* $*$
   unfolding *semantics-mltl.simps WEST-reg-aux.simps* **by** *auto*
  then have *?case* **using** *Until*(*1*) $*$ **by** *auto*
 } moreover {assume $*$: $b > 0$
 then have *cp-F1*: *complen-mltl F1* $\leq$ *length* π
  using *complen-mltl.simps*(*10*)[*of F1 a b F2*] *Until* **by** *simp*
 have *F1-nnf*: $\exists\,\psi.\ F1 = convert\text{-}nnf\ \psi$
  using *Until*
  by (*metis convert-nnf.simps*(*8*) *convert-nnf-convert-nnf mltl.inject*(*7*))
 let *?L1* $=$ (*WEST-reg-aux F1 num-vars*)
 have *match-F1*: *match* π *?L1* $=$ *semantics-mltl* π *F1*
  using *Until*(*2*)[*OF cp-F1 F1-nnf F1-nv*] *Until* **by** *auto*
 have *F2-nv*: *WEST-num-vars F2* $\leq$ *num-vars*
  using *Until* **by** *auto*
 have *cp-F2*: *complen-mltl F2* $\leq$ *length* π
  using *complen-mltl.simps*(*10*)[*of F1 a b F2*] *Until* **by** *simp*
 have *F2-nnf*: $\exists\,\psi.\ F2 = convert\text{-}nnf\ \psi$
  using *Until*
  by (*metis convert-nnf.simps*(*8*) *convert-nnf-convert-nnf mltl.inject*(*7*))
 let *?L2* $=$ (*WEST-reg-aux F2 num-vars*)
 have *match-F2*: *match* π *?L2* $=$ *semantics-mltl* π *F2*
  using *Until*(*3*)[*OF cp-F2 F2-nnf F2-nv*] *Until* **by** *simp*
 have *len-xi*: *length* $\pi \geq$ *complen-mltl* (*Until-mltl F1 a b F2*) **using** *Until* **by** *auto*
 then have *?case* **using** *WEST-until-correct*[*of F1 ?L1 F2 ?L2 num-vars a b* π]
  using *Until F1-nv F2-nv cp-F1 cp-F2 F1-nnf F2-nnf match-F1 match-F2*
  using *WEST-reg-aux.simps*(*9*)[*of F1 a b F2 num-vars*] *WEST-reg-aux-num-vars*
  by (*metis* (*no-types, lifting*) *intervals-welldef.simps*(*9*))
 }
 ultimately show *?case* **using** *Until.prems*(*4*) *Until*(*1*)
  by *fastforce*
next
 case (*Release F F1 F2 a b*)
 have *F1-nv*: *WEST-num-vars F1* $\leq$ *num-vars*
  using *Release* **by** *auto*
 {assume $*$: $a = 0 \wedge b = 0$
  have *complen-leq*: *complen-mltl F2* $\leq$ *length* π
   using *Release*(*1*) *Release.prems*(*1*) **by** *simp*
  have *some-nnf*: $\exists\,\psi.\ F2 = convert\text{-}nnf\ \psi$
   using *Release*(*1*) *Release.prems*(*2*)
   by (*metis convert-nnf.simps*(*9*) *convert-nnf-convert-nnf mltl.inject*(*8*))
  have $F2 \in$ *subformulas* (*Until-mltl F1 a b F2*)
   unfolding *subformulas.simps* **by** *blast*
  then have *num-vars*: *WEST-num-vars F2* $\leq$ *num-vars*
   using *Release*(*1*) *Release.prems*(*3*) *WEST-num-vars-subformulas*[*of F2 F*]

by *auto*

have *match-F2*: *match π* (*WEST-reg-aux F2 num-vars*) = *semantics-mltl π F2*

　using *Release*(*1*) *Release*(*3*)[*OF complen-leq some-nnf num-vars*] *Release.prems*

　by *simp*

have *max* (*complen-mltl F1 − 1*) (*complen-mltl F2*) *>= 1*

　using *complen-geq-one*[*of F2*] **by** *auto*

then have *len-gt*: *length π > 0*

　using *Release.prems*(*1*) *Release*(*1*) **by** *auto*

have *global*: *WEST-global* (*WEST-reg-aux F2 num-vars*) *0 0 num-vars = shift* (*WEST-reg-aux F2 num-vars*) *num-vars 0*

　using *WEST-global.simps*[*of - 0 0*] **by** *auto*

have *map* (*λk. arbitrary-state num-vars*) [*0..<0*] = []

　by *simp*

then have *padis*: *shift* (*WEST-reg-aux F2 num-vars*) *num-vars 0 = WEST-reg-aux F2 num-vars*

　unfolding *shift.simps arbitrary-trace.simps* **using** *append.left-neutral list.simps*(*8*) *map-ident upt-0*

proof −

　have (*@*) (*map* (*λn. arbitrary-state num-vars*) ([]::*nat list*)) = (*λwss. wss*)

　　by *blast*

　then show *map* ((*@*) (*map* (*λn. arbitrary-state num-vars*) [*0..<0*])) (*WEST-reg-aux F2 num-vars*) = *WEST-reg-aux F2 num-vars*

　　by *simp*

　qed

then have *match π* (*WEST-global* (*WEST-reg-aux F2 num-vars*) *0 0 num-vars*) =

(*semantics-mltl π F2*)

　using *match-F2 global padis* **by** *simp*

then have *match π* (*WEST-until* (*WEST-reg-aux F1 num-vars*) (*WEST-reg-aux F2 num-vars*) *0 0 num-vars*) =

(*semantics-mltl π F2*)

　using *WEST-until.simps*[*of - - 0 0 num-vars*] **by** *auto*

then have *match π* (*WEST-until* (*WEST-reg-aux F1 num-vars*) (*WEST-reg-aux F2 num-vars*) *0 0 num-vars*) =

(*semantics-mltl* (*drop 0 π*) *F2 ∧* (*∀j. 0 ≤ j ∧ j < 0 ⟶ semantics-mltl* (*drop j π*) *F1*))

　by *auto*

then have *match π* (*WEST-reg-aux* (*Release-mltl F1 0 0 F2*) *num-vars*) = *semantics-mltl π* (*Release-mltl F1 0 0 F2*)

　using *len-gt* *

　unfolding *semantics-mltl.simps WEST-reg-aux.simps* **by** *auto*

then have *?case* **using** *Release*(*1*) *

　by *auto*

} moreover {assume *: *b > 0*

then have *cp-F1*: *complen-mltl F1 ≤ length π*

　using *complen-mltl.simps*(*10*)[*of F1 a b F2*] *Release* **by** *simp*

have *F1-nnf*: *∃ψ. F1 = convert-nnf ψ*

　using *Release*

　by (*metis convert-nnf.simps*(*9*) *convert-nnf-convert-nnf mltl.inject*(*8*))

    let *?L1 = (WEST-reg-aux F1 num-vars)*
    have *match-F1*: *match π ?L1 = semantics-mltl π F1*
      using *Release(2)[OF cp-F1 F1-nnf F1-nv] Release* **by** *auto*
    have *F2-nv*: *WEST-num-vars F2 ≤ num-vars*
      using *Release* **by** *auto*
    have *cp-F2*: *complen-mltl F2 ≤ length π*
      using *complen-mltl.simps(10)[of F1 a b F2] Release* **by** *simp*
    have *F2-nnf*: *∃ψ. F2 = convert-nnf ψ*
      using *Release*
      by (*metis convert-nnf.simps(9) convert-nnf-convert-nnf mltl.inject(8)*)
    let *?L2 = (WEST-reg-aux F2 num-vars)*
    have *match-F2*: *match π ?L2 = semantics-mltl π F2*
      using *Release(3)[OF cp-F2 F2-nnf F2-nv] Release* **by** *simp*
    have *len-xi*: *length π ≥ (max ((complen-mltl F1)−1) (complen-mltl F2)) + b*
using *∗ Release*
    by *auto*
    have *?case* **using** *WEST-release-correct[of F1 ?L1 F2 ?L2 num-vars a b π]*
      using *Release F1-nv F2-nv cp-F1 cp-F2 F1-nnf F2-nnf match-F1 match-F2*
     using *WEST-reg-aux.simps(10)[of F1 a b F2 num-vars] WEST-reg-aux-num-vars*
     by (*metis (full-types) intervals-welldef.simps(10)*)
    }
    ultimately show *?case* **using** *Release(7) Release(1)* **by** *fastforce*
qed

lemma *complen-convert-nnf*:
  shows *complen-mltl (convert-nnf φ) = complen-mltl φ*
proof(*induction depth-mltl φ arbitrary: φ rule: less-induct*)
  case *less*
  then show *?case* **proof** (*cases φ*)
    case *True-mltl*
    then show *?thesis* **by** *simp*
  next
    case *False-mltl*
    then show *?thesis* **by** *simp*
  next
    case (*Prop-mltl p*)
    then show *?thesis* **by** *simp*
  next
    case (*Not-mltl p*)
    then show *?thesis* **proof** (*induct p*)
      case *True-mltl*
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case *False-mltl*
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next
      case (*Prop-mltl x*)
      then show *?case* **using** *Not-mltl less* **by** *auto*
    next

case (*Not-mltl p*)
then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*And-mltl p1 p2*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*Or-mltl p1 p2*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*Future-mltl a b x*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*Global-mltl a b x*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*Until-mltl x a b y*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
next
  case (*Release-mltl x a b y*)
  then show *?case* **using** *Not-mltl less* **by** *auto*
qed
next
  case (*And-mltl x y*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Or-mltl x y*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Future-mltl a b x*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Global-mltl a b x*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Until-mltl x a b y*)
  then show *?thesis* **using** *less* **by** *auto*
next
  case (*Release-mltl x a b y*)
  then show *?thesis* **using** *less* **by** *auto*
qed
qed

lemma *nnf-int-welldef*:
  assumes *intervals-welldef φ*
  shows *intervals-welldef* (*convert-nnf φ*)
  using *assms*
proof (*induct depth-mltl φ arbitrary*: *φ rule*: *less-induct*)
  case *less*

then show *?case* **proof** (*cases* φ)
  case *True-mltl*
  then show *?thesis* **by** *simp*
next
  case *False-mltl*
  then show *?thesis* **by** *simp*
next
  case (*Prop-mltl p*)
  then show *?thesis* **by** *simp*
next
  case (*Not-mltl* ψ)
  then have *phi-is*: $\varphi = Not\text{-}mltl\ \psi$
    by *auto*
  show *?thesis* **proof** (*cases* ψ)
    case *True-mltl*
    then show *?thesis* **using** *Not-mltl* **by** *simp*
  next
    case *False-mltl*
    then show *?thesis* **using** *Not-mltl* **by** *simp*
  next
    case (*Prop-mltl p*)
    then show *?thesis* **using** *Not-mltl* **by** *simp*
  next
    case (*Not-mltl F*)
    then have *iwd*: *intervals-welldef* (*convert-nnf F*)
      using *phi-is less* **by** *simp*
    have $\varphi = Not\text{-}mltl\ (Not\text{-}mltl\ F)$
      using *phi-is Not-mltl* **by** *auto*
    then show *?thesis* **using** *iwd*
      convert-nnf.simps(*13*)[*of F*] **by** *simp*
  next
    case (*And-mltl x y*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  next
    case (*Or-mltl x y*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  next
    case (*Future-mltl a b x*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  next
    case (*Global-mltl a b x*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  next
    case (*Until-mltl x a b y*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  next
    case (*Release-mltl x a b y*)
    then show *?thesis* **using** *Not-mltl less* **by** *simp*
  qed

next
  case (*And-mltl x y*)
  then show *?thesis* **using** *less* **by** *simp*
next
  case (*Or-mltl x y*)
  then show *?thesis* **using** *less* **by** *simp*
next
  case (*Future-mltl a b x*)
  then show *?thesis* **using** *less* **by** *simp*
next
  case (*Global-mltl a b x*)
  then show *?thesis* **using** *less* **by** *simp*
next
  case (*Until-mltl x a b y*)
  then show *?thesis* **using** *less* **by** *simp*
next
  case (*Release-mltl x a b y*)
  then show *?thesis* **using** *less* **by** *simp*
  qed
qed

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−
 let *?n = WEST-num-vars* φ
  have *match* π (*WEST-reg-aux* (*convert-nnf* φ) (*WEST-num-vars* φ)) = *seman-tics-mltl* π (*convert-nnf* φ)
   using *WEST-reg-aux-correct*[*OF assms*(*2*) - - *nnf-int-welldef*, *of WEST-num-vars* φ] *WEST-num-vars-nnf*[*of* φ]
    using *int-welldef* **by** *auto*
  then show *?thesis*
    unfolding *WEST-reg.simps*
   using *WEST-num-vars-nnf*[*of* φ] *convert-nnf-preserves-semantics*[*OF assms*(*1*)]
    by *simp*
qed

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 −
  show *?thesis*
    using *WEST-correct complen-convert-nnf*
    by (*metis π-long-enough assms*(*1*))
qed

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) $\longleftrightarrow$ *semantics-mltl* π φ
proof −
  let *?unpadded* = *WEST-reg* φ
  let *?complen* = *complen-mltl* φ
  let *?num-vars* = *WEST-num-vars* φ
  let *?len* = *length* (*WEST-reg* φ)
  have *pwr-is*: *pad-WEST-reg* φ = (*map* (λL. *if length* $L <$ *?complen*
                then L @ *arbitrary-trace ?num-vars* (*?complen* − *length* L)
                else L) *?unpadded*)
    unfolding *pad-WEST-reg.simps*
    by (*metis* (*no-types, lifting*) *map-equality-iff pad.elims*)
  then have *length ?unpadded* = *length* (*pad-WEST-reg* φ)
    by *auto*
  then have *pwr-k-is*: (*pad-WEST-reg* φ ! k) = (*if length* (*?unpadded*!k) < *?complen*
                then (*?unpadded*!k) @ *arbitrary-trace ?num-vars* (*?complen* −
length (*?unpadded*!k))
                else (*?unpadded*!k)) **if** *k-lt*: $k<length$ (*pad-WEST-reg* φ) **for** k
    using *k-lt pwr-is*
    by *fastforce*
  have *same-len*: *length* (*pad-WEST-reg* φ) = *length* (*WEST-reg* φ)
    unfolding *pad-WEST-reg.simps*
    by (*meson length-map*)
  have *match-regex* π (*if length* (*WEST-reg* φ ! k) < *complen-mltl* φ
    then WEST-reg φ ! k @
      arbitrary-trace (*WEST-num-vars* φ)
      (*complen-mltl* φ − *length* (*WEST-reg* φ ! k))
    else WEST-reg φ ! k) =
    match-regex π (*WEST-reg* φ ! k) **if** *k-lt*: $k <$ *?len* **for** k
  proof −
    {assume ∗: *length* (*WEST-reg* φ ! k) < *complen-mltl* φ
      then have *len-is*: *length* (*WEST-reg* φ ! k @
      arbitrary-trace (*WEST-num-vars* φ)
      (*complen-mltl* φ − *length* (*WEST-reg* φ ! k))) =
      complen-mltl φ
       by *auto*
      have *univ-prop*: $\bigwedge A$ B::$'a$ *list*. ($\forall$ *time<length*

$(A @ B).\ (time \geq length\ A \longrightarrow$
$P\ time)) \Longrightarrow ((\forall\ time<length$
$(A @ B).\ \ P\ time) = (\forall\ time<length$
$A\ .\ P\ time))$ **for** $P::nat \Rightarrow bool$
    by *auto*
  have *match-timestep* $(\pi\ !\ time)$
        $((WEST\text{-}reg\ \varphi\ !\ k\ @$
          arbitrary-trace $(WEST\text{-}num\text{-}vars\ \varphi)$
          $(complen\text{-}mltl\ \varphi\ -$
          $length\ (WEST\text{-}reg\ \varphi\ !\ k))))\ !\ time)$
      if *time-prop*: $time < length\ (WEST\text{-}reg\ \varphi\ !\ k\ @$
          arbitrary-trace $(WEST\text{-}num\text{-}vars\ \varphi)$
              $(complen\text{-}mltl\ \varphi\ -\ length\ (WEST\text{-}reg\ \varphi\ !\ k))) \wedge time \geq length$
$(WEST\text{-}reg\ \varphi\ !\ k)$
    for *time*
    proof $-$
    have *access*: $((WEST\text{-}reg\ \varphi\ !\ k\ @$
          arbitrary-trace $(WEST\text{-}num\text{-}vars\ \varphi)$
          $(complen\text{-}mltl\ \varphi\ -$
          $length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !\ time)$
      $= (arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
          $(complen\text{-}mltl\ \varphi\ -$
          $length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !\ (time - (length\ (WEST\text{-}reg\ \varphi\ !\ k)))$
      using *time-prop*
      by $(meson\ leD\ nth\text{-}append)$
    have $(arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
          $(complen\text{-}mltl\ \varphi\ -$
          $length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !\ (time - (length\ (WEST\text{-}reg\ \varphi\ !\ k)))$
      $= arbitrary\text{-}state\ (WEST\text{-}num\text{-}vars\ \varphi)$
      unfolding *arbitrary-trace.simps* **using** $*$ *time-prop*
      by $(metis\ diff\text{-}less\text{-}mono\ diff\text{-}zero\ len\text{-}is\ nth\text{-}map\text{-}upt)$
    then have *access2*: $((WEST\text{-}reg\ \varphi\ !\ k\ @$
          arbitrary-trace $(WEST\text{-}num\text{-}vars\ \varphi)$
          $(complen\text{-}mltl\ \varphi\ -$
          $length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !\ time)$
      $= arbitrary\text{-}state\ (WEST\text{-}num\text{-}vars\ \varphi)$
      using *access*
      by *auto*
    have *match-timestep* $(\pi\ !\ time)\ (arbitrary\text{-}state\ (WEST\text{-}num\text{-}vars\ \varphi))$
      unfolding *arbitrary-state.simps*
      match-timestep-def **by** *simp*
    then show *?thesis* **using** *access2* **by** *auto*
    qed
  then have $(\forall\ time<length$
          $(WEST\text{-}reg\ \varphi\ !\ k\ @$
          arbitrary-trace $(WEST\text{-}num\text{-}vars\ \varphi)$
           $(complen\text{-}mltl\ \varphi\ -$
            $length\ (WEST\text{-}reg\ \varphi\ !\ k))).$
        match-timestep $(\pi\ !\ time)$

$((WEST\text{-}reg\ \varphi\ !\ k\ @$
$arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
$(complen\text{-}mltl\ \varphi\ -$
$length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !$
$time)) =$
$(\forall\ time{<}length$
$(WEST\text{-}reg\ \varphi\ !\ k).$
$match\text{-}timestep\ (\pi\ !\ time)$
$((WEST\text{-}reg\ \varphi\ !\ k\ @$
$arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
$(complen\text{-}mltl\ \varphi\ -$
$length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !$
$time))$
  using *univ-prop*[*of WEST-reg* φ *! k arbitrary-trace* (*WEST-num-vars* φ)
$(complen\text{-}mltl\ \varphi\ -$
$length\ (WEST\text{-}reg\ \varphi\ !\ k))$]
  by *auto*
then have $(\forall\ time{<}length$
$(WEST\text{-}reg\ \varphi\ !\ k\ @$
$arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
$(complen\text{-}mltl\ \varphi\ -$
$length\ (WEST\text{-}reg\ \varphi\ !\ k))).$
$match\text{-}timestep\ (\pi\ !\ time)$
$((WEST\text{-}reg\ \varphi\ !\ k\ @$
$arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
$(complen\text{-}mltl\ \varphi\ -$
$length\ (WEST\text{-}reg\ \varphi\ !\ k)))\ !$
$time)) =$
$(\forall\ time{<}length\ (WEST\text{-}reg\ \varphi\ !\ k).$
$match\text{-}timestep\ (\pi\ !\ time)$
$(WEST\text{-}reg\ \varphi\ !\ k\ !\ time))$
  by (*simp add*: *nth-append*)
 then have *match-regex* π (*WEST-reg* φ *! k* @
$arbitrary\text{-}trace\ (WEST\text{-}num\text{-}vars\ \varphi)$
$(complen\text{-}mltl\ \varphi\ -\ length\ (WEST\text{-}reg\ \varphi\ !\ k))) =$
match-regex π (*WEST-reg* φ *! k*)
  using *len-is* π*-long-enough* $*$
  unfolding *match-regex-def*
  by *auto*
 then have *?thesis*
  using $*$ **by** *auto*
  }
 moreover {assume $*$: *length* (*WEST-reg* φ *! k*) $\geq$ *complen-mltl* φ
  then have *?thesis* **by** *simp*
  }
 ultimately show *?thesis*
  by *argo*
qed
 then have *match-regex* π (*pad-WEST-reg* φ *! k*) $=$

149

$match\text{-}regex$ π ($WEST\text{-}reg$ φ ! k) **if** $k\text{-}lt$: $k < ?len$ **for** k
    using $pwr\text{-}k\text{-}is$ $k\text{-}lt$ $same\text{-}len$ **by** $presburger$
  then have $match$ π ($pad\text{-}WEST\text{-}reg$ φ) $\longleftrightarrow$ $match$ π ($WEST\text{-}reg$ φ)
    using $\pi\text{-}long\text{-}enough$ $same\text{-}len$
    unfolding $match\text{-}def$
    by $auto$
  then show $?thesis$
    using $assms$ $WEST\text{-}correct\text{-}v2$
    by $blast$
qed

lemma $WEST\text{-}correct\text{-}pad$:
  fixes φ::(nat) $mltl$
  fixes π::$trace$
  assumes $intervals\text{-}welldef$ φ
  assumes $\pi\text{-}long\text{-}enough$: $length$ π $\geq$ $complen\text{-}mltl$ φ
  shows $match$ π ($simp\text{-}pad\text{-}WEST\text{-}reg$ φ) $\longleftrightarrow$ $semantics\text{-}mltl$ π φ
proof $-$
  let $?unpadded = WEST\text{-}reg$ φ
  let $?complen = complen\text{-}mltl$ φ
  let $?num\text{-}vars = WEST\text{-}num\text{-}vars$ φ
  have $pwr\text{-}is$: $pad\text{-}WEST\text{-}reg$ $\varphi = (map$ $(\lambda L.$ if $length$ $L < ?complen$
          $then$ L @ $arbitrary\text{-}trace$ $?num\text{-}vars$ ($?complen - length$ L)
          $else$ L) $?unpadded$)
    unfolding $pad\text{-}WEST\text{-}reg.simps$
    by ($metis$ ($no\text{-}types$, $lifting$) $map\text{-}equality\text{-}iff$ $pad.elims$)
  then have $length$ $?unpadded = length$ ($pad\text{-}WEST\text{-}reg$ φ)
    by $auto$
  then have $pwr\text{-}k\text{-}is$: ($pad\text{-}WEST\text{-}reg$ φ ! k) = (if $length$ ($?unpadded!k$) $< ?complen$
              $then$ ($?unpadded!k$) @ $arbitrary\text{-}trace$ $?num\text{-}vars$ ($?complen -$
$length$ ($?unpadded!k$))
              $else$ ($?unpadded!k$)) **if** $k\text{-}lt$: $k<length$ ($pad\text{-}WEST\text{-}reg$ φ) **for** k
    using $k\text{-}lt$ $pwr\text{-}is$
    by $fastforce$
  have $length$ ($pad\text{-}WEST\text{-}reg$ φ ! k ! i) =
      $WEST\text{-}num\text{-}vars$ φ **if** $i\text{-}is$: $i<length$ ($pad\text{-}WEST\text{-}reg$ φ ! k) $\wedge k<length$
($pad\text{-}WEST\text{-}reg$ φ)
    for i k
  proof $-$
    {assume $*$: $length$ ($?unpadded!k$) $< ?complen$
      then have $pad\text{-}is$: ($pad\text{-}WEST\text{-}reg$ φ ! k) = ($?unpadded!k$) @ $arbitrary\text{-}trace$
$?num\text{-}vars$ ($?complen - length$ ($?unpadded!k$))
        using $pwr\text{-}k\text{-}is$ $that$ **by** $presburger$
      have $regtrace1$: $trace\text{-}regex\text{-}of\text{-}vars$ ($arbitrary\text{-}trace$ ($WEST\text{-}num\text{-}vars$ φ)
    ($complen\text{-}mltl$ $\varphi - length$ ($WEST\text{-}reg$ φ ! k))) ($WEST\text{-}num\text{-}vars$ φ)
        unfolding $arbitrary\text{-}trace.simps$
        $trace\text{-}regex\text{-}of\text{-}vars\text{-}def$
        by $auto$

have *regtrace2*: *trace-regex-of-vars* (*WEST-reg* φ ! *k*) (*WEST-num-vars* φ)
  using *WEST-reg-num-vars*[*OF assms*(*1*)]
by (*metis* ‹*length* (*WEST-reg* φ) = *length* (*pad-WEST-reg* φ)› *WEST-regex-of-vars-def*
that)
  have *?thesis*
    using *pad-is*
    using *regtrace-append*[*OF regtrace1 regtrace2*]
    by (*metis regtrace1 regtrace2 regtrace-append trace-regex-of-vars-def that*)
  } moreover {assume *∗*: *length* (*?unpadded*!*k*) ≥ *?complen*
    then have (*pad-WEST-reg* φ ! *k*) = (*?unpadded*!*k*)
      using *pwr-k-is that* **by** *presburger*
    then have *?thesis*
      using *WEST-reg-num-vars*[*OF assms*(*1*)]
    by (*metis* ‹*length* (*WEST-reg* φ) = *length* (*pad-WEST-reg* φ)› *WEST-regex-of-vars-def*
trace-regex-of-vars-def that)
  }
  ultimately show *?thesis* **by** *linarith*
qed
then have *trace-regex-of-vars* (*pad-WEST-reg* φ ! *k*)
    (*WEST-num-vars* φ) **if** *k-lt*: *k*<*length* (*pad-WEST-reg* φ) **for** *k*
  unfolding *trace-regex-of-vars-def*
  using *k-lt* **by** *auto*
then have *WEST-regex-of-vars* (*pad-WEST-reg* φ)
  (*WEST-num-vars* φ)
  unfolding *WEST-regex-of-vars-def*
  by *blast*
then show *?thesis*
  using *WEST-correct-pad-aux*[*OF assms*]
  unfolding *simp-pad-WEST-reg.simps*
  using *simp-correct*[*of* (*pad-WEST-reg* φ) (*WEST-num-vars* φ) π]
  by *blast*
qed

end

4   Key algorithms for WEST

theory *Regex-Equivalence*

imports *WEST-Algorithms WEST-Proofs*

begin

fun *depth-dataype-list*:: *state-regex* $\Rightarrow$ *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* (λx. *One#x*) (*enumerate-list T*))
  | *enumerate-list* (*Zero#T*) = (*map* (λx. *Zero#x*) (*enumerate-list T*))
  | *enumerate-list* (*S#T*) = (*enumerate-list* (*Zero#T*))@(*enumerate-list* (*One#T*))
  apply (*metis WEST-and-bitwise.elims list.exhaust*)
  by *simp-all*
termination apply (*relation measure* (λL. *depth-dataype-list L*))
  by *simp-all*

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 (λt. (*map* (λh. *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 (λt. (*map* (λh. *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(*induct state-regex*)
  case *Nil*

then show *?case* **by** *simp*
next
  case (*Cons a state-regex*)
  {assume *zero*: *a = Zero*
    then have ∀ *state* ∈ *set* (*enumerate-list state-regex*).
       length state = length state-regex
     using *Cons* **by** *blast*
    then have *?case* **unfolding** *zero*
     by *simp*
  } moreover {
    assume *one*: *a = One*
    then have ∀ *state* ∈ *set* (*enumerate-list state-regex*).
       length state = length state-regex
     using *Cons* **by** *blast*
    then have *?case* **unfolding** *one*
     by *simp*
  } moreover {
    assume *s*: *a = S*
    then have ∀ *state* ∈ *set* (*enumerate-list state-regex*).
       length state = length state-regex
     using *Cons* **by** *blast*
    then have *?case* **unfolding** *s* **by** *auto*
  }
  ultimately show *?case*
   using *WEST-bit.exhaust* **by** *blast*
qed

lemma *enumerate-list-len*:
  assumes *state* ∈ *set* (*enumerate-list state-regex*)
  shows *length state = length state-regex*
  using *assms enumerate-list-len-alt* **by** *blast*

lemma *enumerate-list-prop*:
  assumes (⋀*k*. *List.member j k* ⟹ *k* ≠ *S*)
  shows *enumerate-list j = [j]*
  using *assms*
proof (*induct j*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons h t*)
  then have *elt*: *enumerate-list t = [t]*
   by *simp*
  then have *h = One* ∨ *h = Zero*
   using *Cons* **by** (*cases h*) *auto*
  then show *?case* **using** *enumerate-list.simps*(*2−3*) *elt*
   by *fastforce*

qed

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*]
  using *assms*
proof (*induct h1*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons h t*)
  then have *ind*: *enumerate-trace t* = [*t*]
    by *auto*
  have *enumerate-list h* = [*h*]
    using *enumerate-list-prop Cons*
    by *auto*
  then show *?case*
    using *Cons ind* **unfolding** *enumerate-trace.simps*
    by *auto*
qed

    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* $\implies$ $(\bigwedge k.$ *List.member j k* $\implies k \neq S)$
  assumes $\bigwedge j.$ *List.member h2 j* $\implies$ $(\bigwedge k.$ *List.member j k* $\implies k \neq S)$
  assumes (*set h1*) $\neq$ (*set h2*)
  shows *set* (*enumerate-trace h1*) $\neq$ *set* (*enumerate-trace h2*)
  using *enumerate-fixed-trace*[*of h1*] *enumerate-fixed-trace*[*of h2*] *assms*
  by *auto*

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 −
  {assume *one*: *h* = *One*
    have *?thesis*
      using *assms* **unfolding** *one enumerate-list.simps* **by** *auto*
  } moreover {
    assume *zero*: *h* = *Zero*
    have *?thesis*
      using *assms* **unfolding** *zero enumerate-list.simps* **by** *auto*
  } moreover {
    assume *s*: *h* = *S*
    have *?thesis*
      using *assms* **unfolding** *s enumerate-list.simps* **by** *auto*

```
      }
    ultimately show ?thesis using WEST-bit.exhaust by blast
qed


lemma enumerate-list-fixed:
  assumes t ∈ set (enumerate-list trace)
  shows (∀ k. List.member t k ⟶ k ≠ S)
  using assms
proof (induct trace arbitrary: t)
  case Nil
  then show ?case by simp
next
  case (Cons h trace)
  obtain head-t tail-t where obt: t = head-t#tail-t
    using Cons.prems enumerate-list-len
    by (metis length-0-conv neq-Nil-conv)
  have tail-t ∈ set (enumerate-list trace)
    using enumerate-list.simps obt Cons.prems enumerate-list-tail-in by blast
  then have hyp: ∀ k. List.member tail-t k ⟶ k ≠ S
    using Cons.hyps(1)[of tail-t] by auto
  {assume one: h = One
    then have head-t = One
      using obt Cons.prems unfolding enumerate-list.simps by auto
    then have ?case
      using hyp obt
      by simp
  } moreover {
    assume zero: h = Zero
    then have head-t = Zero
      using obt Cons.prems unfolding enumerate-list.simps by auto
    then have ?case
      using hyp obt
      by simp
  } moreover {
    assume s: h = S
    then have head-t = Zero ∨ head-t = One
      using obt Cons.prems unfolding enumerate-list.simps by auto
    then have ?case
      using hyp obt s by auto
  }
  ultimately show ?case using WEST-bit.exhaust by blast
qed


lemma map-enum-list-nonempty:
  fixes t::WEST-bit list list
  fixes head::WEST-bit list
  shows map (λh. h # t) (enumerate-list head) ≠ []
proof(induct head arbitrary: t)
```

case *Nil*
then show *?case* **by** *simp*
next
  case (*Cons a head*)
  {**assume** *a*: *a = One*
    then have *?case* **unfolding** *a enumerate-list.simps*
      using *Cons* **by** *auto*
  } **moreover** {
    assume *a*: *a = Zero*
    then have *?case* **unfolding** *a enumerate-list.simps*
      using *Cons* **by** *auto*
  } **moreover** {
    assume *a*: *a = S*
    then have *?case* **unfolding** *a enumerate-list.simps*
      using *Cons* **by** *auto*
  }
  ultimately show *?case* **using** *WEST-bit.exhaust* **by** *blast*
qed

lemma *length-of-flatten-list*:
assumes *flat =*
  foldr (@)
  (*map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)\ H$) *T*) []
shows *length flat = length T $*$ length H*
  using *assms*
proof (*induct T arbitrary*: *flat*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons t1 T2*)
  then have *flat = foldr* (@)
    (*map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)\ H$) (*t1 $\#$ T2*)) []
    by *auto*
  then have *flat = foldr* (@)
    (*map* ($\lambda h.\ h\ \#\ t1$) *H $\#$(map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)\ H$) *T2*)) []
    by *auto*
  then have *flat = map* ($\lambda h.\ h\ \#\ t1$) *H* @ (*foldr* (@) (*map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)$
H) *T2*)) []
    by *simp*
  then have *length flat = length H + length* (*T2*) $*$ *length H*
    using *Cons* **by** *auto*
  then show *?case* **by** *simp*
qed

lemma *flatten-list-idx*:
  assumes *flat = flatten-list* (*map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)\ head$) *tail*)

assumes *i* < *length tail*
assumes *j* < *length head*
shows (*head*!*j*)#(*tail*!*i*) = *flat*!(*i*∗(*length head*) + *j*) ∧ *i*∗(*length head*) + *j* <
length flat
using *assms*
proof(*induct tail arbitrary*: *head i j flat*)
  case *Nil*
  then show *?case*
    by *auto*
next
  case (*Cons a tail*)
  let *?flat* = *flatten-list* (*map* (λ*t. map* (λ*h. h # t*) *head*) *tail*)
  have *cond1*: *?flat* = *?flat* **by** *auto*
  have *equiv*: (*map* (λ*t. map* (λ*h. h # t*) *head*) (*a # tail*)) =
    (*map* (λ*h. h # a*) *head*) # (*map* (λ*t. map* (λ*h. h # t*) *head*) *tail*)
    by *auto*
  then have *flat-is*: *flat* = (*map* (λ*h. h # a*) *head*) @ *flatten-list* (*map* (λ*t. map*
(λ*h. h # t*) *head*) *tail*)
    using *Cons*(*2*) **unfolding** *flatten-list.simps* **by** *simp*

  {**assume** *i0*: *i* = *0*
    then have *bound*: *i* ∗ *length head* + *j* < *length flat*
      using *Cons* **by** *simp*
    have *length* (*map* (λ*h. h # a*) *head*) > *j*
      using *Cons*(*4*) **by** *auto*
    then have (*map* (λ*h. h # a*) *head*) ! *j* = *flat* ! *j*
      using *flat-is*
      by (*simp add*: *nth-append*)
    then have (*head* ! *j*)#*a* = *flat* ! *j*
      using *Cons*(*4*) **by** *simp*
    then have *head* ! *j* # (*a # tail*) ! *i* = *flat* ! (*i* ∗ *length head* + *j*)
      unfolding *i0* **by** *simp*
    then have *?case* **using** *bound* **by** *auto*
  } **moreover** {
    assume *i-ge-0*: *i* > *0*
    have *len-flat*: *length flat* = *length head* ∗ *length* (*a # tail*)
     using *Cons*(*3−4*) *length-of-flatten-list*[*of flat head a#tail*]
     Cons(*2*) **unfolding** *flatten-list.simps*
     by *simp*
    have *i* ∗ *length head* ≤ (*length* (*a # tail*) − *1*)∗*length head*
     using *Cons*(*3*) **by** *auto*
    then have *i* ∗ *length head* ≤ (*length* (*a # tail*))∗*length head* − *length head*
     by *auto*
    then have *i* ∗ *length head* + *j* < (*length* (*a # tail*))∗*length head* − *length head*
+ *length head*
     using *Cons*(*4*) **by** *linarith*
    then have *i* ∗ *length head* + *j* < (*length* (*a # tail*))∗*length head*
     by *auto*
    then have *bound*: *i* ∗ *length head* + *j* < *length flat*

using *len-flat*
by (*simp add*: *mult.commute*)
have *i-minus*: $i - 1 < length\ tail$
using *i-ge-0 Cons(3)*
by *auto*
have *flat* ! ($i * length\ head + j$) = *flat* ! (($i-1$) * $length\ head + j + length\ head$)
using *i-ge-0*
by (*smt* (*z3*) *add.commute bot-nat-0.not-eq-extremum group-cancel.add1 mult-eq-if*)
then have *flat* ! ($i * length\ head + j$) = *flatten-list*
(*map* ($\lambda t.\ map$ ($\lambda h.\ h\ \#\ t$) *head*) *tail*) !
(($i - 1$) * $length\ head + j$)
using *flat-is*
by (*smt* (*verit*, *ccfv-threshold*) *add.commute length-map nth-append-length-plus*)
then have *flat* ! ($i * length\ head + j$) = *head* ! $j\ \#\ tail$! ($i - 1$)
using *Cons.hyps*[*OF cond1 i-minus Cons(4)*]
by *argo*
then have *access*: *head* ! $j\ \#\ (a\ \#\ tail)$! i =
flat ! ($i * length\ head + j$)
using *i-ge-0*
by *simp*
have *?case*
using *bound access*
by *auto*
}
ultimately show *?case* **by** *blast*
qed

lemma *flatten-list-shape*:
assumes *List.member flat x1*
assumes *flat* = *flatten-list* (*map* ($\lambda t.\ map$ ($\lambda h.\ h\ \#\ t$) *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
using *assms*
proof(*induction T arbitrary*: *flat H*)
case *Nil*
have *flat* = (*flatten-list* (*map* ($\lambda t.\ map$ ($\lambda h.\ h\ \#\ t$) *H*) [])))
using *Nil(1)* **unfolding** *Nil* **by** *blast*
then have *flat* = []
by *simp*
then show *?case*
using *Nil*
by *simp*
next
case (*Cons a T*)
have $\exists k.\ x1 = flat$! $k \wedge k < length\ flat$
using *Cons(2)*

by (*auto simp add*: *in-set-conv-nth*)
then obtain *k* **where** *k-is*: *x1 = flat ! k ∧ k < length flat*
  by *auto*
have *len-flat*: *length flat = (length (a#T)∗length H)*
  using *Cons(3) length-of-flatten-list*
  by *auto*
let *?j = k mod (length H)*
have *∃i . k = (i∗(length H)+?j)*
  by (*meson mod-div-decomp*)
then obtain *i* **where** *i-is*: *k = (i∗(length H)+?j)*
  by *auto*
then have *i-lt*: *i < length (a#T)*
  using *len-flat k-is*
  by (*metis add-lessD1 mult-less-cancel2*)
have *j-lt*: *?j < length H*
 by (*metis k-is len-flat length-0-conv length-greater-0-conv mod-by-0 mod-less-divisor mult-0-right*)
have *∃i < length (a # T). k = (i∗(length H)+?j)*
  using *i-is i-lt* **by** *blast*
then have *∃i < length (a # T). ∃j < length H. k = (i∗(length H)+j)*
  using *j-lt* **by** *blast*
then obtain *i j* **where** *ij-props*: *i < length (a#T) j < length H k = (i∗(length H)+j)*
  by *blast*
then have *flat ! k = H ! j # (a # T) ! i*
  using *flatten-list-idx[OF Cons(3) ij-props(1) ij-props(2)]*
    Cons(2) k-is ij-props(3)
  by *argo*
then obtain *x1-head x1-tail* **where** *x1 = x1-head#x1-tail*
and *List.member H x1-head* **and** *List.member (a#T) x1-tail*
  using *ij-props*
  by (*simp add*: *index-of-L-in-L k-is*) (*metis length-Cons nth-mem set-ConsD*)
then show *?case*
  using *Cons(3)* **by** *simp*
qed

lemma *flatten-list-len*:
  assumes *⋀t. List.member T t ⟹ length t = n*
  assumes *flat = flatten-list (map (λt. map (λh. h # t) H) T)*
  shows *⋀x1. List.member flat x1 ⟹ length x1 = n+1*
  using *assms*
proof(*induction T arbitrary*: *flat n H*)
  case *Nil*
  have *flat = (flatten-list (map (λt. map (λh. h # t) H) []))*
    using *Nil(1)* **unfolding** *Nil(3)* **by** *blast*
  then have *flat = []*
    by *simp*
  then show *?case*

159

    using *Nil* **by** *simp*
next
  case (*Cons a T*)
  have $\exists\, k.\ x1 = flat\ !\ k \wedge k < length\ flat$
    using *Cons*(*2*)
    by (*auto simp add: in-set-conv-nth*)
  then obtain *k* **where** *k-is*: $x1 = flat\ !\ k \wedge k < length\ flat$
    by *auto*
  have *len-flat*: $length\ flat = (length\ (a\#T)*length\ H)$
    using *Cons*(*4*) *length-of-flatten-list*
    by *auto*
  let $?j = k\ mod\ (length\ H)$
  have $\exists\, i\ .\ k = (i*(length\ H)+?j)$
    by (*meson mod-div-decomp*)
  then obtain *i* **where** *i-is*: $k = (i*(length\ H)+?j)$
    by *auto*
  then have *i-lt*: $i < length\ (a\#T)$
    using *len-flat k-is*
    by (*metis add-lessD1 mult-less-cancel2*)
  have *j-lt*: $?j < length\ H$
  by (*metis k-is len-flat length-0-conv length-greater-0-conv mod-by-0 mod-less-divisor mult-0-right*)
  have $\exists\, i < length\ (a\ \#\ T).\ k = (i*(length\ H)+?j)$
    using *i-is i-lt* **by** *blast*
  then have $\exists\, i < length\ (a\ \#\ T).\ \exists\, j < length\ H.\ k = (i*(length\ H)+j)$
    using *j-lt* **by** *blast*
  then obtain *i j* **where** *ij-props*: $i < length\ (a\#T)\ j < length\ H\ k = (i*(length\ H)+j)$
    by *blast*
  then have $flat\ !\ k = H\ !\ j\ \#\ (a\ \#\ T)\ !\ i$
    using *flatten-list-idx*[*OF Cons*(*4*) *ij-props*(*1*) *ij-props*(*2*)]
      Cons(*2*) *k-is ij-props*(*3*)
    by *argo*
  then obtain *x1-head x1-tail* **where** $x1 = x1\text{-}head\#x1\text{-}tail$
  and *List.member H x1-head* **and** *List.member* (*a*#*T*) *x1-tail*
    using *ij-props*
    by (*simp add: index-of-L-in-L k-is*) (*metis length-Cons nth-mem set-ConsD*)
  then show *?case*
    using *Cons*(*3*) **by** *simp*
qed

lemma *flatten-list-lemma*:
  assumes $\bigwedge x1.\ List.member\ to\text{-}flatten\ x1 \implies (\bigwedge x2.\ List.member\ x1\ x2 \implies length\ x2 = length\ trace)$
  assumes $a \in set\ (flatten\text{-}list\ to\text{-}flatten)$
  shows $length\ a = length\ trace$
  using *assms* **proof** (*induct to-flatten*)
  case *Nil*

then show *?case* **by** *auto*
next
  case (*Cons h t*)
   have *a-in*: $a \in set\ h \lor a \in set\ (flatten\text{-}list\ t)$
    using *Cons*(*3*) **unfolding** *flatten-list.simps foldr-def* **by** *simp*
  {assume ∗: $a \in set\ h$
   then have *?case*
    using *Cons*(*2*)[*of h*]
    by *simp*
  } moreover {assume ∗: $a \in set\ (flatten\text{-}list\ t)$
   have *ind-h-setup*: $(\bigwedge x1\ x2.\ List.member\ t\ x1 \implies List.member\ x1\ x2 \implies$
    $length\ x2 = length\ trace)$
    using *Cons*(*2*) **by** *auto*
   have $a \in set\ (flatten\text{-}list\ t) \implies length\ a = length\ trace$
    using *Cons*(*1*) *ind-h-setup*
    by *auto*
   then have *?case*
    using ∗ **by** *auto*
  }
  ultimately show *?case*
   using *a-in* **by** *blast*
qed

lemma *enumerate-trace-len*:
  assumes $a \in set\ (enumerate\text{-}trace\ trace)$
  shows $length\ a = length\ trace$
  using *assms*
proof(*induct length trace arbitrary*: *trace a*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc x*)
  then obtain *h t* **where** *trace-is*: $trace = h\#t$
   by (*meson Suc-length-conv*)
  obtain *i* **where** $(enumerate\text{-}trace\ trace)!i = a$
   using *Suc.prems*
   by (*meson in-set-conv-nth*)
  let *?enumerate-H* = *enumerate-list h*
  let *?enumerate-t* = *enumerate-trace t*
  have *enum-tr-is*: *enumerate-trace trace* =
    flatten-list (*map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)$ *?enumerate-H*) *?enumerate-t*)
   using *trace-is* **by** *auto*
  let *?to-flatten* = *map* ($\lambda t.\ map\ (\lambda h.\ h\ \#\ t)$ *?enumerate-H*) *?enumerate-t*

  have *all-w*: $(\bigwedge w.\ List.member\ (enumerate\text{-}trace\ t)\ w \implies length\ w = length\ t)$
   using *Suc*(*1*)[*of t*] *Suc*(*2*) *trace-is*
   by *simp*
  have *a-mem*: *List.member* (*enumerate-trace trace*) *a*

161

using *Suc(3)* **by** *simp*
  show *?case*
    using *flatten-list-len[OF - enum-tr-is a-mem, of length t] all-w*
    trace-is **by** *simp*
qed

definition *regex-zeros-and-ones:: trace-regex ⇒ bool*
  where *regex-zeros-and-ones tr =*
    (∀ j. List.member tr j ⟶ (∀ k. List.member j k ⟶ k ≠ S))

lemma *match-enumerate-state-aux-first-bit*:
  assumes *a-head = Zero ∨ a-head = One*
  assumes *a-head # a-tail ∈ set (enumerate-list (h-head # h))*
  shows *h-head = a-head ∨ h-head = S*
proof −
  {assume *h-head*: *h-head = One*
    then have *?thesis*
      using *assms* **unfolding** *h-head enumerate-list.simps* **by** *auto*
  } moreover {
    assume *h-head*: *h-head = Zero*
    then have *?thesis*
      using *assms* **unfolding** *h-head enumerate-list.simps* **by** *auto*
  } moreover {
    assume *h-head = S*
    then have *?thesis* **by** *auto*
  }
  ultimately show *?thesis* **using** *WEST-bit.exhaust* **by** *blast*
qed

lemma *advance-state-iff*:
  assumes *x > 0*
  shows *x ∈ state ⟷ (x−1) ∈ advance-state state*
proof −
  have *forward*: *x ∈ state ⟶ (x−1) ∈ advance-state state*
    using *assms* **by** *auto*
  have *converse*: *(x−1) ∈ advance-state state ⟶ x ∈ state*
    unfolding *advance-state.simps* **using** *assms*
      by *(smt (verit, best) Suc-diff-1 diff-0-eq-0 diff-Suc-1′ diff-self-eq-0 less-one*
mem-Collect-eq nat.distinct(1) not0-implies-Suc not-gr-zero old.nat.exhaust)
  show *?thesis* **using** *forward converse* **by** *blast*
qed

lemma *match-enumerate-state-aux*:
  assumes *a ∈ set (enumerate-list h)*
  assumes *match-timestep state a*
  shows *match-timestep state h*
  using *assms*
proof (*induct h arbitrary*: *state a*)

case *Nil*
have $a = []$
  using *Nil* **by** *auto*
then show *?case* **using** *Nil* **by** *blast*
next
  case (*Cons h-head h*)
  then obtain *a-head a-tail* **where** *obt*: $a = a\text{-}head \# a\text{-}tail$
    using *enumerate-list-len Cons*
    by (*metis length-0-conv list.exhaust*)
  let *?adv-state* = *advance-state state*
  {**assume** *in-state*: $0 \in state$
    then have *a-head* = *One*
      using *Cons*(*2*) *Cons.prems*(*2*) *obt*
    by (*cases ‹a-head›*) (*auto simp add*: *match-timestep-def dest*: *enumerate-list-fixed*)
    then have *h-head*: *h-head* = *One* $\lor$ *h-head* = *S*
      using *Cons.prems*(*1*) **unfolding** *obt*
      using *match-enumerate-state-aux-first-bit* **by** *blast*
    have *match-adv*: *match-timestep* (*advance-state state*) *h*
      using *Cons.hyps*[*of a-tail ?adv-state*]
      using *Cons.prems*(*1*) *Cons.prems*(*2*) *advance-state-match-timestep enumerate-list-tail-in obt* **by** *blast*
    have $\bigwedge x.\ x{<}length\ (h\text{-}head\ \#\ h) \implies$
      $((h\text{-}head\ \#\ h)\ !\ x = One \longrightarrow x \in state)\ \land$
      $((h\text{-}head\ \#\ h)\ !\ x = Zero \longrightarrow x \notin state)$
    proof $-$
      fix *x*
      assume *x*: $x{<}length\ (h\text{-}head\ \#\ h)$
      let *?thesis* = $((h\text{-}head\ \#\ h)\ !\ x = One \longrightarrow x \in state)\ \land$
      $((h\text{-}head\ \#\ h)\ !\ x = Zero \longrightarrow x \notin state)$
      {**assume** *x0*: $x = 0$
        then have *?thesis* **unfolding** *x0* **using** *h-head in-state* **by** *auto*
      } **moreover** {
        assume *x-ge-0*: $x > 0$
        then have $x{-}1 < length\ h$
          using *x* **by** *simp*
        then have $*$:$(h\ !\ (x{-}1) = One \longrightarrow (x{-}1) \in advance\text{-}state\ state)\ \land$
            $(h\ !\ (x{-}1) = Zero \longrightarrow (x{-}1) \notin advance\text{-}state\ state)$
          using *match-adv* **unfolding** *match-timestep-def* **by** *blast*
        have $h\ !\ (x{-}1) = (h\text{-}head\ \#\ h)\ !\ x$ **using** *x-ge-0* **by** *auto*
        then have $*$: $((h\text{-}head\ \#\ h)\ !\ x = One \longrightarrow (x{-}1) \in advance\text{-}state\ state)\ \land$
            $((h\text{-}head\ \#\ h)\ !\ x = Zero \longrightarrow (x{-}1) \notin advance\text{-}state\ state)$
          using $*$ **by** *argo*
        then have *?thesis* **using** *advance-state-iff x-ge-0* **by** *blast*
      }
      ultimately show *?thesis* **by** *blast*
    qed
    then have *?case*
      using *h-head* **unfolding** *match-timestep-def* **by** *blast*
  } **moreover** {

163

    assume *not-in*: *0* ∉ *state*
    then have *a-head* = *Zero*
      using *Cons.prems obt*
    by (*cases a-head*) (*auto simp add*: *match-timestep-def dest*: *enumerate-list-fixed*)
    then have *h-head*: *h-head* = *Zero* ∨ *h-head* = *S*
      using *Cons.prems*(*1*) **unfolding** *obt*
      using *match-enumerate-state-aux-first-bit* **by** *blast*
    have *match-adv*: *match-timestep* (*advance-state state*) *h*
      using *Cons.hyps*[*of a-tail ?adv-state*]
      using *Cons.prems*(*1*) *Cons.prems*(*2*) *advance-state-match-timestep enumer-*
ate-list-tail-in obt **by** *blast*
    have ⋀*x*. *x*<*length* (*h-head* # *h*) ⟹
      ((*h-head* # *h*) ! *x* = *One* ⟶ *x* ∈ *state*) ∧
      ((*h-head* # *h*) ! *x* = *Zero* ⟶ *x* ∉ *state*)
    proof −
      fix *x*
      assume *x*: *x*<*length* (*h-head* # *h*)
      let *?thesis* = ((*h-head* # *h*) ! *x* = *One* ⟶ *x* ∈ *state*) ∧
      ((*h-head* # *h*) ! *x* = *Zero* ⟶ *x* ∉ *state*)
      {**assume** *x0*: *x* = *0*
        then have *?thesis* **unfolding** *x0* **using** *h-head not-in* **by** *auto*
      } **moreover** {
        assume *x-ge-0*: *x* > *0*
        then have *x−1* < *length h*
          using *x* **by** *simp*
        then have ∗:(*h* ! (*x−1*) = *One* ⟶ (*x−1*) ∈ *advance-state state*) ∧
              (*h* ! (*x−1*) = *Zero* ⟶ (*x−1*) ∉ *advance-state state*)
          using *match-adv* **unfolding** *match-timestep-def* **by** *blast*
        have *h* ! (*x−1*) = (*h-head* # *h*) ! *x* **using** *x-ge-0* **by** *auto*
        then have ∗: ((*h-head* # *h*) ! *x* = *One* ⟶ (*x−1*) ∈ *advance-state state*) ∧
              ((*h-head* # *h*) ! *x* = *Zero* ⟶ (*x−1*) ∉ *advance-state state*)
          using ∗ **by** *argo*
        then have *?thesis* **using** *advance-state-iff x-ge-0* **by** *blast*
      }
      ultimately show *?thesis* **by** *blast*
    qed
    then have *?case*
      using *h-head* **unfolding** *match-timestep-def* **by** *blast*
  }
  ultimately show *?case* **using** *WEST-bit.exhaust* **by** *blast*
qed

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*) ∧
    (*length* (*enumerate-list a*) + *j* < *length* (*enumerate-list* (*S* # *a*)))
  using *assms*

proof(*induct a arbitrary: j*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons a1 a2*)
  then show *?case* **unfolding** *enumerate-list.simps*
    by (*metis* (*mono-tags, lifting*) *length-append length-map nat-add-left-cancel-less nth-append-length-plus nth-map*)
qed

lemma *list-concat-index*:
  assumes *j < length L1*
  shows (*L1@L2*)!*j = L1*!*j*
  using *assms*
  by (*simp add: nth-append*)

lemma *enumerate-list-index-zero*:
  assumes *j < length* (*enumerate-list a*)
  shows *Zero # enumerate-list a* ! *j = enumerate-list* (*S # a*) ! *j* ∧
    j < length (*enumerate-list* (*S # a*))
  using *assms* **unfolding** *enumerate-list.simps*
proof(*induct a arbitrary: j*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a1 a2*)
  then have *j-bound: j < length* (*enumerate-list* (*S # a1 # a2*))
    by *simp*
  let *?subgoal = Zero # enumerate-list* (*a1 # a2*) ! *j = enumerate-list* (*S # a1 # a2*) ! *j*
  have *j < length* (*map* ((#) *Zero*) (*enumerate-list* (*a1 # a2*)))
    using *j-bound Cons* **by** *simp*
  then have (((*map* ((#) *Zero*) (*enumerate-list* (*a1 # a2*)) @
    map ((#) *One*) (*enumerate-list* (*a1 # a2*)))) !
    j) = (*map* ((#) *Zero*) (*enumerate-list* (*a1 # a2*)))!*j*
    using *Cons.prems j-bound list-concat-index* **by** *blast*
  then have *?subgoal* **using** *Cons* **unfolding** *enumerate-list.simps*
    by *simp*
  then show *?case* **using** *j-bound* **by** *auto*
qed

lemma *match-enumerate-list*:
  assumes *match-timestep state a*
  shows ∃*j<length* (*enumerate-list a*).
    match-timestep state (*enumerate-list a* ! *j*)
  using *assms*
proof(*induct a arbitrary: state*)
  case *Nil*

then show *?case* **by** *simp*
next
  case (*Cons head a*)
  let *?adv-state = advance-state state*
  {**assume** *in-state*: *0 ∈ state*
   then have (*head # a*) *! 0 ≠ Zero*
    using *Cons.prems* **unfolding** *match-timestep-def* **by** *blast*
   then have *head*: *head = One ∨ head = S*
    using *WEST-bit.exhaust* **by** *auto*
   have *match-timestep ?adv-state a*
    using *Cons.prems*
    using *advance-state-match-timestep* **by** *auto*
   then obtain *j* **where** *obt*: *match-timestep ?adv-state* (*enumerate-list a ! j*)
                ∧ j < length (*enumerate-list a*)
    using *Cons.hyps*[*of ?adv-state*] **by** *blast*
  let *?state =* (*enumerate-list a ! j*)
  {**assume** *headcase*: *head = One*
   let *?s = One # ?state*
   have ⋀*x. x<length ?s ⟹*
   ((*?s ! x = One ⟶ x ∈ state*) *∧* (*?s ! x = Zero ⟶ x ∉ state*))
   proof −
    fix *x*
    assume *x*: *x<length ?s*
   let *?thesis =* ((*?s ! x = One ⟶ x ∈ state*) *∧* (*?s ! x = Zero ⟶ x ∉ state*))
    {**assume** *x0*: *x = 0*
     then have *?thesis* **using** *in-state* **by** *simp*
    } **moreover** {
     assume *x-ge-0*: *x > 0*
     have *cond1*: (*One = One ⟶ 0 ∈ state*) *∧* (*One = Zero ⟶ 0 ∉ state*)
      using *in-state* **by** *blast*
     have *cond2*: *∀ x<length* (*enumerate-list a ! j*).
  (*enumerate-list a ! j ! x = One ⟶ x + 1 ∈ state*) *∧*
  (*enumerate-list a ! j ! x = Zero ⟶ x + 1 ∉ state*)
       using *obt* **unfolding** *match-timestep-def advance-state-iff* **by** *fastforce*
     have *x<length* (*One # enumerate-list a ! j*)
      using *x* **by** *blast*
     then have *?thesis*
      using *index-shift*[*of One state ?state, OF cond1 cond2*] **by** *blast*
    }
    ultimately show *?thesis* **by** *blast*
   qed
   then have *match*: *match-timestep state ?s*
    using *obt headcase in-state* **unfolding** *match-timestep-def* **by** *blast*
   have (*map* ((*#*) *One*) (*enumerate-list a*) *! j*) *= One #* (*enumerate-list a ! j*)
    using *obt* **by** *simp*
   then have *?case* **unfolding** *headcase enumerate-list.simps*
    using *match obt* **by** *auto*
  } **moreover** {
   assume *headcase*: *head = S*

let *?s = One # ?state*
have $\bigwedge x.$ *x<length ?s* $\Longrightarrow$
((?s ! x = One $\longrightarrow$ *x* $\in$ *state)* $\land$ *(?s ! x = Zero* $\longrightarrow$ *x* $\notin$ *state))*
proof−
  fix *x*
  assume *x*: *x<length ?s*
 let *?thesis = ((?s ! x = One* $\longrightarrow$ *x* $\in$ *state)* $\land$ *(?s ! x = Zero* $\longrightarrow$ *x* $\notin$ *state))*
  {assume *x0*: *x = 0*
   then have *?thesis* **using** *in-state* **by** *simp*
  } moreover {
   assume *x-ge-0*: *x > 0*
   have *cond1*: *(One = One* $\longrightarrow$ *0* $\in$ *state)* $\land$ *(One = Zero* $\longrightarrow$ *0* $\notin$ *state)*
    using *in-state* **by** *blast*
   have *cond2*: $\forall$ *x<length (enumerate-list a ! j).*
(enumerate-list a ! j ! x = One $\longrightarrow$ *x + 1* $\in$ *state)* $\land$
(enumerate-list a ! j ! x = Zero $\longrightarrow$ *x + 1* $\notin$ *state)*
    using *obt* **unfolding** *match-timestep-def advance-state-iff* **by** *fastforce*
   have *x<length (One # enumerate-list a ! j)*
    using *x* **by** *blast*
   then have *?thesis*
    using *index-shift[of One state ?state, OF cond1 cond2]* **by** *blast*
  }
  ultimately show *?thesis* **by** *blast*
qed
then have *match*: *match-timestep state ?s*
  using *obt headcase in-state* **unfolding** *match-timestep-def* **by** *blast*
have $\bigwedge x.$ *x<length (S # enumerate-list a ! j)* $\Longrightarrow$
((S # enumerate-list a ! j) ! x = One $\longrightarrow$ *x* $\in$ *state)* $\land$
((S # enumerate-list a ! j) ! x = Zero $\longrightarrow$ *x* $\notin$ *state)*
proof−
  fix *x*
  assume *x*: *x<length (S # enumerate-list a ! j)*
  let *?thesis = ((S # enumerate-list a ! j) ! x = One* $\longrightarrow$ *x* $\in$ *state)* $\land$
((S # enumerate-list a ! j) ! x = Zero $\longrightarrow$ *x* $\notin$ *state)*
  {assume *x0*: *x = 0*
   then have *?thesis* **by** *auto*
  } moreover {
   assume *x-ge-0*: *x > 0*
   then have *?thesis* **using** *x match* **unfolding** *match-timestep-def* **by** *simp*
  }
  ultimately show *?thesis* **by** *blast*
qed
then have *match-S*: *match-timestep state (S # enumerate-list a ! j)*
  using *match* **unfolding** *match-timestep-def* **by** *blast*
have *j-bound*: *j < length (enumerate-list a)*
  using *obt* **by** *blast*
have *?s = enumerate-list (S # a)!((length (enumerate-list a))+j)*
   $\land$ *(length (enumerate-list a))+j < length (enumerate-list (S # a))*
  using *j-bound enumerate-list-index-one* **by** *blast*

167

then have *?case* **unfolding** *headcase*
  using *match obt match-S* **by** *metis*
}
ultimately have *?case* **using** *head* **by** *blast*
} moreover {
assume *not-in*: *0 ∉ state*
then have (*head # a*) *! 0 ≠ One*
  using *Cons.prems* **unfolding** *match-timestep-def* **by** *blast*
then have *head*: *head = Zero ∨ head = S*
  using *WEST-bit.exhaust* **by** *auto*
have *match-timestep ?adv-state a*
  using *Cons.prems*
  using *advance-state-match-timestep* **by** *auto*
then obtain *j* **where** *obt*: *match-timestep ?adv-state* (*enumerate-list a ! j*)
                   ∧ j < length (*enumerate-list a*)
  using *Cons.hyps*[*of ?adv-state*] **by** *blast*
let *?state = *(*enumerate-list a ! j*)
{assume *headcase*: *head = Zero*
  let *?s = Zero # ?state*
  have $\bigwedge$*x. x<length ?s* $\Longrightarrow$
  ((*?s ! x = One* $\longrightarrow$ *x ∈ state*) *∧* (*?s ! x = Zero* $\longrightarrow$ *x ∉ state*))
  proof −
    fix *x*
    assume *x*: *x<length ?s*
   let *?thesis = *((*?s ! x = One* $\longrightarrow$ *x ∈ state*) *∧* (*?s ! x = Zero* $\longrightarrow$ *x ∉ state*))
    {assume *x0*: *x = 0*
      then have *?thesis* **using** *not-in headcase* **by** *simp*
    } moreover {
      assume *x-ge-0*: *x > 0*
      have *cond1*: (*Zero = One* $\longrightarrow$ *0 ∈ state*) *∧* (*Zero = Zero* $\longrightarrow$ *0 ∉ state*)
        using *not-in* **by** *blast*
      have *cond2*: *∀ x<length* (*enumerate-list a ! j*).
  (*enumerate-list a ! j ! x = One* $\longrightarrow$ *x + 1 ∈ state*) *∧*
  (*enumerate-list a ! j ! x = Zero* $\longrightarrow$ *x + 1 ∉ state*)
        using *obt* **unfolding** *match-timestep-def advance-state-iff* **by** *fastforce*
      have *x<length* (*Zero # enumerate-list a ! j*)
        using *x* **by** *blast*
      then have *?thesis*
        using *index-shift*[*of Zero state ?state, OF cond1 cond2*] **by** *blast*
    }
    ultimately show *?thesis* **by** *blast*
  qed
  then have *match*: *match-timestep state ?s*
    using *obt headcase not-in* **unfolding** *match-timestep-def* **by** *blast*
  have *?case* **unfolding** *headcase enumerate-list.simps*
    using *match obt* **by** *auto*
} moreover {
assume *headcase*: *head = S*
let *?s = Zero # ?state*

168

have $\bigwedge x.\ x{<}length\ ?s \implies$
$((?s\ !\ x = One \longrightarrow x \in state) \wedge (?s\ !\ x = Zero \longrightarrow x \notin state))$
proof $-$
  fix x
  assume *x*: $x{<}length\ ?s$
 let *?thesis* $= ((?s\ !\ x = One \longrightarrow x \in state) \wedge (?s\ !\ x = Zero \longrightarrow x \notin state))$
  {assume *x0*: $x = 0$
   then have *?thesis* **using** *not-in* **by** *simp*
  } moreover {
   assume *x-ge-0*: $x > 0$
   have *cond1*: $(Zero = One \longrightarrow 0 \in state) \wedge (Zero = Zero \longrightarrow 0 \notin state)$
    using *not-in* **by** *blast*
   have *cond2*: $\forall\, x{<}length\ (enumerate\text{-}list\ a\ !\ j).$
$(enumerate\text{-}list\ a\ !\ j\ !\ x = One \longrightarrow x + 1 \in state) \wedge$
$(enumerate\text{-}list\ a\ !\ j\ !\ x = Zero \longrightarrow x + 1 \notin state)$
    using *obt* **unfolding** *match-timestep-def advance-state-iff* **by** *fastforce*
   have $x{<}length\ (Zero\ \#\ enumerate\text{-}list\ a\ !\ j)$
    using *x* **by** *blast*
   then have *?thesis*
    using *index-shift*[*of Zero state ?state, OF cond1 cond2*] **by** *blast*
  }
  ultimately show *?thesis* **by** *blast*
qed
then have *match*: *match-timestep state ?s*
 using *obt headcase not-in* **unfolding** *match-timestep-def* **by** *blast*
have $\bigwedge x.\ x{<}length\ (S\ \#\ enumerate\text{-}list\ a\ !\ j) \implies$
$((S\ \#\ enumerate\text{-}list\ a\ !\ j)\ !\ x = One \longrightarrow x \in state) \wedge$
$((S\ \#\ enumerate\text{-}list\ a\ !\ j)\ !\ x = Zero \longrightarrow x \notin state)$
proof $-$
  fix x
  assume *x*: $x{<}length\ (S\ \#\ enumerate\text{-}list\ a\ !\ j)$
  let *?thesis* $= ((S\ \#\ enumerate\text{-}list\ a\ !\ j)\ !\ x = One \longrightarrow x \in state) \wedge$
$((S\ \#\ enumerate\text{-}list\ a\ !\ j)\ !\ x = Zero \longrightarrow x \notin state)$
  {assume *x0*: $x = 0$
   then have *?thesis* **by** *auto*
  } moreover {
   assume *x-ge-0*: $x > 0$
   then have *?thesis* **using** *x match* **unfolding** *match-timestep-def* **by** *simp*
  }
  ultimately show *?thesis* **by** *blast*
qed
then have *match-S*: *match-timestep state* $(S\ \#\ enumerate\text{-}list\ a\ !\ j)$
 using *match* **unfolding** *match-timestep-def* **by** *blast*
have *j-bound*: $j < length\ (enumerate\text{-}list\ a)$
 using *obt* **by** *blast*
have $?s = enumerate\text{-}list\ (S\ \#\ a)!(j)$
    $\wedge\ j < length\ (enumerate\text{-}list\ (S\ \#\ a))$
 using *j-bound enumerate-list-index-zero* **by** *blast*
then have *?case* **unfolding** *headcase*

using *match obt match-S* **by** *metis*
    }
    ultimately have *?case* **using** *head* **by** *blast*
  }
  ultimately show *?case* **by** *blast*
qed

lemma *enumerate-trace-head-in*:
  assumes *a-head # a-tail ∈ set* (*enumerate-trace* (*h # trace*))
  shows *a-head ∈ set* (*enumerate-list h*)
proof −
    let *?flat = flatten-list*
    (*map* (*λt. map* (*λh. h # t*)
          (*enumerate-list h*))
     (*enumerate-trace trace*))
    have *flat-is*: *?flat = ?flat*
     by *auto*
    have *mem*: *List.member*
     ?flat
     (*a-head # a-tail*)
     using *assms* **by** *simp*
    then obtain *x1-head x1-tail* **where**
     x1-props: *a-head # a-tail = x1-head # x1-tail ∧*
     List.member (*enumerate-list h*) *x1-head ∧*
     List.member (*enumerate-trace trace*) *x1-tail*
     using *flatten-list-shape*[*OF mem flat-is*] **by** *auto*
    then have *a-head = x1-head*
     by *auto*
    then have *List.member* (*enumerate-list h*) *a-head*
     using *x1-props*
     by *auto*
    then show *?thesis*
     by *simp*
qed

lemma *enumerate-trace-tail-in*:
  assumes *a-head # a-tail ∈ set* (*enumerate-trace* (*h # trace*))
  shows *a-tail ∈ set* (*enumerate-trace trace*)
proof −
    let *?flat = flatten-list*
    (*map* (*λt. map* (*λh. h # t*)
          (*enumerate-list h*))
     (*enumerate-trace trace*))
    have *flat-is*: *?flat = ?flat*
     by *auto*
    have *mem*: *List.member*
     ?flat

```
        (a-head # a-tail)
          using assms by simp
      then obtain x1-head x1-tail where
        x1-props: a-head # a-tail = x1-head # x1-tail ∧
        List.member (enumerate-list h) x1-head ∧
        List.member (enumerate-trace trace) x1-tail
          using flatten-list-shape[OF mem flat-is]  by auto
      then have a-tail = x1-tail
        by auto
      then have List.member (enumerate-trace trace) a-tail
        using x1-props
        by auto
      then show ?thesis
        by simp
qed
```

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 ∈ set (enumerate-trace trace)
  assumes match-regex π a
  shows match-regex π trace
proof −
  show ?thesis using assms proof (induct trace arbitrary: a π)
    case Nil
    then show ?case by auto
  next
    case (Cons h trace)
    then obtain a-head a-tail where obt-a: a = a-head#a-tail
      using enumerate-trace-len
      by (metis length-0-conv neq-Nil-conv)
    have length π > 0
      using Cons unfolding match-regex-def obt-a by auto
    then obtain π-head π-tail where obt-π: π = π-head#π-tail
      using min-list.cases by auto
    have cond1: a-tail ∈ set (enumerate-trace trace)
      using Cons.prems(1) unfolding obt-a
      using enumerate-trace-tail-in by blast
    have cond2: match-regex π-tail a-tail
      using Cons.prems(2) unfolding obt-a obt-π match-regex-def by auto
    have match-tail: match-regex π-tail trace
      using Cons.hyps[OF cond1 cond2] by blast
    have a-head: a-head ∈ set (enumerate-list h)
      using Cons.prems(1) unfolding obt-a
      using enumerate-trace-head-in by blast
    have match-timestep π-head a-head
      using Cons.prems(2) unfolding obt-π match-regex-def
      using obt-a by auto
```

171

then have *match-head*: *match-timestep π-head h*
  using *match-enumerate-state-aux*[*of a-head h π-head*] *a-head* **by** *blast*
have $\bigwedge$*time. time<length (h # trace)* $\Longrightarrow$
    match-timestep ((π-head # π-tail) ! time) ((h # trace) ! time)
proof −
  fix *time*
  assume *time*: *time<length (h # trace)*
  let *?thesis = match-timestep ((π-head # π-tail) ! time) ((h # trace) ! time)*
  {**assume** *time0*: *time = 0*
    then have *?thesis* **using** *match-head* **by** *simp*
  } **moreover** {
    assume *time-ge-0*: *time > 0*
    then have *?thesis*
      using *match-tail time-ge-0 time* **unfolding** *match-regex-def* **by** *simp*
  }
  ultimately show *?thesis* **by** *blast*
qed
then show *?case* **using** *match-tail* **unfolding** *match-regex-def obt-a obt-π*
  by *simp*
  qed
qed

lemma *match-enumerate-trace*:
  assumes *a* ∈ *set (enumerate-trace h)* ∧ *match-regex π a*
  shows *match π (h # T)*
proof −
  show *?thesis*
    unfolding *match-def*
    using *match-enumerate-trace-aux assms*
    by *auto*
qed

lemma *match-enumerate-sets1*:
  assumes (∃ *r* ∈ (*enumerate-sets R*). *match-regex π r*)
  shows (*match π R*)
  using *assms*
proof (*induct R*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons h T*)
  then obtain *a* **where** *a-prop*: *a*∈*set (enumerate-trace h)* ∪ *enumerate-sets T* ∧
match-regex π a
    by *auto*
  { **assume** ∗: *a* ∈ *set (enumerate-trace h)*
    then have *?case*
      using *match-enumerate-trace a-prop*

172

by *blast*
  } moreover {assume ∗: *a* ∈ *enumerate-sets T*
    then have *match π T*
      using *Cons a-prop* **by** *blast*
    then have *?case*
      by (*metis Suc-leI le-imp-less-Suc length-Cons match-def nth-Cons-Suc*)
  }
  ultimately show *?case*
    using *a-prop* **by** *auto*
qed

lemma *match-cases*:
  assumes *match π (a # R)*
  shows *match π [a] ∨ match π R*
proof −
  obtain *i* **where** *obt*: *match-regex π ((a # R)!i) ∧ i < length (a # R)*
    using *assms* **unfolding** *match-def* **by** *blast*
  {assume *i0*: *i = 0*
    then have *?thesis*
      using *assms* **unfolding** *match-def* **using** *obt* **by** *simp*
  } moreover {
    assume *i-ge-0*: *i > 0*
    then have *match-regex π (R ! (i−1))*
      using *assms obt* **unfolding** *match-def* **by** *simp*
    then have *match π R*
      unfolding *match-def* **using** *obt i-ge-0*
      by (*metis Suc-diff-1 Suc-less-eq length-Cons*)
    then have *?thesis* **by** *blast*
  }
  ultimately show *?thesis* **using** *assms* **unfolding** *match-def* **by** *blast*
qed

lemma *enumerate-trace-decompose*:
  assumes *state* ∈ *set (enumerate-list h)*
  assumes *trace* ∈ *set (enumerate-trace T)*
  shows *state#trace* ∈ *set (enumerate-trace (h#T))*
proof −
  let *?enumh = enumerate-list h*
  let *?enumT = enumerate-trace T*
  let *?flat = flatten-list (map (λt. map (λh. h # t) ?enumh) ?enumT)*
  have *enum*: *enumerate-trace (h#T) = ?flat*
    unfolding *enumerate-trace.simps* **by** *simp*
  obtain *i* **where** *i*: *?enumT!i = trace ∧ i < length ?enumT*
    using *assms(2)* **by** (*meson in-set-conv-nth*)
  obtain *j* **where** *j*: *?enumh!j = state ∧ j < length ?enumh*
    using *assms(1)* **by** (*meson in-set-conv-nth*)
  have *enumerate-list h ! j # enumerate-trace T ! i =*
    flatten-list (map (λt. map (λh. h # t) (enumerate-list h)) (enumerate-trace T))

173

!
    $(i * length$ $(enumerate\text{-}list\ h) + j)\ \wedge$
    $i * length$ $(enumerate\text{-}list\ h) + j$
    $<\ length$
      $(flatten\text{-}list$
        $(map\ (\lambda t.\ map\ (\lambda h.\ h\ \#\ t)\ (enumerate\text{-}list\ h))\ (enumerate\text{-}trace\ T)))$
    using *flatten-list-idx*[*of ?flat ?enumh ?enumT i j*] *enum i j* **by** *blast*
  then show *?thesis*
    using *i j enum* **by** *simp*
qed

lemma *match-enumerate-trace-aux-converse*:
  assumes *match-regex π trace*
  shows *match π* (*enumerate-trace trace*)
  using *assms*
proof(*induct trace arbitrary*: *π*)
  case *Nil*
  have *enum*: *enumerate-trace* [] = [[]]
    by *simp*
  show *?case* **unfolding** *enum match-def match-regex-def* **by** *auto*
next
  case (*Cons a trace*)
  have *length π > 0*
    using *Cons.prems* **unfolding** *match-regex-def* **by** *auto*
  then obtain *pi-head pi-tail* **where** *pi-obt*: *π = pi-head#pi-tail*
    using *list.exhaust* **by** *auto*
  have *cond*: *match-regex pi-tail trace*
    using *Cons.prems pi-obt* **unfolding** *match-regex-def* **by** *auto*
  then have *match-tail*: *match pi-tail* (*enumerate-trace trace*)
    using *Cons.hyps* **by** *blast*
  then obtain *i* **where** *obt-i*: *match-regex pi-tail* (*enumerate-trace trace ! i*) $\wedge$
      $i<length$ (*enumerate-trace trace*)
    unfolding *match-def* **by** *blast*
  let *?enum-tail* = (*enumerate-trace trace ! i*)

  have *match-head*: *match-timestep pi-head a*
    using *Cons.prems* **unfolding** *match-regex-def*
    by (*metis Cons.prems WEST-and-trace-correct-forward-aux nth-Cons' pi-obt*)
  then have $\exists j < length$ (*enumerate-list a*).
      match-timestep pi-head ((*enumerate-list a*)!*j*)
    using *match-enumerate-list* **by** *blast*
  then obtain *j* **where** *obt-j*: *match-timestep pi-head* ((*enumerate-list a*)!*j*) $\wedge$
        $j < length$ (*enumerate-list a*)
    by *blast*
  let *?enum-head* = (*enumerate-list a*)!*j*

  have (*?enum-head#?enum-tail*) $\in set$(*enumerate-trace* (*a # trace*))
    using *enumerate-trace-decompose*

174

    by (*meson in-set-conv-nth obt-i obt-j*)
  have *match-tail*: *match-regex pi-tail ?enum-tail*
    using *obt-i* **by** *blast*
  have *match-head*: *match-timestep pi-head* ((*enumerate-list a*)!*j*)
    using *obt-j* **by** *blast*
  have *match*: *match-regex π* (*?enum-head#?enum-tail*)
    using *match-head match-tail*
   using *WEST-and-trace-correct-forward-aux-converse*[*OF pi-obt match-head match-tail*]
by *auto*
  let *?flat = flatten-list*
    (*map* (*λt. map* (*λh. h # t*) (*enumerate-list a*))
      (*enumerate-trace trace*))
  have *enumerate-list a* ! *j # enumerate-trace trace* ! *i =*
  flatten-list
  (*map* (*λt. map* (*λh. h # t*) (*enumerate-list a*)) (*enumerate-trace trace*)) !
  (*i ∗ length* (*enumerate-list a*) + *j*) ∧
  i ∗ length (*enumerate-list a*) + *j*
  < *length*
    (*flatten-list*
      (*map* (*λt. map* (*λh. h # t*) (*enumerate-list a*)) (*enumerate-trace trace*)))
    using *flatten-list-idx*[*of ?flat enumerate-list a enumerate-trace trace i j*]
    using *obt-i obt-j* **by** *blast*
  then show *?case*
    unfolding *match-def* **using** *match*
    by *auto*
qed

lemma *match-enumerate-sets2*:
  assumes (*match π R*)
  shows (∃ *r* ∈ *enumerate-sets R. match-regex π r*)
  using *assms*
proof(*induct R arbitrary*: *π*)
  case *Nil*
  then show *?case* **unfolding** *match-def* **by** *auto*
next
  case (*Cons a R*)
  have *enumerate-sets* (*a # R*) = *set* (*enumerate-trace a*) ∪ *enumerate-sets R*
    unfolding *enumerate-sets.simps* **by** *blast*
  {assume *match-a*: *match π* [*a*]
    then have *match-regex π a*
      unfolding *match-def* **by** *simp*
    then have *match π* (*enumerate-trace a*)
      using *match-enumerate-trace-aux*
      using *match-enumerate-trace-aux-converse* **by** *blast*
    then have ∃ *b* ∈ *set* (*enumerate-trace a*). *match-regex π b*
      unfolding *match-def* **by** *auto*
    then have *?case* **by** *auto*
  } moreover {

```
    assume match-R: match π R
    then have ?case
      using Cons by auto
  }
  ultimately show ?case
    using Cons.prems match-cases by blast
qed


lemma match-enumerate-sets:
  shows (∃ r ∈ enumerate-sets R. match-regex π r) ⟷ (match π R)
  using match-enumerate-sets1 match-enumerate-sets2
  by blast

lemma regex-equivalence-correct1:
  assumes (naive-equivalence A B)
  shows match π A = match π B
  unfolding regex-equiv-def
  using match-enumerate-sets[of A π] match-enumerate-sets[of B π]
  using assms
  unfolding naive-equivalence.simps
  by blast


lemma regex-equivalence-correct:
  shows (naive-equivalence A B) ⟶ (regex-equiv A B)
  using regex-equivalence-correct1
  unfolding regex-equiv-def
  by metis


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.